

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Hellas Cube: Ανάπτυξη CLI για την εξαγωγή και
ανάλυση περιβαλλοντικών δεδομένων μέσω του Open
Data Cube»



Hellas Cube

Του φοιτητή
Σαπουνά Χρήστου
Αρ. Μητρώου: 2021152

Επιβλέπων
Δρ. Μπράτσας Χαράλαμπος
Καθηγητής

Ημερομηνία 04/09/2026

Τίτλος Δ.Ε.: Hellas Cube: Ανάπτυξη CLI για την εξαγωγή και ανάλυση περιβαλλοντικών δεδομένων μέσω του Open Data Cube

Κωδικός Δ.Ε.: 26145

Ονοματεπώνυμο φοιτητή: Χρήστος Σαπουνάς

Ονοματεπώνυμο εισηγητή: Μπράτσας Χαράλαμπος

Ημερομηνία ανάληψης Δ.Ε.: 07/03/2026

Ημερομηνία περάτωσης Δ.Ε.: 04/09/2026

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Χρήστου Σαπουνά που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραίτητως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

«Σε όλους τους συμφοιτητές μου»

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στο πλαίσιο της ολοκλήρωσης των προπτυχιακών μου σπουδών στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων της Σχολής Μηχανικών του Διεθνούς Πανεπιστημίου της Ελλάδος. Το σύνολο της ερευνητικής και πρακτικής διαδικασίας πραγματοποιήθηκε υπό την επίβλεψη και καθοδήγηση του Καθηγητή Δρ. Χαράλαμπου Μπράτσα.

Αντικείμενο και κύριος στόχος της εργασίας είναι η σχεδίαση και ανάπτυξη του "Hellas Cube", ενός ολοκληρωμένου συστήματος και εργαλείου Διεπαφής Γραμμής Εντολών για την εξαγωγή, επεξεργασία και ανάλυση ανοικτών περιβαλλοντικών δεδομένων μέσω του framework Open Data Cube. Μέσω του συστήματος αυτού, επιδιώκεται η παροχή μιας αποδοτικής λύσης επεξεργασίας τεράστιου όγκου δορυφορικών δεδομένων. Η διπλωματική αναπτύσσει το θεωρητικό υπόβαθρο γύρω από τα δορυφορικά δεδομένα, τους περιβαλλοντικούς δείκτες και ορισμένες εφαρμογές που χρησιμοποιούν το ίδιο framework. Έτσι, αφού καλυφθεί το απαραίτητο θεωρητικό υπόβαθρο, αναλύεται το ενδιάμεσο στάδιο του σχεδιασμού με τις αντίστοιχες απαιτήσεις και την αρχιτεκτονική που επιλέχθηκε. Η εργασία ολοκληρώνεται με το πρακτικό κομμάτι του συστήματος, όπου παρουσιάζεται η ανάλυση της λογικής του κώδικα, καθώς και τελικά αποτελέσματα.

Περίληψη

Η παρούσα διπλωματική εργασία πραγματεύεται τον σχεδιασμό και την ανάπτυξη του Hellas Cube, ενός ολοκληρωμένου συστήματος ανοικτού κώδικα για την αυτοματοποιημένη εξαγωγή, διαχείριση και ανάλυση πολυδιάστατων δορυφορικών δεδομένων Παρατήρησης της Γης (Big Earth Data). Στόχος του λογισμικού είναι η ουσιαστική διευκόλυνση της επιστημονικής κοινότητας στην περιβαλλοντική παρακολούθηση του Ελλαδικού χώρου, μετατρέποντας δορυφορικά δεδομένα σε αξιοποιήσιμα πορίσματα. Αρχιτεκτονικά, το σύστημα υιοθετεί ένα μοντέλο τριών διακριτών επιπέδων. Η αλληλεπίδραση με τον τελικό χρήστη πραγματοποιείται μέσω μιας Διεπαφής Γραμμής Εντολών (CLI) ανεπτυγμένης σε γλώσσα Rust, η οποία εξασφαλίζει αμεσότητα, εφαρμογή αυστηρών κανόνων επικύρωσης και δυνατότητα ενσωμάτωσης σε αυτοματοποιημένα σενάρια. Το CLI επικοινωνεί με έναν κεντρικό διακομιστή (REST API), ο οποίος λειτουργεί ως API Gateway για την ασφαλή ταυτοποίηση χρηστών, τη διαχείριση ιδιωτικών γεωχώρων και την επικοινωνία με τη σχεσιακή βάση δεδομένων PostgreSQL. Ο υπολογιστικός πυρήνας του συστήματος υλοποιήθηκε σε Python, ενσωματώνοντας το ισχυρό πλαίσιο του Open Data Cube (ODC). Το εσωτερικό αυτό υποσύστημα αναλαμβάνει τη δυναμική ευρετηρίαση (μέσω STAC) και τη φόρτωση δεδομένων από τις δορυφορικές αποστολές Sentinel-2, Landsat 8/9 και το εναρμονισμένο προϊόν HLS. Αξιοποιώντας τεχνικές κατανεμημένης επεξεργασίας και πολυδιάστατων πινάκων, το σύστημα εκτελεί ταχύτατους υπολογισμούς ευρείας κλίμακας περιβαλλοντικών δεικτών (όπως NDVI, NDWI, NDBI, NBR) και εξειδικευμένων αλγορίθμων (όπως ο WofS για πλημμύρες/ξηρασία). Τα παραγόμενα στατιστικά αποτελέσματα παρουσιάζονται δομημένα στο τερματικό και εξάγονται μαζικά σε μορφή CSV, καθιστώντας το Hellas Cube ένα εργαλείο για την επιστημονική έρευνα.

«Hellas Cube: Development of a Command Line Interface for the export and analysis of environmental data through Open Data Cube»

«Christos Sapounas»

Abstract

The volume of Earth Observation data produced by satellite missions is growing exponentially, making its processing a significant challenge for researchers and scientists. This thesis presents "Hellas Cube," a layered software system designed to facilitate the extraction, processing, and analysis of open environmental satellite data. To address the complexities of spatial Big Data, the system utilizes the Open Data Cube (ODC) framework to index and manage large-scale satellite imagery from missions such as Sentinel-2 and Landsat 8/9. The architecture is divided into three layers to maximize performance and scalability. The backend analytical subsystem, developed in Python, computes environmental indices and performs water observation frequency analysis for flood and surface-water monitoring. The middle layer is a high-performance, asynchronous RESTful API built in Rust that orchestrates requests to the analytical subsystem and communicates with a dedicated PostgreSQL database to manage user accounts, API keys, and execution logs. Finally, users interact with the system through a fast Command Line Interface (CLI), also developed in Rust, which supports authentication, the declaration of custom geographical areas via GeoJSON, and automated analysis workflows with minimal computational overhead on local machines. The system can be applied to Greek municipalities and protected areas. The resulting open-source tool provides reliable, easily exportable analytical data, streamlining workflows for environmental monitoring and contributing directly to the Earth observation community.

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή μου, Δρ. Χαράλαμπο Μπράτσα, για την καθοδήγηση, την εμπιστοσύνη και την υποστήριξή του καθ' όλη τη διάρκεια εκπόνησης της παρούσας διπλωματικής εργασίας.

Ένα μεγάλο ευχαριστώ από καρδιάς οφείλω στην οικογένειά μου, για τη συνεχή συμπαράσταση, την υπομονή και την πίστη τους σε μένα σε όλα τα χρόνια των σπουδών μου.

Τέλος, ευχαριστώ θερμά τους φίλους μου για την πολύτιμη ηθική τους στήριξη, την εμπύχωση και την κατανόησή τους σε αυτή την απαιτητική διαδρομή.

Περιεχόμενα

Πρόλογος	5
Περίληψη	6
Abstract	7
Ευχαριστίες	8
Περιεχόμενα	9
Κατάλογος Εικόνων	11
Κατάλογος Πινάκων	13
Συντομογραφίες	14
Κεφάλαιο 1ο: Εισαγωγή	1
1.1 Πρωταρχική Ιδέα	1
1.2 Στόχοι	2
1.3 Δομή & Παραδοτέο	2
1.4 Προκλήσεις	4
Κεφάλαιο 2ο: Η Αρχή και η Παγκόσμια Εξέλιξη των Data Cube	5
2.1 Data Cube	5
2.2 Data Cube ανά τον κόσμο	6
2.3 Υποδομές για Ανάλυση Δεδομένων	10
Κεφάλαιο 3ο: Δορυφορικά Δεδομένα και Περιβαλλοντικοί Δείκτες	11
3.1 Δορυφορικά Δεδομένα	11
3.2 Δορυφορικές Αποστολές	12
3.3 Κατηγορίες Περιβαλλοντικών Δεικτών	18
3.4 Περιβαλλοντικοί Δείκτες & Αλγόριθμος του Hellas Cube	18
3.5 Επιλογή δεικτών και αλγορίθμων	21
Κεφάλαιο 4ο: Ανάλυση Απαιτήσεων Συστήματος	23
4.1 Πλαίσιο Απαιτήσεων	23
4.2 Λειτουργικές Απαιτήσεις (Functional Requirements)	24
4.3 Μη-λειτουργικές Απαιτήσεις (Non-Functional Requirements)	26
Κεφάλαιο 5ο: Τεχνολογίες και Αρχιτεκτονική του Συστήματος	29
5.1 Υψηλού επιπέδου Αρχιτεκτονική	29
5.2 Παράδειγμα Ροής Εκτέλεσης (Use Case)	32
5.3 Η γλώσσα προγραμματισμού Rust	34
5.4 Η γλώσσα προγραμματισμού Python	36
5.5 Το framework Open Data Cube	38
5.6 PostgreSQL	39
5.7 Visual Studio Code (VSC)	40
5.8 Git και GitHub	40
Κεφάλαιο 6ο: Στήσιμο Open Data Cube	42
6.1 Οδηγίες Εγκατάστασης Open Data Cube	42
6.2 Setup Βάσης Δεδομένων Open Data Cube	42
6.3 Setup Περιβάλλοντος	45

Κεφάλαιο 7ο: Ανάπτυξη Python Υποσυστήματος Ανάλυσης	52
7.1 Προδιαγραφές Υποσυστήματος	52
7.2 Αρχιτεκτονική του επιπέδου (Python Layer)	52
7.3 Ευρετηρίαση δεδομένων (Indexing Data)	53
7.4 Φόρτωση datasets	56
7.5 Πρώτο test	58
7.6 Δομή αρχείων Συστήματος Ανάλυσης	59
7.7 Constants	60
7.8 Χάρτες Γεωγραφικών Δεδομένων (Geographic Data Maps)	61
7.9 Η Βασική Κλάση (Parent Class)	62
7.10 Dataset Importer	62
7.11 Data Manager	64
7.12 Ο Επιστημονικός Πυρήνας: Η κλάση env_id	67
7.13 API	74
7.14 Παράδειγμα τελικού αποτελέσματος	76
7.15 Αξιολόγηση	76
Κεφάλαιο 8ο: Η Βάση Δεδομένων του Hellas Cube	80
8.1 RDBMS και Περιβάλλον Ανάπτυξης Βάσης	80
8.2 Δομή Πινάκων	80
8.3 API keys	82
8.4 Τελικό Schema	84
Κεφάλαιο 9ο: Ανάπτυξη του Rust API	86
9.1 Ρόλος και Τεχνολογίες του API	86
9.2 Αρχιτεκτονική Rust API	86
9.3 Εκκίνηση του API	88
9.4 Τομέας Αιτημάτων Ανάλυσης	91
9.5 Τομέας Εκτέλεσης Αιτημάτων Χρηστών	94
9.6 Αξιολόγηση API	96
Κεφάλαιο 10ο: Ανάπτυξη του Command Line Interface	97
10.1 Ρόλος CLI	97
10.2 Αρχιτεκτονική των Εντολών του CLI	97
10.3 Εκτέλεση Εντολών	99
10.4 Εξαγωγή δεδομένων	103
Κεφάλαιο 11ο: Χρήση του συστήματος	109
11.1 Εγκατάσταση εργαλείου	109
11.2 Σενάριο Α : Χρήση Χωρίς Λογαριασμό	110
11.3 Σενάριο Β : Διαχείριση Λογαριασμού	110
11.4 Σενάριο Γ : Χρήση με Λογαριασμό	111
11.5 Σενάριο Δ : Εξαγωγή Ιστορικού	112
Κεφάλαιο 12ο: Συμπεράσματα και Μελλοντικές Επεκτάσεις	114
12.1 Συμπεράσματα	114
12.2 Μελλοντικές Επεκτάσεις	114
ΒΙΒΛΙΟΓΡΑΦΙΑ	116
ΠΑΡΑΡΤΗΜΑ Α: Πηγαίος Κώδικας του Συστήματος Hellas Cube	120

Κατάλογος Εικόνων

Εικόνα 2.1: Η ροή της διαδικασίας από ακατέργαστα δεδομένα σε προϊόντα του Data Cube[1].	5
Εικόνα 2.2: Ετήσιες ακτογραμμές από το DE Africa Coastlines	7
Εικόνα 2.3: Γραφικό Περιβάλλον EL-BIOS	9
Εικόνα 3.1: Δορυφορική εικόνα τηλεπισκόπησης ενός τυπικού αγροκτήματος	11
Εικόνα 3.2: Επεξεργαστικά επίπεδα από το Level-0 έως το Level-2a	14
Εικόνα 3.3: Δορυφόροι του HLS	15
Εικόνα 3.4: Σύνοψη συνολικής ακρίβειας (ποσοστό %) (fF= Fmask, A= ATCOR, S= Sen2Cor). [18]	16
Εικόνα 3.5: Σύνθετη εικόνα HLS SWIR της Ρώμης. Λήψη στις 24-07-2022, επεξεργασμένη μέσω του Sentinel Hub.	17
Εικόνα 4.1: Ανάλυση κατηγοριών απαιτήσεων ενός συστήματος.	24
Εικόνα 4.2: Παράδειγμα help command ενός εργαλείου CLI.	24
Εικόνα 4.3: Διάγραμμα περιπτώσεων χρήσης με την χρήση UML	25
Εικόνα 4.4: Ανάλυση κύκλου απαιτήσεων συστήματος.	28
Εικόνα 5.1: Παράδειγμα αρχιτεκτονικής ενός συστήματος Web	29
Εικόνα 5.2: Επίπεδα αρχιτεκτονικής Hellas Cube	30
Εικόνα 5.3: Διάγραμμα αρχιτεκτονικής CLI	31
Εικόνα 5.4: Κύκλος ζωής εκτέλεσης μιας εντολής	33
Εικόνα 5.5: Rust, η τομή ανάμεσα σε C++ και Java	34
Εικόνα 5.6: Διάγραμμα Ιδιοκτησίας μνήμης της Rust	35
Εικόνα 5.7: Χαρακτηριστικά Object Oriented Programming (OOP)	37
Εικόνα 5.8: Use Case του Open Data Cube [31]	38
Εικόνα 5.9: Παράδειγμα πίνακα σχεσιακής βάσης	39
Εικόνα 5.10: Επέκταση Git στο Visual Studio Code.	41
Εικόνα 6.1: Η σελίδα Setup της ιστοσελίδας του Open Data Cube	42
Εικόνα 6.2: Κατάσταση PostgreSQL μετά από την εκτέλεση της εντολής status	43
Εικόνα 6.3 : Εντολή & αποτέλεσμα login χρήστη PostgreSQL	44
Εικόνα 6.4: Εντολές επιβεβαίωσης δημιουργίας βάση δεδομένων	45
Εικόνα 6.5: Ενεργοποίηση περιβάλλοντος	45
Εικόνα 6.6: Επιβεβαίωση εγκατάστασης των εργαλείων.	46
Εικόνα 6.7: Αρχείο για το Sentinel-2-L2A στο repository του DEA	49
Εικόνα 6.8: Επεξεργασία των HLS προϊόντων	49
Εικόνα 6.10: Προσθήκη προϊόντων στο datacube	51
Εικόνα 6.11: Εμφάνιση λίστας προϊόντων	51
Εικόνα 6.12: Αρχικοποίηση Βάσης και Προϊόντων	51
Εικόνα 7.1: Ιδέα αρχιτεκτονικής Υποσυστήματος Ανάλυσης	53
Εικόνα 7.2: Εμφάνιση στοιχείων	54
Εικόνα 7.3: Αναλυτική μορφή αποθηκευμένου εγγράφου	55
Εικόνα 7.4: Script εκτέλεσης εντολής	56
Εικόνα 7.5 Φόρτωση ενός dataset (όπου de ένα αντικείμενο τύπου DataCube)	57
Εικόνα 7.6: Το xarray. Dataset μετά το dc.load()	58

Εικόνα 7.7: Προγραμματιστική εφαρμογή τύπου NDVI στο xarray.Dataset	58
Εικόνα 7.8: Αποτελέσματα δοκιμής	59
Εικόνα 7.9 Αλλαγή εικόνας με τελικό αποτέλεσμα	59
Εικόνα 7.10: Υπόδειξη αρχείου index	60
Εικόνα 7.11: Αρχεία OverpassQL και geojson	61
Εικόνα 7.12: Παράδειγμα εκτέλεσης overpass query	62
Εικόνα 7.13: Γονική κλάση BaseODCService	62
Εικόνα 7.14: Κατασκευαστής κλάσης check_data()	63
Εικόνα 7.15: Έλεγχος και ανακτηση odc_geom & desired_aoi_geometry	63
Εικόνα 7.16: Αναζήτηση και Ανακτηση datasets	63
Εικόνα 7.17: Φόρτωση datasets με την βοήθεια	65
Εικόνα 7.18: Μέθοδος για φόρτωση dataset με περιβάλλον	65
Εικόνα 7.19: Αρχείο αρχικοποίησης μεταβλητών περιβάλλοντος	66
Εικόνα 7.20: Επιστρεφόμενα στατιστικά της stats()	67
Εικόνα 7.23: Εφαρμογή landsat_qa_clean_mask του datacube.utils	68
Εικόνα 7.25: Συνάρτηση εφαρμογής μάσκας αλγορίθμου fmask	69
Εικόνα 7.26: Περίπτωση εφαρμογής περιβαλλοντικού δείκτη σε δεδομένα sentinel	70
Εικόνα 7.27: Περίπτωση ανάλυσης δεδομένων Landsat	71
Εικόνα 7.28: Περίπτωση ανάλυσης δεδομένων HLS	71
Εικόνα 7.29: Προετοιμασία δεδομένων και εφαρμογή επιπρόσθετων μασκών για τον αλγόριθμο WOfs	72
Εικόνα 7.30: Ανάλυση σκηνών του dataset	73
Εικόνα 7.31: Λογική εντοπισμού ξηρασίας σε υδάτινο σώμα	73
Εικόνα 7.32: Κατηγοριοποίηση δεδομένων	73
Εικόνα 7.33: Εκκίνηση server	74
Εικόνα 7.34: Δήλωση endpoints εσωτερικού API	74
Εικόνα 7.35: Δήλωση API μέσω FastAPI	74
Εικόνα 7.36: Συναρτηση lifespan για δημιουργία εργατών	75
Εικόνα 7.37: Αντληση ονομάτων μεθόδων και handler δημιουργίας endpoint	75
Εικόνα 7.38: Δήλωση ορισμάτων κάθε μεθόδου του endpoint	75
Εικόνα 7.39: Αποτελέσματα Σεναρίου A	77
Εικόνα 7.40: Χρήση CPU και RAM κατά το πρώτο σενάριο	77
Εικόνα 7.41: Αποτελέσματα Σεναρίου B	78
Εικόνα 7.42: Ενδιάμεση φάση εκτέλεσης	79
Εικόνα 7.43: Τελική φάση εκτέλεσης	79
Εικόνα 8.1: Πίνακας user και έλεγχος μορφής email	81
Εικόνα 8.2: Πεδία πίνακα καταγραφής αιτημάτων	81
Εικόνα 8.3: Πίνακες αποθηκευσης αποτελεσμάτων	82
Εικόνα 8.4: Πίνακας api k, συνάρτηση και Trigger της συνάρτησης	83
Εικόνα 8.5: Πίνακας χρονοπρογραμματισμενων διεργασιών του pgcron (Query και ώρα επανάληψης)	84
Εικόνα 8.6: Πίνακας χρονοπρογραμματισμενων διεργασιών του pgcron (Επιπλέον Πληροφορίες)	84
Εικόνα 8.7: Διάγραμμα Συνδέσεων Πινάκων Hellas Cube	85
Εικόνα 9.1: Ροή Αρχιτεκτονικής αιτημάτων ανάλυσης	87
Εικόνα 9.2: Ροή Αρχιτεκτονικής αιτημάτων χρήστη	88
Εικόνα 9.3: Εκκίνηση API	88

Εικόνα 9.4: Σύνδεση με βάση δεδομένων	89
Εικόνα 9.5: Συνάρτηση μονοπατιών του API	90
Εικόνα 9.6: Αρχικοποίηση συνάρτησης ενημέρωσης του reporter κατάστασης	90
Εικόνα 9.7: Πρόσθετα επίπεδα στα μονοπάτια του API	90
Εικόνα 9.8: Συνάρτηση για την ενημέρωση κατάστασης του αιτήματος	90
Εικόνα 9.9: Ασύγχρονες ενημερώσεις κατάστασης στην βάση	91
Εικόνα 9.10: Εκκίνηση του server	91
Εικόνα 9.11: Έλεγχος εγκυρότητας API	92
Εικόνα 9.12: Έλεγχος για υπάρχοντα δεδομένα στην βάση δεδομένων	93
Εικόνα 9.13: Συγγραφή και αποστολή του json στο υποσύστημα	93
Εικόνα 9.14: Διαχείριση απάντησης του υποσυστήματος	94
Εικόνα 9.15: Έλεγχος στοιχείων, δημιουργία API key και ενημέρωση βάσης	95
Εικόνα 9.16: Δημιουργία API key	95
Εικόνα 9.17: Κωδικοποίηση του api key με SHA-256	95
Εικόνα 9.18: Ερώτημα ανάκτησης ιστορικού χρήστη.	96
Εικόνα 10.1: Σχεδιάγραμμα Εντολών CLI	98
Εικόνα 10.2: Enum εντολών CLI	98
Εικόνα 10.3: Παράδειγμα Fail-fast μεθόδου	99
Εικόνα 10.4: Εκτέλεση εντολής ndvi	100
Εικόνα 10.5: Μέθοδος εξαγωγής	100
Εικόνα 10.6: Hashing από το CLI	101
Εικόνα 10.7: Συνάρτηση login	101
Εικόνα 10.8: Αποστολή geojson αποθήκευση του μονοπατιού στο τοπικό config	102
Εικόνα 10.9: Συνάρτηση αποθήκευσης μονοπατιού στο config	102
Εικόνα 10.10: Λήψη του API key απο το config	103
Εικόνα 10.11: Εντολή Init	103
Εικόνα 10.12: Παράδειγμα εξαγωγής με την prettytable-rs	104
Εικόνα 10.13: Παράδειγμα εξαγωγής τερματικό	104
Εικόνα 10.14: Τύπωση μοναδιαίας ανάλυσης στο τερματικό	105
Εικόνα 10.15: Λογική εμφωλευμενου json σε πίνακα του prettytable	106
Εικόνα 10.16: Καθορισμός export μονοπατιού	107
Εικόνα 10.17: Εξαγωγή δεδομένων ανάλογα την τιμή που πήρε ως δεδομένο	108
Εικόνα 11.1: Εκτέλεση Help εντολής, logo συστήματος	109
Εικόνα 11.2: Εκτέλεση εντολής για αίτημα περιβαλλοντικής ανάλυσης	110
Εικόνα 11.3: Εντολές χρήστη	111
Εικόνα 11.4: Εκτέλεση εντολής με προσωπική περιοχή ενδιαφέροντος	112
Εικόνα 11.5: Αποτέλεσμα ιστορικού στο τερματικό	113
Εικόνα 11.6: Αποτέλεσμα ιστορικού σε csv	113

Κατάλογος Πινάκων

Πίνακας 3.7: NDVI κατηγοριοποίηση αποτελεσμάτων [20]	19
Πίνακας 3.11: NDWI κατηγοριοποίηση αποτελεσμάτων [22]	20
Πίνακας 6.9: Πίνακας καναλιών του Harmonized Landsat Sentinel για το προϊόν HLS	50
Πίνακας 7.21: Αποτελέσματα αλγορίθμου SCL	68

Πίνακας 7.22: Πίνακας για το QA_PIXEL του Landsat 8 και 9

68

Πίνακας 7.24: bit μάσκας fmask

69

Συντομογραφίες

AGDC	Australian Geoscience Data Cube
AOT	Aerosol Optical Thickness
API	Application Programming Interface / Προγραμματιστική Διεπαφή Εφαρμογών
AR	Aquatic Reflectance
ARDC	Africa Regional Data Cube
ASGI	Asynchronous Server Gateway Interface
AWS	Amazon Web Services
CLI	Command Line Interface
CRS	Coordinate Reference System / Σύστημα Αναφοράς Συντεταγμένων
D.E.	Διπλωματική Εργασία
DCOD	Data Cube on Demand
DEA	Digital Earth Australia
ΔΙΠΙΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
DTO	Data Transfer Object / Αντικείμενο Μεταφοράς Δεδομένων
dNBR	Differenced Normalized Burn Ratio
EOEPCA	Earth Observation Exploitation Platform Common Architecture
ESA	European Space Agency
FOS	Forest Observation System
GA	Geoscience Australia / Γεωεπιστήμη της Αυστραλίας
GEE	Google Earth Engine
GIS	Γεωγραφικά Συστήματα Πληροφοριών
GUI	Graphical User Interface / Γραφικό Περιβάλλον Διεπαφής
HLS	Harmonized Landsat Sentinel
IDE	Integrated Development Environment / Ολοκληρωμένο Περιβάλλον Ανάπτυξης
ITEM	Intertidal Extents Model
JSON	JavaScript Object Notation
JSONB	Binary JSON

LiDAR	Light Detection and Ranging
MPC	Microsoft Planetary Computer
MRO	Method Resolution Order
NASA	National Aeronautics and Space Administration
NBR	Normalized Burn Ratio
NDBI	Normalized Difference Built-up Index
NDCI	Normalized Difference Chlorophyll Index
NDMI	Normalized Difference Moisture Index
NDSI	Normalized Difference Snow Index
NDTI	Normalized Difference Turbidity Index
NDVI	Normalized Difference Vegetation Index
NDWI	Normalized Difference Water Index
NIR	Near Infrared / Εγγύς Υπέρυθρο
O.D.C.	Open Data Cube
OOP	Object Oriented Programming / Αντικειμενοστραφής Προγραμματισμός
OSM	Open Street Map
Δ.Ε.	Διπλωματική Εργασία
RDBMS	Relational Database Management System / Σύστημα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων
RGB	Red, Green, Blue / Κόκκινο, Πράσινο, Μπλε
SAR	Synthetic Aperture Radar
SCL	Scene Classification Layer
STAC	Spatio-Temporal Asset Catalog
SWIR	Short-Wave Infrared / Βραχύ Υπέρυθρο
USGS	United States Geological Survey
UUID	Universally Unique Identifier
V.S.C.	Visual Studio Code
VPS	Virtual Private Server
WofS	Water Observations from Space

Κεφάλαιο 1ο: Εισαγωγή

1.1 Πρωταρχική Ιδέα

Στη σύγχρονη εποχή, η κατάσταση του περιβάλλοντος αναδεικνύεται σε έναν από τους κρισιμότερους παράγοντες που επηρεάζουν την ποιότητα ζωής τόσο σε εθνικό όσο και σε παγκόσμιο επίπεδο. Στον 21ο αιώνα περιβαλλοντικά ζητήματα, όπως η ρύπανση του αέρα, του νερού και του εδάφους, επιδρούν άμεσα στην καθημερινότητα, καθιστώντας συχνά επιτακτική την ανάγκη της συνεχούς παρακολούθησής τους. Στο παρελθόν, η απουσία προηγμένων τεχνολογικών μέσων καθιστούσε εξαιρετικά δυσχερή τη συστηματική παρακολούθηση του φυσικού κόσμου. Ωστόσο, η ραγδαία τεχνολογική εξέλιξη των τελευταίων δεκαετιών κατέστησε εφικτή τη συνεχή καταγραφή και παραγωγή μεγάλου όγκου περιβαλλοντικών δεδομένων. Η συλλογή και η παραγωγή αυτών των δεδομένων αποτελεί μόνο την αρχή της διαδικασίας. Στην παρούσα διπλωματική εργασία (Δ.Ε.), η σύζευξη της Περιβαλλοντικής Επιστήμης και της Μηχανικής Λογισμικού επιτρέπει την επεξεργασία δεδομένων σε συνδυασμό με υπολογιστικούς δείκτες (NDVI, NDWI, NBR κ.λπ.). Σκοπός είναι η εξαγωγή ασφαλών πορισμάτων και η άντληση πολύτιμης πληροφορίας μέσα από τον διαρκώς αυξανόμενο όγκο των περιβαλλοντικών δεδομένων.

Τα δεδομένα με τα οποία ασχοληθήκαμε στην τρέχουσα Δ.Ε. προήλθαν από δορυφόρους όπως Sentinel-2, Landsat. Για να μπορέσουμε όμως να επεξεργαστούμε τα δορυφορικά δεδομένα που παρέχονται από υπηρεσίες όπως την AWS και Microsoft Planetary, χρησιμοποιήθηκε το framework του Open Data Cube (O.D.C.). Το ODC αποτελεί ένα λογισμικό πλαισίου ανοιχτού κώδικα που καθιστά εφικτή την διαχείριση και επεξεργασία πολυδιάστατων γεωχωρικών δεδομένων συμβάλλοντας στην ερμηνεία πορισμάτων και αποτελεσμάτων. Η χρήση του framework έγινε μέσα σε υποσύστημα Python το οποίο ήταν υπεύθυνο στην απόκτηση των δεδομένων και στην σωστή επεξεργασία τους. Ως γνωστόν τα δορυφορικά δεδομένα έχουν μέγεθος μεγαλύτερο από ότι ο βασικός υπολογιστής-server μπορεί να αποθηκεύσει. Για αυτό και στην βάση δεδομένων που χρησιμοποιήθηκε, αποθηκεύτηκαν οι δείκτες (indexes) των δεδομένων. Καθώς ο πυρήνας του ODC είναι βασισμένος σε Python, η βάση που χρησιμοποιήθηκε για την αποθήκευση των indexes ήταν η PostgreSQL.

Η αποτελεσματικότητα ενός συστήματος λογισμικού τείνει να εξαρτάται σε μεγάλο βαθμό όχι μόνο από την υπολογιστική του ικανότητα στην ανάλυση δεδομένων, αλλά και από τη χρηστικότητα της διεπαφής του. Στην Δ.Ε. επιλέχθηκε να αναπτυχθεί διεπαφή γραμμής εντολών (Command Line Interface, CLI). Ο λόγος για τον οποίο επιλέχθηκε το CLI για την ανάπτυξη της εφαρμογής είναι επειδή το σύστημα επικεντρώνεται στην ταχύτητα και την απλότητα του με στοχευμένα αποτελέσματα και άμεση αξιοποίηση τους από τους χρήστες. Για την ανάπτυξη του CLI χρησιμοποιείται η γλώσσα Rust και το εργαλείο μπορεί να εγκατασταθεί στον υπολογιστή του χρήστη μέσω του εργαλείου cargo.

Η αρχιτεκτονική του συστήματος αποτελείται από τρία βασικά στάδια:

- 1) Την διεπαφή γραμμής εντολών του χρήστη που λαμβάνει τις παραμέτρους εισόδου (π.χ. γεωγραφική περιοχή, ημερομηνία) και τις προωθεί στον διακομιστή μέσω διαδικτυακών μηνυμάτων.
- 2) Η διεπαφή προγραμματισμού εφαρμογών (API) που λειτουργεί ως οργανωτής στην πλευρά του server. Παραλαμβάνει τα αιτήματα του χρήστη και τα προωθεί στο υποσύστημα ανάλυσης.

3) Το API καλεί την Python. Το υποσύστημα ελέγχει αν η βάση έχει ήδη αποθηκευμένα ευρετήρια, τα επαναχρησιμοποιεί, διαφορετικά αντλεί τα ευρετήρια από υποδομές και δημιουργεί νέα. Στην συνέχεια εκτελεί ανάλυση των δεδομένων και επιστρέφει τα αποτελέσματα στο API, το οποίο τα προωθεί στο τερματικό του χρήστη.

1.2 Στόχοι

Ο στόχος της παρούσας Δ.Ε. είναι η παρουσίαση ενός πειραματικού συστήματος, το οποίο φιλοδοξεί να αποτελέσει βασικό εργαλείο υποδομής για περιβαλλοντολόγους, ερευνητές και δημόσιους φορείς. Με τον τρόπο αυτό, τίθενται τα θεμέλια για τη μελλοντική λειτουργία, με απώτερο σκοπό την αποτελεσματική αντιμετώπιση των πρακτικών προκλήσεων στην επεξεργασία και ανάλυση δεδομένων. Η εφαρμογή αποσκοπεί να εστιάζει στην πραγματικότητα και τις ανάγκες του Ελλαδικού χώρου καλύπτοντας περιοχές που θα χρησιμεύουν στους χρήστες όπως πόλεις, λίμνες, βουνά κ.τ.λπ. καθώς και επιλογές παραμετροποίησης για κάθε ανάγκη. Το σύστημα διατίθεται υπό άδεια ανοιχτού κώδικα, εξασφαλίζοντας την ελεύθερη πρόσβαση σε κάθε ενδιαφερόμενο για χρήση και προσωπική προσαρμογή, προωθώντας τη φιλοσοφία της ανοικτής επιστήμης ενθαρρύνοντας την ελεύθερη ανταλλαγή δεδομένων και μεθοδολογιών μεταξύ ερευνητικών ομάδων, πανεπιστημίου και ανεξάρτητων μελετητών. Επιπλέον παρέχονται οι κατάλληλες υποδομές μελλοντικής επέκτασης και βελτίωσης του συστήματος.

Μολονότι η εφαρμογή απευθύνεται σε ανθρώπους της επιστήμης του περιβάλλοντος με στοιχειώδεις γνώσεις χρήσης ηλεκτρονικών υπολογιστών, η απλότητα της διεπαφής και της χρήσης της είναι ένα από τα χαρακτηριστικά του σχεδιασμού της καθώς η αρχιτεκτονική της στοχεύει στην ανάπτυξη των τεχνολογιών του back-end. Συνοψίζοντας η Δ.Ε. επιδιώκει στην χρησιμότητα της ως προς την κοινότητα με αξιόπιστα και πλήρη δεδομένα, καθώς και την μελέτη των τεχνολογιών στις οποίες βασίζεται το σύστημα.

1.3 Δομή & Παραδοτέο

- Κεφάλαιο 1ο: Στο τρέχον κεφάλαιο γίνεται μια γενική εισαγωγή στην Διπλωματική Εργασία όπου περιγράφεται η πρωταρχική ιδέα του Hellas Cube μαζί με μια πρώτη ματιά στην αρχιτεκτονική του συστήματος. Παρουσιάζονται οι στόχοι της Δ.Ε., οι προκλήσεις που αντιμετωπίστηκαν κατά την διάρκεια της υλοποίησης και επιπλέον παρόμοιες εφαρμογές που έχουν ήδη υλοποιηθεί και τις διαφορές με την εφαρμογή της Δ.Ε..
- Κεφάλαιο 2ο: Στο δεύτερο κεφάλαιο πραγματοποιείται μια ιστορική αναδρομή και ανάλυση της παγκόσμιας εξέλιξης των Data Cubes. Παρουσιάζονται οι βασικές έννοιες, το Open Data Cube (ODC) και εξετάζονται παραδείγματα επιτυχημένων υλοποιήσεων σε παγκόσμιο και εθνικό επίπεδο, όπως το Digital Earth Australia, το Digital Earth Africa, το Euro Data Cube και η ελληνική προσπάθεια EL-BIOS.
- Κεφάλαιο 3ο: Στο τρίτο κεφάλαιο παρουσιάζεται το θεωρητικό υπόβαθρο των δορυφορικών δεδομένων, εστιάζοντας στις αποστολές Landsat και Sentinel και στην εναρμόνιση των δεδομένων τους μέσω του Harmonized Landsat Sentinel (HLS). Επιπλέον, αναλύονται οι κατηγορίες των περιβαλλοντικών δεικτών, τα φασματικά κανάλια και οι μαθηματικοί τύποι (όπως το NDVI, NDWI, κ.ά.), καθώς και οι εξειδικευμένοι αλγόριθμοι που ενσωματώνονται στο σύστημα του Hellas Cube.
- Κεφάλαιο 4ο: Στο τέταρτο κεφάλαιο διερευνάται η ανάλυση των απαιτήσεων του συστήματος. Περιγράφονται αναλυτικά οι λειτουργικές απαιτήσεις όπως η διεπαφή του

χρήστη, το υποσύστημα χρηστών και η ανάκτηση δεδομένων καθώς και οι μη-λειτουργικές απαιτήσεις, οι οποίες αφορούν τη διαχείριση πόρων, την απόδοση, την αξιοπιστία και το περιβάλλον εκτέλεσης του λογισμικού.

- Κεφάλαιο 5ο: Το πέμπτο κεφάλαιο καταγράφει τη συνολική αρχιτεκτονική του συστήματος, η οποία δομείται σε τρία διακριτά επίπεδα: τη διεπαφή γραμμής εντολών, το Εξωτερικό API σε Rust και το Υποσύστημα Ανάλυσης Δεδομένων σε Python. Παράλληλα, παρουσιάζονται και αιτιολογούνται οι τεχνολογικές επιλογές που έγιναν, όπως το πλαίσιο Open Data Cube, η σχεσιακή βάση PostgreSQL, και τα εργαλεία ανάπτυξης.
- Κεφάλαιο 6ο: Στο έκτο κεφάλαιο περιγράφεται βήμα προς βήμα το πρακτικό μέρος της εγκατάστασης και διαμόρφωσης του περιβάλλοντος για το Open Data Cube. Αναλύεται το στήσιμο της βάσης δεδομένων με χρήση PostgreSQL και PostGIS, η δημιουργία του απομονωμένου περιβάλλοντος, η προσθήκη των κατάλληλων δορυφορικών προϊόντων και η τελική αρχικοποίηση του συστήματος.
- Κεφάλαιο 7ο: Το έβδομο κεφάλαιο εστιάζει στην υλοποίηση του προγραμματιστικού πυρήνα του συστήματος σε Python. Αναλύεται η αρχιτεκτονική του εσωτερικού API, η διαδικασία ευρετηρίασης, η φόρτωση των δεδομένων, καθώς και η πρακτική εφαρμογή των περιβαλλοντικών δεικτών, των τεχνικών συγκάλυψης και η εξαγωγή των τελικών στατιστικών αποτελεσμάτων.
- Κεφάλαιο 8ο: Στο όγδοο κεφάλαιο αναλύεται ο σχεδιασμός και η δομή της βάσης δεδομένων PostgreSQL που υποστηρίζει το σύστημα. Επεξηγούνται οι πίνακες για τη διαχείριση των χρηστών, την καταγραφή των αιτημάτων και την αποθήκευση των αποτελεσμάτων, καθώς και οι αυτοματοποιημένοι μηχανισμοί για την ασφαλή διαχείριση του κύκλου ζωής των API keys.
- Κεφάλαιο 9ο: Στο ένατο κεφάλαιο παρουσιάζεται ο σχεδιασμός και η υλοποίηση του κεντρικού εξωτερικού API σε γλώσσα Rust. Περιγράφεται η ασύγχρονη δρομολόγηση των αιτημάτων μέσω του πλαισίου Axum, η ασφαλής διαχείριση της ταυτοποίησης των χρηστών με ισχυρά πρότυπα κρυπτογραφίας, η επικοινωνία με τη βάση δεδομένων και η ενορχήστρωση της επικοινωνίας με το υποσύστημα ανάλυσης δεδομένων.
- Κεφάλαιο 10ο: Το δέκατο κεφάλαιο αφορά την ανάπτυξη της Διεπαφής Γραμμής Εντολών (CLI), που αποτελεί το σημείο αλληλεπίδρασης του χρήστη με το σύστημα. Αναλύεται η αρχιτεκτονική των εντολών, ο μηχανισμός επικύρωσης παραμέτρων (Fail-Fast), η εκτέλεση των αιτημάτων και οι μέθοδοι εξαγωγής και οπτικοποίησης των αποτελεσμάτων στο τερματικό και σε μορφή αρχείων CSV.
- Κεφάλαιο 11ο: Στο ενδέκατο κεφάλαιο παρατίθεται ένας πρακτικός οδηγός χρήσης του Hellas Cube μέσα από ολοκληρωμένα σενάρια λειτουργίας. Επιδεικνύεται βήμα-βήμα η διαδικασία εγκατάστασης, η ανώνυμη χρήση, η δημιουργία και διαχείριση λογαριασμού με προσωπικό γεώχωρο, καθώς και η ανάκτηση και μαζική εξαγωγή του ιστορικού αναλύσεων.
- Κεφάλαιο 12ο: Στο δωδέκατο και τελευταίο κεφάλαιο συνοψίζονται τα βασικά συμπεράσματα που προέκυψαν από την εκπόνηση και την ολοκλήρωση της διπλωματικής εργασίας. Επιπλέον, καταγράφονται προτάσεις για τη μελλοντική εξέλιξη του συστήματος, όπως η φιλοξενία σε απομακρυσμένο διακομιστή, η υποστήριξη πρόσθετων μορφότυπων εξαγωγής και η ενσωμάτωση νέων περιβαλλοντικών δεικτών.

1.4 Προκλήσεις

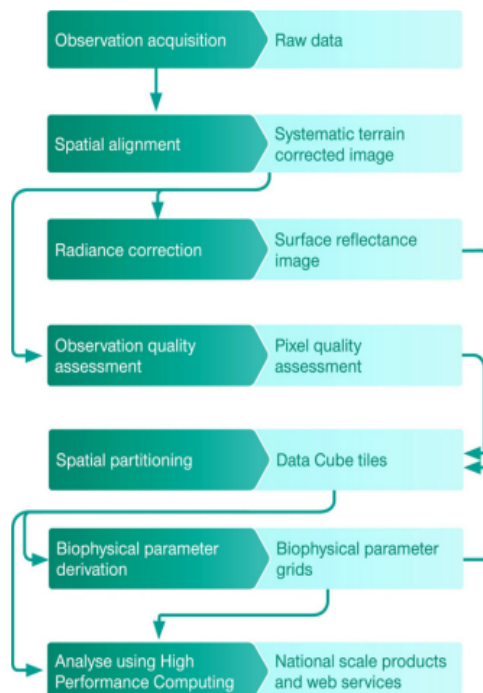
Το Hellas Cube, που αναπτύχθηκε στα πλαίσια της Δ.Ε., αποτελεί ένα αρκετά περίπλοκο και εξαιρετικά απαιτητικό σύστημα. Μια μεγάλη και πολύ πιθανόν η σημαντικότερη πρόκληση του Hellas Cube ήταν η ανάπτυξη ενός server για τις ανάγκες της εφαρμογής. Για να καλυφθούν οι ανάγκες της εφαρμογής, οι τεχνικές απαιτήσεις που έχουμε από τον διακομιστή είναι υψηλές όπως η ύπαρξη αρκετών κεντρικών πυρήνων, μεγάλη χωρητικότητα RAM, καθώς και αρκετή χωρητικότητα δίσκου. Πέραν των τεχνικών προδιαγραφών, μια ακόμα πρόκληση αποτέλεσε το API, καθώς ήταν σημαντικός παράγοντας για την εκτέλεση requests από τον χρήστη, τα οποία ο server όφειλε να διαχειριστεί μέσω αυτού. Η διαχείριση των request είναι ένας σημαντικός πυλώνας για την ομαλή λειτουργία της εφαρμογής και μία από τις δυσκολίες που αντιμετώπισε η εφαρμογή. Ο χρόνος απόκρισης και η βέλτιστη διαχείριση πόρων αποτελούν πάγιες προκλήσεις στον τομέα της μηχανικής λογισμικού [40], παραμέτρους τις οποίες καλείται να διαχειριστεί επιτυχώς και το Hellas Cube που ως κύριο της χαρακτηριστικό είναι η ανάλυση και σωστή παράδοση επεξεργασμένων δορυφορικών δεδομένων. Επομένως είναι απαραίτητη η άριστη χρήση εργαλείων, τεχνικών (συνδυασμός προγραμματιστικών γλωσσών και βιβλιοθηκών) ώστε η εμπειρία του χρήστη να βρίσκεται στο βέλτιστο επίπεδο. Η πολυπλοκότητα της αρχιτεκτονικής του συστήματος, η οποία καλείται να ενορχηστρώσει διεπαφές γραμμής εντολών, ενδιάμεσο API και σύστημα ανάλυσης, έχει την ανάγκη διαχείρισης τεράστιου όγκου γεωχωρικών δεδομένων (Big Earth Data). Για την πλήρη κατανόηση της επιλογής του πλαισίου Open Data Cube ως της κεντρικής τεχνολογίας που θα υποστηρίξει αυτό το υπολογιστικό φορτίο, είναι αναγκαία η περιγραφή της εξέλιξης των συστημάτων Data Cube παγκοσμίως και του τρόπου με τον οποίο έχουν μετασχηματίσει ριζικά την ανάλυση δορυφορικών παρατηρήσεων τα τελευταία έτη.

Κεφάλαιο 2ο: Η Αρχή και η Παγκόσμια Εξέλιξη των Data Cube

2.1 Data Cube

2.1.1. Australian Geoscience Data Cube (AGDC)

Το AGDC αποτελεί το πρώτο και ιδρυτικό Data Cube που ξεκίνησε ως μία καινοτόμα τεχνολογική υποδομή “Big Data” [1] (Μεγάλα Δεδομένα: Σύνολα δεδομένων, υπερβολικά μεγάλα και σύνθετα για να υποστούν επεξεργασία ή διερεύνηση με συμβατικές μεθόδους ή εργαλεία [1]) που αναπτύχθηκε για να αξιοποιήσει δορυφορικά δεδομένα παρατηρήσεων την Γης στην Αυστραλία. Το AGDC κλήθηκε να αντιμετωπίσει προκλήσεις των Big Data όπως τον τεράστιο όγκο (volume), την ταχύτητα (velocity) και την ποικιλομορφία (variety). Μια από τις καινοτομίες που έφερε το AGDC είναι η μετατροπή των εικόνων σε μετρήσεις [1]. Δηλαδή δεν αποθηκεύει τις εικόνες ως απλές δορυφορικές εικόνες αλλά εφαρμόζει γεωμετρικές και ραδιομετρικές διορθώσεις, αξιολογώντας την ποιότητα του κάθε pixel και επιτρέποντας την αξιόπιστη ανάλυση χρονοσειρών. Αναπτύχθηκε με άδεια ανοιχτού κώδικα επιτρέποντας και σε άλλες χώρες να αναπτύξουν το δικό τους Data Cube. Η αρχιτεκτονική του πρώτου Data Cube αποτελείται από τέσσερα βασικά επίπεδα: 1) Συλλογή δεδομένων και προετοιμασία σε μορφή έτοιμα προς ανάλυση. 2) Υποδομή Data Cube όπου τα δεδομένα αποθηκεύονται σε netCDF4 [1] 3) Πλατφόρμα δεδομένων που επιτρέπουν την δημιουργία νέων προϊόντων 4) Διεπαφή μέσω API και web services. Η τελευταία έκδοση του AGDC (Version 2) αποτέλεσε ένα τεράστιο βήμα στην εξέλιξη του κύβου καθώς η κύρια ικανότητα του AGDCv2 είναι η “καταχώρηση” (indexing) εξωτερικών δεδομένων χωρίς να χρειάζεται να τα αντιγράψει και να αλλάξει την μορφή τους εξασφαλίζοντας την άνοξη αξιοποίηση αρχείων κλίμακας petabyte. Το AGDC ως πρώτο Data Cube αποτέλεσε τα θεμέλια για την αξιοποίηση των Big Data σε όλο τον κόσμο και όχι μόνο στην Αυστραλία.



Εικόνα 2.1: Η ροή της διαδικασίας από ακατέργαστα δεδομένα σε προϊόντα του Data Cube[1].

2.1.2. Open Data Cube (ODC)

Το Open Data Cube και το AGDC έχουν μια άμεση σχέση εξέλιξης [2] καθώς το AGDC αποτελεί θεμέλιο για το ODC. Το Data Cube (DC) της Αυστραλίας από τοπικός κύβος μετασχηματίστηκε ως ανεξάρτητος κύβος ώστε να υποστηρίζει διαφορετικά είδη δορυφορικών δεδομένων (πέραν της Αυστραλίας). Το ODC δημιουργήθηκε ως εμπορικό σήμα (brand) στις αρχές του 2017 από ιδρυτικούς εταίρους, με στόχο τα 20 λειτουργικά εθνικά data cubes. Το ODC αποδείχτηκε μια καινοτομική λύση για τα Big Data και μέχρι το 2018 υπήρχαν ήδη 4 λειτουργικά Data Cubes σε Κολομβία, Ελβετία, Ταιβάν, Βιετνάμ. Επιπλέον 11 DC βρίσκονταν υπό ανάπτυξη και 28 χώρες που εξέφρασαν ενδιαφέρον [2][3]. Μέχρι το 2020 υπήρχαν σημαντικές εξελίξεις με την ανάπτυξη 15 λειτουργικών data cubes σε εθνικό επίπεδο και με τον συνολικό παγκόσμιο αντίκτυπο να πλησιάζει τις 106 χώρες[3]. Το 2023 οι κύβοι κατέστησαν πλέον δυσδιαχειρίσιμοι λόγω της αλλαγής της τεχνολογίας όπου αναπτύχθηκε το Data Cube on Demand (DCoD). Η σύγχρονη φιλοσοφία που έχει επικρατήσει στον κόσμο των data cubes είναι «ο χρήστης στα δεδομένα», δηλαδή αντί ο χρήστης να κατεβάζει τα δεδομένα η επεξεργασία γίνεται εκεί που αποθηκεύονται τα δεδομένα [4]. Πολύ απλά οι σημερινοί κύβοι βρίσκονται σε υποδομές Cloud (Amazon Web Services ή Google Cloud) ή σε συστήματα Υψηλής Υπολογιστικής Απόδοσης [4] και οι χρήστες έχουν πρόσβαση σε περιβάλλοντα εργασίας όπως το Jupyter Notebook όπου μπορούν να γράψουν κώδικα και να εκτελέσουν αναλύσεις δίπλα στα δεδομένα, δημιουργώντας έναν προσωρινό προσωπικό κύβο.

Σε αντίθεση με τη δομή των παραδοσιακών βάσεων δεδομένων, τα σύγχρονα Data Cubes συνιστούν εξαιρετικά δυναμικά και αυτοματοποιημένα συστήματα [4]. Παράλληλα, μολονότι εξακολουθούν να λειτουργούν με επιτυχία εθνικοί και περιφερειακοί κύβοι όπως το eocube.ro (Ρουμανία), ACUBE(Αυστρία) και DEA(Αφρική) κ.α. η τάση πλέον ενισχύεται από την ανάπτυξη παγκόσμιων πλατφορμών [3] όπως το Microsoft Planetary Computer (MPC), Google Earth Engine(GEE). Η μελλοντική κατεύθυνση των κύβων δεδομένων διαφαίνεται να στρέφεται προς την Ομοσπονδιοποίηση (Federation) αντί για τις μεμονωμένες εθνικές υποδομές, όπως προτείνεται από την αρχιτεκτονική EOEPKA [4]. Η νέα εποχή του ODC παρουσίασε νέα πρότυπα όπως τα Cloud-Optimized μορφότυπα (Formats) και τον κατάλογο STAC. Τα Cloud-Optimized Formats μετατρέπουν τα δεδομένα σε «cloud-friendly» μορφότυπα, δηλαδή για την πρόσβαση ενός μικρού γεωχώρου δεν υπάρχει πλέον ανάγκη εγκατάστασης μιας ολόκληρης δορυφορικής εικόνας. Ο STAC (Spatio-Temporal Asset Catalog) είναι ένας εύκολος τρόπος αναζήτησης δεδομένων, αντικαθιστώντας το πολύπλοκο σύστημα αναζήτησης του κάθε Data Cube και παρέχει μια κοινή γλώσσα αναζήτησης. Το Data Cube από τα χρόνια της ίδρυσης του έχει εξελιχθεί ραγδαία, ξεκινώντας ως μια λύση για την χώρα της Αυστραλίας και στην συνέχεια μια παγκόσμια λύση για να ανάλυση γεωχωρικών Big Data καταλήγοντας στην δυνατότητα ατομικής χρήσης, όμως δεν θα σταματήσει εκεί η εξέλιξη του Open Data Cube.

2.2 Data Cube ανά τον κόσμο

2.2.1. Digital Earth Australia (DEA)

Το Digital Earth Australia αποτελεί τη μετεξέλιξη του AGDC (τεχνολογικά και φιλοσοφικά) και πρόκειται για ένα μεγάλο κυβερνητικό πρόγραμμα για την αξιοποίηση δεδομένων για την οικονομία και το περιβάλλον της Αυστραλίας. Η κυβέρνηση της Αυστραλίας έχει επενδύσει 15,3 εκατομμύρια δολάρια για την ίδρυση του DEA που βοηθά στην παρακολούθηση των επενδύσεων για το περιβάλλον καθώς την παρακολούθηση της γεωργικής παραγωγικότητας και την διαχείριση των υδάτων [5]. Το DEA για την αξιοποίηση των δεδομένων εφαρμόζει αλγορίθμους ανεπτυγμένους για

τα δεδομένα του Αυστραλέζικου γεωχώρου όπως τον «Water Observations from Space (WOfS)» για την παρακολούθηση επιφανειακών υδάτων και το μοντέλο Intertidal Extents Model (ITEM) για την χαρτογράφηση παλιρροιακών ζωνών. Ως απόγονος του AGDC και χτισμένο πάνω στην φιλοσοφία του Open Data Cube, το DEA επιτρέπει στους αλγορίθμους να μοιράζονται πολύ εύκολα με χρήστες και να προσαρμόζονται σε τοπικές ανάγκες. Το πρόγραμμα έχει σχεδιαστεί με τέτοιο τρόπο ώστε τα δεδομένα να μετατρέπονται σε φυσικές μετρήσεις (ανακλαστικότητα, ταχύτητα, θερμοκρασία) επιτρέποντας στους χρήστες την λήψη άμεσων αποφάσεων. Η επένδυση DEA έχει αποτελέσει επιτυχία για την Κυβέρνηση της Αυστραλίας καθώς έχει προσφέρει περίπου 15.000 νέες θέσεις εργασίας και μέχρι το 2025 προβλέπεται να προσφέρει περίπου 8 δισεκατομμύρια δολάρια ετησίως [5].

2.2.2. Digital Earth Africa (DE Africa)

Ένας από τους μεγαλύτερους κύβους σε όλον τον κόσμο είναι ο κύβος που αντιπροσωπεύει την ήπειρο της Αφρικής. Το Digital Earth Africa (DE Africa) έκανε τα πρώτα του βήματα το 2018, έναν χρόνο μετά την εκκίνηση του ODC, με την ονομασία Africa Regional Data Cube (ARDC). Το ARDC ξεκίνησε ως μια περιφερειακή προσπάθεια για να αποδειχθεί η αξία της τεχνολογίας. Οι 5 χώρες που υποστήριζε ήταν οι εξής: Κένυα, Τανζανία, Γκάνα, Σιέρα Λεόνε και Σενεγάλη [3]. Αποτέλεσε ένα ιστορικό ορόσημο, καθώς ήταν η πρώτη περιφερειακή πρωτοβουλία που ένωσε πολλές χώρες μαζί κάτω από την ίδια υποδομή. Με αφορμή την επιτυχία του ARDC παρουσιάστηκε χρηματοδότηση για την δημιουργία μιας μεγαλύτερης υποδομής, του DE Africa, με σκοπό την υποστήριξη και των 54 χωρών της Αφρικής [3]. Απώτερος σκοπός του DE Africa είναι η ενδυνάμωση των αφρικανικών χωρών όσον αφορά το κλίμα και την ενίσχυση της εγχώριας συμμετοχής και συνεργασίας. Το σύστημα λειτουργεί με την χρήση εκτενής βιβλιοθήκης “Python notebooks” προσφέροντας καθοδήγηση σε χρήστες για την χρήση, ανάλυση και εκμετάλλευση των δορυφορικών δεδομένων. Παράλληλα επεκτείνοντας τον κατάλογο του με εικόνες υψηλής ανάλυσης και εξειδικευμένων δεδομένων προσφέρει σε κρίσιμες εφαρμογές [6]. Το πρόγραμμα αποτελεί χρήσιμο εργαλείο για την αντιμετώπιση πραγματικών προβλημάτων και ένα από αυτά είναι η χαρτογράφηση ακτογραμμών, χρησιμοποιώντας δεδομένα για την συνεχή (κάθε χρόνο) παρακολούθηση των αλλαγών στις ακτογραμμές της ηπείρου, βοηθώντας τροπικά νησιά με έντονη κλιματική αλλαγή.



Εικόνα 2.2: Ετήσιες ακτογραμμές από το DE Africa Coastlines

Η συμβολή της πλατφόρμας στην παρακολούθηση της ποιότητας των υδάτων είναι αρκετά σημαντική στην Αφρική με την ζήτηση των χρηστών να είναι σημαντικά μεγάλη, συγχρόνως συνεισφέρει στην

γεωργία και τον αστικό σχεδιασμό, εκμεταλλεύοντας δορυφορικές εικόνες υψηλής ανάλυσης για την χαρτογράφηση και οριοθέτηση μικρών γεωργικών εκτάσεων. Το σύστημα του DE Africa διακρίνεται για σημαντικά τεχνολογικά επιτεύγματα. Ένα από αυτά είναι η χρήση συστήματος ραντάρ Sentinel-1 που ξεπερνά το εμπόδιο των σύννεφων στις δορυφορικές εικόνες. Έτσι, επιτρέπει τον ακριβή εντοπισμό της ακτογραμμής, μετατρέποντας δεδομένα που διαφορετικά θα παρέμεναν ανεκμετάλλευτα σε απολύτως αξιοποιήσιμα. Αξιοσημείωτη είναι και η βελτίωση της ποιότητας νερού με το σύστημα να μεταφέρεται από την παραδοσιακή χρήση των surface reflectance δεδομένων που είχαν περιορισμούς και να εξετάζεται ένα νέο και πειραματικά προϊόν που ονομάζεται Aquatic Reflectance (AR), ειδικό για υδάτινες επιφάνειες. Το DE Africa είχε υψηλό κόστος και πολλά εμπόδια όπως τις άδειες των εμπορικών δεδομένων αλλά και την διαλειτουργικότητα μεταξύ διαφόρων δεδομένων, αλλά η πρόοδος του επιβεβαιώνει ότι είναι ένα επιτυχές Data Cube.

2.2.3. Euro Data Cube (EDC)

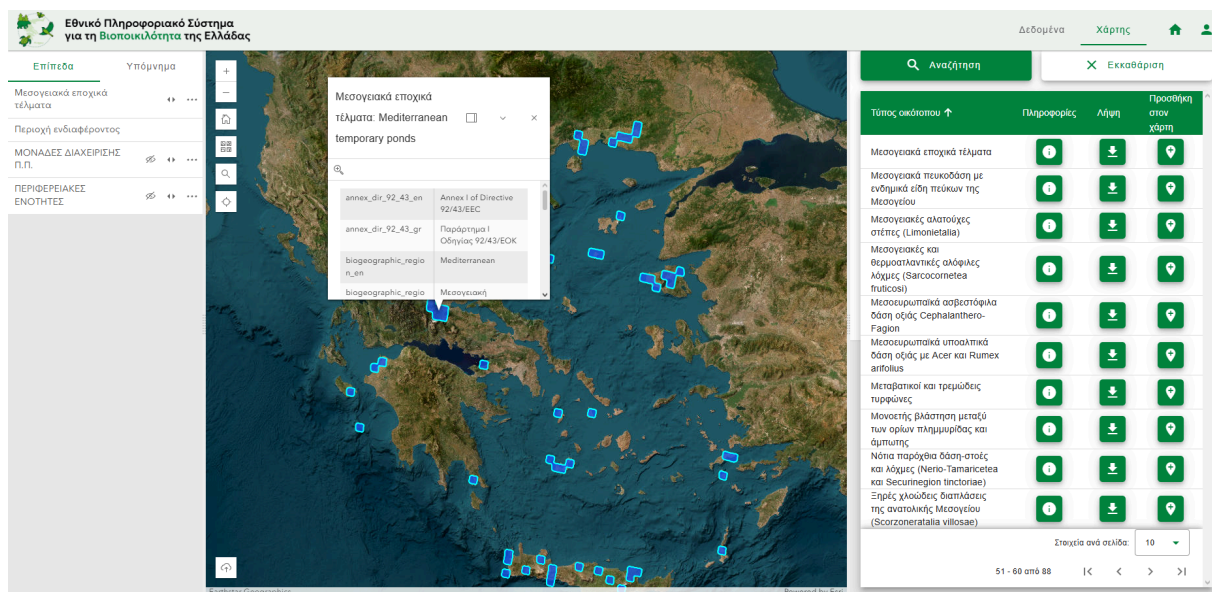
Παράλληλα με την εξέλιξη των ανοικτών κύβων δεδομένων ανοικτής πρόσβασης, αναπτύχθηκαν και εμπορικές λύσεις. Ένα χαρακτηριστικό σύστημα αυτής της κατηγορίας αποτελεί το Euro Data Cube (EDC). Το πρόβλημα που προσπαθεί να λύσει το EDC είναι αξιοποίηση δεκάδων Terabyte δεδομένων που παράγουν οι δορυφορικές αποστολές. Ο τεράστιος όγκος των διαθέσιμων δεδομένων καθιστά την επαναχρησιμοποίησή τους ιδιαίτερα προβληματική. Εξαιτίας της πολυπλοκότητας στη διαχείριση και επεξεργασία τους, η εξαγωγή επιστημονικών συμπερασμάτων καθυστερεί συχνά από ένα έως δύο έτη. Η λύση που προτείνει ο ευρωπαϊκός κύβος είναι να αναπτύσσονται οι αλγόριθμοι ανάλυσης δεδομένων στο ίδιο περιβάλλον που θα ληφθούν τα αποτελέσματα, δηλαδή προσφέρει ένα cloud native περιβάλλον γι' αυτόν ακριβώς τον σκοπό επιτρέποντας τους χρήστες να έχουν πρόσβαση και να επεξεργάζονται τα δεδομένα εκεί [7]. Κάποια από τα εργαλεία που προσφέρει το EDC είναι τα εξής:

1. Sentinel Hub: Μία μηχανή επεξεργασίας Petabyte δορυφορικών δεδομένων από δορυφόρους όπως Sentinel, Landsat, Modis και Airbus.
2. GeoDB: Υπολογιστικό σύστημα που υποστηρίζει την εκτέλεση διανυσματικών αναλύσεων και την εξασφάλιση σε γεωχωρικά δεδομένα διοικητικών ορίων και επίσημων στοιχείων.[7]
3. EOxHub Workspace: Ένα workspace με την κατάλληλη υπολογιστική δύναμη για αποθήκευση δεδομένων και εκτέλεση κώδικα (μέσω περιβαλλόντων Jupyter Notebook) καθώς και φιλοξενία εφαρμογών νέφους[7].

Λόγω των πόρων και εργαλείων που παρέχει ως υπηρεσία είναι ιδανικό για μεγάλο φόρτο εργασίας, τόσο σε όγκο δεδομένων όσο και ως προς την υπολογιστική ισχύ σε αλγόριθμους που πιθανόν θα γίνεται να δοκιμαστούν, εκτελεστούν καθώς και να επαναχρησιμοποιηθούν. Ένα παράδειγμα εφαρμογής του EDC είναι για την δημιουργία Analysis Ready Data (data) με την χρήση datasets από Forest Observation System(FOS). Δηλαδή, με την χρήση ωμών δεδομένων από το FOS και την χρήση βιβλιοθηκών για δημιουργία έτοιμων δεδομένων για εκμάθηση τεχνητής νοημοσύνης (AI/REO) το EDC αποτελεί ο χώρος εργασίας όπου όλα συνδέονται. Συμπερασματικά το Euro Data Cube αποτελεί ένα εμπορικά διαθέσιμο (επί-πληρωμή) περιβάλλον εργασίας για την επεξεργασία και ανάλυση Earth observation (EO) δεδομένων με πολλές εφαρμογές να εξαρτώνται από την λειτουργία και την ελαστικότητα του καθιστώντας το έναν τεχνολογικό πυλώνα στα εμπορικά Data Cubes λόγω των πολλών δυνατοτήτων που προσφέρει με τα Sentinel Hub, GeoDB και EOxHub.

2.2.4. EL-BIOS Earth Observation Data Cube (EODC)

Η γεωγραφική έκταση του Ελλαδικού χώρου αποτελείται από ένα μεγάλο εύρος οικοσυστημάτων και ζωικών ειδών [8], καθιστώντας την ανάγκη της επιτήρησης τους μεγάλη. Το EL-BIOS είναι ένα σύστημα που βοηθάει στην ανάκτηση και ανάλυση Big Data δεδομένων με σκοπό της παρακολούθησης της βιοποικιλότητας της Ελλάδος που η χλωρίδα της φτάνει τα 5.700 είδη και η πανίδα τα 23.000 είδη [8]. Αποτελεί πρώτο στο είδος του, ως η πρώτη προσπάθεια υλοποίησης Data Cube για την χώρα που έχει ως επίκεντρο του την διατήρηση την πλούσια φύση της. Η ανάπτυξη ολοκληρώθηκε το έτος του 2024, με κύριο φορέα υλοποίησης το Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης(ΑΠΘ). Το τελικό προϊόν αποτελεί μία υποδομή, η οποία ενσωματώνει τεχνολογίες όπως παράλληλη επεξεργασία δεδομένων με σκοπό τη μείωση του απαιτούμενου χρόνου, ευρετηρίαση δεδομένων από διακομιστές AWS(Amazon Web Services) και υποστήριξη της εφαρμογής σε ένα γραφικό περιβάλλον διεπαφής (GUI) Web GIS, μία πλατφόρμα οπτικοποίησης δορυφορικών δεδομένων προσφέροντας μια ολοκληρωμένη και διαδραστική εμπειρία. Σχεδιασμένο ειδικά για να καλύπτει ανάγκες όπως λήψη δεδομένων, γεφύρωση και παράλληλη απεικόνιση των δεδομένων. Τέθηκε και δοκιμάστηκε σε λειτουργία σε μία από τις μεγαλύτερες πυρκαγιές που έχει αντιμετωπίσει η Ελλάδα στην περιοχή του Έβρου στην Θράκη το 2023 [8] υπολογίζοντας το δείκτη υγείας βλάστησης (Leaf Area Index) σε δύο διαφορετικές χρονικές στιγμές, πριν και μετά την πυρκαγιά, εντοπίζοντας αμέσως την περιοχή της ζημιάς. Συμπερασματικά, το πληροφοριακό σύστημα του «Ελληνικού Κύβου Δεδομένων» αποτελεί ένα κρίσιμο χρήσιμο εργαλείο για τα Ελληνικά δεδομένα και την διατήρηση του πλούσιου οικοσυστήματος αξιοποιώντας Big Data δεδομένα παραχωρώντας πολύτιμη γνώση διευκολύνοντας το έργο ερευνητών, επιστημόνων και διαχειριστών περιβάλλοντος.



Εικόνα 2.3: Γραφικό Περιβάλλον EL-BIOS

Ενώ το EL-BIOS εστιάζει κατά κύριο λόγο στην παρακολούθηση της βιοποικιλότητας της Ελλάδος και προσφέρει την εμπειρία του μέσω ενός γραφικού περιβάλλοντος, το Hellas Cube ακολουθεί μια διαφορετική, πιο γενικευμένη και αυτοματοποιημένη φιλοσοφία. Σχεδιάστηκε με γνώμονα το CLI για μέγιστη ταχύτητα και ενσωμάτωση σε ροές εργασίας pipelines χωρίς την ανάγκη οπτικοποίησης μέσω γραφικών στοιχείων. Παράλληλα, αντί να περιορίζεται αυστηρά στη χλωρίδα και την πανίδα, παρέχει ένα ευρύτερο φάσμα γενικών περιβαλλοντικών δεικτών (π.χ. ανίχνευση υδάτων, αστικής ανάπτυξης,

χιονοκάλυψης). Παρά τις διαφορετικές αρχιτεκτονικές τους προσεγγίσεις, και τα δύο συστήματα υπογραμμίζουν την ανάγκη ανάπτυξης εθνικών εργαλείων για την εκμετάλλευση των δορυφορικών Big Data.

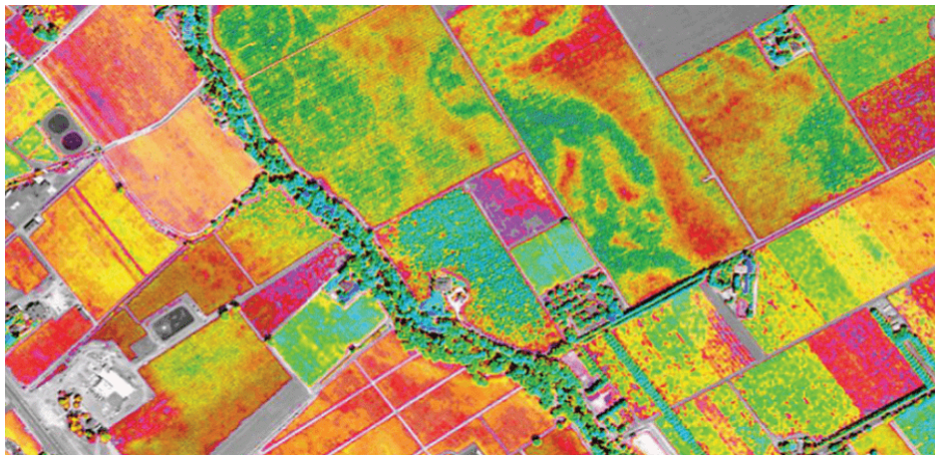
2.3 Υποδομές για Ανάλυση Δεδομένων

Η παγκόσμια εξάπλωση των τεχνολογιών Data Cube αποτελεί έναν πυλώνα στην ανάλυση δορυφορικών Big Data για την παρακολούθηση του περιβάλλοντος. Η δυναμική αυτή καταδεικνύεται τόσο από κρατικές υποδομές όπως ο πρωτοπόρος κύβος της Αυστραλίας (DEA) και η φιλόδοξη πλατφόρμα της Αφρικής (DE Africa) που ενσωματώνει 54 χώρες σε ένα σύστημα όσο και από εμπορικές λύσεις έναντι αντιτίμου, όπως το Euro Data Cube, αλλά και από εγχώριες προσπάθειες διατήρησης βιοτόπων, όπως το ελληνικό EL-BIOS. Ωστόσο, παρά τη ραγδαία ωρίμανση και την ελαστικότητα αυτών των υπολογιστικών συστημάτων νέφους, ο καθοριστικός παράγοντας για την διαμόρφωση έγκυρων περιβαλλοντικών συμπερασμάτων παραμένει η ποιότητα, η συχνότητα και η ανάλυση της ίδιας της «πρώτης ύλης». Ως εκ τούτου, η λεπτομερής εξέταση των δορυφορικών αποστολών που τροφοδοτούν αυτές τις υποδομές, καθώς και των μαθηματικών περιβαλλοντικών δεικτών που προκύπτουν από τις φασματικές τους μετρήσεις, αποτελεί το απαραίτητο επόμενο βήμα για τη θεμελίωση του συστήματος.

Κεφάλαιο 3ο: Δορυφορικά Δεδομένα και Περιβαλλοντικοί Δείκτες

3.1 Δορυφορικά Δεδομένα

Με τον όρο δορυφορικά δεδομένα αναφερόμαστε συνήθως στις καταγραφές από διάφορες δορυφορικές αποστολές, οι οποίες παρέχουν πληροφορίες για την παρατήρηση της Γης, εξυπηρετώντας επιστημονικούς και εμπορικούς σκοπούς [9]. Τα δορυφορικά δεδομένα θεωρούνται ευρέως ως ένας από τους θεμελιώδεις πυλώνες της σύγχρονης τηλεπισκόπησης. Οι δορυφόροι δεν καταγράφουν απλές φωτογραφίες, αλλά μέσω των εργαλείων με τα οποία είναι εξοπλισμένοι, καταγράφουν ακτινοβολία και αντανάκλαση, ως αποτέλεσμα τη δημιουργία τεράστιων σετ δεδομένων, έτοιμα προς ανάλυση.



Εικόνα 3.1: Δορυφορική εικόνα τηλεπισκόπησης ενός τυπικού αγροκτήματος

Καθώς οι δορυφόροι προσφέρουν τη δυνατότητα συνεχών παρατηρήσεων σε βάθος χρόνου, η ανάλυση των παραγόμενων χρονοσειρών μπορεί να αποδειχθεί καθοριστική για τη μελέτη μακροχρόνιων φαινομένων, όπως η κλιματική αλλαγή. Τα δορυφορικά δεδομένα χωρίζονται σε πολλές υποκατηγορίες [9] με κάποιες από αυτές να είναι οι:

- Οπτικές εικόνες υψηλής ανάλυσης για λεπτομερή αστική ανάλυση.
- Εικόνες μέτριας ανάλυσης για παρακολούθηση γεωργίας.
- Εικόνες χαμηλής ανάλυσης για κλιματολογικές και περιβαλλοντικές εφαρμογές όπως πρόγνωση καιρού όπου η συλλογή ατμοσφαιρικών δεδομένων είναι τα θεμέλια της Αριθμητικής Πρόγνωσης Καιρού.
- Εικόνες radar (SAR) που διαπερνούν σύννεφα, κατάλληλες για την επιτήρηση παραμόρφωσης εδάφους και νερού.
- Εικόνες φασματικής απεικόνισης για την εξερεύνηση ορυκτών και ανάλυση βλάστησης.
- Εικόνες θερμικής χαρτογράφησης.
- LiDAR (Ανίχνευση και Μέτρηση Απόστασης μέσω φωτός) για την χαρτογράφηση υψομέτρου.

Η διαχείριση των δεδομένων από τις κατηγορίες αυτές αποτελούν Big Earth Data χρειάζεται εξειδικευμένες πλατφόρμες όπως το Cloud Processing (πλατφόρμες τύπου απομακρυσμένου νέφους

με ισχυρή υπολογιστική δύναμη) και εξειδικευμένους αλγορίθμους. Τα δορυφορικά δεδομένα ως κατά παράδοση φιλοξενούνται σε διακομιστές διαστημικών υπηρεσιών, όμως λόγω του μεγάλου όγκου δημιουργούνται επίσημα αντίγραφα και φιλοξενούνται σε μεγάλες υποδομές Cloud [10]. Δηλαδή, η πλατφόρμα Copernicus Data Space Ecosystem και το Nasa Earth Data είναι οι κύριες πηγές για τα δεδομένα δορυφόρων αλλά τα τεράστια αντίγραφα βρίσκονται σε υποδομές όπως το AWS, το Google Cloud και το Microsoft Azure επιτρέποντας σε χρήστες να φέρουν τον κώδικα δίπλα στα δεδομένα. Για να πετύχει η καινοτομική ιδέα του Cloud και η αξιοποίηση των δεδομένων η χρήση περιβαλλοντικών, ειδικά σχεδιασμένων frameworks γι' αυτό το σκοπό όπως το Google Earth Engine (GEE) μια από τις πιο γνωστές εφαρμογές για Cloud Computing, το Microsoft Planetary Computer με παρόμοια φιλοσοφία με το GEE, το Copernicus Data and Information Access Services και το βασικό framework που χρησιμοποιήθηκε από το σύστημα του Hellas Cube ήταν το Open Data Cube ένα εργαλείο που με την βοήθεια του AWS καλύπτει τις ανάγκες για γρήγορες και αποτελεσματικές εφαρμογές χωρίς να υπάρχει η ανάγκη για αποθήκευση δεδομένων. Δύο από τις μεγαλύτερες αποστολές δορυφόρων και πυλώνες των ανοιχτών δεδομένων Παρατήρησης της Γής είναι η αποστολή Sentinel της European Space Agency και η αποστολή Landsat της NASA. Οι δύο αποστολές μοιράζονται την κοινή λογική ελεύθερων δεδομένων (open data) με αποτέλεσμα να εξελίσσεται ραγδαία η επεξεργασία των δεδομένων τους. Συμπερασματικά, τα δορυφορικά δεδομένα λόγω του μεγάλου όγκου παρουσιάζουν δυσκολίες στην διαχείριση και την επεξεργασία τους αλλά με την κατάλληλη υποδομή και τα κατάλληλα εργαλεία τα δεδομένα αξιοποιούνται καθημερινά ενώ παράλληλα οι δορυφόροι παράγουν “φρέσκα” δεδομένα συνεχώς.

3.2 Δορυφορικές Αποστολές

3.2.1. Landsat

Η αποστολή Landsat αποτελεί τον «πατέρα» των δορυφορικών αποστολών και θεωρείται αδιαμφισβήτητη η αρχή της σύγχρονης δορυφορικής τηλεπισκόπησης [13][14]. Ο πρώτος δορυφόρος ονόματι Landsat-1 εκτοξεύτηκε το 1972 με απόλυτη επιτυχία. Στην συνέχεια των χρόνων έχουν εκτοξευθεί άλλες οκτώ δορυφορικές αποστολές (7 επιτυχημένες λόγω αποτυχίας του Landsat-6) με τις δύο από αυτές Landsat-8, Landsat-9 να είναι ενεργές και τα δεδομένα να χρησιμοποιούνται μέχρι και σήμερα. Για να αποκτήσει πρόσβαση στα δεδομένα των προγραμμάτων Landsat οποιοσδήποτε φορέας θα έπρεπε να πληρώσει, με τις τιμές να κινούνται μέσα στα χρόνια ξεκινώντας το 1972 με 20\$ για απλά δεδομένα και να κορυφώνεται το 1998 με 4.000\$ για δορυφορικές εικόνες [13], το 1999 όμως οι τιμές πέσανε στα 600\$, μία ακραία πτώση τιμής και η αρχή για μία αλλαγή πολιτικής. Το 2008 η Αμερικανική Γεωλογική Υπηρεσία (USGS) αποφάσισε να διαθέσει το περιεχόμενο των αποστολών δωρεάν, μία από τις μεγαλύτερες αλλαγές πολιτικής στον χώρο της τηλεπισκόπησης. Η αλλαγή αυτή είχε ως συνέπεια τη σημαντική ενίσχυση της συνεισφοράς των δεδομένων του προγράμματος Landsat σε πληθώρα επιστημονικών πεδίων. Ειδικότερα, σύμφωνα με στοιχεία του Google Scholar, έως το 2022 το Landsat καταγράφηκε σε σχεδόν 800.000 δημοσιεύσεις, καταλαμβάνοντας την πρώτη θέση, ενώ ο MODIS ακολούθησε στη δεύτερη θέση με περίπου 600.000 αναφορές [13].

Στο σύστημα Hellas Cube χρησιμοποιήθηκαν σερβερ δεδομένων από τις τελευταίες 2 αποστολές (Landsat-8, Landsat-9) λόγω ότι οι γενιές 1-5 των δορυφόρων χρησιμοποιούσαν έναν παλαιότερης τεχνολογίας αισθητήρα τον MSS (Multispectral Scanner System), ο οποίος παρείχε μόνο 4 φασματικά κανάλια (Πράσινο, Κόκκινο, Εγγύς υπέρυθρο 1&2) και μία σχετικά μεγάλη ανάλυση των 80m, με αποτέλεσμα τα δεδομένα να καθίστανται ξεπερασμένα [13], [14]. Σε συνέχεια των προηγούμενων αποστολών, ο δορυφόρος Landsat-7 εξοπλίστηκε με τον αναβαθμισμένο αισθητήρα ETM+ (Enhanced

Thematic Mapper Plus), ο οποίος αποτέλεσε τη μετεξέλιξη του αισθητήρα TM του Landsat-6. Ο ETM+ κατέγραψε δεδομένα σε οχτώ συνολικά φασματικές ζώνες, με τη χωρική τους ανάλυση να κυμαίνεται από τα 15 έως τα 60 μέτρα (έξι ζώνες στα 30m, μία θερμική στα 60m και μία παγχρωματική στα 15m), απαιτώντας συχνά τεχνικές επαναδειγματοληψίας (resampling) για την ομογενοποίησή τους [14]. Η αποστολή νούμερο 7 σταμάτησε να τρέχει τον Απρίλιο του 2022, όμως τα δεδομένα του είναι έτοιμα για ιστορική ανάλυση [13], [14]. Παρ' όλα αυτά, ο Landsat-7 δεν συμπεριλαμβάνεται στα ενεργά datasets που αξιοποιεί ο πυρήνας του Hellas Cube κυρίως λόγω της σοβαρής μηχανικής βλάβης στον μηχανισμό σάρωσης που υπέστη τον Μάιο του 2003, η οποία προκαλεί κενές, διαγώνιες λωρίδες (gaps) χωρίς δεδομένα στα άκρα των εικόνων του, αλλά και λόγω της έλλειψης αυτόματης συμβατότητας με σύγχρονα πρότυπα εναρμόνισης. Οι Landsat-8 και Landsat-9 (οι κύριες πηγές δεδομένων για το Hellas Cube) έφεραν πολλές καινοτομίες με τον αισθητήρα OLI (Operational Land Imager) που παρουσιάστηκε για πρώτη φορά το 2013 και σχεδιάστηκε με ανάλυση παγχρωματικού καναλιού 15 μέτρων καθώς και με τα πολυφασματικά κανάλια αντίστοιχης λογικής του ETM+ [13], [14]. Ο OLI και ο OLI-2 που βρίσκονται στις αποστολές 8 και 9 αντίστοιχα έχουν ως κύρια διαφορά τα bit συλλογής δεδομένων με 12 bit και 14 bit αντίστοιχα, μία σημαντική αύξηση στα 8 bit του ETM+, καθιστώντας τα πιο εξελιγμένα δεδομένα που χρησιμοποίησε το Hellas Cube [14]. Με τις νέες προσθήκες του Landsat-8 αναπτύχθηκε και ένας τρόπος προσθήκης ενός pixel με το χαρακτηριστικό όνομα qa_pixel για τον εντοπισμό σύννεφων [13], [14]. Συμπερασματικά οι αποστολές Landsat αποτελούν τις πιο διάσημες και χρήσιμες δορυφορικές αποστολές του ανθρώπινου είδους με τα δεδομένα του από τους διάφορους και περίπλοκους με τα χρόνια αισθητήρες έχουν επηρεάσει πολλές επιστήμες εκτός της τηλεπισκόπησης και αποδείχθηκαν χρήσιμα για τον σκοπό του Hellas Cube [13], [14].

3.2.2. SPOT

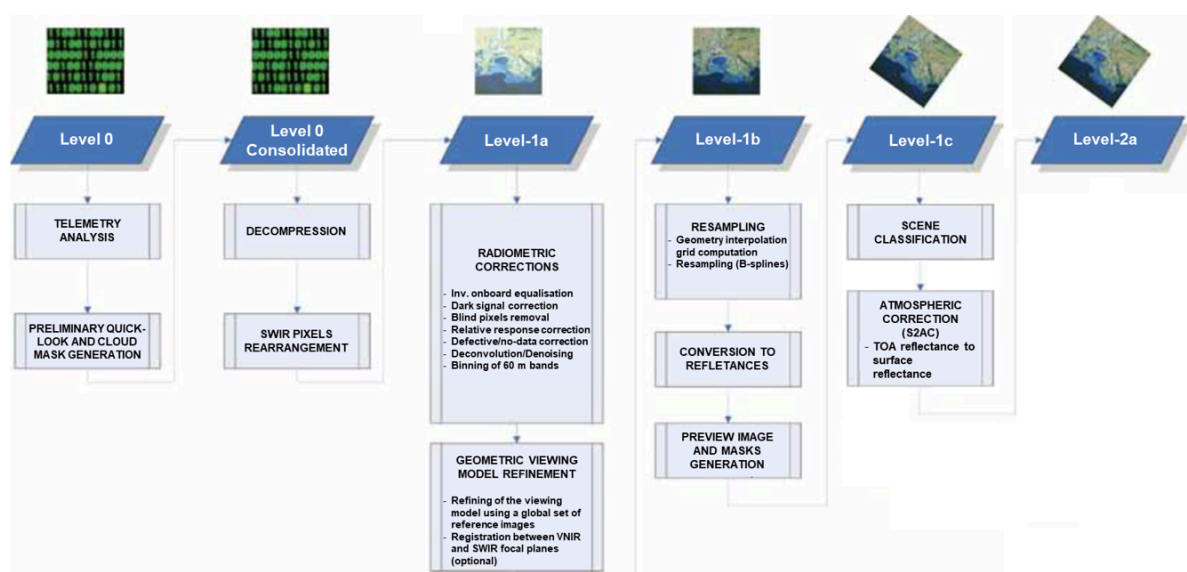
Μία από τις βασικές δορυφορικές αποστολές είναι οι ευρωπαϊκές αποστολές SPOT (Satellite Pour l'Observation de la Terre), προγράμματα που ξεκίνησαν το 1978 σε συνεργασία της Γαλλίας, Σουηδίας και Βελγίου. Συνολικά έχουν υπάρξει 7 αποστολές και η πρώτη αποστολή (SPOT 1) εκτοξεύτηκε το 1986, στην συνέχεια κάθε αποστολή βελτιώνει σταδιακά τις δυνατότητες του δορυφόρου με την χωρική ανάλυση στο SPOT 1 να βρίσκεται στα 20m και στην αποστολή του SPOT 5 το 2002 η ανάλυση έφτασε μέχρι και τα 5m [11]. Οι πρώτες 5 αποστολές SPOT βρισκόντουσαν σε κρατικό επίπεδο όμως οι αποστολές 6 και 7 μετατράπηκαν σε εμπορικό μοντέλο για απόλυτη ιδιωτική χρήση. Έτσι τα SPOT 6 και SPOT 7 με την ευφυΐα των προηγούμενων 5 αποστολών εκτοξεύθηκαν το 2012 και το 2014 με ένα νέο εμπορικό μοντέλο, βελτιωμένα χαρακτηριστικά όπως την χωρική ανάλυση να φτάνει πλέον στα 1.5m [12] και εκτιμώμενο χρόνο ζωής από τα 5 χρόνια των προηγούμενων αποστολών στα 10 χρόνια. Στις μέρες μας ο SPOT 6 είναι ακόμα λειτουργικός και έχει αυξηθεί η διάρκεια ζωής του έως το 2026, σε αντίθεση με τον SPOT 7 που σταμάτησε να λειτουργεί στο 2023 και έχει μετονομαστεί σε "Azersky" καθώς μεταπωλήθηκε στην χώρα του Αζερμπαϊτζάν. Οι αποστολές SPOT χρησιμοποιήθηκαν κυρίως για χαρτογραφία (επιστήμη της συλλογής γεωχωρικών δεδομένων, επεξεργασίας τους και οπτικοποίησής τους), ταξινόμηση κάλυψης Γης (μέθοδος κατηγοριοποίησης της επιφάνειας της Γης) και εντοπισμό αλλαγής γεωχώρου [11] και αποτέλεσαν τα θεμέλια για την αποστολή Sentinel.

3.2.3. Sentinel

Το πρόγραμμα Copernicus Sentinel του Ευρωπαϊκού Οργανισμού Διαστήματος (ESA) είναι ένας ισχυρός απόγονος των αποστολών SPOT που εστιάζει στα ανοιχτά δεδομένα, στην κάλυψη ηπείρων και αποτελείται μέχρι σήμερα από έξι βασικές οικογένειες αποστολών (Sentinel 1-6), με την κάθε μία

να εστιάζει σε διαφορετικές πτυχές παρατήρησης της Γής. Οι σειρές από αποστολές καλύπτουν ανάγκες όπως καιρικές συνθήκες, παρατήρηση εδάφους και ωκεανών, παρατήρηση κλιματικής αλλαγής, παρατήρηση ατμόσφαιρας καθώς και ωκεανογραφία [15].

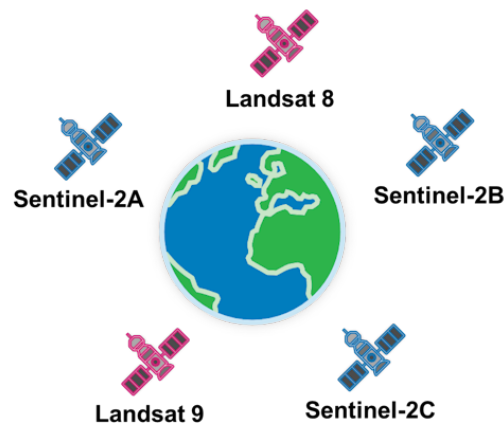
Κάθε δορυφόρος Sentinel φέρει εξειδικευμένους αισθητήρες, καθιστώντας τον μοναδικό στο πεδίο εφαρμογής του, όπως τον Sentinel-3, εξοπλισμένος με αισθητήρες όπως τον OLCI(ocean and land colour instrument) και SLSTR(sea and land surface temperature radiometer) ιδανικό για παρατήρηση και παραγωγή δεδομένων για ωκεανούς και θάλασσες. Στο Hellas Cube όμως αξιοποιήθηκαν δεδομένα της αποστολής Sentinel-2. Η αποστολή αποτελείται από ένα σμήνος δορυφόρων και συγκεκριμένα δύο δορυφόρους με διαφορά φάσης 180 μοιρών, με αποτέλεσμα να βρίσκονται σε διαφορετικά σημεία πάνω στην Γη. Οι δορυφόροι Sentinel-2A και Sentinel-2B που εκτοξεύθηκαν στο 2013 με διαφορά 18 μήνες είναι ακόμα ενεργοί και σε λίγα χρόνια το A-unit θα αντικατασταθεί με τον Sentinel-2C [16]. Η σειρά 2 των αποστολών Sentinel φέρει έναν πολυφασματικό αισθητήρα (Multispectral Imager, MSI), ο οποίος καταγράφει δεδομένα σε 13 διαφορετικές φασματικές ζώνες καταγράφοντας σκηνές κάθε 10 μέρες για το ίδιο σημείο, με αποτέλεσμα να υπάρχει μία μεγάλη ποικιλία σε δεδομένα για τις αναλύσεις του συστήματος. Ένας ακόμα λόγος προτίμησης των συγκεκριμένων dataset είναι ότι τα εν λόγω dataset αναδεικνύονται από την επιστημονική κοινότητα ως θεμελιώδη για περιβαλλοντικές εφαρμογές καθώς δεν είναι “raw” και έχουν υποστεί επεξεργασία από τον επεξεργαστή Sen2Cor ο οποίος διορθώνει τα δεδομένα που παράγει ο αισθητήρας του δορυφόρου. Με λίγα λόγια τα δεδομένα περνάνε από διάφορα επίπεδα επεξεργασίας με το πρώτο επίπεδο να είναι το επίπεδο Level-0 για όπου τα δεδομένα θεωρούνται ακατέργαστα. Μετά από ραδιομετρικές και γεωμετρικές διορθώσεις (Level-1a & Level-1b) τα δεδομένα των εικόνων έχουν γίνει πιο ξεκάθαρα από ποτέ (Level-1c) και ο Sen2Cor κατηγοριοποιεί τα pixel της δορυφορικής φωτογραφίας και τα κατατάσσει σε 11 κλάσεις. Οι κλάσεις αυτές αναγνωρίζουν σύννεφα και βοηθούν στην προσθήκη μάσκας στα δεδομένα για την απομάκρυνση παραπλανητικών (pixel συννεφων) από τα δεδομένα μας για πιο ακριβή αποτελέσματα. Τα δεδομένα του Sentinel και συγκεκριμένα του Sentinel-2 εκμεταλλεύονται ευρέως, λόγω της ανοιχτής τους “φύσης” και συνεργάζονται τέλεια με τα δεδομένα των Landsat γι’ αυτό και ενσωματώθηκαν από το σύστημα του Hellas Cube.



Εικόνα 3.2: Επεξεργαστικά επίπεδα από το Level-0 έως το Level-2a

3.2.4. Η αρμονία μεταξύ των Landsat και Sentinel

Οι δορυφορικές αποστολές Landsat και Sentinel αποτελούν δύο κολοσσούς προγράμματα με μεγάλη επιρροή στην εξέλιξη των δορυφορικών αποστολών. Τα δεδομένα που παράγουν είναι εξαιρετικά χρήσιμα και γι' αυτό τον λόγο στο σύστημα του Hellas Cube υλοποιήθηκε ένας συνδυασμός των δεδομένων για μέγιστα αποτελέσματα και μεγάλη επεξεργαστική ποικιλία. Όπως έχει προαναφερθεί στο Hellas Cube αξιοποιήθηκαν δεδομένα από τους Landsat 8,9 και Sentinel-2 Level-2(A,B,C). Οι αισθητήρες των αποστολών 8 και 9 (OLI, OLI-2) και του Sentinel-2 (MSI) παίρνουν παρόμοιες μετρήσεις με αποτέλεσμα η NASA(National Aeronautics and Space Administration) να αναπτύξει ένα προϊόν που συνδυάζει τα δεδομένα των τριών αποστολών που μοιράζονται τόσα κοινά. Το Harmonized Landsat Sentinel (HLS) καλύπτει τις ανάγκες την κοινότητας των περιβαλλοντικών εφαρμογών καθώς με την αποφυγή των πολλαπλών προϊόντων, τα συστήματα απλοποιούνται.



Εικόνα 3.3: Δορυφόροι του HLS

Για την επίτευξη αρμονίας μεταξύ των δεδομένων, τα δεδομένα του Sentinel “ευθυγραμμίζονται” με τις προδιαγραφές του Landsat. Συγκεκριμένα, δεδομένου ότι η χωρική ανάλυση του Sentinel είναι 10X10 μέτρα ανά pixel, ενώ του Landsat 30X30 μέτρα ανά pixel, ένα pixel του Landsat αντιστοιχεί σε 9 pixels του Sentinel (βλ. Μαθηματικές πράξεις 3.2.4.1).

$$A_{Landsat} = 30m \times 30m = 900m^2 \quad A_{Sentinel} = 10m \times 10m = 100m^2 \quad \frac{A_{Landsat}}{A_{Sentinel}} = \frac{900m^2}{100m^2} = 9 \text{ pixels} \quad (3.2.4.1)$$

Με την “γενικοποίηση” της ανάλυσης των pixel χάνεται αρκετή πληροφορία αλλά εξοικονομείται αρκετή υπολογιστική δύναμη. Γι' αυτό στο Hellas Cube εφαρμόστηκε ένας συνδυασμός προϊόντων Sentinel-2 Level 2A που δέχεται επεξεργασία Sen2Cor (Εικόνα 3.2) και HLS που τα δεδομένα Landsat και Sentinel-2 δέχονται την ίδια επεξεργασία Fmask έτσι ώστε κατά την ανάλυση δεδομένων και εφαρμογή μαθηματικών τύπος να μην χάνεται πληροφορία. Σε σύγκριση των δύο επεξεργασιών πάνω σε δεδομένα του Sentinel όπου εφαρμόστηκαν πάνω σε 20 σκληρές γύρω στον κόσμο τα αποτελέσματα παρουσιάστηκαν μικτά με τον Fmask να αναδεικνύεται περισσότερο σε εύκρατο κλίμα ενώ ο Sen2Cor παρουσίασε καλύτερα αποτελέσματα σε αγροτικές περιοχές (Εικόνα 3.4) [18]. Όσον αφορά τη συνολική αξιολόγηση της ακρίβειας των δύο αλγορίθμων, ο Sen2Cor επικράτησε με ποσοστό 92% έναντι 89% του Fmask. Η μικρή αυτή απόκλιση καθιστά και τις δύο επιλογές κατάλληλες για αξιοποίηση στο πλαίσιο του Hellas Cube.

Scene	Location	OA Difference Area			OA Same Area			OA Total Area		
		F	A	S	F	A	S	F	A	S
ID	Average (all scenes)	45	56	62	97	97	97	89	91	92
1	Antarctic	36	51	56	95	98	100	79	86	88
2	Argentina, Buenos Aires	90	59	59	98	98	98	98	96	96
3	Australia, Lake Lefroy	59	49	67	100	100	100	96	95	97
4	Bolivia, Puerto Siles	90	22	58	100	100	100	99	98	99
5	China, Dunhuang	56	40	74	97	97	100	73	64	85
6	Estonia, Tallin	40	71	78	98	98	99	90	94	96
7	Germany, Berlin	57	76	67	100	100	100	97	98	98
8	Italy, Etna	32	70	71	100	100	100	88	95	95
9	Kazakhstan, Balkhash	47	75	45	92	91	91	87	90	87
10	Mexico, Cancun	44	59	66	99	99	99	89	92	93
11	Morocco, Quarzazate	71	86	45	100	100	100	96	98	93
12	Mosambique, Maputo	75	35	45	85	85	85	84	83	83
13	Netherlands, Amsterdam	38	63	64	96	96	96	86	90	91
14	Phillipines, Manila	46	69	67	98	98	98	92	95	94
15	Russia, Sachalin	85	51	63	99	98	98	98	93	94
16	Russia, Yakutsk	64	49	53	97	97	97	94	92	93
17	Spain, Barrax-1	25	74	62	99	99	99	83	93	91
18	Spain, Barrax-2	64	92	90	100	100	100	97	99	99
19	Switzerland, Davos	16	42	46	86	97	97	54	72	74
20	USA, Rimrock	50	52	63	99	99	99	98	99	98

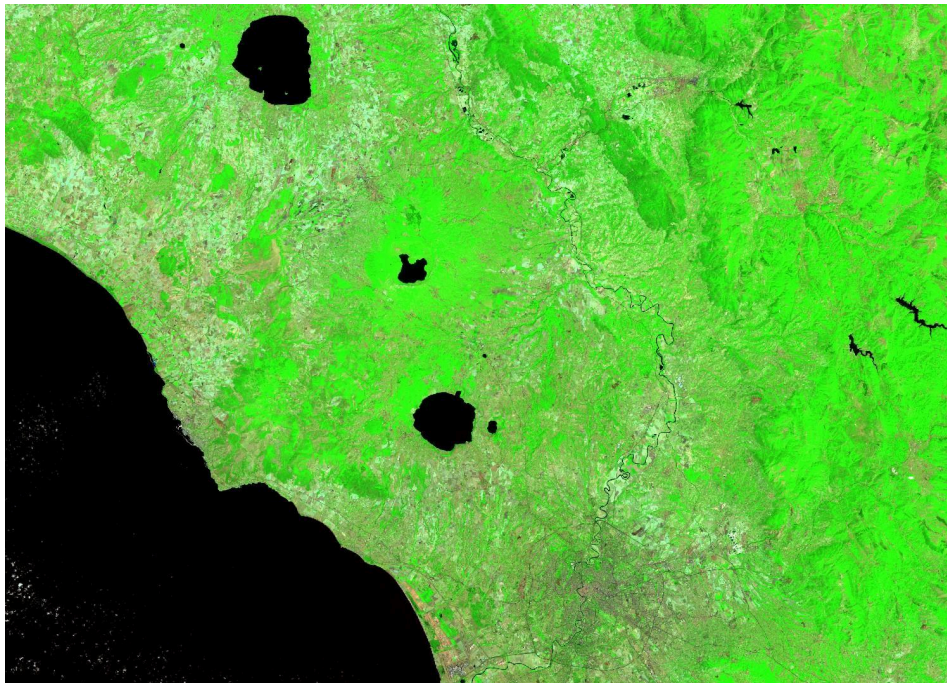
Εικόνα 3.4: Σύνοψη συνολικής ακρίβειας (ποσοστό %) (F = Fmask, A = ATCOR, S = Sen2Cor). [18]

3.2.5. Φασματικά Κανάλια

Τα φασματικά κανάλια που παράγουν οι αισθητήρες των αποστολών μοιράζονται κοινά χαρακτηριστικά γι' αυτό και μπορούν να συνδυαστούν τόσο εύκολα μέσω του HLS. Τα φασματικά μπορούν να χαρακτηριστούν σαν τα χρώματα των pixel. Τα φασματικά κανάλια των Landsat-8 & Landsat-9 είναι τα ίδια 11 κανάλια καθώς ο Sentinel-2 παρέχει 13. Όμως δεν χρησιμοποιούνται όλα για ανάλυση καθώς πολλά παράγονται για εφαρμογή μάσκας στα υπόλοιπα δεδομένα. Μερικά από τα φασματικά κανάλια που μοιράζονται είναι οι εξής:

1. **Red, Green, Blue (Κόκκινο, Πράσινο, Μπλε):** Τα συγκεκριμένα κανάλια αποδίδουν τα φυσικά χρώματα (True Color), προσομοιώνοντας την ανθρώπινη αντίληψη του οπτικού φάσματος. Για παράδειγμα, η υγιής βλάστηση απεικονίζεται με πράσινο χρώμα. Σε όλα τα προϊόντα, η κατανομή των RGB καναλιών παραμένει σταθερή: το κανάλι B02 αντιστοιχεί στο Μπλε, το B03 στο Πράσινο και το B04 στο Κόκκινο.
2. **Near Infrared - Nir (Εγγύς Υπέρυθρο):** Το NIR είναι ένα φασματικό κανάλι το οποίο δεν μπορεί να το εντοπίσει το ανθρώπινο μάτι και βρίσκεται αμέσως μετά το κόκκινο. Πιο συγκεκριμένα, το κόκκινο βρίσκεται σε μήκη κύματος από περίπου 0.625μm έως 0.75μm και το εγγύς υπέρυθρο βρίσκεται σε μήκη περίπου 0.75μm έως 1.4μm. Το NIR καθίσταται ένα από τα σημαντικότερα κανάλια στην τηλεπισκόπηση με την χρήση του να κυμαίνεται από εφαρμογές σε τύπους βλάστησης καθώς και νερού, δηλαδή τα υγιή φυτά αντανακλούν το NIR σε τεράστιες ποσότητες ενώ το καθαρό νερό τις απορροφά. Στον Sentinel-2 το NIR χωρίζεται σε δύο διαφορετικά κανάλια, στο B08 “ευρύ” εγγύς υπέρυθρο και στο B08A “στενό” εγγύς υπέρυθρο με την κύρια τους διαφορά να είναι το ευρύ κανάλι να καλύπτει μήκος κύματος περίπου 0.0115μm ενώ το στενό 0.02μm. Σε αντίθεση οι Landsat 8/9 τοποθετούν το NIR στο κανάλι B05 καλύπτοντας 0.04μm. Για να επιτευχθεί αρμονία μεταξύ των δύο καναλιών για το προϊόν HLS η NASA επέλεξε να λάβει υπόψη μόνο το B08A του Sentinel και ο λόγος είναι γιατί το B05 των Landsat και το B08A του Sentinel-2 έχουν το ίδιο κεντρικό μήκος κύματος στα 0.865 μm.

- 3. Short-Wave Infrared – SWIR (Βραχύ Υπέρυθρο):** Η φασματική ζώνη του SWIR το καθιστά ένα πιο βαθύ και ακόμα πιο πολύ μη ορατό φάσμα από το NIR, με μήκη κύματος περίπου από 1.0μm έως 3.0μm. Καθώς το SWIR καλύπτει τόσο μεγάλο εύρος μήκους κύματος τα κανάλια χωρίζονται σε SWIR1 και SWIR2 όπου σε Landsat 8/9 στις θέσεις B06 και B07 και Sentinel στις θέσεις B11 και B12 να κυμαίνονται και στα δύο προϊόντα από 1.565μm έως 1.655μm και 2.1μm έως 2.294μm αντίστοιχα με ενοποίηση τους γίνεται άψογα για το προϊόν HLS. Το βραχύ υπέρυθρο έχει πολλές ιδιότητες με αποτέλεσμα η χρήση του να είναι σημαντική για εντοπισμό υγρασίας καθώς το SWIR απορροφάται από το νερό ακόμα περισσότερο από ότι το NIR (Εικόνα 3.4 παρατηρούμε ότι το νερό εμφανίζεται σκοτεινό-μαύρο). Σε σύγκριση με το NIR που απορροφάται μόνο από το καθαρό νερό το SWIR θεωρείται πιο ευαίσθητο με αποτέλεσμα να απορροφάται και από άλλες μορφές νερού όπως χιόνι και πάγο, ενώ ανακλάται από τα σύννεφα, καθιστώντας το ιδανικό για την διαφοροποίηση χιονιού και σύννεφων καθώς τα άλλα κανάλια δεν μπορούν να την εντοπίσουν. Τέλος το SWIR μπορεί και διαπερνά τον καπνό με αποτέλεσμα να μπορούμε να εντοπίσουμε έδαφος ανάμεσα και από εμπόδια καπνών.



Εικόνα 3.5: Σύνθετη εικόνα HLS SWIR της Ρόδης. Λήψη στις 24-07-2022, επεξεργασμένη μέσω του Sentinel Hub.

- 4. Red Edge (Ερυθρό Άκρο):** Το φασματικό κανάλι αντιστοιχεί σε περιοχή του ηλεκτρομαγνητικού φάσματος μεταξύ του κόκκινου και του εγγύς υπέρυθρου καλύπτοντας ένα στενό μήκος κύματος περίπου 0.68μm έως 0.78μm με μεγάλη ακρίβεια σε μετρήσεις βλάστησης. Βασίζεται πάνω σε δύο ιδιότητες των φύλλων να απορροφούν το ερυθρό χρώμα και να ανακλά το εγγύς υπέρυθρο. Στον Sentinel-2, καθώς δεν υπάρχει μέτρηση στους Landsat 8/9, αφιερώθηκαν τρία κανάλια για το Red Edge (Vegetation 1,2,3 στα κανάλια B05, B06, B07 αντίστοιχα) με το κάθε band να έχει και διαφορετικές ιδιότητες όπως το B05 ο οποίος είναι εξαιρετικά ευαίσθητος σε μεταβολές της χλωροφύλλης εντοπίζοντας εύκολα την υγεία των φυτών ακόμα και προβλέποντας ξηρασία, τοξικότητα και προσβολή σε παθογόνους μικροοργανισμούς 1-2 εβδομάδες πριν εμφανιστούν συμπτώματα [19].

- 5. Κοινά και μη κοινά κανάλια μεταξύ Sentinel-2 και Landsat 8/9:** Τα προϊόντα των αποστολών αποτελούνται από περισσότερα άλλα κανάλια τα οποία δεν εξυπηρετούν τις ανάγκες σε επίπεδο εφαρμογής του Hellas Cube, αλλά υπηρετούν διαφορετικό και πολλές φορές σημαντικότερο σκοπό. Μερικά από αυτά να είναι το κανάλι υδάτων και αερολυμάτων που εμφανίζεται και στα δύο προϊόντα, σχεδιασμένο για ατμοσφαιρική διόρθωση, δηλαδή την αφαίρεση του ατμοσφαιρικού θορύβου από τα υπόλοιπα κανάλια της εικόνας καθώς και για χαρτογράφηση του θαλάσσιου πυθμένα, η παγχρωματική ζώνη που παρουσιάζεται αποκλειστικά στο Landsat 8/9 για την μετατροπή pixel διάστασης 30 μέτρα σε 15 μέτρα, τα κανάλια Cirrus (Sentinel-2, Landsat 8/9) και Water Vapour (Sentinel-2) που σχεδιάστηκαν αποκλειστικά για ατμοσφαιρική διόρθωση και εφαρμόζονται σε αλγόριθμους μασκών όπως τον Fmask και Sen2Cor. Τέλος το θερμικό υπέρυθρο που συνεισφέρει σε θερμοδυναμικές μελέτες λόγω της ακριβούς μέτρησης της θερμοκρασίας επιφάνειας εδάφους.

3.3 Κατηγορίες Περιβαλλοντικών Δεικτών

Τα φασματικά κανάλια των δορυφορικών αποστολών αξιοποιούνται για τον υπολογισμό περιβαλλοντικών δεικτών, με στόχο την εξαγωγή χρήσιμων συμπερασμάτων για την κατάσταση του εδάφους. Κάθε δείκτης κατατάσσεται σε μία συγκεκριμένη κατηγορία και βασίζεται σε έναν εξειδικευμένο τύπο:

1. Δείκτες βλάστησης, οι οποίοι αξιοποιούν κανάλια RGB, NIR και Red Edge για την εκτίμηση παραμέτρων όπως η πρασινότητα, η πυκνότητα και η περιεκτικότητα σε χλωροφύλλη.
2. Οι δείκτες υγρασίας εκμεταλλεύονται παρόμοια φασματικά κανάλια με τους δείκτες βλάστησης εστιάζοντας στον εντοπισμό υδατικής κατάστασης της βλάστησης.
3. Οι δείκτες νερού έχουν παρόμοιο ρόλο με τους δείκτες υγρασίας αλλά είναι σχεδιασμένοι να εφαρμόζονται σε υδάτινα σώματα για τον εντοπισμό διαύγειας του νερού και γενική ύπαρξης.
4. Οι δείκτες καμένου εδάφους που εντοπίζουν καμένο έδαφος και αναλύουν την ζημία πυρκαγιών.
5. Η κατηγορία δεικτών χιονιού που εντοπίζουν χιονισμένες περιοχές και την φωτεινότητα χιονιού.
6. Τέλος οι δείκτες αστικών περιοχών/εδάφους που είναι σχεδιασμένοι για την σύγκριση βλάστησης σε αστικές περιοχές και σε γυμνό έδαφος.

Με βάση αυτές τις κατηγορίες επιλέχθηκαν οι περιβαλλοντικοί δείκτες που θα έχει την δυνατότητα να εφαρμόσει ο χρήστης στις προσωπικές περιοχές ενδιαφέροντος, στο σύστημα του Hellas Cube.

3.4 Περιβαλλοντικοί Δείκτες & Αλγόριθμος του Hellas Cube

Όπως έχει προαναφερθεί ένα κύριο μέρος του Hellas Cube είναι η ανάλυση των δορυφορικών δεδομένων. Ένα βασικό μέρος τη ανάλυσης των δορυφορικών δεδομένων, είναι η εφαρμογή περιβαλλοντικών δεικτών οι οποίοι έχουν σχεδιαστεί για τα δεδομένα των αποστολών Landsat και Sentinel. Στην πρακτική εφαρμογή τους, οι φασματικοί δείκτες υπολογίζονται συνήθως μέσω της εφαρμογής συγκεκριμένων μαθηματικών πράξεων μεταξύ των φασματικών καναλιών. Μέσω αυτών

των δεικτών, είναι δυνατή η εξαγωγή σύνθετων πληροφοριών και η ανάλυση περιβαλλοντικών παραμέτρων που δεν είναι άμεσα προσεγγίσιμες με άλλες μεθόδους. Οι δείκτες που ενσωματώθηκαν στο σύστημα Hellas Cube είναι:

1. **Normalized Difference Vegetation Index (NDVI):** Ένας βασικός και ευρέως χρησιμοποιούμενος δείκτης εξέτασης βλάστησης. Ο δείκτης εκμεταλλεύεται τα κανάλια NIR και RED, υπολογίζοντας τη διαφορά τους και διαιρώντας τη με το άθροισμα τους (Μαθηματικός τύπος (3.4.1)). Με αυτόν τον τρόπο, εντοπίζεται η χλωρή βλάστηση στα datasets καθώς τα υγιή φυτά απορροφούν το “χρώμα” RED και αντανακλούν το “χρώμα” NIR.

$$NDVI = \frac{NIR-RED}{NIR+RED} \quad (3.4.1)$$

Οι τιμές του δείκτη κυμαίνονται από το -1 έως το 1 με τα συμπεράσματα για κάθε ανάλυση να διαφέρουν ανάλογα με τα datasets και τις περιοχές που εξετάζονται. Μία γενική εικόνα των αποτελεσμάτων είναι του Μαθηματικού Τύπου (3.6) όπου οι αρνητικές τιμές εκφράζουν νερό, χιόνι και σύννεφα ενώ οι θετικές τιμές 0-0.2 εκφράζουν μη-πράσινο έδαφος. Οι θετικές τιμές μεγαλύτερες του 0.2 εκφράζουν πράσινες περιοχές δηλαδή υγιή βλάστηση [20].

Πίνακας 3.7: NDVI κατηγοριοποίηση αποτελεσμάτων [20]

Τιμές NDVI	Συμπέρασμα
[-1 , 0]	Νερό, Χιόνι, Σύννεφα
[0 , 0.2]	Άγονο Έδαφος, Δομημένη περιοχή, Βράχοι
[0.2 , 1]	Βλάστηση

2. **Normalized Difference Chlorophyll Index (NDCI):** Ο δείκτης κανονικοποίησης διαφοράς χλωροφύλλης μπορεί να χρησιμοποιηθεί μόνο με τα dataset της αποστολής Sentinel-2 καθώς παρέχει το κανάλι Red-edge. Ο σκοπός του δείκτη είναι να εντοπίσει την χλωροφύλλη σε υδάτινα σώματα. Υπολογίζεται η κανονικοποιημένη διαφορά των Red-edge(1) και Red. Ο δείκτης είναι αποτέλεσμα των σωμάτων με υψηλή χλωροφύλλη (και των φυτών) να ανακλούν το Red-edge πολύ περισσότερο από ότι το NIR και, όπως προαναφέρθηκε, να απορροφά το Red. Οι τιμές του δείκτη (όπως σε όλες τις κανονικοποιημένες διαφορές) κυμαίνεται από τις τιμές -1 έως 1 και οι τιμές κοντά στο 1 αντικατοπτρίζουν σώματα με υψηλή χλωροφύλλη. [21]

$$NDCI = \frac{RedEdge1-RED}{RedEdge1+RED} \quad (3.4.2)$$

3. **Normalized Difference Turbidity Index (NDTI):** Ο συγκεκριμένος δείκτης εστιάζει στον εντοπισμό διαύγειας νερού, δηλαδή μπορεί να εντοπίσει πόση ρύπανση, λάσπη κ.λπ. βρίσκεται σε ένα υδάτινο σώμα. Η φόρμουλα αποτελείται από τα κανάλια Red (ερυθρό) και Green (πράσινο), δεδομένου ότι το καθαρό νερό παρουσιάζει υψηλότερη φασματική ανάκλαση στο πράσινο τμήμα του φάσματος, ενώ το θολό/βρώμικο νερό παρουσιάζει αυξημένη ανάκλαση στο ερυθρό. Με αποτέλεσμα οι τιμές που αποδίδει πλησίον του -1 για καθαρά ύδατα και τιμές πλησίον του +1 για θολά ή ρυπασμένα νερά [42].

$$NDTI = \frac{RED-GREEN}{RED+GREEN} \quad (3.4.3)$$

4. **Normalized Difference Water Index (NDWI):** Ο παρών δείκτης χρησιμοποιείται στον εντοπισμό υδάτινου σώματος ή υγρασίας σε μια περιοχή βλάστησης ή εδάφους. Η κανονικοποίηση διαφοράς αποτελείται από το πράσινο το οποίο δεν απορροφάται από το νερό και την βλάστηση και το NIR που απορροφάται από το νερό και αντανακλάται έντονα από την βλάστηση.

$$NDWI = \frac{GREEN-NIR}{GREEN+NIR} \quad (3.4.4)$$

Τα αποτελέσματα της φόρμουλα σε Dataset Sentinel & Landsat δεν διαφέρουν με τις τιμές κοντά στο +1 να χαρακτηρίζουν επιφάνειες νερού, τιμές κοντά στο 0 πλημμύρες/υγρασία και τιμές κοντά στο -1 περιοχές ξηρασίας. [22]

Πίνακας 3.11: NDWI κατηγοριοποίηση αποτελεσμάτων[22]

Τιμές NDWI	Συμπέρασμα
[-1 , -0.3]	Ξηρασία
[-0.3 , 0]	Ξηρασία μέτριου επιπέδου, μη-υδατινές περιοχές
[0 , 0.2]	Πλημμύρες , υγρασία
[0.2 , 1]	Υδάτινη περιοχή

5. **Normalized Difference Moisture Index (NDMI):** Ο δείκτης κανονικοποίησης διαφοράς υγρασίας ή αλλιώς ως NDWI Gao style έχει παρόμοια λογική με τον NDWI. Εστιάζει στον εντοπισμό της υγρασία του στον γεωχώρο και το καταφέρνει μεγαλύτερη επιτυχία καθώς στη φόρμουλα του αντί του καναλιού NIR εφαρμόζεται το κανάλι SWIR, στο οποίο η ευαισθησία απορρόφησης του από ύδατα είναι μεγαλύτερη. [41]

$$NDMI = \frac{NIR-SWIR1}{NIR+SWIR1} \quad (3.4.5)$$

6. **Normalized Difference Built-up Index (NDBI):** Ο δείκτης NDBI χρησιμοποιείται για την ανάδειξη δομημένων περιοχών και αστική ανάπτυξη όπως δρόμους κτήρια κ.λπ.. Ως παράμετροι υπολογισμού παίρνει το κανάλι SWIR το οποίο αντανακλάται από επιφάνειες τσιμέντου ενώ επιφάνειες βλάστησης το απορροφούν και αντανακλούν το κανάλι NIR. Με αποτέλεσμα όσο μεγαλύτερο είναι το NDBI τόσο περισσότερο αστικοποιημένη και πυκνοδομημένη είναι η περιοχή. [23]

$$NDBI = \frac{SWIR1-NIR}{SWIR1+NIR} \quad (3.4.6)$$

7. **Normalized Difference Snow Index (NDSI):** Ο κανονικοποιημένος δείκτης διαφοράς χιονιού αποτελεί το βασικότερο εργαλείο για τον εντοπισμό χιονοκάλυψης. Ωστόσο, παρουσιάζει σημαντικό περιορισμό, καθώς μπορεί να εμφανίσει εσφαλμένες ταξινομήσεις,

συγχέοντας το χιόνι με τα νέφη, παρόλο που στην φόρμουλα εφαρμόζονται τα κανάλια SWIR και Green που απορροφάται από το χιόνι (και όχι από τα σύννεφα) και ανακλάται αντίστοιχα. Για τον δείκτη NDSI οι τιμές >0.42 έχουν μεγάλη πιθανότητα να υπάρχει χιονοκάλυψη. [24]

$$NDSI = \frac{GREEN-SWIR1}{GREEN+SWIR1} \quad (3.4.7)$$

- 8. Normalized Burn Ratio (NBR):** Ο δείκτης NBR χρησιμοποιείται για τον εντοπισμό καμένων εκτάσεων και την εκτίμηση της έντασης πυρκαγιών. Ο δείκτης βασίζεται στη χρήση του NIR και του SWIR. Η υγιής βλάστηση παρουσιάζει υψηλή ανάκλαση στο NIR και χαμηλή στο SWIR (SWIR2, με χαμηλότερη ανάκλαση από το SWIR1), ενώ στις καμένες περιοχές παρατηρείται η αντίστροφη συμπεριφορά. Για την ακριβέστερη εκτίμηση της ζημιάς, προτιμάται ο υπολογισμός του δείκτη dNBR(differenced NBR), ο οποίος προκύπτει από την σύγκριση των τιμών πριν και μετά το συμβάν. Στην κλίμακα του δείκτη, τιμές πλησίον του -1 υποδεικνύουν καμένο έδαφος ή στάχτη, ενώ τιμές πλησίον του +1 αντιστοιχούν σε υγιή, πυκνή βλάστηση [25].

$$NBR = \frac{NIR-SWIR2}{NIR+SWIR2} \quad dNBR = NBR_{pre} - NBR_{post} \quad (3.4.8)$$

Ο αλγόριθμος που χρησιμοποιείτε στο Hellas Cube:

- **Water Observation from Space (WOfS):** Ο αλγόριθμος WOfS αναπτύχθηκε από την Γεωεπιστημη της Αυστραλίας (Geoscience Australia, GA) ως αντίδραση στις σοβαρές πλημμύρες στην ανατολική Αυστραλία στα τέλη του 2010. Στον αλγόριθμο αξιοποιήθηκαν δεδομένα του Landsat καθώς εφαρμόστηκε για κάθε pixel, που διαθέτει από την εκτόξευση της αποστολής το 1987, και είχαν ως αποτέλεσμα ενός frequency layer, δηλαδή το ποσοστό όπου το pixel χαρακτηρίστηκε ως νερό σε σχέση με το πόσες φορές ήταν ορατό (ratio/clear_pixels). Ο πυρήνας του αλγορίθμου είναι η χρήση δέντρων απόφασης συνδυάζοντας την ανακλαστικότητα των φασματικών ζωνών: Blue, Green, Red, NIR SWIR(1&2) και εφαρμόζοντας δείκτες κανονικοποίησης διαφοράς διαχωρίζει το νερό από σκιές, σύννεφα, σκιές σύννεφων, έδαφος, βλάστηση και το κάθε pixel χαρακτηρίζεται ως υγρό ή ξηρό (WET or DRY). Ο WOfS είναι ένας αλγόριθμος με καλύτερα αποτελέσματα από ότι οι απλοί δείκτες NDWI, NDMI κ.λπ., λόγω την πολυπλοκότητας του. [26]

3.5 Επιλογή δεικτών και αλγορίθμων

Η επιλογή των συγκεκριμένων δεικτών και αλγορίθμων έγινε με βάση τις ανάγκες των χρηστών που προσπαθεί να προσελκύσει και να αξιοποιηθεί το σύστημα και εργαλείο Hellas Cube. Ως ένα σύστημα ανάλυσης δορυφορικών δεδομένων είναι δίκαιο να καλυφθούν όλες οι κατηγορίες δεικτών (βλάστησης, υγρασίας, υδάτων, χιονιού και αστικών περιοχών) με τους βασικότερους και πιο γνωστούς δείκτες που εφαρμόζονται παγκόσμια και δείκτες ενσωματωμένους ήδη στο framework του OpenDataCube. Δεδομένου ότι οι δείκτες και αλγόριθμοι για τη βλάστηση και τα ύδατα παρουσιάζουν μεγάλη ποικιλία και υψηλή χρηστικότητα, η εφαρμογή ενός ευρύτερου συνόλου από αυτούς κρίνεται επιβεβλημένη για την εξαγωγή αξιόπιστων συμπερασμάτων. Οι ανάγκες όμως είναι απειριορίστες και το Hellas Cube μπορεί να καλύψει ένα μικρό μέρος αλλά η συνεισφορά να είναι αρκετά μεγάλη στην Γεω-επιστήμη της Ελλάδας. Η ενσωμάτωση του συνόλου αυτών των δεικτών και αλγορίθμων από τον δείκτη NDVI έως τον ιδιαίτερα απαιτητικό, σε επίπεδο διασταύρωσης δεδομένων, αλγόριθμο WOfS διασφαλίζει ότι το σύστημα δεν αποτελεί έναν απλό μηχανισμό ανάκτησης εικόνων, αλλά ένα

ισχυρό εργαλείο επιστημονικής ανάλυσης. Συνεπώς, η εκτέλεση αυτών των αλγορίθμων σε εκατομμύρια pixels δορυφορικών δεδομένων δημιουργεί σημαντικές προκλήσεις. Η αντιμετώπισή τους απαιτεί τον άρτιο συνδυασμό του θεωρητικού υποβάθρου με την τεχνική υλοποίηση, ώστε να διασφαλιστεί η επιτυχής λειτουργία του συστήματος.

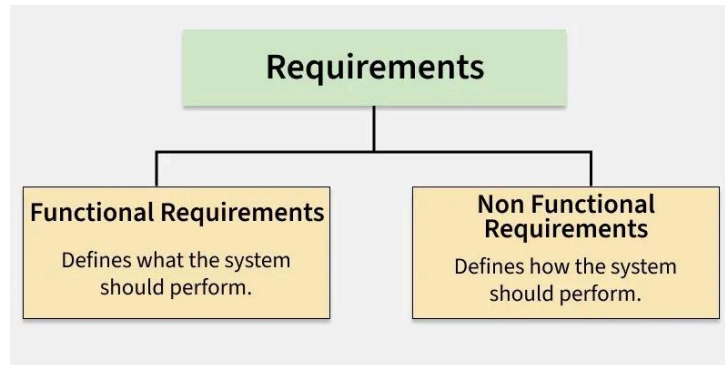
Κεφάλαιο 4ο: Ανάλυση Απαιτήσεων Συστήματος

4.1 Πλαίσιο Απαιτήσεων

Η κύρια προτεραιότητα του συστήματος είναι αξιοποίηση του από τους χρήστες, οι οποίοι θα διευκολυνθούν και να εξελίσσουν τις έρευνές τους. Οι χρήστες οι οποίοι θα βρίσκουν χρήσιμο το Hellas Cube θα είναι επιστήμονες-ερευνητές που αναλύουν δεδομένα Παρατήρησης Γής (Earth Observation). Οι ερευνητές θα πρέπει να είναι εξοικειωμένοι με την “φύση” μιας διεπαφής γραμμής εντολών μέσω τερματικού Linux, καθώς η εγκατάσταση του τοπικού εργαλείου Hellas Cube μέσω ενός τρίτου εργαλείου (cargo) απαιτεί τις απαραίτητες γνώσεις. Η φύση των διεπαφών τερματικού τις καθιστά κάτι λιγότερο από αυτόματες με το Hellas Cube να προσπαθεί να καταρρίψει ένα τέτοιο στερεότυπο. Το CLI πρέπει να είναι εύχρηστο και να αυτοματοποιεί τις αναλύσεις του χρήστη, με λίγα λόγια ο χρήστης να παρέχει όσο το δυνατόν λιγότερους παραμέτρους, με το πάτημα ενός κουμπιού και με μία αναμονή να δέχεται αποτελέσματα. Οι διαθέσιμες παράμετροι εισόδου πρέπει να είναι ευπροσάρμοστες, παρέχοντας πολλαπλές επιλογές που καλύπτουν πλήρως τις ερευνητικές ανάγκες του χρήστη. Το σύστημα στοχεύοντας στην αυτοματοποίηση της επεξεργασίας των δεδομένων παρέχει έναν αόρατο, στον χρήστη, σωλήνα διεργασιών. Η εγκατάσταση του τοπικού εργαλείου, η εκτέλεση εντολών και η εμφάνιση των αποτελεσμάτων είναι οι μόνες ενέργειες που έχει επίγνωση ο χρήστης. Οι διεργασίες, δηλαδή η επικοινωνία μεταξύ των API, η ανάλυση των δεδομένων με τους δείκτες, γενικότερα ο κυκλικός σωλήνας (χρήστης → server → χρήστης), καθώς και διεργασίες αφαίρεσης των δεδομένων από την βάση είναι αυτοματοποιημένες.

Πέρα από την εξοικείωση με το περιβάλλον τερματικού, οι χρήστες απαιτείται να διαθέτουν εξειδικευμένο επιστημονικό υπόβαθρο. Δεδομένου ότι η εφαρμογή απευθύνεται σε ερευνητές, κρίνεται απαραίτητο οι χρήστες να έχουν την ικανότητα ορθής ερμηνείας των δεδομένων που παράγει το σύστημα. Με κάθε αίτημα (εκτέλεση εντολής) ο server επιστρέφει αποτελέσματα ανάλυσης datasets και αριθμούς οι οποίοι για κάθε δείκτη διαφέρουν και ερμηνεύουν και διαφορετικά αποτελέσματα. Το Hellas Cube δεν εξάγει τελικά περιβαλλοντικά πορίσματα, αλλά παρέχει ποσοτικά δεδομένα, δείκτες και βασικές ταξινομήσεις, παρέχοντας έτσι στους ερευνητές τη βάση για να προβούν στις δικές τους ερμηνείες.

Το σύστημα, όπως και κάθε σύστημα λογισμικού έχει ορισμένες απαιτήσεις με τις λειτουργικές να παρέχουν δυνατότητες στους χρήστες και να είναι ένας μεγάλος παράγοντας επιλογής λογισμικού για να μπορέσει να καλύψει τις ανάγκες του ο χρήστης, δηλαδή αν το σύστημα του λογισμικού δεν παρέχει βασικούς δείκτες ή εάν δεν παρέχει υποσύστημα χρηστών πιθανώς να μην το επιλέξει για τις αναλύσεις του. Αντιθέτως οι μη λειτουργικές απαιτήσεις αποτελούν την επιθυμία του χρήστη οι αναλύσεις να γίνονται όσο το δυνατόν ταχύτερα από την μεριά του χρήστη, απλά και ενοποιημένα και από την μεριά του backend το σύστημα να λειτουργεί αποτελεσματικά με την ελάχιστη χρήση υπολογιστικών πόρων.



Εικόνα 4.1: Ανάλυση κατηγοριών απαιτήσεων ενός συστήματος.

4.2 Λειτουργικές Απαιτήσεις (Functional Requirements)

Οι λειτουργικές απαιτήσεις του συστήματος αποτελούν έναν σημαντικό κλάδο του συστήματος. Είναι η βάση και μία παράμετρος για το επόμενο βήμα ανάπτυξης αρχιτεκτονικής του συστήματος. Με μία πρώτη όψη το σύστημα αποτελείται από διαφορετικά σημεία με την διεπαφή, το σύστημα χρηστών, την ανάκτηση δεδομένων, επεξεργασία δεδομένων καθώς και μηχανισμούς με σκοπό την ομαλή λειτουργία του συστήματος.

4.2.1. Διεπαφή Χρήστη

Η διεπαφή όπως έχει προαναφερθεί υλοποιήθηκε σε μορφή γραμμής εντολών, με αποτέλεσμα χωρίς τις απαραίτητες γνώσεις και λειτουργίες του εργαλείου του Hellas Cube θα είναι αδύνατον να το χρησιμοποιήσει. Γι' αυτό ενσωματώθηκε (όπως στα περισσότερα cli tools) μια εντολή η οποία εμφανίζει όλες τις εντολές που είναι διαθέσιμες και τις παραμέτρους τις οποίες χρειάζεται για την εκτέλεση κάθε εντολής. Κάθε περιγραφή εντολής παρέχει και ένα μικρό κείμενο με την λειτουργία και τις οδηγίες της, ένα βασικό στοιχείο για την κατανόηση του συστήματος από τον χρήστη.

```

IT Global CLI

Demo application for IT Global CLI

USAGE
cli-features-example [OPTIONS] COMMAND

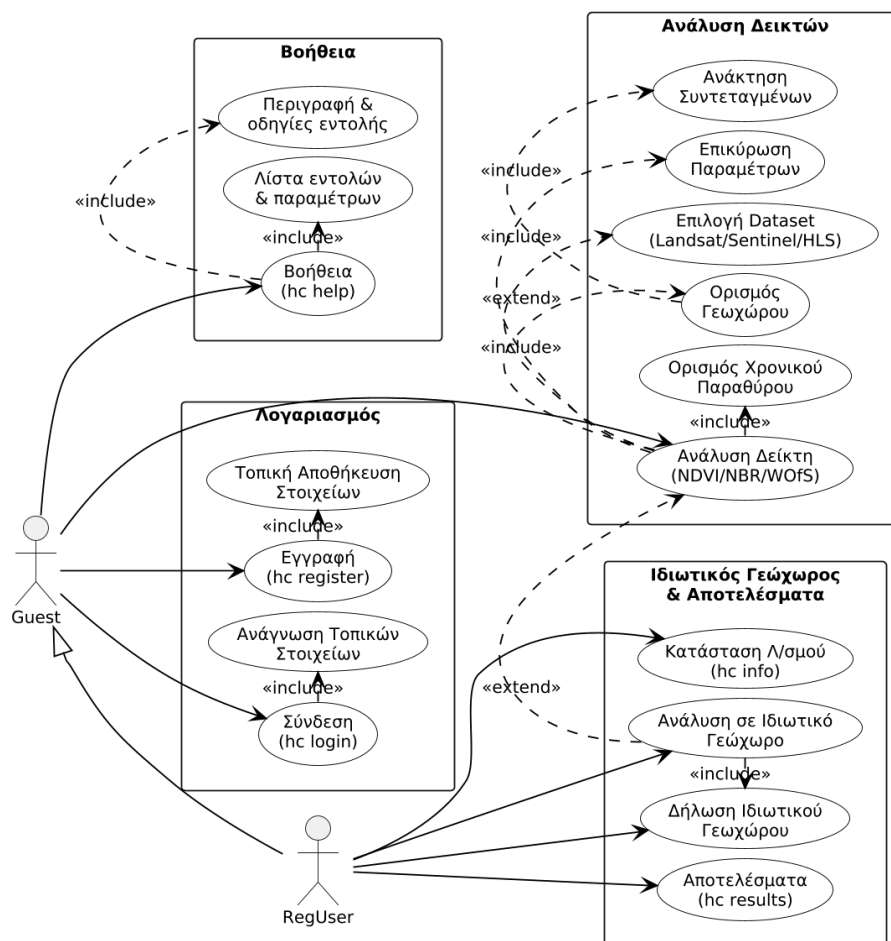
OPTIONS
--no-color  Disable colors
--help      Print help and exit

COMMANDS
colors      Run a 'colors' demo
live        Run a 'live' demo
nested-colors  Run a 'nested-colors' demo
nested-help-output  Run a 'nested-help-output' demo
sample      Various samples
table       Run a 'table' demo
  
```

Εικόνα 4.2: Παράδειγμα help command ενός εργαλείου CLI.

Οι επιλογές εντολών προδίδουν τι δυνατότητες που έχει ο χρήστης με το εργαλείο, με μερικές από αυτές τις επιλογές να είναι δείκτες που θα ήθελε να εκτελέσει. Για κάθε δείκτη ο χρήστης θα πρέπει να παρέχει παραμέτρους όπως το όνομα του γεωχώρου, τα χρονικά παράθυρα με την εφαρμογή να ελέγχει τα δεδομένα που παρέχει και να τον καθοδηγεί ώστε να μπορέσει να παρέχει τις κατάλληλα

συντακτικές παραμέτρους. Το σύστημα είναι εξοπλισμένο με τα κατάλληλα δεδομένα για την ανάκτηση των συντεταγμένων από τα ονόματα γεωχώρων που παρέχει ο χρήστης. Σε μερικές αναλύσεις ο χρήστης θα έχει την δυνατότητα να επιλέξει και το δικό του datasets (Landsat, Sentinel, HLS) για μεγαλύτερη ποικιλία αποτελεσμάτων. Μέσω της διεπαφής ο χρήστης θα έχει και την δυνατότητα να αλληλεπιδρά με το σύστημα των χρηστών που είναι ενσωματωμένο στο Hellas Cube. Η διεπαφή αφιερώνει εξειδικευμένες εντολές για αυτήν την αλληλεπίδραση. Ο σκοπός της διεπαφής είναι να γεφυρώσει τον ερευνητή με τον κεντρικό διακομιστή, προσφέροντας ομαλή και ασφαλή εμπειρία. Όπως αναλύσαμε, ο ρόλος της διεπαφής δεν είναι ένας απλός τρόπος μεταφοράς δεδομένων από και προς τον server αλλά μία μινιμαλιστική καθοδήγηση. Τέλος, το περιβάλλον γραμμής εντολών είναι υπεύθυνο για τον έλεγχο πρόσβασης σε τοπικό επίπεδο, επιστρέφοντας σαφή και κατανοητά μηνύματα σφάλματος σε περίπτωση που επιχειρηθεί κάποια ενέργεια (όπως η αποθήκευση ενός γεωχώρου) χωρίς να έχει γίνει σύνδεση του χρήστη πρώτα.



Εικόνα 4.3: Διάγραμμα περιπτώσεων χρήσης με την χρήση UML

4.2.2. Υποσύστημα χρηστών

Το σύστημα του Hellas Cube εξοπλίστηκε και από ένα υποσύστημα χρηστών του οποίου ο ρόλος είναι η διαχείριση των χρηστών και η διαχείριση των αιτημάτων σχετικά με τους λογαριασμούς των χρηστών. Το σύστημα παρέχει την δυνατότητα στον χρήστη να εγγραφεί στο σύστημα καθώς και να του επιτρέψει να συνδεθεί στο λογαριασμό ώστε να δέχεται access tokens και να αποθηκεύονται τοπικά. Ο χρήστης εκτός από τα βασικά του στοιχεία έχει την δυνατότητα να διαχειρίζεται τον προσωπικό του γεωχώρο ώστε να τον καλεί σε μελλοντικές αναλύσεις. Το υποσύστημα του χρήστη είναι πολύ σημαντικό για την υλοποίηση του Hellas Cube καθώς δίνει στον χρήστη την δυνατότητα να

εφαρμόζει δεδομένα όχι με τους περιορισμένους χώρους της βάσης του Hellas Cube αλλά και προσωπικές περιοχής ενδιαφέροντος, εναρμονίζοντας τις χωρικές αναλύσεις στις ανάγκες του.

4.2.3. Ανάκτηση και Επεξεργασία Δεδομένων

Τα δεδομένα και πιο συγκεκριμένα τα δορυφορικά δεδομένα είναι ο πυρήνας του συστήματος. Χωρίς την ανάκτηση τους και την τροφοδοσία τους το σύστημα δεν αποδίδει αποτελέσματα. Έτσι το σύστημα πρέπει να έχει σωστή επικοινωνία εξωτερικών API, ώστε να αναζητά τα dataset που έχει ανάγκη μία ανάλυση ενός χρήστη καθώς και να τα αποθηκεύει ώστε να επαναχρησιμοποιηθούν στο μέλλον. Το σύστημα έχει σχεδιαστεί ώστε να αντλεί και να επεξεργάζεται πληροφορία αποκλειστικά από τρεις προκαθορισμένες δορυφορικές πηγές: Landsat, Sentinel και τον εναρμονισμένο συνδυασμό τους (Harmonized Landsat Sentinel - HLS). Μέσω της διεπαφής γραμμής εντολών (CLI), παρέχεται στον χρήστη η δυνατότητα επιλογής του επιθυμητού συνόλου δεδομένων για τις εκάστοτε αναλύσεις. Ωστόσο, η τρέχουσα αρχιτεκτονική δεν υποστηρίζει την τροφοδότηση του συστήματος με εξωτερικές, προσωπικές τοπικές βάσεις δορυφορικών δεδομένων από την πλευρά του χρήστη. Η συγκεκριμένη λειτουργική απαίτηση διασφαλίζει την απόλυτη εναρμόνιση, συμβατότητα και ορθή ευρετηρίαση των δεδομένων εντός του αυστηρά ελεγχόμενου περιβάλλοντος του Open Data Cube. Σε επόμενο επίπεδο η επεξεργασία γίνεται με τέτοιο τρόπο ώστε να διαχειρίζεται τα data από τα API και να εφαρμόζονται οι περιβαλλοντικοί αλγόριθμοι και οι μαθηματικοί τύποι των δεικτών. Τα αποτελέσματα της επεξεργασίας ποικίλλουν και παρέχουν όλες τις κατάλληλες μετρήσεις μετά από την ανάλυση των datasets. Συμπερασματικά ο πυρήνας του Hellas Cube έχει τις σημαντικές προδιαγραφές, καθώς καλείται να εναρμονίσει την πολυπλοκότητα των μαθηματικών υπολογισμών. Αυτές ακριβώς οι απαιτήσεις έθεσαν τα θεμέλια για τον αρχιτεκτονικό σχεδιασμό και τις τεχνολογικές επιλογές του συστήματος που θα αναλυθούν στην συνέχεια.

4.2.4. Μηχανισμοί Αυτοματοποίησης

Το σύστημα απαιτεί την ενσωμάτωση αυτοματοποιημένων μηχανισμών για την βελτιστοποίηση της ροής και της διαχείρισης των δεδομένων. Η διαχείριση των δεδομένων των χρηστών πραγματοποιείται μέσω διεργασιών παρασκήνιου (background processes), οι οποίες διασφαλίζουν τη συνεχή εποπτεία της εγκυρότητας των πληροφοριών, όπως την διαχείριση κύκλου ζωής των access tokens. Επιπροσθέτως, η υλοποίηση ενός μηχανισμού αποθήκευσης αποτελεσμάτων κρίνεται επιβεβλημένη, καθώς επιτρέπει την επαναχρησιμοποίηση ήδη αναλυθέντων δεδομένων, περιορίζοντας την κατανάλωση υπολογιστικών πόρων. Τέλος, το σύστημα ενσωματώνει αυστηρά πρωτόκολλα ασφάλειας για την προστασία από ευπάθειες του δικτύου και την αποτροπή μη εξουσιοδοτημένης πρόσβασης στην βάση δεδομένων.

4.3 Μη-λειτουργικές Απαιτήσεις (Non-Functional Requirements)

Το λογισμικό Hellas Cube όπως και κάθε άλλο λογισμικό έχει περιορισμούς ειδικά ένα λογισμικό ανάλυσης τεράστιων σε χωρητικότητα και πολυπλοκότητα δεδομένων (Big Data). Η ομαλή και αυτοματοποιημένη εκτέλεση των διεργασιών που παίρνουν μέρος στο “πίσω μέρος” (backend) του συστήματος δεν εξαρτάται μόνο από τις ικανότητες και τις δεξιότητες χρησιμοποιήσαμε και χτίσαμε, αλλά και από το πόσο αποδοτικά μπορεί να το κάνει. Οι τεχνικοί περιορισμοί, τα κριτήρια απόδοσης και οι ποιοτικές προδιαγραφές διασφαλίζουν την βέλτιστη λειτουργία του συστήματος κάτω από απαιτητικές συνθήκες. Οι περιορισμοί που θα αναλύσουμε σε αυτό το κεφάλαιο αποτελούν τις μη-λειτουργικές απαιτήσεις του συστήματος.

4.3.1. Διαχείριση πόρων και Απόδοση

Όπως έχουμε αναφέρει η ανάλυση δορυφορικών δεδομένων απαιτεί σοβαρή υπολογιστική δύναμη λόγω του όγκου των δεδομένων. Όμως μέσω της μηχανικής λογισμικού διαχειριζόμαστε τους πόρους που χρησιμοποιεί το σύστημα μας για καλύτερη συνεργασία μεταξύ πόρων-ανάλυσης. Η απαίτηση ορίζει ότι το σύστημα επεξεργασίας απαγορεύεται να φορτώνει ολόκληρα τα αρχεία στην κύρια μνήμη, καθώς αυτό οδηγεί σε σφάλματα εξάντλησης μνήμης, προκαλώντας πιθανώς κατάρρευση του διακομιστή. Αντίθετα, επιβάλλεται να εφαρμοστούν τεχνικές τμηματοποίησης (chunking) ή χρήση νημάτων (threading). Το σύστημα πρέπει να διαβάσει, επεξεργάζεται μαθηματικά και να γράφει τα δεδομένα στο δίσκο σε μικρά, διαχειρίσιμα τμήματα, διατηρώντας το αποτύπωμα μνήμης σταθερό ανεξαρτήτως του μεγέθους της δορυφορικής εικόνας.

Η εμπειρία χρήσης στη γραμμή εντολών εξαρτάται άμεσα από τον τύπο των εντολών που θα εκτελεστούν. Εντολές που εκτελούνται εντελώς τοπικά, όπως η κλήση εντολής βοήθειας (--help) ή η εμφάνιση των τοπικά αποθηκευμένων στοιχείων, ολοκληρώνονται και εμφανίζουν αποτελέσματα σε χρόνο μικρότερο του ενός δευτερολέπτου (< 1s). Για τις χρονοβόρες διαδικασίες που απαιτούν επικοινωνία με το backend API (π.χ. υπολογισμός ενός δείκτη NDVI), πέραν της ανάλυσης των δεδομένων που απαιτεί τον μεγαλύτερο χρόνο εκτέλεσης, οι κλήσεις του API ολοκληρώνονται σε λίγα δευτερόλεπτα, λαμβάνοντας υπόψη ότι η ταχύτητα δικτύου κάθε χρήστη είναι διαφορετική. Συμπερασματικά, η απόδοση και η διαχείριση πόρων είναι κλάδοι που αλληλοεπηρεάζονται, για την καλύτερη χρηστικότητα του λογισμικού.

4.3.2. Αξιοπιστία και Σφάλματα

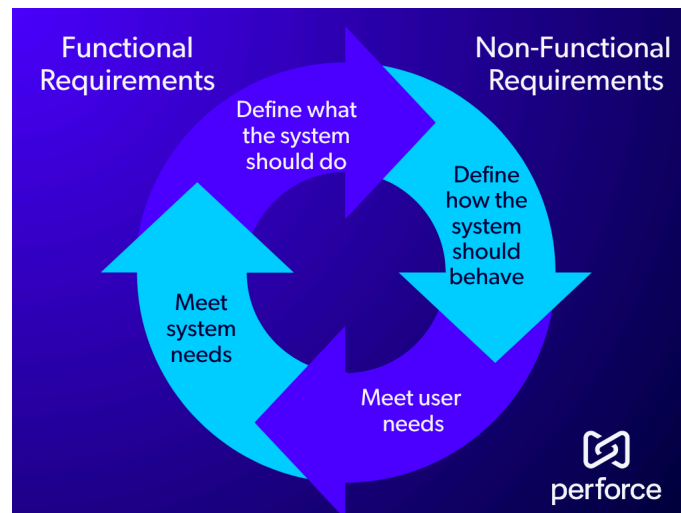
Ένα παραγωγικό σύστημα δεδομένων πρέπει να είναι ανθεκτικό σε εξωγενείς παράγοντες εξασφαλίζοντας ότι ακόμα και σε σφάλματα, η διαχείριση του γίνεται ελεγχόμενη. Δεδομένου ότι το Hellas Cube λειτουργεί βάσει μια αρχιτεκτονικής client-server, εξαρτάται άμεσα από την σταθερότητα του δικτύου. Σε περίπτωση αστάθειας ή μη-διαθέσιμων εξωτερικών πηγών δορυφορικών δεδομένων, το σύστημα ενημερώνει για τα σφάλματα και δίνει οδηγίες για επαναπροσπάθεια από τον χρήστη. Η αξιοπιστία από το σύστημα είναι ένα μεγάλο ζήτημα, με το σύστημα να πρέπει να δέχεται δεδομένα μόνο από επίσημους φορείς (Επίσημα Cloud, API).

Παρόλο που το CLI σχεδιάστηκε με γνώμονα την αλληλεπίδραση με ανθρώπους-ερευνητές, είναι σημαντικό να παραμένει συμβατό με πρότυπα. Με την ολοκλήρωση κάθε εντολής, το πρόγραμμα πρέπει να επιστρέφει status (κωδικούς ή text). Για παράδειγμα, το status OK υποδηλώνει επιτυχή εκτέλεση, το status ERROR γενικό σφάλμα του server. Επιπλέον, το σύστημα διαχωρίζει αυστηρά την έξοδο: τα χρήσιμα δεδομένα κατευθύνονται στο stdout (Standard Output), ενώ τα μηνύματα συστήματος και τα σφάλματα στο stderr (Standard Error) ώστε να διαχωρίζονται τα σφάλματα με τα αποτελέσματα. Συνοψίζοντας, η αξιοπιστία των δεδομένων και των σφαλμάτων απαιτούνται ώστε το σύστημα να καλύπτει σενάρια των αλληλεπιδράσεων και να είναι ανθεκτικό σε οποιαδήποτε περίπτωση.

4.3.3. Περιβάλλον εκτέλεσης

Η κοινότητα της επιστήμης δεδομένων (Data Science) και τα συστήματα γεωχωρικών δεδομένων βασίζονται κατά κύριο λόγο σε λειτουργικά συστήματα ανοιχτού κώδικα. Έτσι, αποτελεί απαίτηση το HellaCube να εκτελείται απρόσκοπτα και χωρίς περίπλοκες εξαρτήσεις σε διανομές Linux. Είτε σε enterprise περιβαλλόντα είτε σε συστήματα γενικής χρήσης, το εργαλείο πρέπει να εγκαθίσταται εύκολα και να λειτουργεί άμεσα. Η απουσία δέσμευσης από ιδιόκτητα λειτουργικά συστήματα

μειώνει το κόστος υποδομής και ευθυγραμμίζει το λογισμικό με τη φιλοσοφία των εργαλείων ανοιχτού κώδικα στην παρατήρηση της Γης.



Εικόνα 4.4: Ανάλυση κύκλου απαιτήσεων συστήματος.

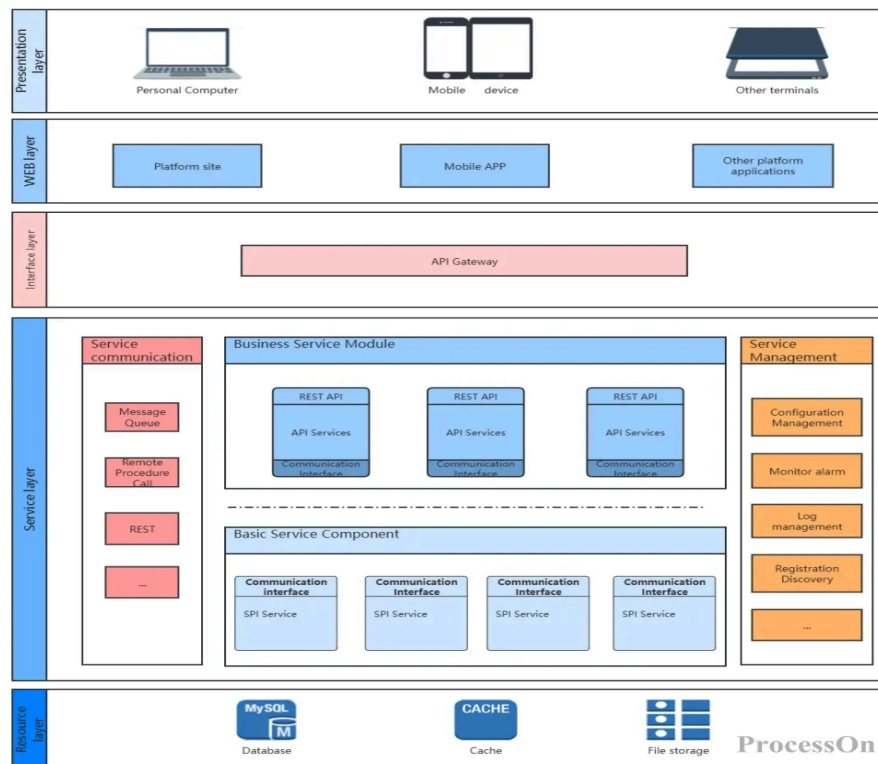
Συνολικά, η ανάγκη να εναρμονιστούν αυτές οι φαινομενικά αντίρροπες δυνάμεις η υψηλή πολυπλοκότητα των γεωχωρικών υπολογισμών από τη μία πλευρά, και η βέλτιστη διαχείριση των υπολογιστικών πόρων (μνήμης και δικτύου) από την άλλη αποτελεί τον καθοριστικό παράγοντα για τις στρατηγικές τεχνολογικές επιλογές που ακολουθούν. Ο τρόπος με τον οποίο η αρχιτεκτονική του συστήματος ανταποκρίνεται σε αυτές τις αυστηρές προδιαγραφές εξετάζεται αναλυτικά στην παρουσίαση της επιλεγμένης τεχνολογικής στοίβας.

Κεφάλαιο 5ο: Τεχνολογίες και Αρχιτεκτονική του Συστήματος

Η διαδικασία ανάπτυξης ενός λογισμικού παρουσιάζει πολλά βήματα με ένα από αυτά να είναι η σχεδίαση και η απόφαση της κατάλληλης αρχιτεκτονικής για την υλοποίηση του συστήματος είναι ένα από τα σημαντικότερα πριν την τεχνική υλοποίηση του έργου. Με την σχεδίαση αρχιτεκτονικής λογισμικού δίνεται η δυνατότητα να αναπαριστάται το σύστημα με τα διαθέσιμα εργαλεία και τεχνικές ανάπτυξης λογισμικού. Η σχεδίαση αρχιτεκτονικής μπορεί να χωριστεί σε δύο διακριτές υποκατηγορίες [27]:

- **Αρχιτεκτονική λογισμικού:** Η αρχιτεκτονική λογισμικού όπως και η αρχιτεκτονική των κτιρίων αποτελεί το στρατηγικό πλάνο που ακολουθείται κατά την υλοποίηση του. Η βασική του λειτουργία είναι ο συνδυασμός των οντοτήτων τεχνολογίας που χρησιμοποιήθηκαν.
- **Σχεδίαση Λογισμικού:** Η σχεδίαση του λογισμικού αποτελείται κυρίως από τις μεθόδους σχεδιασμού των απαιτήσεων που έχει το σύστημα μας καθώς και των διαθέσιμων τεχνολογικών υποβάθρων.

Οι δύο αυτές υποκατηγορίες επηρεάζουν σε μεγάλο βαθμό την ποιότητα, την απόδοση καθώς και την μελλοντική συντήρηση. Στην συνέχεια αναλύεται η αρχιτεκτονική τριών επιπέδων (Ενότητα 5.2), ακολουθούμενη από την ανάλυση των τεχνολογιών που επιλέχθηκαν.



Εικόνα 5.1: Παράδειγμα αρχιτεκτονικής ενός συστήματος Web

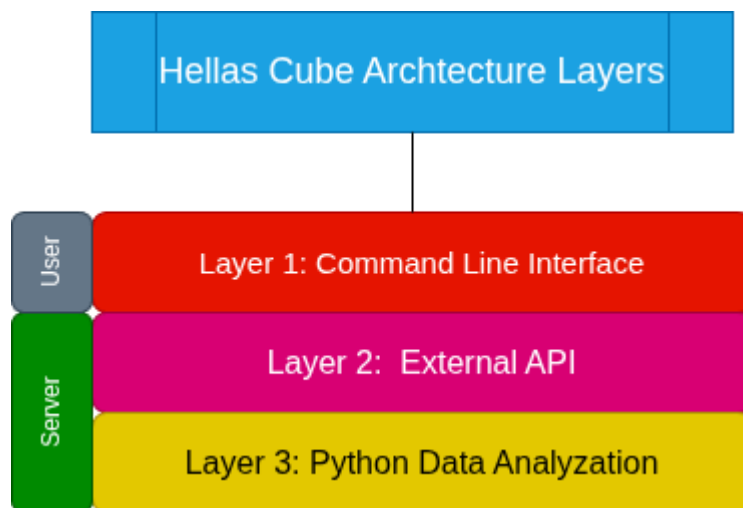
5.1 Υψηλού επιπέδου Αρχιτεκτονική

Η αρχιτεκτονική του Hellas Cube σχεδιάστηκε με γνώμονα τον σαφή διαχωρισμό των αρμοδιοτήτων, κατανέμοντας τον υπολογιστικό φόρτο και τη λογική του συστήματος σε διακριτά επίπεδα εκτέλεσης. Αυτή η προσέγγιση εξασφαλίζει τη βέλτιστη διαχείριση και επεξεργασία των δορυφορικών

δεδομένων, διατηρώντας παράλληλα την βέλτιστη υψηλή ταχύτητα απόκρισης, ασφάλεια και επεκτασιμότητα.

Η ροή της πληροφορίας στο σύστημα ακολουθεί ένα μοντέλο τριών επιπέδων (3-tier architecture), υποστηριζόμενο από ένα ισχυρό σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων. Όπως απεικονίζεται στην Εικόνα 5.2, το σύστημα συνεργάζεται αρμονικά για την διεκπεραίωση κάθε αιτήματος. Η αποσύνδεση αυτών των επιπέδων επιτρέπει την ανεξάρτητη συντήρηση και αναβάθμιση του κάθε τμήματος, χωρίς να επηρεάζεται.

Μια αρχή της μηχανικής λογισμικού που εφαρμόζεται στο επίπεδο του CLI είναι η προσέγγιση "Fail-Fast" [40]. Πριν το αίτημα προωθηθεί στον πυρήνα του συστήματος (Rust API) ή καταναλώσει πόρους από τη βάση δεδομένων, το CLI εκτελεί αυστηρούς ελέγχους ορθότητας (data validation). Ελέγχει, για παράδειγμα, εάν η μορφή των ημερομηνιών είναι έγκυρη (π.χ. ISO 8601), εάν η ημερομηνία έναρξης προηγείται λογικά της ημερομηνίας λήξης, και εάν οι συντεταγμένες εμπίπτουν στα αναμενόμενα γεωγραφικά όρια. Με αυτόν τον τρόπο, αποτρέπεται η διάδοση εσφαλμένων καταστάσεων (invalid states) στα υποσυστήματα και επιστρέφεται άμεσα ένα κατανοητό μήνυμα σφάλματος στον χρήστη.



Εικόνα 5.2: Επίπεδα αρχιτεκτονικής Hellas Cube

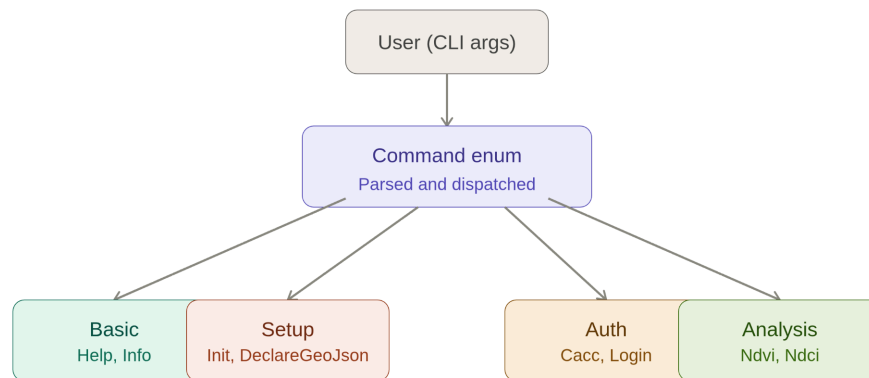
5.1.1. Επίπεδο Διεπαφής Χρήστη

Το πρώτο επίπεδο αρχιτεκτονικής του Hellas Cube αποτελεί το μοναδικό σημείο αλληλεπίδρασης μεταξύ του τελικού χρήστη και του συστήματος. Κατα την διάρκεια σχεδιασμού εξετάστηκε η χρήση μιας Γραφικής Διεπαφής Χρήστη (Graphical User Interface), όμως απορρίφθηκε με σκοπό την απλοποίηση του συστήματος και την εστίαση του στην ανάλυση και εξαγωγή δεδομένων. Η επιλογή αυτή μας δίνει την δυνατότητα να χρησιμοποιήσουμε τους όσον δυνατόν ελάχιστους πόρους του υπολογιστή του χρήστη.

Πρωταρχικός ρόλος του CLI είναι η λήψη και η συντακτική ανάλυση που παρέχει ο χρήστης. Μέσω συγκεκριμένων εντολών και σημαιών εντολών (flags) ο χρήστης καλείται να ορίσει με ακρίβεια τα όρια της αναζήτησης του, περιορίζοντας τον όγκο των δορυφορικών δεδομένων που θα επεξεργαστεί το σύστημα. Οι εντολές περιλαμβάνουν:

➔ Γενικές/Βασικές εντολές: Εντολές βοήθειας και παροχής πληροφοριών

- Εντολές προετοιμασίας: Εντολές που αρχικοποιούν ορίσματα στο σύστημα του χρήστη για την ομαλή λειτουργία της εφαρμογής.
- Εντολές πιστοποίησης: Εντολές που ασχολούνται με τα στοιχεία του χρήστη για δημιουργία λογαριασμού και login. Με σημαίες όπως:
 - Cacc (Create Account): email και κωδικός πρόσβασης
 - Login (Log in to Account): email και κωδικός πρόσβασης
- Εντολές ανάλυσης: Εντολές οι οποίες εκτελούν τα αιτήματα για ανάλυση στον server. Θα έχουν σημαίες όπως:
 - Default: Για χρήστες με προσωπικούς γεωχώρους.
 - Area: Εισαγωγή ονόματος περιοχής ανάλυσης.
 - From: Αρχική ημερομηνία ανάλυσης.
 - Till: Τελική ημερομηνία ανάλυσης.



Εικόνα 5.3: Διάγραμμα αρχιτεκτονικής CLI

Το σημαντικότερο ίσως πλεονέκτημα της CLI προσέγγισης είναι η εγγενής δυνατότητα αυτοματοποίησης. Το Hellas Cube δεν λειτουργεί ως ένα απομονωμένο εργαλείο, αλλά μπορεί να ενσωματωθεί απρόσκοπτα σε ευρύτερες ροές εργασίας δεδομένων (data pipelines). Οι εντολές του συστήματος μπορούν να εκτελεστούν από το τερματικό, με αποτέλεσμα να γίνεται δυνατή η μαζική επεξεργασία (batch processing) μέσω σεναρίων κελύφους (shell scripts), καθώς και μία αυτοματοποιημένη κλήση του εργαλείου.

5.1.2. Επίπεδο Εξωτερικής Προγραμματιστικής Διεπαφής Εφαρμογών (Rust API)

Το μεσαίο επίπεδο της αρχιτεκτονικής φιλοξενεί την Εξωτερική Προγραμματιστική Διεπαφή Εφαρμογών (API), η οποία είναι εξ ολοκλήρου ανεπτυγμένη σε γλώσσα Rust. Ο βασικός ρόλος αυτής της στρώσης της αρχιτεκτονικής είναι ο κεντρικός διαχειριστής/ενορχηστρωτής και ο ρόλος του ενδιαμέσου λογισμικού (middleware), έχοντας επαφή με τον χρήστη μέσω των αιτημάτων που λαμβάνει και απαντάει, παράλληλα με την επαφή του με το σύστημα ανάλυσης δεδομένων.

Κατά την εκκίνηση του συστήματος και την υποδοχή ενός νέου αιτήματος, το API ακολουθεί μία αυστηρή αλληλουχία αρχικοποίησης για την εγκαθίδρυση του περιβάλλοντος εκτέλεσης. Εφαρμόζει ασφαλείς πρακτικές διαχείρισης ρυθμίσεων όπως σύνδεση στην βάση μέσω μεταβλητών περιβάλλοντος (environment variables).

Μία από τις πιο κρίσιμες λειτουργίες του διαχειριστή είναι η βελτιστοποίηση της αναζήτησης των δεδομένων. Το API δεν αλληλεπιδρά απευθείας με τα δορυφορικά δεδομένα, αλλά αλληλεπιδρά με τα δεδομένα της εφαρμογής που είναι αποθηκευμένα στην PostgreSQL. Σε κάθε αίτημα το σύστημα επικοινωνεί σε πολλά σημεία της ζωής ενός αιτήματος. Σε περίπτωση που ο χρήστης θα έχει επιλέξει την ανάλυση προσωπικού γεωχώρου το API θα συλλέγει τα δεδομένα του θα τα παραχωρεί στο σύστημα της ανάλυσης των δεδομένων. Για κάθε αίτημα το API θα καταγράφει κάθε βήμα του κύκλου ζωής τους αιτήματος στην βάση ενώ παράλληλα καλεί την ανάλυση δεδομένων.

Συμπερασματικά το API είναι ένα σημαντικός πυλώνας αρχιτεκτονικής του pipeline του Hellas Cube καθώς συνδέει και διαχειρίζεται χρήση, δεδομένα και ανάλυση δεδομένων.

5.1.3. Επίπεδο Ανάλυσης δεδομένων

Το τρίτο και κατώτερο επίπεδο της αρχιτεκτονικής τους Hellas Cube είναι αφιερωμένο στην ανάκτηση, τον μετασχηματισμό και την ανάλυση των ογκωδών περιβαλλοντικών δορυφορικών δεδομένων. Το επίπεδο αυτό υλοποιείται σε γλώσσα Python, ενσωματώνοντας το ισχυρό πλαίσιο του Open Data Cube (ODC), και λειτουργεί αυστηρά ως ένα εσωτερικό API (Internal API) που δέχεται αιτήματα αυστηρά από το Rust API. Κατά την διάρκεια σχεδιασμού του συστήματος προέκυψαν προκλήσεις όσον αφορά την μέγιστη αποδοτικότητα. Αρχικά το σύστημα καλούσε τα script της python με αποτέλεσμα η ανάλυση να παίρνει σημαντικά μεγάλη χρονική διάρκεια, όμως η ριζική αλλαγή στην αρχιτεκτονική άλλαξε την απόδοση. Η χρήση ενός εσωτερικού API μας δίνει την δυνατότητα να επικοινωνούμε με έναν “ζωντανό” οργανισμό ανάλυσης δεδομένων έτοιμο να δεχτεί αιτήματα χωρίς να τρώει χρόνο σε αρχικοποιήσεις και συνδέσεις σε βάσεις.

Όταν το επίπεδο ανάλυσης κληθεί από το Rust API δέχεται τα δεδομένα και ξεκινάει την ανάλυση. Το σύστημα με τα δεδομένα που έχει λάβει από τον οργανωτή πρέπει να ετοιμάσει τα δεδομένα προς ανάλυση. Το σύστημα αρχικά ψάχνει για τυχόν υπάρχοντα δεδομένα στην σχεσιακή βάση δεδομένων του ODC και σε περίπτωση που του λείπουν πηγαίνει και τα ανακτά. Όταν πλέον έχει τα δεδομένα στην βάση τα προετοιμάζει για επεξεργασία. Όταν πλέον τα δεδομένα είναι επεξεργάσιμα εφαρμόζεται ο κατάλληλος δείκτης ή αλγόριθμος και επιστρέφουν τα αποτελέσματα πίσω στο API. Η Python στην συνέχεια παραμένει “ζωντανή” έτσι ώστε να έτοιμη να διεκπεραιώσει και τις επόμενες αναλύσεις.

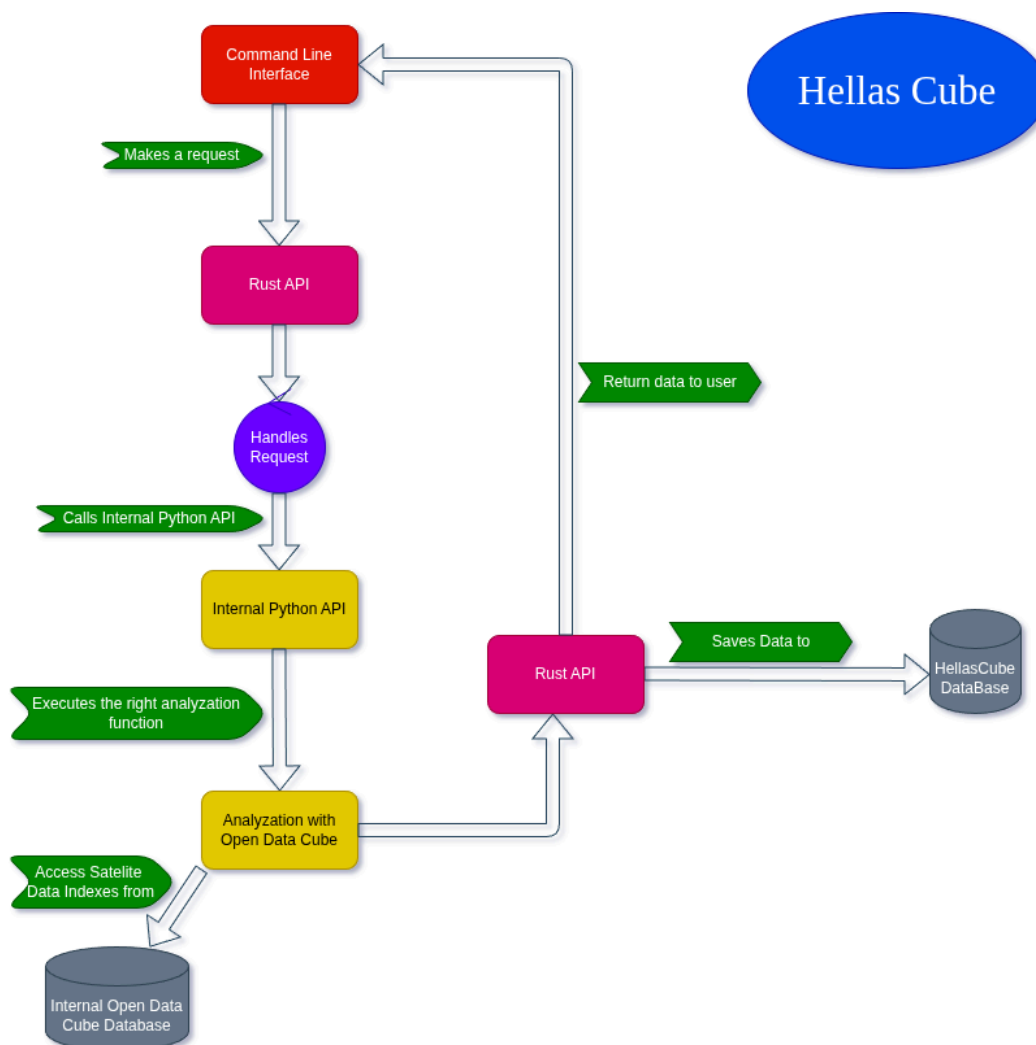
Συμπερασματικά, η σχεδίαση του επιπέδου ανάλυσης δεν περιορίζεται αποκλειστικά στην εξαγωγή πληροφορίας. Αποτελώντας το πιο απομονωμένο τμήμα του backend, με την αρχιτεκτονική του διασφαλίζει τη βέλτιστη διαχείριση των υπολογιστικών πόρων, εγγυώμενη έτσι τη μέγιστη συνολική απόδοση (performance) του συστήματος.

5.2 Παράδειγμα Ροής Εκτέλεσης (Use Case)

Τα επίπεδα που αναφέρθηκαν στο προηγούμενο κεφάλαιο παρουσιάζονται αναλυτικά στην παράθεση ενός ολοκληρωμένου σεναρίου χρήσης. Όπως απεικονίζεται στην Εικόνα 5.4, ο κύκλος ζωής ενός αιτήματος στο Hellas Cube Ακολουθεί καθορισμένη αλληλουχία βημάτων. Η ροή εκτέλεσης αναλύεται στα παρακάτω στάδια:

- **Εκκίνηση Αιτήματος (Makes request):** Η διαδικασία ξεκινάει ο χρήστης εισάγει μία εντολή στο CLI. Το επίπεδο αυτό δρομολογεί το αίτημα στο επόμενο στάδιο.

- **Υποδοχή και Διαχείριση (Handles Request):** Το κεντρικό Rust API παραλαμβάνει το αίτημα, αναλύει τις απαιτήσεις της εντολής και προετοιμάζει το απαραίτητο φορτίο (payload) για την επεξεργασία.
- **Κλήση Εσωτερικού API (Calls Internal Python API):** Το API εκκινεί την επικοινωνία με το επίπεδο επεξεργασίας, δίνοντας εντολή στο εσωτερικό API της Python να εκτελέσει την κατάλληλη συνάρτηση ανάλυσης.
- **Ανάκτηση και Ανάλυση Δεδομένων(Analysis with Open Data Cube):** Το περιβάλλον του ODC μας δίνει πρόσβαση στα δορυφορικά δεδομένα με την χρήση της εσωτερικής βάσης(Access Satellite Data Indexes from), και αναλύουμε τα δεδομένα.
- **Επιστροφή και αποθήκευση Αποτελεσμάτων (Saves Data to):** Με την ολοκλήρωση την ανάλυσης από την Python, τα παραγόμενα αποτελέσματα επιστρέφουν στο Rust API το οποίο τα αποθηκεύει στη Σχεσιακή Βάση Δεδομένων του συστήματος (Hellas Cube DataBase).
- **Ολοκλήρωση Κύκλου Ζωής (Return data to user):** Τέλος, το Rust API ενημερώνει το επίπεδο διεπαφής για την επιτυχή ολοκλήρωση των διεργασιών, επιστρέφοντας το τελικό αποτέλεσμα στον χρήστη μέσω του Command Line Interface, κλείνοντας έτσι τον κύκλο ζωής του αιτήματος.

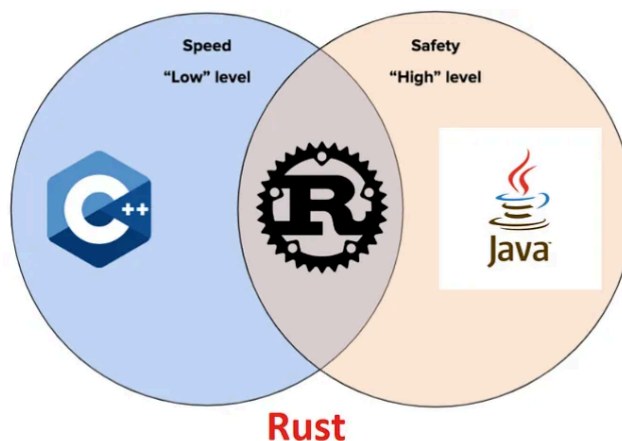


Εικόνα 5.4: Κύκλος ζωής εκτέλεσης μιας εντολής

5.3 Η γλώσσα προγραμματισμού Rust

Η επιλογή της γλώσσας προγραμματισμού Rust για τη διεπαφή και τον κεντρικό κόμβο δεδομένων (εξωτερικό API) του πολυεπίπεδου συστήματος αποτελεί μία από τις κρισιμότερες επιλογές αρχιτεκτονικής. Η επιλογή δεν έγινε με βάση τις σύγχρονες τάσεις της τεχνολογίας λογισμικών, αλλά κυρίως στα μοναδικά δομικά χαρακτηριστικά της γλώσσας, τα οποία ευθυγραμμίζονται με τις απαιτήσεις του συστήματος.

Η Rust είναι μία γλώσσα προγραμματισμού συστημάτων (system-level), η οποία αναπτύχθηκε με κύριο σκοπό την αλληλεπίδραση της με το hardware και την αξιοπιστία σε λογισμικά. Όπως και η C++ μεταφράζεται κατευθείαν σε γλώσσα μηχανής, δημιουργώντας ένα εκτελέσιμο σε αντίθεση με άλλες γλώσσες προγραμματισμού όπως την Java που χρησιμοποιεί εικονική μηχανή (Virtual Machine) για την εκτέλεση της. Αυτό προσφέρει στον προγραμματιστή τον απόλυτο έλεγχο και την ευελιξία όσον αφορά χρόνους εκτέλεσης και μνήμη, χωρίς όμως να θυσιάζει την ασφάλεια μνήμης που προσφέρουν γλώσσες σαν την Java [28]. Ο compiler της Rust χρησιμοποιεί στατικό σύστημα τύπων (static type system) δηλαδή πριν από κάθε compile ελέγχει το συντακτικό του γραμμένου κώδικα και δεν αφήνει το πρόγραμμα να εκτελεστεί έως ότου είναι σωστό, αυτό έχει αποτέλεσμα το τελικό εκτελέσιμο να είναι ασφαλές και εγγυημένα λειτουργικό. Μέσω αυτού του συστήματος, και σε συνδυασμό με το περιβάλλον εκτέλεσης της, η Rust εγγυάται την πλήρη απουσία κρίσιμων σφαλμάτων όπως υπερχειλίση buffer, υπερχειλίση στοίβας καθώς και μην αρχικοποιημένη μνήμη ή ήδη απελευθερωμένη μνήμη.



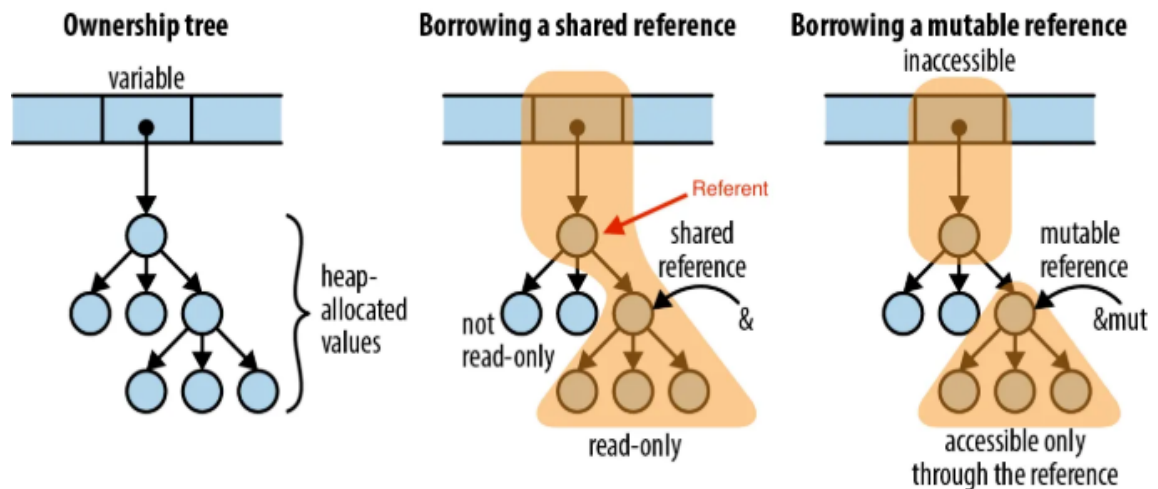
Εικόνα 5.5: Rust, η τομή ανάμεσα σε C++ και Java

5.3.1. Τα κύρια χαρακτηριστικά της Rust

Η Rust ως γλώσσα προγραμματισμού ακολουθεί κάποιες ιδιαίτερες τεχνικές όσον αφορά την διαχείριση μνήμης με κάποια συγκεκριμένα δομικά χαρακτηριστικά που επιλύουν κλασικά προβλήματα μηχανικής λογισμικού [40]. Αυτά τα χαρακτηριστικά αποτέλεσαν μεγάλο ρόλο στην επιλογή της και είναι τα εξής:

- **Ασφάλεια μνήμης μέσω Ιδιοκτησίας (Ownership):** Για τον έλεγχο των αναφορών (aliasing) και τη διασφάλιση της ορθότητας των τύπων, η Rust εισάγει την έννοια της "ιδιοκτησίας" στο σύστημα τύπων της. Η έννοια αυτή εφαρμόζει κανόνες όπως ιδιοκτήτης μίας μεταβλητής, όπου κάθε μεταβλητή έχει έναν ιδιοκτήτη και όταν ο ιδιοκτήτης βγει εκτός "εμβέλειας" η μνήμη απελευθερώνεται. Όπως παρουσιάζεται και στην Εικόνα 5.5 οι μεταβλητές έχουν

ιδιοκτησία πάνω στην μνήμη heap της μεταβλητής, όμως υπάρχει δυνατότητα να δανειστούμε μία μεταβλητή αναφέροντας στην μνήμη heap της, καθώς και να τροποποιήσουμε τα δεδομένα μέσα στην ίδια την μνήμη.



Εικόνα 5.6: Διάγραμμα Ιδιοκτησίας μνήμης της Rust

- **Απουσία Garbage Collector και πρότυπα RAIL(Resource Acquisition Is Initialization):** Η έννοια της ιδιοκτησίας μεταβλητών και η ικανότητα της να απελευθερώνει μνήμη όταν ο ιδιοκτήτης βρίσκεται εκτός εμβέλειας έχει ως αποτέλεσμα η Rust να μην έχει ανάγκη ενός Garbage Collector (η διαδικασία διαγραφής αντικειμένων που δεν χρησιμοποιούνται), αξιοποιώντας την ικανότητα υποστήριξης καταστροφικών (destructors) ορισμένων από τον χρήστη, ενισχύοντας την χρήση προτύπων RAIL.
- **Προβλέψιμο Μοντέλο Απόδοσης:** Προσφέρει ένα πολύ ξεκάθαρο μοντέλο απόδοσης (performance), το οποίο κάνει ευκολότερη την πρόβλεψη της αποδοτικότητας του προγράμματος, παρέχοντας άμεση υποστήριξη για κατανομή μνήμης στη στοίβα (stack allocation) και λεπτομερή έλεγχο αναπαράστασης της μνήμης.

5.3.2. Επιλογή χρήσης για το CLI και το API

Βάσει των παραπάνω χαρακτηριστικών, η επιλογή της Rust εφαρμόζει ιδανικά στα δύο βασικά υποσυστήματα της αρχιτεκτονικής:

- ➔ **Command Line Interface :** Ένα εργαλείο γραμμής εντολών απαιτεί ταχύτητα και αμεσότητα. Το γεγονός ότι όλοι οι τύποι της Rust μπορούν να δεσμευτούν απευθείας στη στοίβα (stack) και ότι η γλώσσα δεν χρειάζεται απολύτως κανένα περιβάλλον εκτέλεσης, την καθιστά ιδανική. Παρέχει την αμεσότητα ελέγχου του χρόνου εκτέλεσης και της μνήμης που συνήθως συναντάται στη C++, επιτρέποντας στο CLI του Hellas Cube να φορτώνεται και να εκτελείται ακαριαία στο τερματικό του χρήστη.
- ➔ **Server-Side API:** Η χρήση της Rust για την υλοποίηση ενός API μπορεί να θεωρηθεί μια πειραματική συμβατική επιλογή σε σχέση με εξειδικευμένα frameworks και γλώσσες (π.χ. σε Python, [Node.js](#) ή Java Spring Boot). Ωστόσο, εκτός από πειραματική επιλογή είναι και στρατηγική. Ένα API που λειτουργεί ως ορχηστρωτής πρέπει να διαχειρίζεται πολλαπλά αιτήματα ταυτόχρονα. Η Rust έχει σχεδιαστεί ρητά για να υποστηρίζει τον ταυτοχρονισμό και

τον παραλληλισμό, εκμεταλλευόμενη πλήρων το σύγχρονο υλικό (modern hardware). Επομένως, επιλέγοντας τη Rust για το API, θυσιάζεται ενδεχομένως η ταχύτητας συγγραφής του κώδικα, αλλά ελαχιστοποιείται η πιθανότητα ότι ο διακομιστής θα καταρρεύσει από προβλήματα διαχείρισης μνήμης κάτω από συνθήκες έντονου παραλληλισμού.

5.4 Η γλώσσα προγραμματισμού Python

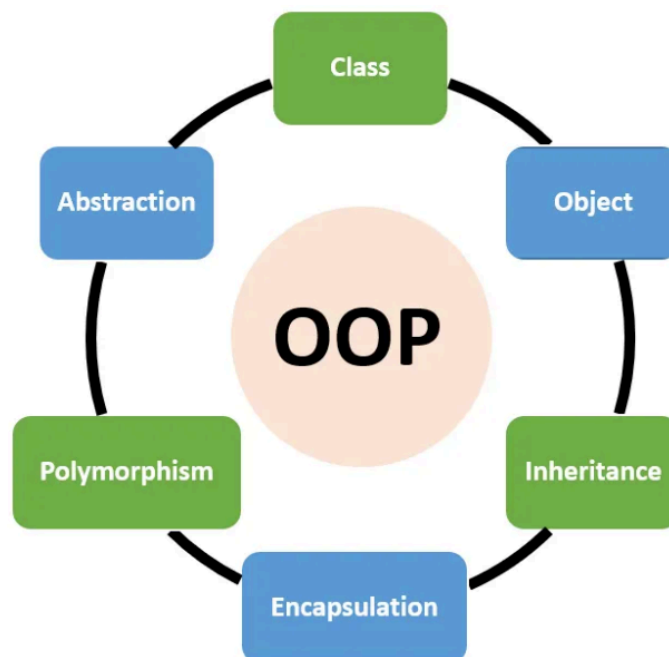
Η Python είναι μία γλώσσα προγραμματισμού η οποία χρησιμοποιείται ευρέως στη βιομηχανία [29], την επιστήμη και την εκπαίδευση λόγω ότι είναι μία διερμηνευόμενη, με ευανάγνωστο κώδικα, γλώσσα προγραμματισμού υψηλού επιπέδου. Υποστηρίζει διάφορα προγραμματιστικά παραδείγματα (programming paradigm) όπως την αντικειμενοστρέφεια, τον διαδικαστικό και τον συναρτησιακό προγραμματισμό. Το πιο σημαντικό και διαδεδομένο προγραμματιστικό στυλ είναι ο αντικειμενοστραφής προγραμματισμός που η python προωθεί τον χρήστη να εφαρμόσει ([29], [40]). Επιπλέον η Python διαθέτει δυναμικό σύστημα τύπων, δηλαδή μας δίνει την δυνατότητα να δηλώσουμε μεταβλητές χωρίς να ξέρουμε από την αρχή τι τύπος θα είναι καθώς και αν τροποποιήσουμε τον τύπο της μεταβλητής, με αποτέλεσμα να κάνει τον προγραμματιστή να ξεχάσει την δήλωση τύπων. Σε αντίθεση με την Rust, η Python διαθέτει δυναμική διαχείριση μνήμης, μέσω Garbage Collector ο οποίος διαγράφει αντικείμενα όταν δεν χρησιμοποιούνται. Αυτά τα χαρακτηριστικά ελαχιστοποιούν τον περιττό κώδικα, και επιτρέπουν την ταχεία ανάπτυξη εφαρμογών και καθιστούν την Python την κυρίαρχη γλώσσα παγκοσμίως στον τομέα των επιστημονικών υπολογισμών και της επιστήμης των δεδομένων.

5.4.1. Αντικειμενοστραφής Προγραμματισμός (Object Oriented Programming OOP) της Python

Το σύστημα αντικειμενοστραφούς προγραμματισμού της Python είναι ευρύ και δυναμικό και αποτέλεσε την κύρια σχεδιαστική προσέγγιση που υιοθετήθηκε, προσφέροντας ισχυρά χαρακτηριστικά όπως:

- **Κλάση (Class):** Θεωρείται η αρχή και το τέλος του αντικειμενοστραφούς προγραμματισμού, καθώς λειτουργεί ως το προσχέδιο για την δημιουργία αντικειμένων, ορίζοντας τη δομή και τη συμπεριφορά τους. Μια θεμελιώδης ιδιαιτερότητα της Python είναι ότι οι ίδιες οι κλάσεις αποτελούν αντικείμενα πρώτης τάξης δηλαδή συμπεριφέρονται ως μεταβλητές στην γλώσσα.
- **Αντικείμενο (Object):** Το αντικείμενο αποτελεί το στιγμιότυπο (instance) μίας κλάσης. Στον πυρήνα της Python κάθε τιμή είναι είτε ένα αντικείμενο είτε μια αναφορά σε ένα αντικείμενο στη μνήμη. Δομικά, κάθε αντικείμενο αποτελείται από τρία μέρη: την κλάση του, το πρωτογενές του περιεχόμενο και ένα λεξιλόγιο (dictionary) το οποίο αποθηκεύει τα πεδία του αντικειμένου με την χρήση αλφαριθμητικών δεικτών (string). Το λεξιλόγιο αυτό επιτρέπει την δυναμική ανάθεση ή διαγραφή πεδίων στο αντικείμενο ανά πάσα στιγμή. [29]
- **Κληρονομικότητα (Inheritance):** Η κληρονομικότητα επιτρέπει τη δημιουργία νέων κλάσεων που αντλούν ιδιότητες από υφιστάμενες κλάσεις. Το μοντέλο της Python είναι εξαιρετικά ισχυρό, καθώς υποστηρίζει πολλαπλή κληρονομικότητα. Η διαχείριση της πολυπλοκότητας (π.χ. ποια μέθοδο θα εκτελεστεί αν υπάρχει σε πολλαπλές γονικές κλάσεις) επιλύεται μέσω του αλγορίθμου Method Resolution Order MRO ο οποίος εξασφαλίζει την σταθερή και προβλέψιμη αναζήτηση μεθόδων ώστε ο compiler να έχει γνώση για το ποιά μέθοδο θα εκτελέσει.

- **Ενθυλάκωση (Encapsulation):** Η ενθυλάκωση αφορά την απόκρυψη της εσωτερικής κατάστασης ενός αντικειμένου και την προστασία των δεδομένων του. Σε αντίθεση με γλώσσες όπως η Java, η Python δεν επιβάλλει ενθυλάκωση σε επίπεδο μεταγλωττιστή, δηλαδή δεν χρειάζεται να δηλωθεί η ορατότητα μιας μεταβλητής. Μπορεί όμως να υποδειχθεί ότι μία μεταβλητή θα χρησιμοποιηθεί εσωτερικά.
- **Πολυμορφισμός (Polymorphism):** Ο πολυμορφισμός στην Python διαφέρει από την κοινή υλοποίηση του σε άλλες γλώσσες προγραμματισμού, καθώς για να εκτελεστεί μία μέθοδος για ένα συγκεκριμένο αντικείμενο δεν χρειάζεται να υπάρχει κληρονομική προέλευση αλλά να υλοποιούνται οι απαραίτητες μέθοδοι. Για παράδειγμα αν δύο κλάσεις οι οποίες υλοποιούν μία συνάρτηση με διαφορετικό περιεχόμενο αλλά ίδιο όνομα π.χ. fly(), μία τρίτη συνάρτηση π.χ. fly_test() που καλεί συνάρτηση fly(), θα λειτουργήσει για αντικείμενο που υλοποιεί μέθοδο fly() [30]. Στην Python ένας επιπλέον πυλώνας είναι οι μαγικές μέθοδοι.
- **Αφαιρετικότητα (Abstraction):** Η αφαιρετικότητα στοχεύει στην απόκρυψη της πολυπλοκότητας της υλοποίησης, εκθέτοντας μόνο τα απαραίτητα χαρακτηριστικά στον χρήστη. Στην Python, αυτό επιτυγχάνεται μέσω σχεδιασμού μεθόδων πάνω σε αυστηρό επίπεδο Αφηρημένων Βασικών Κλάσεων (Abstract Base Classes).



Εικόνα 5.7: Χαρακτηριστικά Object Oriented Programming (OOP)

5.4.2. Κύρια χαρακτηριστικά Επιλογής Python

Η χρήση της Python για την ανάλυση των δεδομένων ήταν αναπόφευκτη καθώς η λειτουργία του συστήματος βασίζεται στην άντληση και διαχείριση γεωχωρικών δεδομένων μέσω του framework Open Data Cube (ODC) το οποίο έχει αναπτυχθεί και υποστηρίζεται εντελώς σε οικοσύστημα Python. Υπάρχουν όμως κάποια χαρακτηριστικά τα οποία την καθιστούν ιδανική για την ανάλυση δεδομένων του Hellas Cube:

- **Ευελιξία (Flexibility):** Η Python φημίζεται για την τεράστια ευελιξία που προσφέρει στους προγραμματιστές [29]. Ιδιότητες, πέραν των προαναφερόμενων:
 - Εκτεταμένη Βασική Βιβλιοθήκη

- Μαγικά ειδικά πεδία (παρακάμπτουν ενσωματωμένες λειτουργίες της γλώσσας)
- Εύκολη διαλειτουργικότητα με άλλες γλώσσες
- Ευελιξία στα arguments
- Κάλυψη πολλών επιστημονικών πεδίων

Αποτελούν την αξεπέραστη ευελιξία που μας δίνει την δυνατότητα να διαχειριστούμε, να μετασχηματίσουμε και να αναλύσουμε τα πολύπλοκα δορυφορικά δεδομένα με δυναμικό τρόπο, προσαρμόζοντας τη συμπεριφορά του κώδικα στις εκάστοτε χωροχρονικές απαιτήσεις.

- **Υψηλή απόδοση:** Παρόλο που η Python είναι μια διερμηνευόμενη γλώσσα, η υψηλή απόδοση και η μεγάλη χρήση της κρύβεται στην διαχείριση υπολογιστικών πόρων. Η γλώσσα μπορεί να χαρακτηριστεί σχετικά αργή αλλά ο λόγος της υψηλής της απόδοσης είναι επειδή λειτουργεί ουσιαστικά ως ένας wrapper που καλεί βιβλιοθήκες και εργαλεία τα οποία είναι γραμμένα σε μεταγλωτισμένες γλώσσες, όπως η C και C++. Με αυτόν τον τρόπο, το Hellas Cube εκμεταλλεύεται της εκφραστικότητας της Python για την οργάνωση της λογικής, ενώ ταυτόχρονα επιτυγχάνει ταχύτητες επιπέδου C++ όταν πρέπει να εφαρμόσει βαρείς μαθηματικούς αλγορίθμους πάνω σε εκατομμύρια pixels.

5.5 Το framework Open Data Cube

Όπως έχει προαναφερθεί το Open Data Cube είναι το εργαλείο με το οποίο διαχειριζόμαστε τα δορυφορικά δεδομένα που “προμηθευόμαστε”. Το Open Data Cube είναι ένα λογισμικό το οποίο αναπτύχθηκε με σκοπό την απλοποίηση των δορυφορικών δεδομένων, με αποτέλεσμα να προσφέρει κάποια χαρακτηριστικά. Ένα από αυτά είναι η ευελιξία αποθήκευσης δεδομένων και τοπικά σε ένα τοπικό σύστημα αρχείων αλλά και άντληση από υποδομές νέφους. Παράλληλα σε περίπτωση αποθήκευσης τοπικά το σύστημα χρησιμοποιείται ως ευρετήριο των δεδομένων έτσι ώστε τα βρίσκει αυτόματα.

Γι’ αυτούς τους λόγους, τα χαρακτηριστικά του λογισμικού, τις τεχνικές ιδιότητες κ.α. επιλέχθηκε για να αναλαμβάνει το υπολογιστικό φορτίο των δορυφορικών Big Data μας. Παρακάτω μία εικόνα χρησιμοποίησης του Open Data Cube όπως προορίζεται.



Εικόνα 5.8: Use Case του Open Data Cube [31]

5.6 PostgreSQL

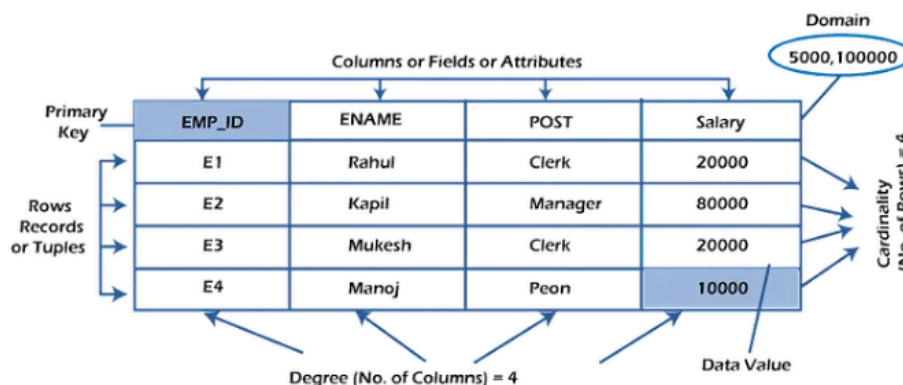
Η στρατηγική επιλογή για τη σχεδίαση της βάσης δεδομένων εδράζεται στην τεχνολογική υποδομή του Open Data Cube, το οποίο υλοποιεί μια τοπική βάση για την ευρετηρίαση των δορυφορικών δεδομένων. Δεδομένου ότι η PostgreSQL αποτελεί ήδη το θεμέλιο για την εξυπηρέτηση του πλαισίου ODC, κρίθηκε αναπόφευκτο η υιοθέτηση του ίδιου τύπου συστήματος διαχείρισης για την κάλυψη των διευρυμένων λειτουργικών απαιτήσεων του συστήματος. Η PostgreSQL αποτελεί ένα από τα πιο ισχυρά και αξιόπιστα Συστήματα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων (RDBMS) ανοιχτού κώδικα. Τα χαρακτηριστικά των σχεσιακών βάσεων και η ικανότητα τους να υποστηρίζουν σύγχρονους, σύνθετους τύπους δεδομένων την κάνουν ιδανική για το Hellas Cube παρόλο που η χρήση της ήταν αναπόφευκτη.

5.6.1. Τι είναι μια Σχεσιακή Βάση δεδομένων

Μία σχεσιακή βάση δεδομένων (Relational Database) είναι ένας βασικός τύπος που αποθηκεύει και παρέχει πρόσβαση σε δεδομένα τα οποία σχετίζονται μεταξύ τους. Βασίζεται στο σχεσιακό μοντέλο, μια προσέγγιση όπου τα δεδομένα, κατατάσσονται σε πίνακες. Κάθε πίνακας αποτελείται από στήλες, οι οποίες καθορίζουν τους τύπους των δεδομένων (π.χ. ακέραιοι, αλφαριθμητικά, ημερομηνίες κ.λπ.), και γραμμές όπου η καθεμία περιέχει μία μοναδική συλλογή δεδομένων για τα πεδία των στηλών. Το βασικό χαρακτηριστικό του μοντέλου είναι η δημιουργία συσχετίσεων μεταξύ των πινάκων μέσω της χρήσης κύριο κλειδίων (primary keys) και ξένων κλειδίων (foreign keys). Τα πρωτεύοντα κλειδιά λειτουργούν ως η μοναδική και υποχρεωτική ταυτότητα κάθε εγγραφής, η οποία δύναται να ενσωματωθεί σε έτερους πίνακες ως αναφορά (ξένο κλειδί), παραπέμποντας απευθείας στην αρχική μητρική οντότητα. Αυτή η δομή αποτρέπει τον πλεονασμό δεδομένων και διασφαλίζει της ακεραιότητα τους.

5.6.2. Σχεσιακή Χρήση στο Σύστημα

Στην αρχιτεκτονική του συστήματος, η PostgreSQL δεν λειτουργεί απλά ως μία βάση δεδομένων, ως ένας απλός αποθηκευτικός χώρος, αλλά αξιοποιείται πλήρως τη σχεσιακή της φύση για τη βέλτιστη απόδοσης της. Αυτό επιτυγχάνεται με τους πίνακες που χρειαστήκαμε στην βάση δεδομένων, όπως χρήστες, αποτελέσματα, log και API να συνδέονται μεταξύ τους με πιθανά ID ως primary keys κι αυτά τα κλειδιά να βρίσκονται ως ξένα κλειδιά μεταξύ, δημιουργώντας ένα δυνατό πλέγμα συσχετίσεων.



Εικόνα 5.9: Παράδειγμα πίνακα σχεσιακής βάσης

5.6.3. Υποστήριξη Πολλαπλών Τύπων Δεδομένων

Ένα από τα σημαντικότερα πλεονεκτήματα της PostgreSQL, το οποίο ξεφεύγει από τα αυστηρά όρια των παραδοσιακών σχεσιακών βάσεων, είναι η εκτεταμένη υποστήριξη που προσφέρει σε πολλαπλούς, σύγχρονους τύπους δεδομένων. Πέρα τους τυπικούς (όπως αριθμούς και κείμενο), η PostgreSQL έχει ενσωματώσει πλήρη και βελτιστοποιημένη υποστήριξη για δεδομένα τύπων UUID, JSON και JSONB (Binary JSON).

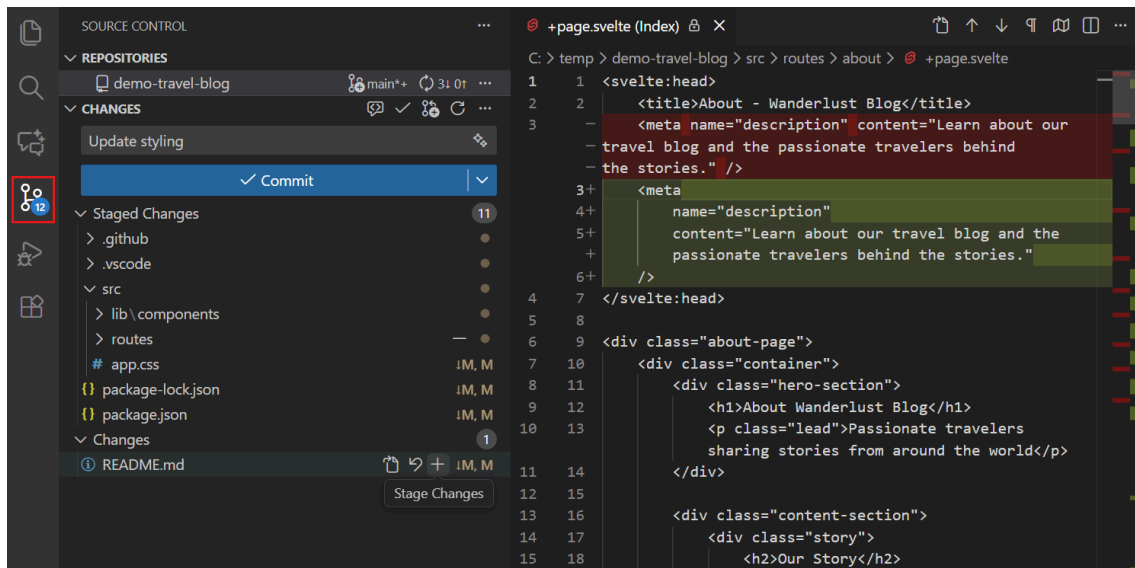
5.7 Visual Studio Code (VSC)

Κατά την διάρκεια ανάπτυξης του συστήματος Hellas Cube, για τις ανάγκες ανάπτυξης του κώδικα επιλέχθηκε ως βασικό περιβάλλον ανάπτυξης (Integrated Development Environment, IDE) το Visual Studio Code. Το συγκεκριμένο περιβάλλον αναπτύχθηκε και συντηρείται ενεργά από τη Microsoft και έχει καθιερωθεί στο χώρο της μηχανικής λογισμικού χάρη στην ελαφριά του αρχιτεκτονική, η οποία επιτρέπει την ταχύτατη εκτέλεση του χωρίς να απαιτεί υπερβολικούς υπολογιστικούς πόρους, διατηρώντας παράλληλα τις δυνατότητες ενός πλήρους IDE. Ένα από τα ισχυρότερα πλεονεκτήματα του είναι η δυνατότητα να υποστηρίζει πρακτικά οποιαδήποτε γλώσσα προγραμματισμού, αυτό επιτυγχάνεται μέσω των extensions (επεκτάσεις). Οι επεκτάσεις μας δίνουν την δυνατότητα να χρησιμοποιήσουμε εργαλεία συμπλήρωσης κώδικα, αποσφαλμάτωσης (debugging) και ελέγχου σύνταξης για τις απαιτήσεις του έργου. Στα πλαίσια της διπλωματικής η ευελιξία επέτρεψε την ταυτόχρονη ανάπτυξη της backend υποδομής στις γλώσσες Python και Rust καθώς και στο frontend στην γλώσσα Rust. Επιπροσθέτως, το εργαλείο διαθέτει ενσωματωμένο το σύστημα ελέγχου εκδόσεων Git. Αυτό μας παρείχε την δυνατότητα να παρακολουθούμε τις αλλαγές στον κώδικα στο κύριο branch που δουλέψαμε μέσα από το γραφικό περιβάλλον του VSC. Με όλες τις λειτουργίες που προαναφέρθηκαν το VS Code αποτελεί ένα απολύτως πλήρως σύστημα ενοποιώντας τον κύκλο ανάπτυξης, βελτιστοποιώντας σημαντικά την παραγωγικότητα.

5.8 Git και GitHub

Η διαχείριση του κώδικα, η παρακολούθηση των αλλαγών και των εκδόσεων αποτελούν θεμελιώδεις πρακτικές στη σύγχρονη μηχανική λογισμικού. Για τον σκοπό αυτό, ενσωματώθηκε το Git, ένα κατακευματισμένο σύστημα ελέγχου εκδόσεων (Distributed Version Control System). Η αρχιτεκτονική του Git επιτρέπει την τοπική αποθήκευση ολόκληρου του ιστορικού του έργου, προσφέροντας ταχύτητα και ασφάλεια καθώς και ανάπτυξη κλάδων (branches).

Ενώ το Git αποτελεί το εργαλείο ελέγχου εκδόσεων, για την φιλοξενία του αποθετηρίου (repository) και την διαχείριση του επιλέχθηκε η πλατφόρμα του GitHub. Το GitHub επεκτείνει τις λειτουργίες του Git προσφέροντας ένα κεντρικό, απομακρυσμένο σημείο αποθήκευσης και ενσωματώνοντας ισχυρά συνεργατικά εργαλεία. Δεδομένου ότι το σύστημα Hellas Cube αναπτύσσεται ως λογισμικού ανοιχτού κώδικα, η επιλογή του GitHub είναι ιδανική καθώς διασφαλίζει την ελεύθερη πρόσβαση της κοινότητας στον κώδικα. Συνοψίζοντας, η συνδυαστική χρήση του Git και του GitHub προσφέρει τεχνική αρτιότητα, ασφάλεια και υψηλή αποτελεσματικότητα, ενώ ολοκληρώνεται αρμονικά με το περιβάλλον του Visual Studio Code. (Εικόνα 5.10).



Εικόνα 5.10: Επέκταση Git στο Visual Studio Code.

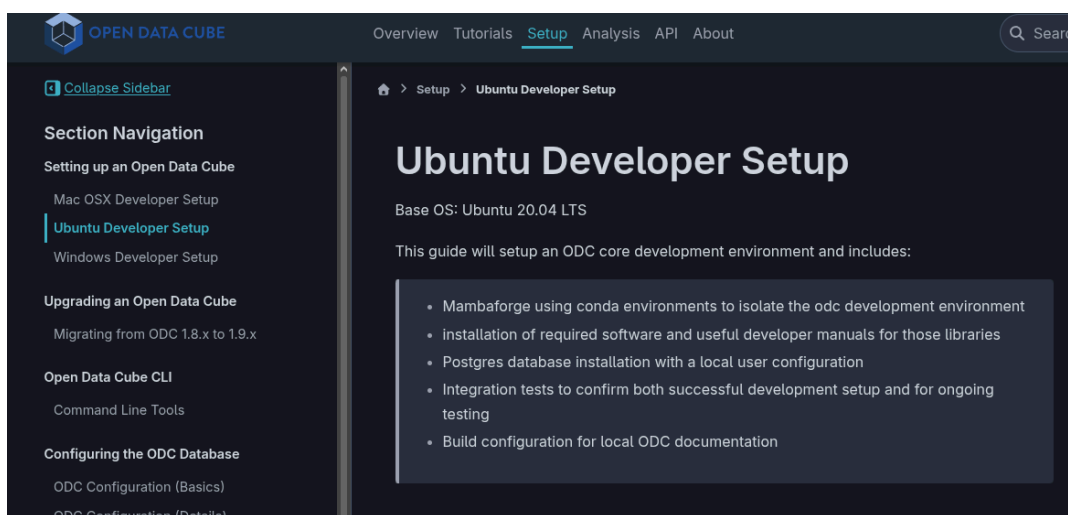
Η χρήση του Git σε συνδυασμό με την πλατφόρμα GitHub εξασφαλίζει την ισορροπία, την ασφάλεια και την αποτελεσματική διαχείριση των εκδόσεων του κώδικα, ολοκληρώνοντας το οικοσύστημα των εργαλείων ανάπτυξης. Ωστόσο, πριν η αρχιτεκτονική των τριών επιπέδων (CLI, Rust API, Python Analytics) μεταφραστεί σε εκτελέσιμο κώδικα, έχει μεγάλη σημασία η προετοιμασία του περιβάλλοντος φιλοξενίας των δεδομένων. Καμία ανάλυση και κανένα API δεν δύναται να λειτουργήσει αποδοτικά εάν δεν έχει προηγηθεί η εγκατάσταση, παραμετροποίηση και διασύνδεση του πλαισίου Open Data Cube με το σχεσιακό σύστημα της PostgreSQL, διαδικασία που αποτελεί την αφετηρία της πρακτικής υλοποίησης του συστήματος.

Κεφάλαιο 6ο: Στήσιμο Open Data Cube

6.1 Οδηγίες Εγκατάστασης Open Data Cube

Έχοντας ολοκληρώσει το θεωρητικό υπόβαθρο και τον αρχιτεκτονικό σχεδιασμό του συστήματος στο προηγούμενο κεφάλαιο, το παρόν κεφάλαιο εστιάζει στο πρώτο πρακτικό μέρος της εφαρμογής, το στήσιμο του περιβάλλοντος που θα “ζει” το framework του Open Data Cube. Το επίπεδο αυτό αποτελεί μέρος του backend (server-side) κομματιού ως ο υπολογιστικός πυρήνας της υποδομής, αναλαμβάνοντας τη άμεση διαχείριση, ανάκτηση και επεξεργασία των δορυφορικών δεδομένων.

Πρωταρχικό βήμα για την ανάπτυξη αυτού του επιπέδου αποτελεί η εγκατάσταση, η σωστή παραμετροποίηση του ODC, η εγκατάσταση των απαραίτητων λογισμικών καθώς και την εγκατάσταση της βάσης δεδομένων. Προκειμένου να ξεκινήσει η αρχική διαμόρφωση του τεχνικού πλαισίου, κρίνεται απαραίτητο να συμβουλευτούμε την επίσημη ιστοσελίδα του Open Data Cube [32]. Η ιστοσελίδα μας παρέχει πλήρες πληροφορίες για τις κατάλληλες προδιαγραφές που πρέπει να έχει το περιβάλλον που θα εφαρμόσουμε το πλαίσιο καθώς και πλούσιο εκπαιδευτικό υλικό με το οποίο μπορούμε να πειραματιστούμε. Στο συγκεκριμένο κεφάλαιο ακολουθήθηκαν οι οδηγίες της σελίδας “Setup” που προορίζεται για συστήματα Linux. Παρόλο που οι οδηγίες μας παροτρύνουν να χρησιμοποιήσουμε Ubuntu Linux Distribution, η παρούσα υλοποίηση βασίζεται στο λειτουργικό σύστημα Fedora. Συνεπώς είναι απαραίτητη η προσαρμογή των εντολών των οδηγιών με τις αντίστοιχες του λειτουργικού συστήματος της συσκευής. Στο επόμενο κεφάλαιο ξεκινάμε με την εγκατάσταση της βάσης δεδομένων και των εργαλείων που θα βρίσκεται εκτός του απομονωμένου περιβάλλοντος της Python.



Εικόνα 6.1: Η σελίδα Setup της ιστοσελίδας του Open Data Cube

6.2 Setup Βάσης Δεδομένων Open Data Cube

Για την εγκατάσταση την βάσης και την παραμετροποίηση της επρεπε να εγκατασταθούν τα απαραίτητα αρχεία βάσεων και επεκτάσεων της βάσης. Το Open Data Cube μας υποδεικνύει τα πακέτα PostgreSQL και postgis όπου το πακέτο PostgreSQL αποτελεί το RDBMS του συστήματος και το postgis μία απαραίτητη επέκταση. Στο τερματικό εκτελέστηκαν οι παρακάτω εντολές:

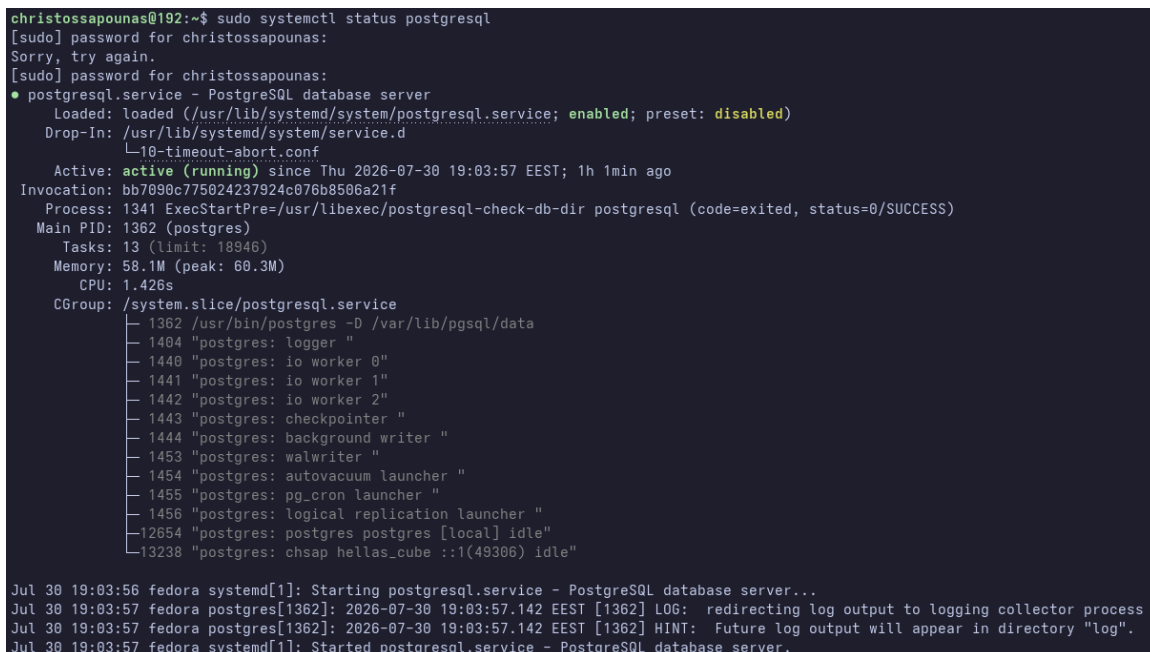
```
christossapounas@192:~$ sudo dnf install PostgreSQL-server PostgreSQL-contrib
christossapounas@192:~$ sudo dnf install postgres
```

Μέσω της ανωτέρω εντολής αιτήθηκαν προνόμια υπερχρήστη (sudo) και, χρησιμοποιώντας τον κατάλληλο διαχειριστή πακέτων dnf για το περιβάλλον Fedora, πραγματοποιεί την ανάκτηση όλων των απαιτούμενων στοιχείων από το διαδίκτυο. Εφόσον έχουμε επιβεβαιώσει ότι τα εργαλεία μας είναι εγκατεστημένα, με τις κατάλληλες εκδόσεις:

```
christossapounas@192:~$ psql --version
psql (PostgreSQL) 18.3
christossapounas@192:~$ rpm -qa | grep -i postgres
postgres-3.6.4-1.fc44.x86_64
```

Στη συνέχεια, το set up προχώρησε με τη δημιουργία της βάσης δεδομένων για την υποστήριξη του ODC. Απαραίτητη προϋπόθεση πριν από τη δημιουργία της βάσης ήταν η αρχικοποίηση και η εκκίνηση της υπηρεσίας PostgreSQL στο λειτουργικό σύστημα. Με τον τρόπο αυτό, η διεργασία της βάσης δεδομένων εκτελείται στο παρασκήνιο ώστε να παραμένει ενεργή και έτοιμη για την υποδοχή συνδέσεων. Η προετοιμασία του περιβάλλοντος πραγματοποιείται με τις ακόλουθες εντολές:

```
christossapounas@192:~$ sudo PostgreSQL-setup --initdb #Αρχικοποιούμε την PostgreSQL
christossapounas@192:~$ sudo systemctl start PostgreSQL #Ξεκινάμε τον εσωτερικό διακομιστή
christossapounas@192:~$ sudo systemctl enable PostgreSQL #Ενεργοποιούμε τον διακομιστή
christossapounas@192:~$ sudo systemctl status PostgreSQL #Επιβεβαιώνουμε την κατάσταση του
```



```
christossapounas@192:~$ sudo systemctl status postgresql
[sudo] password for christossapounas:
Sorry, try again.
[sudo] password for christossapounas:
● postgresql.service - PostgreSQL database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql.service; enabled; preset: disabled)
   Drop-In: /usr/lib/systemd/system/service.d
            └─10-timeout-abort.conf
   Active: active (running) since Thu 2026-07-30 19:03:57 EEST; 1h 1min ago
   Invocation: bb7090c775024237924c076b8506a21f
   Process: 1341 ExecStartPre=/usr/libexec/postgresql-check-db-dir postgresql (code=exited, status=0/SUCCESS)
   Main PID: 1362 (postgres)
   Tasks: 13 (limit: 18946)
   Memory: 58.1M (peak: 60.3M)
   CPU: 1.426s
   CGroup: /system.slice/postgresql.service
           └─1362 /usr/bin/postgres -D /var/lib/pgsql/data
             └─1404 "postgres: logger "
               └─1440 "postgres: io worker 0"
                 └─1441 "postgres: io worker 1"
                   └─1442 "postgres: io worker 2"
                     └─1443 "postgres: checkpointing "
                       └─1444 "postgres: background writer "
                         └─1453 "postgres: walwriter "
                           └─1454 "postgres: autovacuum launcher "
                             └─1455 "postgres: pg_cron launcher "
                               └─1456 "postgres: logical replication launcher "
                                 └─12654 "postgres: postgres postgres [local] idle"
                                   └─13238 "postgres: chsap hellas_cube ::1(49306) idle"

Jul 30 19:03:56 fedora systemd[1]: Starting postgresql.service - PostgreSQL database server...
Jul 30 19:03:57 fedora postgres[1362]: 2026-07-30 19:03:57.142 EEST [1362] LOG:  redirecting log output to logging collector process
Jul 30 19:03:57 fedora postgres[1362]: 2026-07-30 19:03:57.142 EEST [1362] HINT:  Future log output will appear in directory "log".
Jul 30 19:03:57 fedora systemd[1]: Started postgresql.service - PostgreSQL database server.
```

Εικόνα 6.2: Κατάσταση PostgreSQL μετά από την εκτέλεση της εντολής status

Η δημιουργία έγινε και πάλι μέσω τερματικό για καλύτερο έλεγχο (και όχι σε λογισμικό όπως ο DBBeaver που θα μας φανεί χρήσιμο στην συνέχεια). Έπρεπε πρώτα να συνδεθούμε στον default χρήστη της PostgreSQL για να μπορούμε να τρέξουμε SQL εντολές.

```
christossapounas@192:~$ sudo -i -u postgres psql
[sudo] password for christossapounas:
Password for user postgres:
psql (18.3)
Type "help" for help.

postgres=#
```

Εικόνα 6.3 : Εντολή & αποτέλεσμα login χρήστη PostgreSQL

Αφού συνδεθήκαμε στον χρήστη, δημιουργούμε έναν νέο χρήστη ώστε να ελέγχουμε την βάση δεδομένων μέσω ενός προσωπικού superuser και όχι του default (Superuser είναι ο ρόλος με τα υψηλότερα δυνατότερα προνόμια). Η SQL σύνταξη είναι:

CREATE USER chsap WITH PASSWORD 'security_reasons'; —Δημιουργία χρήστη
ALTER USER chsap SUPERUSER; —Αλλαγή χρήστη από κανονικό χρήστη σε superuser

Στην συνέχεια ήταν αναγκαίο να δημιουργηθεί η βάση δεδομένων η οποία θα ανήκει στον χρήστη που δημιουργήθηκε και έχει όλα τα δικαιώματα πάνω στην βάση. Υιοθετείται η ονομασία «opendatacube», εξασφαλίζοντας τη μελλοντική διακριτότητα και την άμεση αναγνωρισιμότητα της βάσης δεδομένων εντός του συστήματος. Η SQL σύνταξη είναι:

CREATE DATABASE opendatacube; —Δημιουργία βάσης με το κατάλληλο όνομα
ALTER DATABASE opendatacube OWNER TO chsap; —Δήλωση owner
GRANT ALL PRIVILEGES ON DATABASE opendatacube TO chsap; —Απόδοση δικαιωμάτων

Στο στάδιο αυτό, ήταν αναγκαία η ενσωμάτωση της επέκτασης PostGIS στη βάση δεδομένων μας. Η συγκεκριμένη επέκταση αποτελεί δομικό στοιχείο για το σύστημά μας, καθώς παρέχει την υποστήριξη γεωγραφικών τύπων δεδομένων, χωρικών ευρετηρίων και εξειδικευμένων συναρτήσεων. Έχοντας διατηρήσει την ενεργή σύνδεση με τη βάση «opendatacube», ολοκληρώνουμε τη διαδικασία με τη δημιουργία της επέκτασης.

\c opendatacube — Εισαγωγή στην βάση δεδομένων
CREATE EXTENSION postgis; —Δημιουργία επέκτασης
\q —Εξαγωγή από την βάση δεδομένων

Καταλήγοντας, πραγματοποιήθηκε επαλήθευση ώστε να διαπιστωθεί ότι η βάση δεδομένων έχει συσταθεί στον ορθό χρήστη με τα ενδεδειγμένα δικαιώματα πρόσβασης, καθώς και ότι έχει ενεργοποιηθεί επιτυχώς η επέκταση PostGIS.

```
postgres=# \l opendatacube
```

Name	Owner	Encoding	Locale Provider	Collate	Ctype	Locale	ICU Rules	Access privileges
opendatacube	chsap	UTF8	libc	en_US.UTF-8	en_US.UTF-8			=Tc/chsap + chsap=CtC/chsap + agdc_admin=CtC/chsap

```
(1 row)
```

```
postgres=# \c opendatacube
You are now connected to database "opendatacube" as user "postgres".
opendatacube=# \dx postgis
```

Name	Version	Default version	Schema	Description
postgis	3.6.4	3.6.4	public	PostGIS geometry and geography spatial types and functions

```
(1 row)
```

Εικόνα 6.4: Εντολές επιβεβαίωσης δημιουργίας βάση δεδομένων

6.3 Setup Περιβάλλοντος

6.3.1. Conda Environment και Απαραίτητα Λογισμικά

Το επόμενο κομμάτι για την εγκατάσταση του ODC είναι το περιβάλλον της Python-Conda. Το περιβάλλον conda όπου μας καθοδηγεί να εγκαταστήσουμε είναι ένας απομονωμένος και αφιερωμένος φάκελος με συλλογή από πακέτα λογισμικών (packages), βιβλιοθήκες (libraries) και εξαρτήσεις (dependencies), η διαχείριση του οποίου γίνεται μέσω της Conda, ενός συστήματος διαχείρισης πακέτων και περιβαλλόντων ανοιχτού κώδικα. Το ODC μας προτρέπει να χρησιμοποιήσουμε και να χτίσουμε (conda-forge) το conda environment μας με ένα εργαλείο ανοιχτού κώδικα, το miniforge [33]. Το mini-forge θα μας δώσει τα βασικά στοιχεία που χρειαζόμαστε για την δημιουργία του περιβάλλοντος. Αρχικά εγκαταστήθηκε το εργαλείο (clone) mini-forge από το github, εκτελέστηκε το εκτελέσιμο αρχείο του ώστε να δημιουργηθεί το εργαλείο στον υπολογιστή μας και αρχικοποιήθηκε σύμφωνα με τις οδηγίες του mini-forge. Εντολές:

```
christossapounas@192:~$ wget
https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-Linux-x86_64.sh
#Κλωνοποίηση εργαλείου από το github
christossapounas@192:~$ bash Miniforge3-Linux-x86_64.sh #Εκτελέσιμο
christossapounas@192:~$ conda init
```

Εφόσον αρχικοποιήθηκε το εργαλείο conda μας δόθηκε η δυνατότητα να δημιουργήσουμε το περιβάλλον μας (με την σωστή έκδοση python, η οποία είναι η 3.10), και να το ενεργοποιήσουμε.

```
christossapounas@192:~$ conda create --name odc_env python=3.10 #Δημιουργία περιβάλλοντος
```

```
christossapounas@192:~$ conda activate odc_env
(odc_env) christossapounas@192:~$
```

Εικόνα 6.5: Ενεργοποίηση περιβάλλοντος

Με την ενεργοποίηση και την αρχικοποίηση του περιβάλλοντος το σύστημα είναι έτοιμο για την εγκατάσταση των κατάλληλων λογισμικών που θα χρειαστεί το Data Cube καθώς και το ίδιο το DataCube. Το ODC καθιστά αναγκαία τα λογισμικά:

- **GDAL:** Μεταφραστής για γεωχωρικά δεδομένα raster και vector.
- **HDF5:** Ένα μορφότυπο αρχείων και βιβλιοθήκη για την αποθήκευση τεράστιων, πολύπλοκων επιστημονικών συνόλων δεδομένων.

- **netCDF4:** Μορφότυπο για επιστημονικά δεδομένα εστιασμένο σε πίνακες.

```
(odc_env) christossapounas@192:~$ sudo dnf install gdal-devel hdf5-devel netcdf-devel
#Εγκατάσταση λογισμικών
```

```
(odc_env) christossapounas@192:~$ conda install -c conda-forge datacube #Εγκατάσταση datacube
```

Η ορθή εγκατάσταση των προτεινόμενων πακέτων λογισμικού και του κεντρικού πλαισίου αποτελεί κρίσιμο σημείο για τη διαμόρφωση του Open Data Cube. Για την επιβεβαίωση της επιτυχούς ένταξης των εργαλείων στο απομονωμένο περιβάλλον (odc_env), αξιοποιείται η εντολή list του διαχειριστή conda, η οποία ανακτά το σύνολο των βιβλιοθηκών, σε συνδυασμό με το φίλτρο grep για τον εντοπισμό των ονομάτων των στοιχείων. Επιπροσθέτως, η εκτέλεση της εντολής datacube επιτρέπει τον προσδιορισμό της τρέχουσας έκδοσης στο σύστημα, επιβεβαιώνοντας την ακεραιότητα της εγκατάστασης.

```
(odc_env) christossapounas@192:~$ conda list | grep -iE "gdal|hdf5|netcdf"
gdal                3.10.3                py310hb501871_27      conda-forge
hdf5                 1.14.6                nompi_h19486de_106    conda-forge
libgdal-core        3.10.3                h95ec890_18           conda-forge
libnetcdf            4.10.0                nompi_hbf2fc22_100    conda-forge
netcdf4              1.7.4                nompi_py310h21e55d4_104 conda-forge
(odc_env) christossapounas@192:~$ datacube --version
Open Data Cube core, version 1.9.12
```

Εικόνα 6.6: Επιβεβαίωση εγκατάστασης των εργαλείων.

6.3.2. Αρχείο Παραμετροποίησης Βάσης

Στο παρόν σημείο έχουμε πετύχει δύο ορόσημα στο setup, το στήσιμο της βάσης δεδομένων που θα χρησιμοποιηθεί από το ODC και το περιβάλλον ODC με τα κατάλληλα εργαλεία. Επόμενο βήμα όμως αποτελεί ο απόλυτος συγχρονισμός των δύο “σταθμών”. Για το σωστό συγχρονισμό ο οδηγός προτείνει πολλούς τρόπους παροχής στοιχείων της βάσης στο περιβάλλον, με αυτούς να είναι μεταβλητές περιβάλλοντος και δημιουργία ενός αρχείου παραμετροποίησης (configuration file). Έτσι ώστε το σύστημα να έχει σταθερή πρόσβαση στα στοιχεία της βάσεων δεδομένων. Για το παρόν σύστημα επιλέχθηκε το αρχείο config καθώς αποτελεί ένας απλούστερος τρόπος και εύκολα διαχειρίσιμος. Το αρχείο διαμορφώθηκε ως κρυφό (με την προσθήκη της τελείας «.» πριν από την ονομασία datacube) και αποθηκεύτηκε στον αρχικό κατάλογο (Home directory) του υπολογιστή, ώστε να καθίσταται άμεσα εντοπίσιμο από το σύστημά μας. Το αρχείο θα αποτελείται από μία επικεφαλίδα datacube και στην συνέχεια από μεταβλητές απαραίτητες για την αυθεντικοποίηση των στοιχείων τη βάσης δεδομένων.

Directory: [/home/christossapounas/.datacube.config](#)

```
[datacube]
db_hostname: localhost
db_port: 5432
db_database: opendatacube
db_username: chsap
db_password: security_reasons
```

6.3.3. Data Cube Προϊόντα

Για την πλήρη ολοκλήρωση του datacube εργαλείο και να είναι έτοιμο να χρησιμοποιηθεί από ένα script python έπρεπε να προστεθούν και προϊόντα δεδομένων στο σύστημα μας. Για αρχή προστέθηκε ένα προϊόν (προς το παρόν ένα για να είναι λειτουργικό) και στην συνέχεια υπάρχει η δυνατότητα πρόσθεσης περισσότερων. Στο οικοσύστημα του ODC, ο όρος «Προϊόν» (Product) δεν αναφέρεται στα ίδια τα αρχεία των δορυφορικών εικόνων, αλλά στο προσχέδιο που τα περιγράφει. Προτού το σύστημα μπορεί να διαβάσει και να επεξεργαστεί μια δορυφορική παρατήρηση, είναι απαραίτητο να καθοριστεί με ακρίβεια το συγκεκριμένο Προϊόν στο οποίο αυτή υπάγεται. Το προϊόν υποδεικνύει στο σύστημα ποιες πληροφορίες να περιμένει, πόσα κανάλια (bands) υπάρχουν και τι τύπου. Παρακάτω υποδεικνύεται ένα configuration file τύπου YAML για τα δεδομένα Sentinel-2(Level 2A) και τριών παραδειγματικών bands (στην πράξη θα προστεθούν όλα τα bands). Στο αρχείο υποδηλώνεται το όνομα του προϊόντος όπως θα εμφανίζεται στην λίστα προϊόντων μας του datacube και στην συνέχεια όπως θα πραγματοποιηθεί αναζήτηση στις υποδομές νέφους του παρόχου δεδομένων. Μετά το σημείο της ονοματοδοσίας των προϊόντων δηλώνονται τα bands ως measurements και για μεταβλητές δέχονται:

- name: Όνομα του καναλιού
- aliases: Ποικιλία ονομάτων καναλιού
- units: Μονάδα μέτρησης τιμών ανακλαστικότητας του καναλιού
- dtype: Ο τύπος δεδομένων σε επίπεδο μνήμης
- nodata: Δήλωση κενών ή άκυρων pixels

Directory: /home/christossapounas/product_hellas_cube.yaml

```
[
---
name: sentinel_2_l2a
description: Sentinel-2a and Sentinel-2b imagery, processed to Level
  2A (Surface Reflectance) and converted to Cloud Optimized GeoTIFFs
metadata_type: eo3

metadata:
  product:
    name: sentinel_2_l2a

measurements:
  - name: "coastal"
    aliases: [band_01, B01, coastal_aerosol]
    units: "1"
    dtype: uint16
    nodata: 0

  -name: "The rest of the bands"
]
```

Όπως έχει προαναλυθεί, τα products είναι κρίσιμα για την λειτουργία του datacube. Για την εξασφάλιση της ακεραιότητας των προϊόντων του Data Cube μας τα YAML (configuration αρχεία)

ακολουθούν τις κατάλληλες προδιαγραφές που μας ορίζει το ODC, οι οποίες είναι:

1. Να είναι συμβατοί με το σύγχρονο πρότυπο μεταδεδομένων EO3(Earth Observation 3), το οποίο απαιτείται για τη δυναμική ανάγνωση δεδομένων από υπολογιστικά νέφη (Cloud Optimized GeoTIFFS).
2. Να έχουν ελεγχθεί εκτενώς σε περιβάλλοντα παραγωγής (production environments) από την ευρύτερη κοινότητα της τηλεπισκόπησης.
3. Να περιλαμβάνουν ακριβείς ρυθμίσεις και χαρακτηριστικά για τα κανάλια.

Με σκοπό τη διασφάλιση της μέγιστης δυνατής αξιοπιστίας, η άντληση των προδιαγραφών των προϊόντων πραγματοποιήθηκε από υφιστάμενα αποθετήρια (repositories) που εξυπηρετούν παρόμοιες ανάγκες. Ειδικότερα, όπως αποτυπώνεται και στην Εικόνα 6.7, υιοθετείται ο ορισμός για το προϊόν Sentinel-2-L2a από το αποθετήριο ανοικτού κώδικα του Digital Earth Africa. Από το συγκεκριμένο αποθετήριο αξιοποιήθηκαν οι ορισμοί προϊόντων τόσο για τους δορυφόρους Landsat 8/9 όσο και για τον Sentinel-2-L2a [34]. Τα προϊόντα που πήραμε από το να αποθετήριο είναι τα:

1. esa_s2_l2a.odc-product.yaml για τον Sentinel-2-L2a [34]
2. ls8_sr.odc-product.yaml για Landsat 8 [34]
3. ls9_sr.odc-product.yaml για τον Landsat 9 [34]

Αν και η εσωτερική δομή των αρχείων παραμένει αμετάβλητη, πραγματοποιήθηκε μια μικρή τροποποίηση στην ονοματολογία δεδομένου ότι το ODC αναγνωρίζει το ίδιο κανάλι με πολλαπλές ονομασίες, η αλλαγή αυτή εφαρμόστηκε για την καλύτερη κατανόηση του κώδικα. Συγκεκριμένα αλλάζει από:

```
- name: SR_B1
  aliases: [band_1, coastal_aerosol]
```

και μετατρέπεται σε:

```
- name: "coastal"
  aliases: [band_01, B01, coastal_aerosol]
```

```
config / products / esa_s2_l2a.odc-product.yaml
```

Bskr-P Merge pull request #623 from digitalearthfrica/add-eo3-validate-action

Code Blame 118 lines (102 loc) · 2.16 KB

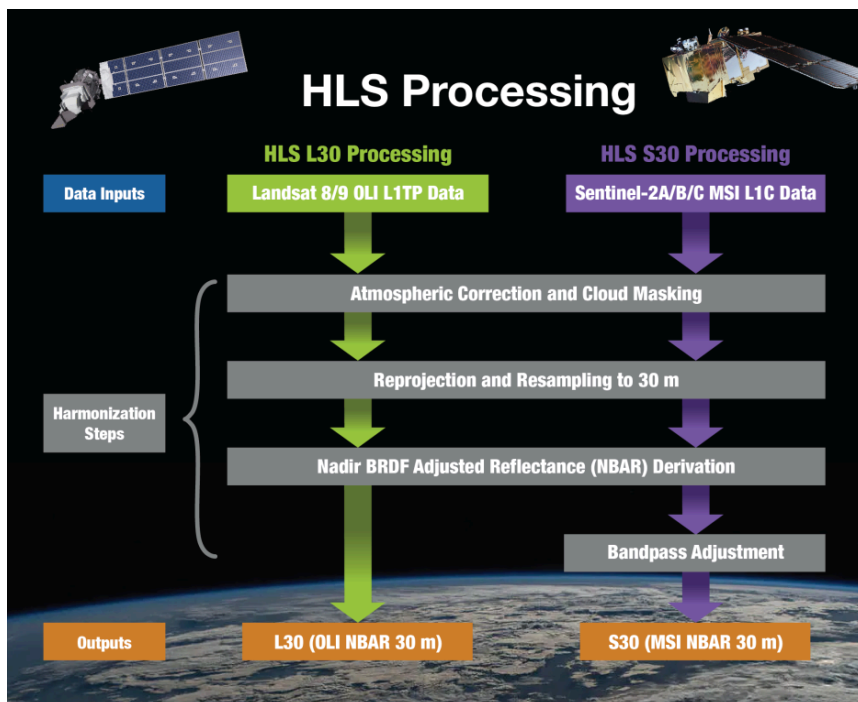
```

1  ---
2  name: s2_l2a
3  description: Sentinel-2a and Sentinel-2b imagery, processed to Level
4    2A (Surface Reflectance) and converted to Cloud Optimized GeoTIFFs
5  metadata_type: eo3
6  license: Copernicus
7
8  metadata:
9    product:
10     name: s2_l2a
11
12  measurements:
13    - name: "B01"
14      aliases: [band_01, coastal_aerosol]
15      units: "1"
16      dtype: uint16
17      nodata: 0
18
19    - name: "B02"
20      aliases: [band_02, blue]
21      units: "1"
22      dtype: uint16
23      nodata: 0
24

```

Εικόνα 6.7: Αρχείο για το Sentinel-2-L2A στο repository του DEA

Στην συνέχεια προστέθηκαν προϊόντα HLS τα οποία είναι απαραίτητα για την αξιοποίηση δύο διακριτών, αλλά εναρμονισμένων δορυφορικών προϊόντων. Η λογική πίσω από το HLS είναι η δημιουργία μία πυκνής και αξιόπιστης χρονοσειράς παρατηρήσεων, γεγονός που επιτυγχάνεται μόνο μέσω του συνδυασμού δεδομένων από διαφορετικές πηγές. Οι διαφορετικές πηγές αυτές διακρίνονται σε HLS L30 για τους Landsat 8/9 και HLS S30 για τον Sentinel-2A. Για να καταλήξουν τα δεδομένα συγκρίσιμα και να προκύψουν τα τελικά προϊόντα, περνούν από μία σειρά βημάτων εναρμόνισης.



Εικόνα 6.8: Επεξεργασία των HLS προϊόντων

Το τελικό αποτέλεσμα αυτής της συνδυαστικής επεξεργασίας αποτυπώνεται αναλυτικά στην Εικόνα 6.8. Ο πίνακας στην εικόνα παρουσιάζει την δομή, τα χαρακτηριστικά και την αντιστοίχιση των φασματικών ζωνών (Bands) ανάμεσα στα δύο προϊόντα (HLS L30 & S30). Για παράδειγμα, παρατηρούμε την κοινή ονοματολογία στα Layer ID όπου βασικές ζώνες όπως το Μπλε (B02), το Πράσινο (B03) και Κόκκινο (B04), η οποία επιτρέπει τη διαφανή χρήση των δεδομένων από το σύστημα. Επιπλέον ο πίνακας ορίζει κρίσιμες παραμέτρους όπως ο τύπος των δεδομένων, ο συντελεστής κλίμακας και η τιμή συμπλήρωσης για κενά δεδομένα, καθώς και τα επίπεδα της Fmask για masking. Συμπερασματικά, λαμβάνοντας υπόψη και παίρνοντας ως πρότυπα τα προϊόντα από το DE Africa και των δεδομένων από τον Πίνακα 6.9 θα χτίσουμε τα Config αρχεία για τα HLS L30 και HLS S30. Η τελική μορφή του θα είναι ως εξής:

```
[
name: hls_l30
description: >
  NASA HLS L30 v2.0 - Harmonized Landsat 8/9 OLI 30m Nadir BRDF-Adjusted
  Reflectance (NBAR), gridded to the Sentinel-2 MGRS tiling system.
metadata_type: eo3
metadata:
  product:
    name: hls_l30
measurements:
  - name: 'B01'
    aliases: [coastal_aerosol]
    dtype: int16
    nodata: -9999
    units: 'reflectance'
-name: τα υπόλοιπα κανάλια]
```

Πίνακας 6.9: Πίνακας καναλιών του Harmonized Landsat Sentinel για το προϊόν HLS

Layer ID		Band Name	Data Type	Scaling Factor	Fill Value	Description* (units)
HLS L30	HLS S30					
B01	B01	Coastal Aerosol	int16	0.0001	-9999	SR (n/a)
B02	B02	Blue				
B03	B03	Green				
B04	B04	Red				
-	B05	Red-Edge 1				
-	B06	Red-Edge 2				
-	B07	Red Edge 3				
-	B08	NIR Broad				
B05	B8A	NIR (Narrow)				
B06	B11	SWIR 1	int16	0.0001	-9999	TOA (n/a)
B07	B12	SWIR 2				
-	B09	Water Vapor				
B09	B10	Cirrus	int16	0.01	-9999	TOA BT (°C)
B10	-	Thermal Infrared 1				
B11	-	Thermal Infrared 2				
Fmask	Fmask	Quality Assessment	uint8	1	255	Per-pixel QA (bit-packed categorical)
SAA	SAA	Solar Azimuth Angle	uint16	0.01	40000	Per-pixel acquisition geometry (degrees)
SZA	SZA	Solar Zenith Angle				
VAA	VAA	View Azimuth Angle				
VZA	VZA	View Zenith Angle				

Προκειμένου να επιτευχθεί η ενσωμάτωση των ανώτερων προϊόντων στο τοπικό datacube του συστήματος, εκτελείται η ειδική εντολή προσθήκης, προσδιορίζοντας με ακρίβεια τη διαδρομή του αντίστοιχου αρχείου YAML:

```
christossapounas@192:~$ datacube product add /home/christossapounas/sentinel_2_l2a.yaml
```

```
christossapounas@192:~$ datacube product add /home/christossapounas/ls8_c2l2_sr.yaml
```

```
christossapounas@192:~$ datacube product add /home/christossapounas/ls9_c2l2_sr.yaml
```

#προσθήκη προϊόντων

```
(odc_env) christossapounas@192:~$ datacube product add /home/christossapounas/hls_l30.yaml
Adding "hls_l30" (this might take a while) DONE
(odc_env) christossapounas@192:~$ datacube product add /home/christossapounas/hls_s30.yaml
Adding "hls_l30" (this might take a while)2026-08-10 23:08:15,678 18386 datacube.index.postgres._products WARNING Product
hls_l30 is already in the database, checking for differences
DONE
```

Εικόνα 6.10: Προσθήκη προϊόντων στο datacube

Επιβεβαιώθηκε ότι το προϊόν εφαρμόστηκε στο datacube τρέχοντας την εντολή:

```
(odc_env) christossapounas@192:~$ datacube product list
hls_l30      NASA HLS L30 v2.0 - Harmonized Landsat 8/9 OLI 30m Nadir BRDF-Adjusted Reflectance (NBAR), gridded to the
Sentinel-2 MGRS tiling system.

hls_s30      NASA HLS S30 v2.0 - Harmonized Sentinel-2 MSI 30m Nadir BRDF-Adjusted Reflectance (NBAR), spectrally bandp
ass-adjusted to Landsat, on the native Sentinel-2 MGRS tiling system.

ls8_c2l2_sr  USGS Landsat 8 Collection 2 Level-2 Surface Reflectance
ls9_c2l2_sr  USGS Landsat 9 Collection 2 Level-2 Surface Reflectance
sentinel_2_l2a Sentinel-2a and Sentinel-2b imagery, processed to Level 2A (Surface Reflectance) and converted to Cloud Op
timized GeoTIFFs
```

Εικόνα 6.11: Εμφάνιση λίστας προϊόντων

Η οποία να μας δίνει ως αποτέλεσμα το όνομα των προϊόντων καθώς και την περιγραφή description που βάλαμε στο YAML αρχείο.

6.3.4. Αρχικοποίηση Συστήματος

Τέλος, για την επίτευξη του συγχρονισμού των αρχείων παραμετροποίησης (config και YAML) της βάσης δεδομένων και των προϊόντων, ήταν αναγκαστική η πραγματοποίηση της αρχικοποίησης του συστήματος του Open Data Cube.

```
(odc_env) christossapounas@192:~$ datacube system init
Initialising database...
Updated.
Checking indexes/views.
Done.
```

Εικόνα 6.12: Αρχικοποίηση Βάσης και Προϊόντων

Με την επιτυχή εκτέλεση της εντολής αρχικοποίησης, διασφαλίζεται ο πλήρης συγχρονισμός της σχεσιακής βάσης δεδομένων με τα αρχεία παραμετροποίησης των δορυφορικών προϊόντων. Η υποδομή του Open Data Cube καθίσταται πλέον έτοιμη να εξυπηρετήσει το υποσύστημα ανάλυσης της Python, παρέχοντας οργανωμένη και ταχύτατη πρόσβαση στα γεωχωρικά ευρετήρια.

Κεφάλαιο 7ο: Ανάπτυξη Python Υποσυστήματος Ανάλυσης

7.1 Προδιαγραφές Υποσυστήματος

Με την υποδομή του Open Data Cube (ODC) και της σχεσιακής βάσης δεδομένων πλήρως εγκατεστημένη και λειτουργική, το παρόν κεφάλαιο εστιάζει στην πρακτική υλοποίηση του προγραμματιστικού πυρήνα του συστήματος Hellas Cube. Το υποσύστημα αυτό αναπτύχθηκε εξ ολοκλήρου σε γλώσσα Python, και έχει ως ρόλο την επεξεργασία δορυφορικών αρχείων και την ανάπτυξη της τελικής πληροφορίας που ζητά ο χρήστης.

Το στάδιο αυτό αναδεικνύεται ο κύριος στόχος και το κεντρικό ερευνητικό αντικείμενο της παρούσας εργασίας: η πολυδιάστατη ανάλυση περιβαλλοντικών δεδομένων. Όλα τα βήματα στα επίπεδα αρχιτεκτονικής, εγκατάστασης και διαχείρισης API αποτελούν τον υποστηρικτικό πλαίσιο που εξυπηρετεί αυτόν ακριβώς τον σκοπό. Το βασικό επίκεντρο αυτού του υποσυστήματος είναι ο τρόπος αλληλεπίδρασης του κώδικα με τον τεράστιο όγκο της Παρατήρησης της Γης για να εξάγει ουσιαστική πληροφορία. Η αρχιτεκτονική δομή και η λειτουργικότητα του υποσυστήματος στηρίζονται σε μια στέρεη τεχνική υποδομή, η οποία αναλαμβάνει τον κομβικό ρόλο της επεξεργασίας. Η υλοποίηση βασίζεται στην αυστηρή εφαρμογή σύγχρονων αρχιτεκτονικών προτύπων, τη χρήση εξειδικευμένων προγραμματιστικών κλάσεων και τη βέλτιστη αξιοποίηση των σεναρίων εντολών (scripts) της Python. Μέσα από ένα σύνολο κρίσιμων συναρτήσεων, διαχειρίζεται αποδοτικά η πολυπλοκότητα των δεδομένων, διαμορφώνοντας μια αδιάλειπτη ροή εκτέλεσης που εγγυάται τη σταθερότητα και την αξιοπιστία ολόκληρου του έργου.

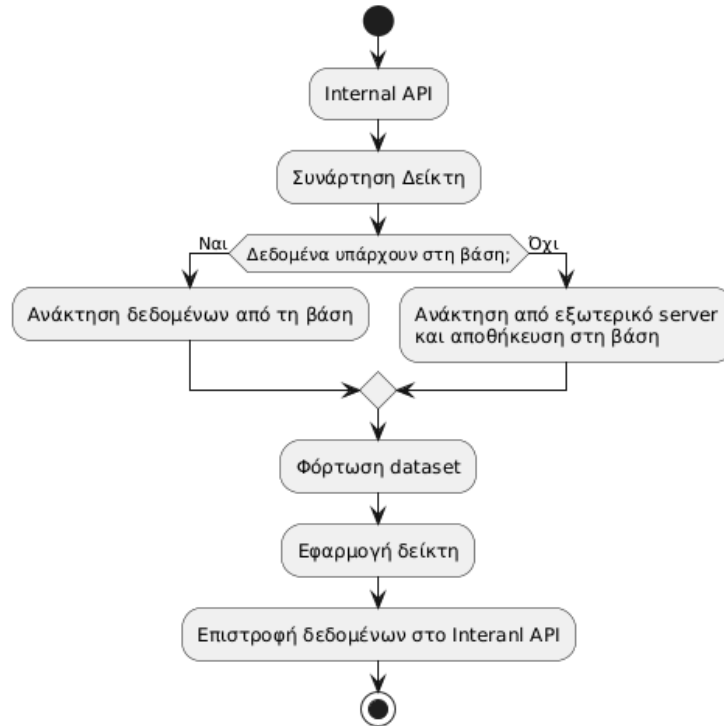
7.2 Αρχιτεκτονική του επιπέδου (Python Layer)

Στο σύστημα Hellas Cube, το υποσύστημα της Python καθώς είναι ένα αρκετά σημαντικό επίπεδο οπότε η σχεδίαση της αρχιτεκτονικής του είναι εξίσου σημαντική. Όπως αποτυπώνεται και στην Εικόνα 7.1 η διαχείριση των εργαλείων γίνεται μέσω ενός API (βιβλιοθήκη python για υλοποίηση API) το οποίο είναι η αρχή και το τέλος του υποσυστήματος.

Η ροή εργασιών μιας ανάλυσης δεδομένων (όπως για παράδειγμα η εξαγωγή ενός περιβαλλοντικού δείκτη) ακολουθεί μια προκαθορισμένη σειρά βημάτων, με το προσχέδιο για την ανάπτυξη του υποσυστήματος να διαμορφώνεται ως εξής:

1. **Υποδοχή αιτήματος (Internal API):** Το σύστημα ενεργοποιείται όταν το API λάβει ένα αίτημα ανάλυσης, το οποίο περιέχει τις απαραίτητες παραμέτρους (όπως χωρικά και χρονικά όρια).
2. **Κλήση συνάρτησης δείκτη/αλγόριθμου:** Το αίτημα προωθεί στην αντίστοιχη εξειδικευμένη συνάρτηση που αναπτύχθηκε για τον υπολογισμό του ζητούμενου δείκτη.
3. **Έλεγχος διαθεσιμότητας δεδομένων:** Πριν οποιαδήποτε υπολογιστική διεργασία, το σύστημα ελέγχει αν τα απαιτούμενα δορυφορικά δεδομένα είναι ήδη ευρετηριασμένα.
 - Εάν Ναι : Το σύστημα συνεχίζει ανακτώντας τα δεδομένα από τη βάση.
 - Εάν Όχι: Το σύστημα επικοινωνεί δυναμικά με εξωτερικές πηγές και αντλεί τις απαραίτητες πληροφορίες
4. **Φόρτωση Dataset:** Με τα δεδομένα πλέον να είναι διαθέσιμα, το σύστημα επικοινωνεί με το Data Cube και φορτώνει το dataset στη μνήμη

5. **Εφαρμογή Δείκτη:** Στο φορτωμένο dataset εφαρμόζονται οι απαραίτητοι δείκτες ή αλγόριθμοι καθώς και τεχνικές συγκάλυψης (masking) για την παραγωγή της τελικής πληροφορίας.
6. **Επιστροφή δεδομένων στο API:** Το τελικό αποτέλεσμα μορφοποιείται και επιστρέφεται πίσω στο API το οποίο το προωθεί στο κεντρικό σύστημα.



Εικόνα 7.1: Ιδέα αρχιτεκτονικής Υποσυστήματος Ανάλυσης

7.3 Ευρετηρίαση δεδομένων (Indexing Data)

7.3.1. Εντολή stac-to-dc

Όπως έχει ήδη αναλυθεί, ο ευρετηριασμός αποτελεί τον μηχανισμό με τον οποίο το σύστημα καταχωρεί και οργανώνει τα μεταδεδομένα των δορυφορικών παρατηρήσεων στη βάση δεδομένων. Ουσιαστικά το σύστημα καταγράφει δεδομένα εύρεσης των πραγματικών δεδομένων των οποίων η πραγματική τοποθεσία είναι στο cloud. Στην σύγχρονη διαχείριση γεωχωρικών δεδομένων, το πρότυπο STAC (SpatioTemporal Asset Catalog) αποτελεί ο κυρίαρχος τρόπος περιγραφής και ο τρόπος που το ODC υποστηρίζει και προτείνει [32]. Παρόλο που η εντολή product add δέχεται προϊόντα της μορφής EO3 (Earth Observation Metadata type 3) όσο και έγγραφα STAC το Data Cube μπορεί να επεξεργαστεί προϊόντα μορφής EO3. Αυτό έχει ως αποτέλεσμα όταν δεχόμαστε έγγραφα μέσω STAC, να μετατρέπονται σε μορφή EO3 και τα έγγραφα μας να ακολουθούν μία και μόνο μορφή παρέχοντας μία αρμονία μέσα στο σύστημα του κύβου. Η παροχή των δεδομένων μας παρέχεται μέσω μιας υποδομής αποκλειστικά για την βέλτιστη ευρετηρίαση των δεδομένων, με τον πάροχο να είναι η Amazon Web Services (AWS). Η προτίμηση της AWS στηρίζεται στη συμβατότητα του Open Data Cube, το οποίο σχεδιάστηκε με γνώμονα τις δικές της υποδομές, καθώς και στο γεγονός ότι αποτελεί τον επιλεγμένο πάροχο για υλοποιήσεις μεγάλης κλίμακας, όπως το Digital Earth Africa.

Για την ευρετηρίαση των δεδομένων χρειάστηκε να χρησιμοποιήσουμε ένα εργαλείο το οποίο ονομάζεται stac-to-dc. Το εργαλείο παρέχεται μέσω εγκατάστασης του με την εκτέλεση της εντολής:

```
christossapounas@192:~$ conda install -c conda-forge odc-stac
```

Για την εκτέλεση της εντολής του stac-to-dc χρειαζόμαστε τα εξής ορίσματα:

- Τον σύνδεσμο του καταλόγου της AWS για να ξέρει που να κοιτάξει η εντολή για δεδομένα.
- Την συλλογή από την οποία αντλούμε δεδομένα και το προϊόν που λαμβάνουμε.
- Τις συντεταγμένες του γεωχώρου των δορυφορικών δεδομένων που θέλουμε.
- Το διάστημα χρόνου δεδομένων που θέλουμε να αντλήσουμε.

Μια ενδεικτική εκτέλεση εντολής είναι:

<code>stac-to-dc</code>	#Εντολή
<code>--catalog-href https://earth-search.aws.element84.com/v1</code>	#Λινκ καταλόγου παρόχου
<code>--collections sentinel-2-l2a</code>	#Συλλογή δεδομένων
<code>--bbox 22.8,40.6,23.0,40.7</code>	#Γεωχωρος
<code>--datetime 2023-05-01/2023-05-31</code>	#

Στην βάση μας μετά την εκτέλεση της εντολής stac-to-dc λαμβάνουμε ένα σύνολο από συνδέσμους. Για να εντοπίσουμε αυτούς τους συνδέσμους, ανατρέχουμε αρχικά σε ένα από τα ευρετηριασμένα στοιχεία. Στην Εικόνα 7.2 φαίνεται το μοναδικό ID που του έχει αποδοθεί από το datacube, ενώ τα δεδομένα που αποθηκεύτηκαν για το έγγραφο παρουσιάζονται στην Εικόνα 7.3.

```
(odc_env) christossapounas@192:~$ datacube dataset search product=sentinel_2_l2a -f csv
id,status,product,location
f57296b6-0d7c-53d1-8d3a-e394f602fe28,active,sentinel_2_l2a,https://earth-search.aws.element84.com/v1/collections/sentinel-2-l2a/items/S2B_35SKB_20220525_0_L2A
```

Εικόνα 7.2: Εμφάνιση στοιχείων

```
(odc_env) christossapounas@192:~$ datacube dataset info f57296b6-0d7c-53d1-8d3a-e394f602fe28
id: f57296b6-0d7c-53d1-8d3a-e394f602fe28
product: sentinel_2_l2a
status: active
indexed: 2026-05-22 02:35:24.591822+03:00
location: https://earth-search.aws.element84.com/v1/collections/sentinel-2-l2a/items/S2B_35SKB_20220525_0_L2A
fields:
  cloud_cover: 0.223765
  creation_dt: '2022-11-06T12:12:15.285000+00:00'
  creation_time: 2022-11-06 12:12:15.285000+00:00
  dataset_maturity: null
  format: GeoTIFF
  grid_spatial:
    geo_ref_points:
      ll:
        x: 199980.0
        y: 4090200.0
      lr:
        x: 309780.0
        y: 4090200.0
      ul:
        x: 199980.0
        y: 4200000.0
      ur:
        x: 309780.0
        y: 4200000.0
    spatial_reference: EPSG:32635
  valid_data:
    coordinates:
      - - 199981.0000000003
        - 4199998.999999999
        - 309779.0
        - 4199998.999999999
        - 309778.999999999
        - 4090200.9999999995
        - 199980.9999999994
        - 4090200.999999999
        - 199981.0000000003
        - 4199998.999999999
      type: Polygon
    id: f57296b6-0d7c-53d1-8d3a-e394f602fe28
    instrument: MSI
    label: null
    lat: {begin: 36.91012847809043, end: 37.927671217101775}
    lon: {begin: 23.588039291075088, end: 24.863963480483836}
    measurements:
      aot:
        path: https://sentinel-cogs.s3.us-west-2.amazonaws.com/sentinel-s2-l2a-cogs/35/S/KB/2022/5/S2B_35SKB_20220525_0_L2A/AOT.tif
      aot-jp2:
        path: s3://sentinel-s2-l2a/tiles/35/S/KB/2022/5/25/0/AOT.jp2
```

Εικόνα 7.3: Αναλυτική μορφή αποθηκευμένου εγγράφου

Όπως μπορούμε να παρατηρήσουμε από την Εικόνα 7.3 το τελικό αποτέλεσμα του ευρετηριασμού αποτελείται από πεδία με τα στοιχεία του εγγράφου και τους συνδέσμους για τα μεταδεδομένα. Συγκεκριμένα μπορούμε να παρατηρήσουμε δεδομένα σχετικά με την εντολή stac που εκτελέσαμε καθώς και τα πραγματικά δεδομένα όπως ότι η συγκεκριμένη εικόνα έχει περίπου 22% νέφωση. Η αξία του indexing όμως αναδεικνύεται στο πεδίο των measurements όπου για το πεδίο aot (Aerosol Optical Thickness) παρατηρούμε το μονοπάτι του συνδέσμου για το ογκώδες αρχείο .tif.

7.3.2. Εκτέλεση εντολής stac-to-dc μέσω την βιβλιοθήκης subprocess

Μέσω αυτής της διαδικασίας, η ευρετηρίαση των δορυφορικών εικόνων ολοκληρώνεται επιτυχώς, επιτρέποντάς μας να προχωρήσουμε πλέον στο στάδιο της ανάλυσής τους. Η εντολή αυτή όμως εκτελείται μέσω της Python δυναμικά και κάθε φορά που δεν υπάρχουν τα δεδομένα μας στην βάση. Με την βοήθεια της βιβλιοθήκης subprocess μπορούμε να εκτελούμε μια εντολή όποτε χρειαστεί κατά την εκτέλεση του συστήματος στην Python. Απλούστερα αυτοματοποιείται η εκτέλεση της και χτίζεται μία “γέφυρα” ανάμεσα σε κώδικα και τερματικό.

Η βιβλιοθήκη subprocess αποτελεί ένα ενσωματωμένο module python, το οποίο έχει σχεδιαστεί για την εκκίνηση και διαχείριση νέων εσωτερικών διεργασιών στο επίπεδο του λειτουργικού συστήματος [36]. Μέσω την εντολής subprocess μπορούμε να διαχειριστούμε το κανάλι εισόδου (stdin), δηλαδή το ίδιο το command, το κανάλι εξόδου (stdout) καθώς και των σφαλμάτων (stderr) της διεργασίας.

7.3.2.1 Τεχνική υλοποίηση

Για να υλοποιήσουμε πρακτικά την εκτέλεση της εντολής, παρατηρούμε στην Εικόνα 7.4 ότι πρέπει να αρχικοποιήσουμε πρώτα κάποιες μεταβλητές που χρησιμοποιούνται κατά την εκτέλεση της εντολής stac. Με τους δείκτες x_min, x_max, y_min, y_max οριοθετούμε τον γεωχώρο και με την desired_data_range επιλέγουμε το χρονικό διάστημα των δεδομένων μας. Οι δύο αρχικοποιήσεις μετατρέπονται σε String έτσι ώστε να παίρνονται ως ορίσματα στο χτίσιμο της εντολής. Για την σύνταξη της εντολής χρησιμοποιούμε μία λίστα δεδομένων όπου το κάθε της στοιχείο αντικατοπτρίζει και ένα σημείο της εντολής. Τέλος, πραγματοποιείται η κλήση της βιβλιοθήκης subprocess, εισάγοντας ως παραμέτρους τη λίστα εντολών (command list) και ενεργοποιώντας τον έλεγχο σφαλμάτων (check=True) για την επαλήθευση της επιτυχούς ολοκλήρωσης. Στη συγκεκριμένη υλοποίηση, η έξοδος ορίζεται ως κενή (στην παρούσα φάση), ενώ παράλληλα προβλέπεται η εμφάνιση κατάλληλου μηνύματος σε περίπτωση εντοπισμού σφάλματος.

```
print("No data found! staccng....")
#bbox
x_min, x_max = (22.90, 22.95)
y_min, y_max = (40.65, 40.7)
bbox_str = f"{x_min},{y_min},{x_max},{y_max}"

#dates
desired_date_range = ("2023-05-01", "2023-05-31")
date_str = f"{desired_date_range[0]}/{desired_date_range[1]}"

#command
command = [
    "/home/christossapounas/.conda/envs/odc_env/bin/stac-to-dc",
    "--catalog-href", "https://earth-search.aws.element84.com/v1",
    "--collections", "sentinel-2-l2a",
    "--bbox", bbox_str,
    "--datetime", date_str,
    "--rename-product", "sentinel_2_l2a"
]

#execution
subprocess.run(command, check=True, stdout=subprocess.DEVNULL, stderr="stac-to-dc failed")
```

Εικόνα 7.4: Script εκτέλεσης εντολής

7.4 Φόρτωση datasets

Μετά την επιτυχή ολοκλήρωση του αυτόματου ευρετηριασμού, το σύστημα διαθέτει πλήρη γνώση για τα διαθέσιμα datasets indexes της βάσης και είναι έτοιμα να αξιοποιηθούν. Για να καταστούν αυτά διαθέσιμα προς αλγοριθμική ανάλυση και εξαγωγή περιβαλλοντικών δεικτών είναι απαραίτητη η φόρτωση τους στην κύρια μνήμη RAM του υπολογιστικού περιβάλλοντος. Αυτό το στάδιο γεφυρώνει το χάσμα μεταξύ της απλής αναζήτησης, μεταδεδομένων και της ουσιαστικής επεξεργασίας της γεωχωρικής πληροφορίας.

7.4.1. Η συνάρτηση `dc.load()`

Η βασική μέθοδος για την ανάκτηση των δεδομένων μέσω της Python και του ODC είναι η συνάρτηση `dc.load()`. Η συγκεκριμένη εντολή είναι εξαιρετικά σημαντική και λειτουργεί ως ένας πρώιμος μηχανισμός επεξεργασίας των δορυφορικών δεδομένων. Κατά την κλήση της ορίζει συγκεκριμένες παραμέτρους όπως:

- Το όνομα του προϊόντος (στην Εικόνα 7.5 για τον Sentinel)
- Τον άξονα X του γεωχώρου προς επεξεργασία
- Τον άξονα Y του γεωχώρου προς επεξεργασία
- Το CRS το οποίο δηλώνει την ζώνη αναζήτησης των δεδομένων (στην Εικόνα 7.5 δηλώνουμε την παγκόσμια ζώνη)
- Το διάστημα των δεδομένων προς επεξεργασία
- Τις μετρήσεις καναλιών που θέλουμε να φορτώσουμε (στην Εικόνα 7.5 ορίζονται τα κανάλια του κόκκινου και του εγγύς ερυθρού)
- Το CRS εξαγωγής που προδίδει το σύστημα συντεταγμένων που προβάλλονται τα δεδομένα εξόδου
- Την ανάλυση των δεδομένων (Στην Εικόνα 7.5 η χωρική ανάλυση της εικόνας ορίζεται στα 10 μέτρα, η τιμή που υποστηρίζεται από τα δεδομένα Sentinel)
- Την ομαδοποίηση πολλαπλών παρατηρήσεων δορυφόρου που έγιναν την ίδια ημερολογιακή μέρα ως ένα ενιαίο χρονικό διάστημα και όχι ξεχωριστά

```
#load the dataset
ds = dc.load(
    product="sentinel_2_l2a",
    x=(22.90, 22.95),
    y=(40.65, 40.7),
    crs="EPSG:4326",
    time=desired_date_range,
    measurements=["red", "nir"],
    output_crs="EPSG:32635",
    resolution=(-10, 10),
    group_by='solar_day'
)
```

Εικόνα 7.5 Φόρτωση ενός dataset (όπου dc ένα αντικείμενο τύπου DataCube)

7.4.2. `xarray.Dataset`

Το αποτέλεσμα που επιστρέφει η συνάρτηση `dc.load()` δεν είναι ένας απλός πίνακας ή μία λίστα, αλλά ένα αντικείμενο τύπου `xarray.Dataset`. Η βιβλιοθήκη `xarray` αποτελεί ένα από τα θεμελιώδη εργαλεία του οικοσυστήματος της Python για την επιστημονική ανάλυση δεδομένων. Ουσιαστικά, το `xarray` επεκτείνει τις δυνατότητες της βιβλιοθήκης `NumPy`, επιτρέποντας την επεξεργασία πολυδιάστατων πινάκων με την χρήση ετικετών αντί για ακέραιους δείκτες. Ένα `xarray` στο `Open Data Cube` οργανώνει την πληροφορία σε τρεις κύριες διαστάσεις: το χρόνο, τον άξονα Y και τον άξονα X όπως απεικονίζεται και στην Εικόνα 7.6.

```

sh-5.3$ /home/christossapounas/.conda/envs/odc_env/bin/python /run/media/christossapounas/experiment.py
<xarray.Dataset> Size: 2MB
Dimensions:      (time: 6, y: 289, x: 225)
Coordinates:
  * time          (time) datetime64[ns] 48B 2020-05-03T09:29:10.790000 ... 202...
  * y             (y) float64 2kB 4.514e+06 4.514e+06 ... 4.508e+06 4.508e+06
  * x             (x) float64 2kB 1.533e+05 1.534e+05 ... 1.578e+05 1.578e+05
    spatial_ref   int32 4B 32635
Data variables:
  red             (time, y, x) uint16 780kB 2004 1924 1908 ... 4920 4860 5088
  nir             (time, y, x) uint16 780kB 3960 4036 3988 ... 5404 4796 5420
Attributes:
  crs:            EPSG:32635
  grid_mapping:   spatial_ref

```

Εικόνα 7.6: Το xarray.Dataset μετά το dc.load()

7.5 Πρώτο test

Αφού έχουμε εξετάσει και υλοποιήσει δύο κρίσιμα κομμάτια, την αποθήκευση των index στην βάση δεδομένων και την φόρτωση των datasets στην μνήμη ως xarray datasets μπορούμε να κάνουμε μία πρώτη δοκιμή, στο οποίο βασιστήκαμε στη συνέχεια της ανάπτυξης του συστήματος. Αρχικά ακολουθήσαμε την αρχιτεκτονική που έχουμε ορίσει στην αρχή του κεφαλαίου, όπου βλέπουμε αν έχει φορτωθεί dataset. Αν δεν έχει φορτωθεί τότε πρέπει να εκτελεστεί η εντολή stac-to-dc. Όταν είναι έτοιμη η βάση (ή αν ήταν ήδη έτοιμη) τότε φορτώνεται το dataset και εφαρμόζεται ένας τυχαίος περιβαλλοντικός τύπος, στην προκειμένη περίπτωση τον τύπο NDVI. Όπως βλέπουμε και στην Εικόνα 7.7 παίρνουμε τα κανάλια του τύπου τα οποία είναι το κόκκινο και το εγγύς ερυθρό από το dataset, τα μετατρέπουμε σε τύπο float για μεγαλύτερη δεκαδική ακρίβεια και το πολλαπλασιάζουμε με κλίμακα 0.0001 για την μετατροπή του σε πραγματικό. Στην συνέχεια εφαρμόζουμε τον μαθηματικό τύπο και με την βοήθεια της συνάρτησης isel απομονώνουμε τα δεδομένα (x,y) του dataset επιλέγοντας τον χρόνο 0 (time=0) ως το χρονικό στιγμιότυπο. Τέλος εμφανίζουμε τον μέσο όρο από το υπολογισμένο NDVI.

```

red = ds["red"].astype("float32")
nir = ds["nir"].astype("float32")
ndvi = (nir - red) / (nir + red)
calc_ndvi = ndvi.isel(time=0)
print(f"NDVI Mean : {calc_ndvi.mean().values:.3f}")

```

Εικόνα 7.7: Προγραμματιστική εφαρμογή τύπου NDVI στο xarray.Dataset

Κατά την εκτέλεση του σεναρίου dc_query_experiment.py, διαπιστώνεται στην Εικόνα 7.8 η επιτυχής ολοκλήρωση της δοκιμής. Συγκεκριμένα, πραγματοποιήθηκε επιτυχώς ο υπολογισμός του δείκτη NDVI για μια τυχαία επιλεγμένη γεωγραφική περιοχή, αφού προηγουμένως ανακτήθηκαν τα απαραίτητα δεδομένα μέσω της εντολής STAC, καθώς δεν ήταν διαθέσιμα στην βάση. Η συγκεκριμένη δοκιμή θέτει τα θεμέλια όλης αρχιτεκτονικής, καθώς με τις κατάλληλες βελτιστοποιήσεις και την ενσωμάτωσή της σε ένα εσωτερικό API, αποτέλεσε το βασικό υποσύστημα ανάλυσης του Hellas Cube.

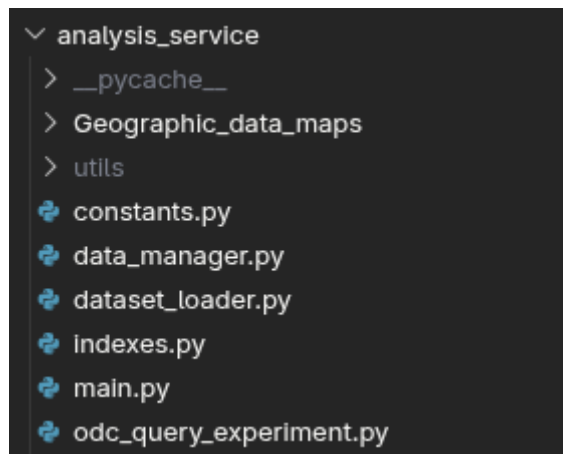
```
(odc_env) christossapounas@192:~$ python /run/media/christossapounas/SAPOUNASUSB/Thesis_Hellas_Cube/Hellas_Cube/analysis_service/odc_query_experiment.py
No data found! staccating....
Done
NDVI Mean : 0.193
```

Εικόνα 7.8: Αποτελέσματα δοκιμής

7.6 Δομή αρχείων Συστήματος Ανάλυσης

Έχοντας εξασφαλίσει τα απαραίτητα προϊόντα για την σωστή αναγνώριση των δορυφορικών δεδομένων, η ανάπτυξη πέρασε στο προγραμματιστικό κομμάτι. Η επιτυχής έκβαση της δοκιμής μάς επιτρέπει να αναπτύξουμε το υπόλοιπο σύστημα γύρω από αυτόν τον πυρήνα, διαμορφώνοντάς το ως μια πλήρως αυτόνομη υπηρεσία. Όπως αποτυπώνεται στην Εικόνα 7.9, το κεντρικό πακέτο “analysis_service” του υποσυστήματος Python διαχωρίζει διακριτά τις λειτουργικότητες του συστήματος στα παρακάτω αρχεία:

- **main.py:** Το σημείο εισόδου της εφαρμογής. Εδώ φιλοξενείται η αρχικοποίηση του εσωτερικού API το οποίο υποδέχεται τα αιτήματα από τον εξωτερικό διακομιστή και ελέγχει την εκτέλεση των υπόλοιπων ενοτήτων.
- **data_manager.py:** Ο διαχειριστής της ροής δεδομένων. Είναι υπεύθυνος για την σωστή φόρτωση των dataset καθώς και για την σωστή εξαγωγή τους στον διακομιστή.
- **dataset_importer.py:** Εξειδικεύεται στον δυναμικό έλεγχο της βάσης. Επιβεβαιώνει την ύπαρξη των απαιτούμενων δεδομένων και, εφόσον κριθεί απαραίτητο, αναλαμβάνει την εκτέλεση εντολής stac-to-dc για την άντληση δεδομένων.
- **indexes.py:** Ο επιστημονικός πυρήνας του συστήματος. Εδώ έχουν συγκεντρωθεί και υλοποιηθεί όλοι οι μαθηματικοί αλγόριθμοι που εφαρμόζονται πάνω στα πολυδιάστατα δεδομένα για την εξαγωγή περιβαλλοντικών δεικτών, αλγορίθμων και εφαρμογή масκών.
- **constants.py:** Ένα βοηθητικό αρχείο που συγκεντρώνει όλες τις σταθερές μεταβλητές του συστήματος.
- **ndi_request.py:** Ένα βοηθητικό αρχείο που περιέχει το Αντικείμενο Μεταφοράς Δεδομένων (DTO)
- **odc_query_experiment.py:** Το πείραμα του προηγούμενου κεφαλαίου.
- **Φάκελος utils/:** Περιέχει βοηθητικές συναρτήσεις και εργαλεία από το ODC
- **Φάκελος Geographic_data_maps/:** Αφιερωμένος φάκελος για τα γεωχωρικά αρχεία σε μορφή .geojson από όπου αντλούμε τις συντεταγμένες για συγκεκριμένες περιοχές.



Εικόνα 7.9 Αλλαγή εικόνας με τελικό αποτέλεσμα

7.7 Constants

Η διασπορά σταθερών τιμών σε πολλαπλά σημεία του εκτελέσιμου κώδικα δυσκολεύει σημαντικά τη μελλοντική διαχείριση και την αναγνωσιμότητα του συστήματος. Η κλάση constants λειτουργεί ως η μοναδική πηγή σταθερών για όλο το σύστημα ανάλυσης. Οποιαδήποτε αλλαγή σε παραμέτρους του συστήματος απαιτεί τροποποίηση μόνο σε αυτό το αρχείο, η οποία κληρονομείται αυτόματα σε όλα τα υπόλοιπα αρχεία (indexes.py, dataset_importer.py, data_manager.py). Τεχνικά οι κλάσεις περιέχουν δεδομένα όπως:

- **Συντελεστές κλιμάκωσης:** Τιμές όπως Scale και Offset καθώς κάθε dataset προϊόντος έχει διαφορετικές τιμές για την μετατροπή τους σε πραγματικές φυσικές τιμές
- **Αντιστοίχιση Προϊόντων και Καναλιών:** Διαφορετικές δορυφορικές αποστολές χρησιμοποιούν διαφορετική ονοματολογία για τα ίδια κανάλια. Για παράδειγμα το κόκκινο κανάλι στα προϊόντα Sentinel και Landsat είναι δηλωμένο ως “red” ενώ στα κανάλια HLS (L30 και S30) είναι ως B04.
 RED="red"
 LANDSAT_RED="red"
 HLS_L30_RED="B04"
 HLS_S30_RED="B04"
- **Στατικές μέθοδοι:** Στατικές (σταθερές) μέθοδοι οι οποίες δίνουν σταθερά δεδομένα ανάλογα το προϊόν. Για παράδειγμα για τα url των καταλόγων από τα οποία παίρνουμε τα δεδομένα μέσω stac.
- **Συστήματα αναφοράς Συντεταγμένων (CRS):** Δήλωση των σταθερών CRS
- **Κανόνες Mask:** Καθώς πρέπει να εφαρμόσουμε μάσκες στα dataset που θέλουμε να επεξεργαστούμε, με αποτέλεσμα να δηλώσουμε ως σταθερές τις μάσκες με τα bit από τους αλγόριθμους που θέλουμε να κρατήσουμε/διώξουμε.

```
class constants:

    #Scale for sentinel-12
    S2_SCALE = 0.0001
    #Scale for Landsat
    LANDSAT_SCALE = 0.0000275
    LANDSAT_OFFSET = -0.2
    #Scale for HLS
    HLS_SCALE = 0.0001

    #RESOLUTIONS
    RES_10=(-10, 10)
    RES_20=(-20,20)
    RES_30=(-30,30)

    #Directory of the geojson, this may change
    GEOS_DIR="/home/christossapounas/Projects/Hellas_Cube/analysis_service/Geographic_data_maps"

    #SENTINEL
    SENTINEL=["sentinel_2_12a"]
```

Εικόνα 7.10: Υπόδειξη αρχείου index

7.8 Χάρτες Γεωγραφικών Δεδομένων (Geographic Data Maps)

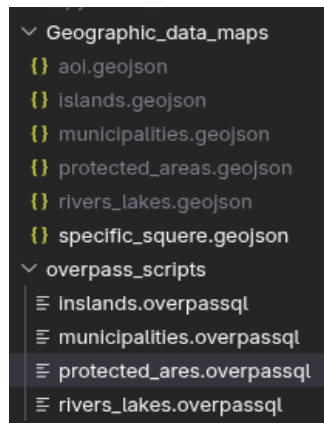
7.8.1. Τα αρχεία geojson

Σε αυτό το σημείο είναι σημαντικό να αναφέρουμε τον φάκελο ο οποίος αποτελεί το τοπικό αποθετήριο χωρικής πληροφορίας του υποσυστήματος και εμπεριέχει πολλαπλά αυτόνομα αρχεία τύπου geojson. Κάθε αρχείο του καταλόγου αναπαριστά τα ακριβή γεωχωρικά όρια (διανυσματικά πολύγωνα) πολλαπλών προκαθορισμένων περιοχών της Ελλάδας. Τα εξής αρχεία αναπαριστούν χωρικά δεδομένα γνωστών ελληνικών γεωγραφικών οντοτήτων:

- rivers_lakes.geojson: Λίμνες και Ποτάμια της Ελλάδος
- protected_areas.geojson: Προστατευόμενες περιοχές και Εθνικά πάρκα της Ελλάδος
- municipalities.geojson: Δήμοι της Ελλάδος
- islands.geojson: Νησιά της Ελλάδος

7.8.2. Ανάκτηση Γεωγραφικών Χαρτών δεδομένων

Για την δημιουργία των δεδομένων αξιοποιήθηκαν ελεύθερα δεδομένα από την πλατφόρμα του OpenStreetMap (OSM). Η άντληση των δεδομένων έγινε μέσω του Overpass Turbo [39] ενός εξειδικευμένου διαδικτυακού εργαλείου εξόρυξης δεδομένων του OSM. Όπως αποτυπώνεται και στην Εικόνα 7.12 η συλλογή πραγματοποιήθηκε με την συγγραφή κατάλληλων script στην γλώσσα ερωτημάτων Overpass QL. Όπως παρουσιάζεται στην Εικόνα 7.12, απεικονίζεται το σύνολο των λιμνών και των ποταμών της ελληνικής επικράτειας. Το τελικό γεωχωρικό αποτέλεσμα που προκύπτει από το αρχείο GeoJSON εμφανίζεται στον χάρτη στο δεξί τμήμα της εικόνας. Για την παραγωγή του, καθορίζονται προηγουμένως στο σχετικό σενάριο εντολών (Script) η μέθοδος εξαγωγής του αρχείου (JSON), η γεωγραφική περιοχή κάλυψης (Greece), καθώς και η κατηγορία των προς επεξεργασία δεδομένων (λίμνες και ποταμοί). Ακολούθως, τα δεδομένα αυτά αποθηκεύονται σε μορφότυπο GeoJSON, καθιστώντας τα διαθέσιμα προς περαιτέρω αξιοποίηση στο υποσύστημα.

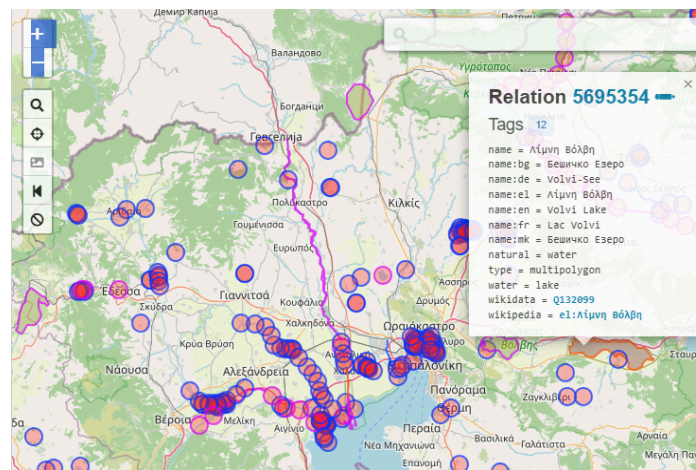


Εικόνα 7.11: Αρχεία OverpassQL και geojson

```

1  /*
2  This has been generated by the overpass-turbo wizard.
3  The original search was:
4  "Lake in Greece"
5  */
6  [out:json][timeout:25];
7  // fetch area "Greece" to search in
8  {{geocodeArea:Greece}}->.searchArea;
9  // gather results
10 (
11   way["natural"="water"]["water"="lake"](area.searchArea);
12   relation["natural"="water"]["water"="lake"](area.searchArea);
13   way["natural"="water"]["water"="river"](area.searchArea);
14   relation["natural"="water"]["water"="river"](area.searchArea);
15 );
16 // print results
17 out geom;

```



Εικόνα 7.12: Παράδειγμα εκτέλεσης overpass query

7.9 Η Βασική Κλάση (Parent Class)

Για την αποφυγή πλεονασμού κώδικα και την αρχικοποίησης αντικειμένων σχεδιάστηκε η θεμελιώδη κλάση `BaseODCSERVICE`. Η κλάση αυτή λειτουργεί ως η γονική οντότητα από την οποία κληρονομούν χαρακτηριστικά οι υπόλοιπες δομικές κλάσεις του υποσυστήματος (όπως `check_data`, `data_manager` `env_ind` που θα αναλυθούν στην συνέχεια). Ο κατασκευαστής (constructor) της `BaseODCSERVICE` εκτελεί δύο λειτουργίες:

1. Δημιουργία αντικειμένου `DataCube` το οποίο χρησιμοποιείται καθόλη τη ροή εκτέλεσης αιτήματος, και δημιουργείται μία φορά αν δεν έχει δημιουργηθεί ήδη.
2. Ενσωμάτωση σταθερών οι οποίοι χρησιμοποιούνται σε όλες τις κλάσεις που κληρονομούν την `Base` κλάση μας.

```

class BaseODCSERVICE:
    def __init__(self, dc: Datacube=None):
        #initialize Datacube as an object
        self.dc = dc if dc is not None else Datacube(app='Hellas_Cube')
        #initializing the constants to get them throughout
        self.const = constants

```

Εικόνα 7.13: Γονική κλάση `BaseODCSERVICE`

7.10 Dataset Importer

Ακολουθώντας την αρχιτεκτονική του υποσυστήματος ανάλυσης, υλοποιήθηκε η μονάδα `dataset_importer`. Το συγκεκριμένο αρχείο περιέχει κλάσεις που είναι υπεύθυνο για τον συστηματικό έλεγχο και την εκτέλεση των απαραίτητων λειτουργιών, διασφαλίζοντας την διαθεσιμότητα των indexes των dataset στη βάση.

7.10.1. Constructor

Η κλάση `check_data` αποτελεί υποκλάση της `BaseODCSERVICE`. Μέσω του μηχανισμού της κληρονομικότητας, αποκτά αυτόματα εσωτερική πρόσβαση στο αντικείμενο `Data Cube` και στις σταθερές του συστήματος (μέσω των μεταβλητών `self.dc` και `self.const` αντίστοιχα). Η κύρια προσθήκη στον δικό της κατασκευαστή είναι η αρχικοποίηση μιας δευτερεύουσας, εσωτερικής κλάσης με το όνομα `geo_searcher`. Κύριος σκοπός αυτής της δευτερεύουσας κλάσης είναι η μαζική ενοποίηση των αρχείων GeoJSON που έχουν προκύψει από την εξαγωγή μέσω του εργαλείου `Overpass Turbo`. ώστε η χωρική αναζήτηση των συντεταγμένων μιας περιοχής ενδιαφέροντος να

εκτελείται ταχύτατα σε ένα ενοποιημένο ευρετήριο.

```
#constructor, getting the dc object of datacube and also set
def __init__(self, dc: Datacube):
    super().__init__(dc)
    self.geosearch=geo_searcher(constants.GEOS_DIR)._gdf
    self.constants = self.const
```

Εικόνα 7.14: Κατασκευαστής κλάσης check_data()

7.10.2. Κύριος έλεγχος δεδομένων

Στην συνέχεια η μέθοδος που καλείται για τον έλεγχο και την αποθήκευση των δεδομένων είναι η συνάρτηση checking. Η μέθοδος πρέπει να κάνει κρίσιμες διεργασίες για την ανάκτηση μεταβλητών όπως, το σχήμα της περιοχής ενδιαφέροντος σε μαθηματική γεωμετρία (desired_aoi_geometry, Εικόνα 7.15) καθώς και ένα αντικείμενο κλάσης Geometry της βιβλιοθήκης odc.geo το οποίο συνδέει το πολύγωνο του γεωχώρου με το Σύστημα Αναφοράς Συντεταγμένων (CRS) (odc_geom Εικόνας 7.15). Αρχικά πρέπει να αναφερθεί ότι υπάρχουν δύο τρόποι με τους οποίους παίρνουμε τις δύο αυτές μεταβλητές. Με την απλή παροχή ενός ονόματος της περιοχής ενδιαφέροντος και την αναζήτηση των γεωχωρικών της συντεταγμένων στο συμπυκνωμένο αρχείο που δημιούργησε ο κατασκευαστής ή με την παροχή του ίδιου geojson το οποίο δίνεται από τον ίδιο τον χρήστη. Σε αυτόν τον έλεγχο μας βοηθούν οι συναρτήσεις get_by_geojson_of_aoi(place) και get_by_name_of_aoi(place) οι οποίες γυρίζουν ότι χρειαζόμαστε, δηλαδή το odc_geom και το desired_aoi_geometry (place).

```
#This method is reliable for the checking if the data exist in the databa
def checking(self,place, date1, date2, catalog, req_type):
    if (req_type=="DEFAULT"):
        #getting the geom and geometry by receiving a geojson
        odc_geom, desired_aoi_geometry= self.get_by_geojson_of_aoi(place)
    elif(req_type=="TARGET"):
        #other than that we need to get the geojson only by the name of it
        odc_geom, desired_aoi_geometry= self.get_by_name_of_aoi(place)
```

Εικόνα 7.15: Έλεγχος και ανάκτηση odc_geom & desired_aoi_geometry

Στην συνέχεια εφόσον έχουν ανακτηθεί τα κρίσιμα δεδομένα του γεωχώρου, του χρονικού διαστήματος των δεδομένων καθώς και τον κατάλογο ανάκτησης τους, μέσω της μεθόδου dc.find_dataset η οποία μας επιστρέφει τα dataset για τις μεταβλητές που παρέχουμε. Σε περίπτωση που το Data Cube δεν βρει dataset στο σύστημα τότε χρειάζεται να αναζητήσουμε από τις εξωτερικές πηγές για datasets. Για την ανάκτηση των απαιτούμενων συνόλων δεδομένων, η μέθοδος get_datasets διευκολύνει τη διαδικασία.

```
#searching for existing dataset
datasetsfound=self.dc.find_datasets(
    product=desired_collections,
    geopolygon=odc_geom,
    time=desired_date_range
)

#if we find no dataset, try to get them
if len(datasetsfound) == 0:
    odc_geom, desired_date_range, datasetsfound= self.get_datasets(desired_aoi_geometry,
                                                                    desired_date_range,
                                                                    catalog,
                                                                    desired_collections,
                                                                    odc_geom)
```

Εικόνα 7.16: Αναζήτηση και Ανάκτηση datasets

7.10.3. Ανάκτηση datasets (get_datasets())

Όπως προαναφέρθηκε στο κεφάλαιο 7.5 η ανακτηση των datasets γίνεται με την βοήθεια της εντολής `stac-to-dc` και εκτελείται εσωτερικά του συστήματος. Έτσι με την επαναχρησιμοποίηση του κώδικα από το πρώτο τεστ του κεφαλαίου 7.5 παίρνουμε τα datasets στην βάση και στην συνέχεια πάλι με την μέθοδο `dc.find_datasets`, εντοπίζονται και επιστρέφονται τα datasets.

7.11 Data Manager

Η κύρια αρμοδιότητα του αρχείου `data_manager.py` (και της αντίστοιχης κλάσης) είναι η φόρτωση των datasets στην κύρια μνήμη (RAM), με σκοπό τη δημιουργία ενός πολυδιάστατου αντικειμένου `xarray.Dataset`, το οποίο καθίσταται έτοιμο για την εφαρμογή масκών και τον υπολογισμό των περιβαλλοντικών δεικτών. Όπως και η `check_data`, η συγκεκριμένη κλάση κληρονομεί δομικά από τη `BaseODCService`, αντλώντας αυτόματα το αντικείμενο του Data Cube. Παράλληλα, κατά την αρχικοποίησή της, δέχεται ως εξάρτηση το αντικείμενο της `check_data`, ώστε να μπορεί να επικοινωνεί μαζί της και να επιβεβαιώνει τη διαθεσιμότητα των δεδομένων (μέσω του `dataset importer`) πριν προχωρήσει στη φόρτωσή τους. Συνοπτικά, η κλάση αναλαμβάνει με ασφάλεια τη μεταφορά των δεδομένων στο υπολογιστικό περιβάλλον και, σε μεταγενέστερο στάδιο, την εξαγωγή των τελικών στατιστικών στοιχείων.

7.11.1. Φόρτωση dataset sentinel

Για την φόρτωση των dataset ου sentinel έχει δημιουργηθεί μία ξεχωριστή συνάρτηση για τους λόγους ότι τα δεδομένα sentinel μας δίνουν την δυνατότητα να τα επεξεργαστούμε με την βοήθεια πολλών νημάτων. Η επιλογή της χρησιμοποίησης νημάτων βασίζεται στην μέγιστη αποδοτικότητα του συστήματος και στην διαχείριση κατανάλωσης διαθέσιμης μνήμης RAM. Αφού ολοκληρωθεί η κλήση της συνάρτησης `checking` (η οποία αναλύθηκε στην Ενότητα 7.9.2) και επιστραφούν το αντικείμενο γεωμετρίας `ode`, το χρονικό διάστημα ημερομηνιών καθώς και τα εντοπισμένα datasets, προστίθεται το κανάλι `SCL`, το συγκεκριμένο κανάλι αφορά τον αλγόριθμο μάσκας `SCL` ο οποίος θα εξεταστεί αναλυτικά στα επόμενα κεφάλαια. Στη συνέχεια, καλείται η `dc.load()` για τη φόρτωση των πινάκων `xarray` με την μόνη διαφορά την προσθήκη της μεταβλητής `dask_chunks`. Η μεταβλητή `dask_chunks` αποτελεί την πιο σημαντική της συνάρτησης. Ορίζοντας την τιμή `{"time": 1, "x": 1024, "y": 1024}` όπως εμφανίζεται στην Εικόνα 7.17, τα δεδομένα δεν φορτώνονται απευθείας στη μνήμη του συστήματος. Αντιθέτως εφαρμόζεται μία ετεροχρονισμένη φόρτωση μέσω της βιβλιοθήκης `Dask`. Η γεωγραφική έκταση χωρίζεται σε μικρότερες πλάκες διαστάσεων `1024x1024 pixel` (x και y). Αυτό επιτρέπει στο σύστημα να αναθέσει στον υπολογισμό κάθε πλάκας από pixel ένα διαφορετικό νήμα. Με την χρήση πολλαπλών νημάτων αυξάνεται η αποδοτικότητα του συστήματος και η ταχύτητα επεξεργασίας των δεδομένων.

```
#loading the dataset only for sentinel2 because sentinel dat
def load_s2(self, place: str, date1: str, date2: str, req_ty
    #Checking for data in the database and making sure they
    odc_geom, desired_dates, datasets = self.check.checking(

    #loading dataset
    meas=list(dict.fromkeys(measurements + ["scl"])) #addin
    ds=self.dc.load(
        product=product, #pro
        datasets=datasets, #dat
        geopolygon=odc_geom, #geo
        time=desired_dates, #dat
        output_crs=constants.CRS_GREECE, #the
        resolution=resolution, #res
        measurements=meas, #cha
        dask_chunks={"time": 1, "x": 1024, "y": 1024}, #das
        group_by="solar_day", #gro
    )
    return ds
```

Εικόνα 7.17: Φόρτωση datasets με την βοήθεια

7.11.2. Φόρτωση dataset για Landsat και HLS με περιβάλλον rasterio

Η συνάρτηση dc.load για τα δεδομένα Landsat και HLS έχει διαφορετική συμπεριφορά από ότι για του Sentinel. Η βασική διαφορά είναι ότι τα δεδομένα δεν υποστηρίζουν “τεμπέλικη” φόρτωση με αποτέλεσμα να φορτώνονται και να υπολογίζονται πιο αργά από ότι τα δεδομένα του Sentinel. Μία δεύτερη διαφορά είναι η χρήση περιβάλλοντος rasterio.

```
#Before calling this method, set enviroment variables
def load_dataset_with_env(self, place: str, date1: str, date2: str
    odc_geom, desired_dates, datasets = self.check.checking(place,
    with rasterio.Env():
        ds=self.dc.load(
```

Εικόνα 7.18: Μέθοδος για φόρτωση dataset με περιβάλλον

Η χρήση του περιβάλλοντος rasterio είναι απαραίτητη, καθώς χρησιμοποιείται για την αρχικοποίηση μεταβλητών. Οι μεταβλητές αυτές είναι καθοριστικές προκειμένου το DataCube να είναι σε θέση να ανακτήσει με ακρίβεια τα πραγματικά δεδομένα, βασιζόμενο στα αποθηκευμένα ευρετήρια. Στην Εικόνα 7.19 παρατηρούμε ότι οι μεταβλητές αποτελούν κυρίως μεταβλητές αυθεντικοποίησης για τις πολιτικές της AWS και NASA για τα Landsat και HLS δεδομένα αντίστοιχα. Η αρχικοποίηση έχει τρεις άξονες:

- 1. Αυθεντικοποίηση AWS:** απαραίτητα στοιχεία πρόσβασης για τον λογαριασμό AWS, καθορισμός της περιοχής άντλησης δεδομένων, ρύθμιση χρέωσης και υποχρεωτική επαλήθευση ταυτότητας.
- 2. Αυθεντικοποίηση NASA:** Αντίστοιχα λήψη στοιχείων αυθεντικοποίησης μέσω cookie file ο οποίος είναι ένα κρυφός φάκελος στο σύστημα (.netrc).
- 3. Ρυθμίσεις GDAL:** Γενικές ρυθμίσεις όπως ταυτόχρονη αποστολή αιτημάτων, συγχώνευση συνεχόμενων αιτημάτων ανάγνωσης και διαχείριση μνήμης cache.

```

#AWS AUTH
os.environ["AWS_ACCESS_KEY_ID"] = "" # AWS access key (empty = a
os.environ["AWS_SECRET_ACCESS_KEY"] = "" # AWS secret key (empty = a
os.environ["AWS_DEFAULT_REGION"] = "us-west-2" # region where the S3 bucke
os.environ["AWS_REQUEST_PAYER"] = "requester" #Just telling who pays (it'
os.environ["AWS_NO_SIGN_REQUEST"] = "NO" # don't skip auth signing (

#NASA AUTH
os.environ["GDAL_HTTP_COOKIEFILE"] = "/tmp/gdal_cookies.txt" # where GDAL reads coo
os.environ["GDAL_HTTP_COOKIEJAR"] = "/tmp/gdal_cookies.txt" # where GDAL writes/st

#general settings
os.environ["GDAL_HTTP_MERGE_CONSECUTIVE_RANGES"] = "YES" # merge S3 range requests
os.environ["GDAL_HTTP_MULTIPLEX"] = "YES" # HTTP/2 multiplexing
os.environ["GDAL_DISABLE_READDIR_ON_OPEN"] = "EMPTY_DIR" # skip directory listing
os.environ["VSI_CACHE"] = "TRUE" # cache S3 reads
os.environ["VSI_CACHE_SIZE"] = "10000000" # 10MB per file cache
os.environ["CPL_VSIL_CURL_CACHE_SIZE"] = "200000000" # 200MB global cache

```

Εικόνα 7.19: Αρχείο αρχικοποίησης μεταβλητών περιβάλλοντος

Με την ολοκλήρωση της αρχικοποίησης του περιβάλλοντος, η συνάρτηση `dc.load()` είναι πλέον διαθέσιμη για τη φόρτωση των datasets, καθιστώντας τα έτοιμα προς περαιτέρω επεξεργασία και ανάλυση.

7.11.3. Στατιστικά αποτελεσμάτων

Οι πολυδιάστατοι πίνακες (`xarray.Dataarray`), οι οποίοι περιέχουν εκατοντάδες χιλιάδες ή εκατομμύρια τιμές pixel, πρέπει να μετατραπούν σε διαχειρίσιμα, κατανοητά δεδομένα. Για τον σκοπό αναπτύχθηκε η συνάρτηση `stats()`, η οποία εξάγει ένα σύνολο στατιστικών από τα αποτελέσματα της επεξεργασίας των dataset. Η συνάρτηση εκδίδει στατιστικά σε δύο επίπεδα:

1. **Έγκυρα Pixel:** Για την μέγιστη ενημέρωση του χρήστη πρέπει να τον ενημερώνουμε ότι από το dataset το οποίο επεξεργάστηκε το σύστημα, πόσα pixel είναι έγκυρα και τι ποσοστό από τα συνολικά ήταν έγκυρα. Στην Εικόνα 7.20 παρουσιάζονται ότι οι μεταβλητές είναι τα `valid_px` για τα έγκυρα pixel, `total_pixel` τα συνολικά pixel και η `coverage` το ποσοστό κάλυψης έγκυρων pixel.
2. **Εξαγωγή Πλήρη Στατιστικών:** Εφόσον υπάρχουν έγκυρα pixel, εξάγεται ένα σύνολο στατιστικών (αντί απλώς ενός μέσου όρου) για την καλύτερη επιστημονική αξιοπιστία. Τα στατιστικά τα οποία που επιλέχθηκαν για το dataset δημιουργούνται με την βοήθεια έτοιμων συναρτήσεων της βιβλιοθήκης `xarray` και είναι (με επιπλέον εφαρμογή στρογγυλοποίησης και παράκαμψης των κενών):
 - `mean` (Μέσος όρος): Μέση τιμή των δεδομένων του περιβαλλοντικού δείκτη
 - `median` (Διάμεσος): Η μεσαία τιμή όταν τα δεδομένα ταξινομηθούν σε αύξουσα σειρά. (καλύτερος μέσος όρος)
 - `min & max` (ελάχιστο και μέγιστο) : Τα όρια τιμών του dataset
 - `std` (Τυπική Απόκλιση): Απόκλιση τιμών από τον μέσο όρο. Υψηλές τιμές αποτυπώνουν διασπορά ενώ μικρή τιμή ομοιομορφία.
 - Εκατοστημόρια (`p10`, `p25`, `p75`, `p90`): Οι τιμές κάτω από την οποία βρίσκεται το αντίστοιχο ποσοστό (10%, 25%, 50%, 75% και 90%) του συνόλου των εγκύρων pixels.

```

    }
    return {
        "index":      index_name,
        "mean":       round(float(da.mean(skipna=True).values),3),
        "median":     round(float(da.median(skipna=True).values),3),
        "min":        round(float(da.min(skipna=True).values),3),
        "max":        round(float(da.max(skipna=True).values),3),
        "std":        round(float(da.std(skipna=True).values),3),
        "p10":        round(float(da.quantile(0.10, skipna=True).values),3),
        "p25":        round(float(da.quantile(0.25, skipna=True).values),3),
        "p75":        round(float(da.quantile(0.75, skipna=True).values),3),
        "p90":        round(float(da.quantile(0.90, skipna=True).values),3),
        "valid_px":   valid_px,
        "total_px":   total_px,
        "coverage_pct": f"{coverage} %"
    }

```

Εικόνα 7.20: Επιστρεφόμενα στατιστικά της stats()

7.12 Ο Επιστημονικός Πυρήνας: Η κλάση env_id

Η κλάση env_ind (Environmental Indexes) αποτελεί τον κύριο επιστημονικό και υπολογιστικό πυρήνα του συστήματος. Είναι η κλάση η οποία χρησιμοποιεί όλες τις παραπάνω μεθόδους που αναλύθηκαν στα προηγούμενα κεφάλαια με τελικό σκοπό την εφαρμογή των τύπων πάνω στα dataset για την εξαγωγή των τελικών στατιστικών.

7.12.1. Αρχικοποίηση

Αξιοποιώντας την Base κλάση, ο constructor της env_id κληρονομεί την BaseODCService (όπως και οι κλάσεις των προηγούμενων κεφαλαίων) παίρνοντας τις απαραίτητες πληροφορίες για το αντικείμενο του DataCube και τις σταθερές. Πέραν της κληρονόμησης η __init__(self) δημιουργεί αντικείμενα για την check_data και data_manager για την αξιοποίηση τους στην διαδικασία υπολογισμού των δεικτών.

7.12.2. Μέθοδοι masking

Η διαχείριση ποιότητας των pixels (αφαίρεση νέφων, σκιών κλ.π) διαφοροποιείται ριζικά ανάλογα με την δορυφορική αποστολή και τον αλγόριθμο της. Οι συναρτήσεις για τις μάσκες χωρίζονται ως εξής:

1. **Sentinel-2 (SCL Masking):** Η αποστολή παρέχει το κανάλι SCL, το οποίο λαμβάνει ακέραιες τιμές, όπως απεικονίζεται στον Πίνακα 7.21. Αξιοποιώντας τη συνάρτηση .isin(mask) της βιβλιοθήκης xarray (Εικόνα 7.22), αποδίδεται η λογική τιμή False στα pixels που εμπίπτουν στη μάσκα. Ως τελικό αποτέλεσμα, επιστρέφεται το φιλτραρισμένο σύνολο δεδομένων (masked dataset).

Πίνακας 7.21: Αποτελέσματα αλγορίθμου SCL

Label	Classification
0	NO_DATA
1	SATURATED_OR_DEFECTIVE
2	DARK_AREA_PIXELS
3	CLOUD_SHADOWS
4	VEGETATION
5	NOT_VEGETATED
6	WATER
7	UNCLASSIFIED
8	CLOUD_MEDIUM_PROBABILITY
9	CLOUD_HIGH_PROBABILITY
10	THIN_CIRRUS
11	SNOW

Figure 3: Scene Classification Values

2. **Landsat(QA_PIXEL):** Όσον αφορά τα προϊόντα Landsat, η διαδικασία masking υλοποιείται μέσω του αλγορίθμου QA_PIXEL (Πίνακας 7.22) σε συνδυασμό με τις βοηθητικές βιβλιοθήκες (utilities) του Open Data Cube. Συγκεκριμένα, αξιοποιείται η εξειδικευμένη συνάρτηση `landsat_qa_clean_mask()` (Εικόνα 7.23), η οποία, δεχόμενη ως ορίσματα τις ονομαστικές παραμέτρους των pixels από το αρχείο των σταθερών (constants), προβαίνει στην εκκαθάριση και επιστρέφει το τελικό, επεξεργασμένο dataset.

Πίνακας 7.22: Πίνακας για το QA_PIXEL του Landsat 8 και 9

Landsat Collection 1 LandsatLook 8-bit Quality Images Designations						
Landsat 8 OLI/TIRS		Landsat 7 ETM+, Landsat 4-5 TM		Landsat 1-5 MSS		Color
Bit	Description	Bit	Description	Bit	Description	
0	Designated Fill	0	Designated Fill	0	Designated Fill	
1	Terrain Occlusion	1	Dropped Pixel	1	Dropped Pixel	
2	Radiometric Saturation	2	Radiometric Saturation	2	Radiometric Saturation	
3	Cloud	3	Cloud	3	Cloud	
4	Cloud Shadow	4	Cloud Shadow	4	Unused	
5	Snow/Ice	5	Snow/Ice	5	Unused	
6	Cirrus	6	Unused	6	Unused	
7	Unused	7	Unused	7	Unused	Unused

```
def mask_landsat(self, ds, cover_types, platform="LANDSAT_8", collection='c2', level='l2'):
    return landsat_qa_clean_mask(
        ds,
        platform=platform,
        cover_types=cover_types,
        collection=collection,
        level=level
    )
```

Εικόνα 7.23: Εφαρμογή `landsat_qa_clean_mask` του `datacube.utils`

3. **HLS (Fmask):** Για το mask των HLS προϊόντων (Sentinel και Landsat) χρησιμοποιούμε τον αλγόριθμο `fmask`. Ο αλγόριθμος `fmask` έχει την ιδιαιτερότητα να κρατάει πολλαπλά pixel ανοιχτά ταυτόχρονα. Για παράδειγμα για pixel με τιμή `Fmask=00000011` τα bit 1 και 2 είναι ανοιχτά, το οποίο έχει ως αποτέλεσμα το pixel να είναι σύννεφο και να είναι δίπλα σε σύννεφο (Πίνακας 7.24). Έτσι στην συνάρτηση μετατρέπουμε τα bit που δεν θέλουμε σε θέσεις bit και στην συνέχεια εφαρμόζουμε την μάσκα bit που δημιουργήσαμε στο dataset και

επιστρέφεται (Εικόνα 7.25).

Πίνακας 7.24: bit μάσκας fmask

Bit Number	Mask Name	Bit Value	Mask Description
0	Cirrus	Reserved, but not used in HLS V2.0	N/A
1	Cloud	1	Yes
		0	No
2	Adjacent to cloud/shadow	1	Yes
		0	No
3	Cloud shadow	1	Yes
		0	No
4	Snow/ice	1	Yes
		0	No
5	Water	1	Yes
		0	No
6-7	AOT level	00	Climatology aerosol
		10	Low aerosol
		01	Medium aerosol
		11	High aerosol

```
def fmask(self, ds, exclude_bits):
    #Convert the list of bit positions
    bit_mask = 0
    for bit in exclude_bits:
        bit_mask |= (1 << bit)

    # Extract the Fmask array and ensure it is an integer
    fmask = ds[self.const.HLS_L30_FMASK].astype("int16")

    # Apply bitwise AND
    valid_pixels = (fmask & bit_mask) == 0

    return valid_pixels
```

Εικόνα 7.25: Συνάρτηση εφαρμογής μάσκας αλγορίθμου fmask

7.12.3. Υπολογισμός Δεικτών: Το παράδειγμα του NDVI

Όλοι οι τυπικοί περιβαλλοντικοί δείκτες (όπως ο NDWI για το νερό ή ο NBR για καμένες εκτάσεις) ακολουθούν την ίδια ακριβώς αρχιτεκτονική εκτέλεσης. Επομένως, ο δείκτης NDVI παρουσιάζεται ως αντιπροσωπευτικό παράδειγμα για την ροή του κώδικα. Οι μέθοδοι των δεικτών χωρίζονται σε τρεις κλάδους ροής, αντιμετωπίζοντας τις ιδιαιτερότητες κάθε δορυφόρου. Αρχικά η κάθε μέθοδος δείκτη παίρνει ως παραμέτρους την περιοχή ενδιαφέροντος (aoi), το διάστημα ημερομηνιών, τον client (για την πολλαπλή χρήση νημάτων κατά των υπολογισμό Sentinel Dataset) τον τύπο αίτησης (TARGET ή DEFAULT για τον προσδιορισμό του τύπου του aoi), καθώς και την πηγή (source), η οποία προσδιορίζει την αποστολή από την οποία αντλούνται τα προς επεξεργασία δεδομένα. Μετά από κάθε υπολογισμό Data Array καλώντας την συνάρτηση stats απο τον data_manager . Είναι σημαντικό να σημειωθεί ότι για τον δείκτη NDCI λόγω ότι στα προϊόντα Landsat και HLS L30 δεν υπάρχει το κανάλι του κόκκινου άκρου (red edge), με αποτέλεσμα να υπάρχουν μόνο οι δυνατότητες του Sentinel και HLS S30.

7.12.3.1 Εφαρμογή Sentinel

Στην περίπτωση του Sentinel το Dataset φορτώνεται μέσω της συνάρτησης load_s2 από την κλάση data_manager που αναλύθηκε σε προηγούμενο κεφάλαιο. Αφού φορτωθεί εφαρμόζεται η ειδική

μάσκα για τον SCL αλγόριθμο του Sentinel-2. Μετά την εφαρμογή μάσκας το dataset είναι έτοιμο για την εφαρμογή του δείκτη, όπου στην Εικόνα 7.26 παίρνουμε τα χρώματα (εγγύς ερυθρό και ερυθρό) του δείκτη (NDVI) και αφού τα έχουμε πολλαπλασιάσει με τον συντελεστή κλιμάκωσης και έχουμε πάρει τα θετικά δεδομένα με την βοήθεια του client υπολογίζουμε το τελικό dataset. Ο υπολογισμός του dataset στην περίπτωση του sentinel γίνεται στο τέλος λόγω της “lazy” φόρτωσης και της τεχνικής πολλαπλών νημάτων που εφαρμόζουμε.

```
#NDVI(NORMALIZED DIFFERENCE VEGETATION INDEX)
def ndvi(self, place, date1, date2, client, req_type, source):

    #SENTINEL
    if source == "sentinel":
        #load the dataset
        ds=self.data_manager.load_s2( place,
                                     date1,
                                     date2,
                                     req_type,
                                     [self.const.NIR, self.const.RED],
                                     self.const.RES_10,
                                     self.const.SENTINEL)

        #check if data are empty
        if len(ds.time) == 0:
            return {"error": "no_data"}

        #Mask the data
        mask=self.mask_scl(ds, self.const.STRICT_MASK)

        #load first color
        nir=(ds[self.const.NIR].astype("float32") * self.const.S2_SCALE).where(mask).where(lambda x: x > 0)

        #load second color
        red=(ds[self.const.RED].astype("float32") * self.const.S2_SCALE).where(mask).where(lambda x: x > 0)

        #apply index
        index=((nir - red) / (nir + red)).clip(-1, 1)

        #compute the median, computing with multiple threads
        index = client.compute(index, sync=True)
```

Εικόνα 7.26: Περίπτωση εφαρμογής περιβαλλοντικού δείκτη σε δεδομένα sentinel

7.12.3.2 Εφαρμογή Landsat

Ο υπολογισμός του ίδιου δείκτη αλλά με δεδομένα από Landsat έχει μικρές διαφορές με τον Sentinel. Αρχικά το dataset φορτώνεται με την συνάρτηση του data_manager, ειδική για δεδομένα Landsat. Μετά την εφαρμογή της μάσκας, οι κατάλληλες παράμετροι εφαρμόζονται στο dataset και υπολογίζεται ο περιβαλλοντικός τύπος. Όπως έχει προαναφερθεί τα δεδομένα του Landsat δεν υποστηρίζουν χρήση νημάτων μέσω dask, έτσι ο υπολογισμός γίνεται μέσω της μεθόδου compute() της xarray βιβλιοθήκης.

```
# LANDSAT (Native)
elif source == "landsat":
    ds = self.data_manager.load_dataset_with_env(
        place=place, date1=date1, date2=date2, req_type=req_type,
        measurements=[self.const.LANDSAT_NIR, self.const.LANDSAT_RED, self.const.LANDSAT_QA_PIXEL],
        resolution=self.const.RES_30,
        product= self.const.LANDSAT8
    )

    if ds is None or len(ds.time) == 0:
        return {"error": "no_data"}

    # Mask using the QA_PIXEL algorythm
    mask = self.mask_landsat(ds, self.const.LANDSAT_STRICT_MASK)

    # Apply Landsat specific scale and offset
    nir = ((ds[self.const.LANDSAT_NIR].astype("float32") * self.const.LANDSAT_SCALE) +
            self.const.LANDSAT_OFFSET).where(mask).where(lambda x: x > 0)
    red = ((ds[self.const.LANDSAT_RED].astype("float32") * self.const.LANDSAT_SCALE) +
            self.const.LANDSAT_OFFSET).where(mask).where(lambda x: x > 0)

    index = ((nir - red) / (nir + red)).clip(-1, 1)
    index = index.compute()
```

Εικόνα 7.27: Περίπτωση ανάλυσης δεδομένων Landsat

7.12.3.3 Εφαρμογή HLS

Στην περίπτωση του HLS υπάρχει μία ριζική αλλαγή υπολογισμού. Καθώς τα προϊόντα είναι δύο για Landsat (HLS L30) και Sentinel (S30) πρέπει ο τύπος να εφαρμοστεί και στα δύο προϊόντα και στην συνέχεια να συμπτυχθούν μεταξύ τους για να υπολογιστούν. Έτσι για κάθε προϊόν L30 και S30 διαχωρίζουμε τα κανάλια με τα σωστά ονόματα και τα φορτώνουμε στην RAM. Αφού εφαρμοστεί ο περιβαλλοντικός τύπος, τα dataset μπαίνουν σε μία λίστα. Όταν τελειώσουν και τα δυο προϊόντα τότε με την χρήση της μεθόδου concat (της βιβλιοθήκης xarray) μας δίνεται η δυνατότητα να συμπτύξουμε τα δύο αποτελέσματα σε ένα και να τα υπολογίσουμε (Εικόνα 7.28). Στην περίπτωση του NDCI υπολογίζεται μόνο το HLS S30 ακριβώς όπως ο απλός Landsat.

```
# HLS (L30 + S30)
elif source == "hls":
    index_arrays = []

    for product in (self.const.HLS_L + self.const.HLS_S):
        if product in self.const.HLS_L:
            nir_band, red_band, fmask_band = self.const.HLS_L30_NIR, self.const.HLS_L30_RED, self.const.HLS
        else:
            nir_band, red_band, fmask_band = self.const.HLS_S30_NIR, self.const.HLS_S30_RED, self.const.HLS

        ds = self.data_manager.load_dataset_with_env(
            place=place, date1=date1, date2=date2, req_type=req_type,
            measurements=[nir_band, red_band, fmask_band],
            resolution=self.const.RES_30, product=[product]
        )

        if ds is None or len(ds.time) == 0:
            continue

        # Mask using the Fmask algorythm
        mask = self.fmask(ds, self.const.F_STRICT_MASK)

        nir = (ds[nir_band].astype("float32") * self.const.HLS_SCALE).where(mask).where(lambda x: x > 0)
        red = (ds[red_band].astype("float32") * self.const.HLS_SCALE).where(mask).where(lambda x: x > 0)

        product_index = ((nir - red) / (nir + red)).clip(-1, 1)
        index_arrays.append(product_index)

    if not index_arrays:
        return {"error": "no_data"}

    # Combine the arrays and compute
    index = xr.concat(index_arrays, dim="time")
    index = index.compute()
```

Εικόνα 7.28: Περίπτωση ανάλυσης δεδομένων HLSΥπολογισμός πλημμύρας/ξηρασίας μέσω

WOFS

Στο πλαίσιο του Hellas Cube, αναπτύχθηκε μία ξεχωριστή συνάρτηση με τη οποία παρακολουθούμε πλημμύρες και ξηρασία. Η προσέγγιση της συνάρτησης με την βοήθεια του αλγορίθμου WOFS από τα utilities του Data Cube είναι πιο σύνθετη από έναν δείκτη όπως το NDVI λόγω ότι ο αλγόριθμος εντοπίζει με μεγαλύτερη ακρίβεια την παρουσία νερού. Η συνάρτηση floods_wofs ενορχηστρώνει την πολύπλοκη διαδικασία και χωρίζεται σε διαφορετικά στάδια.

7.12.3.4 Προετοιμασία δεδομένων

Αρχικά, το σύστημα φορτώνει έξι διαφορετικά κανάλια (Red, Green, Blue, Nir, SWIR16, SWIR22) από τον δορυφόρο Landsat και τα μεταφέρει απευθείας στην κύρια μνήμη με τον υπολογισμό του. Σε αντίθεση με τους δείκτες που δουλεύουν με τιμές διαστήματος (0,-1) ο αλγόριθμος WOFS απαιτεί τα δεδομένα σε ακέραια μορφή 16-bit (με κλίμακα 0-10000). Για τον λόγο αυτό τα δεδομένα πολλαπλασιάζονται με το 10000 (Εικόνα 7.29). Το masking στο σημείο αυτό γίνεται αρκετά πιο αυστηρά λόγω ευαισθησίας του αλγορίθμου, με αποτέλεσμα να γίνεται ένας συνδυασμός αυστηρής μάσκας και μάσκας σύννεφων καθώς και μιας μάσκας απώλειας δεδομένων.

```
#define surface reflectance bands
sr_bands = [self.const.LANDSAT_RED, self.const.LANDSAT_GREEN, self.const.LANDSAT_BLUE, self.const.LANDSAT_NIR, self.const.LANDSAT_SWIR16, self.const.LANDSAT_SWIR22]

# generate cloud and quality masks
cloud_mask = self.mask_landsat(ds, cover_types=['clear', 'water'], platform="LANDSAT_8", collection='c2')

# strict mask reusing the same QA_PIXEL bit definitions as every other index (dilated cloud, cirrus, cloud shadow)
strict_mask = self.mask_landsat(ds, cover_types=self.const.LANDSAT_STRICT_MASK, platform="LANDSAT_8")

#create no-data mask
nodata_mask = (ds[sr_bands] != 0).to_array(dim='band').all(dim='band')

#apply scaling factors to bands
int_scale = int(10000)
for band in sr_bands:
    ds[band] = ((ds[band] * self.const.LANDSAT_SCALE + self.const.LANDSAT_OFFSET) * int_scale).clip(0, 10000)

#combine all masks
combined_mask = cloud_mask & nodata_mask & strict_mask
```

Εικόνα 7.29: Προετοιμασία δεδομένων και εφαρμογή επιπρόσθετων μαस्कών για τον αλγόριθμο WOFS

7.12.3.5 WOFS και σκηνές

Μετά την εφαρμογή μαस्कών εκτελείται η συνάρτηση wofs_classify(), επιστρέφοντας έναν νέο πολυδιάστατο πίνακα (wofs_stack), όπου κάθε pixel έχει πάρει την τιμή 1 (Νερό) ή 0 (Ξηρά). Στην συνέχεια, ο κώδικας δεν εξάγει απλώς έναν μέσο όρο, αλλά πραγματοποιεί μια αναλυτική χρονολογική διάγνωση. Στην περίπτωση που μια χρονική λήψη διαθέτει λιγότερα από 100 έγκυρα pixels, ο αλγόριθμος την απορρίπτει. Η επιλογή του ορίου των 100 pixels αντιστοιχεί στο 50% του συνόλου των pixels για χωρική ανάλυση 30×30 (300 pixels).. Για κάθε λήψη αν περνάει το στάδιο των έγκυρων pixel υπολογίζεται το ποσοστό νερού, χαρακτηρίζοντας το ποσοστό πάνω από 25% έως υγιές. (Εικόνα 7.30)

```

scenes = []
for i in range(len(wofs_stack.time)):
    scene = wofs_stack.isel(time=i)
    date = str(wofs_stack.time.isel(time=i).values)[:10]
    clear_water = int((scene == 1).sum().item())
    clear_not_water = int((scene == 0).sum().item())
    total_clear = clear_water + clear_not_water

    #skip if scene is too cloudy
    if total_clear < 150:
        scenes.append({"date": date, "status": "too_cloudy", "continue": True})
        continue

    #compute water percentage for scene
    water_pct = round((clear_water / total_clear) * 100, 2)
    scenes.append({
        "date": date,
        "water_pct": water_pct,
        "clear_px": total_clear,
        "status": "dry" if water_pct < 25 else "healthy"
    })

```

Εικόνα 7.30: Ανάλυση σκηνών του dataset

7.12.3.6 Εντοπισμός Πλημμύρας/Ξηρασίας

Το ισχυρότερο χαρακτηριστικό της μεθόδου είναι η ικανότητα της να εξάγει αυτοματοποιημένα συμπεράσματα για την περιβαλλοντική κατάσταση, συγκρίνοντας ιστορικά δεδομένα με πρόσφατες παρατηρήσεις:

- Ο αλγόριθμος συγκρίνει την πιο πρόσφατη λήψη με το συνολικό μέσο όρο των προηγούμενων και σε περίπτωση μεγάλης πτώσης (30%) τότε το σύστημα αυτομάτως σημαίνει την περιοχή ως ξηρασία. Αν ο συνολικός λόγος των ξηρών λήψεων υπερβεί το 50%, η περιοχή κρίνεται ως αποξηραμένη. (Εικόνα 7.31)

```

#calculate dry ratio
dry_ratio = len(dry_scenes) / len(valid_scenes) if valid_scenes else 0

#calculate historic vs recent drying trend
N = min(5, len(valid_scenes))
recent = valid_scenes[-N:]
recent_avg = sum(s["water_pct"] for s in recent) / len(recent) if recent else None
historic_avg = sum(s["water_pct"] for s in valid_scenes) / len(valid_scenes) if valid_scenes else None

if recent_avg is not None and historic_avg is not None and historic_avg > 0:
    drop_pct = (historic_avg - recent_avg) / historic_avg
else:
    drop_pct = 0

#determine conclusion status
if dry_ratio >= 0.5:
    conclusion = "dried"
elif drop_pct >= 0.3:
    conclusion = "drying_trend"
else:
    conclusion = "healthy"

```

Εικόνα 7.31: Λογική εντοπισμού ξηρασίας σε υδάτινο σώμα

- Για κάθε μεμονωμένο pixel υπολογίζεται η συχνότητα εμφάνισης νερού στο πέρασμα του χρόνου και μέσω της συνάρτησης pct_pixels_above() (Εικόνα 7.32) κατηγοριοποιούμε τα δεδομένα σε μόνιμο νερό (>95%), επίμονο νερό (>80%), εποχιακό νερό (>50%), και περιστασιακό νερό (>25%).

```

#define helper for threshold calculation
def pct_pixels_above(threshold_pct):
    if n_valid_px == 0:
        return "0.00%"
    hits = int(((freq >= threshold_pct / 100) & valid_px).sum().item())
    return f"{hits / n_valid_px * 100:.2f}%"

```

Εικόνα 7.32: Κατηγοριοποίηση δεδομένων

7.13 API

Το τελικό δομικό στοιχείο του υποσυστήματος Python αφορά την έκθεση των επιστημονικών λειτουργιών στο κεντρικό σύστημα του Hellas Cube. Για τον σκοπό αυτό, υλοποιήθηκε ένα ασύγχρονο RESRful API βασισμένο στο framework Fast API το οποίο εκτελείται μέσω του διακομιστή Unicorn. Η επιλογή του FastAPI εξασφαλίζει υψηλές ταχύτητες απόκρισης, αυτόματη τεκμηρίων και εγγενή υποστήριξη ασύγχρονου προγραμματισμού.

7.13.1. Εκκίνηση του εσωτερικού API

Η βάση της λειτουργίας στο αρχείο [main.py](#) βρίσκεται στο τέλος όπως ξεκινάει με την μέθοδο που ενεργοποιεί τον έναν ασύγχρονο διακομιστή Unicorn (Asynchronous Server Gateway Interface). με την κλήση της Εικόνας 7.33 εξασφαλίζει ότι το API ακούει όλες τις διαθέσιμες διευθύνσεις του δικτύου (0.0.0.0) στην θύρα 8080.

```
#running the server
def main():
    uvicorn.run("main:app", host="0.0.0.0", port=8080, reload=True)
```

Εικόνα 7.33: Εκκίνηση server

Στην Εικόνα 7.33 παρατηρείται ότι το πρώτο όρισμα είναι το “main:app”, το οποίο καθορίζει το Fast API instance που δημιουργείται με την δημιουργία του Server. Μετά την δημιουργία αντικειμένου app(Fast API Εικόνα 7.34) τα endpoints δημιουργούνται με την βοήθεια μίας κλάσης δυναμικά, έτσι ώστε η μελλοντική πρόσθεση νέων σημείων να είναι ευκολότερη.

```
analyzation = indexes.env_ind()

#Constructing a router with a dask
router = IndexRouter(app, analyzation)
router.register("/analyzation/ndvi", "NDVI", "ndvi")
router.register("/analyzation/ndci", "NDCI", "ndci")
router.register("/analyzation/ndti", "NDTI", "ndti")
router.register("/analyzation/ndwi", "NDWI", "ndwi")
router.register("/analyzation/ndmi", "NDMI", "ndmi")
router.register("/analyzation/ndbi", "NDBI", "ndbi")
router.register("/analyzation/ndsi", "NDSI", "ndsi")

# Landsat data do not use the Dask client
no_duck_router = IndexRouter(app, analyzation, needs_dask=False)
no_duck_router.register("/analyzation/wofs", "WOFS", "flood_wofs")
```

Εικόνα 7.34: Δήλωση endpoints εσωτερικού API

Επιπλέον παίρνει μία παράμετρος lifespan η οποία καθορίζει την συμπεριφορά του API καθόλη την διάρκεια ζωής του API. Η χρησιμότητα του lifespan της Εικόνα 7.35 είναι για την εργατών. Οι εργάτες, όπως παρατηρούμε στην Εικόνα 7.36, που δημιουργούνται μέσω του LocalCluster ενεργοποιούνται μέσω της κλάσης Client του Dask Client περιέχουν νήματα καλύπτοντας έως και 4Gb μνήμης. Όταν κλείνει το σύστημα οι “εργάτες” απελευθερώνονται.

```
app = FastAPI(title="HellasCube", version="0.0.1", lifespan=lifespan)
```

Εικόνα 7.35: Δήλωση API μέσω FastAPI

```
@asynccontextmanager
async def lifespan(app: FastAPI):

    # TODO: security issue (use enviromental variables)
    #setting the aws credentials
    set_env_vars()

    #cluster workers for mutliple threads now we use 8
    cluster = LocalCluster(n_workers=4,
                           threads_per_worker=2,
                           #memory limit per worker, f
                           memory_limit="4GB")

    #From Client Documendation: The Client connects
    client = Client(cluster)

    #setting the dask client with the client we just c
    app.state.dask_client = client
    print(f"Dask dashboard: {client.dashboard_link}")
```

Εικόνα 7.36: Συνάρτηση lifespan για δημιουργία εργατών

7.13.2. Δυναμική Δημιουργία Endpoints

Όπως έχει ήδη επισημανθεί, η συγκρότηση των τελικών σημείων (endpoints) του API πραγματοποιείται μέσω ενός δυναμικού μηχανισμού. Η εξειδικευμένη συνάρτηση `register` δέχεται ως παραμέτρους τη διαδρομή (path), την κατηγορία της ανάλυσης και το όνομα της αντίστοιχης μεθόδου προς εκτέλεση. Αξιοποιώντας την ενσωματωμένη συνάρτηση `getattr()`, το σύστημα ανακτά τη λειτουργικότητα κάθε συνάρτησης βάσει της ονομαστικής της αναφοράς, βελτιστοποιώντας τη διαχείριση των κλήσεων σε κάθε POST αίτημα. Ο διαχειριστής δημιουργίας κάθε κλήση δέχεται ως όρισμα ένα json και το μετατρέπει κατευθείαν σε αντικείμενο του DTO που προαναφέρθηκε (Εικόνα 7.37).

```
#method we are going to execute (name of the method)
method = getattr(self.analyzation, method_name)

#asynchronous function
async def handler(req: dto, request: Request):
```

Εικόνα 7.37: Αντληση ονομάτων μεθόδων και handler δημιουργίας endpoint

Η τελική δρομολόγηση διαφοροποιείται ανάλογα με τη φύση του αιτήματος (DEFAULT ή TARGET), δηλαδή εάν η ανάλυση βασίζεται σε υφιστάμενο γεωχωρικό αρχείο (geojson) ή εάν απαιτείται η δυναμική παροχή του από τις υποδομές του συστήματος. Τα arguments κάθε συνάρτησης διαχωρίζονται σε αυτές οι οποίες χρειάζονται `dask_client` και αυτές που όχι (Εικόνα 7.38) (μόνο η `wofs` δεν χρειάζεται λόγω απόλυτης λειτουργίας με δεδομένα Landsat).

```
#threading calling the dask client, we are going to use multiple threads with dask client
dask_client = request.app.state.dask_client if self.needs_dask else None

#calling the methods with the arguments required
args = (req.place, req.date1, req.date2, dask_client, req.req_type, req.source) \
        if self.needs_dask else (req.place, req.date1, req.date2, req.req_type)

result = method(*args)
```

Εικόνα 7.38: Δήλωση ορισμάτων κάθε μεθόδου του endpoint

Στο τέλος εκτέλεσης κάθε συνάρτησης επιστρέφουμε το αποτέλεσμα της μεθόδου από το `env_ind` ενσωματώνεται σε απάντηση json μορφής και επιστρέφεται έτσι ώστε να δοθεί ως αποτέλεσμα στο

API το οποίο το καταχωρεί και το επιστρέφει στον χρήστη.

7.14 Παράδειγμα τελικού αποτελέσματος

Το τελικό αποτέλεσμα για ένα ερώτημα TARGET στην Λιμνοθάλασσα Κλείσοβας για δεδομένα ενός μήνα για το προϊόν δεδομένων sentinel είναι τα εξής:

```
{
  "STATUS": "OK",
  "analyzation": "NDVI",
  "place": "Λιμνοθάλασσα Κλείσοβας",
  "result": {
    "index": "NDVI",
    "mean": -0.131,
    "median": -0.193,
    "min": -0.994,
    "max": 0.986,
    "std": 0.374,
    "p10": -0.57,
    "p25": -0.403,
    "p75": 0.157,
    "p90": 0.385,
    "valid_px": 2810657,
    "total_px": 2886555,
    "coverage_pct": "97.37 %"
  },
  "time": "20.09s"
}
```

Η μέση τιμή του δείκτη NDVI υπολογίστηκε σε -0.131. Το αποτέλεσμα αυτό είναι συμβατό με την αναμενόμενη συμπεριφορά, δεδομένου ότι η Λιμνοθάλασσα Κλείσοβας αποτελεί υδάτινο σώμα, το οποίο, βάσει του Πίνακα 3.7, αποδίδει αποκλειστικά αρνητικές τιμές.

7.15 Αξιολόγηση

7.15.1. Αξιολόγηση με Μετρικές

Προκειμένου να εκτιμηθεί η πραγματική συμπεριφορά του συστήματος, θα τρέξουμε μερικά τεστ με πραγματικά σενάρια και θα μετρήσουμε τη χρήση σε CPU και RAM. Σκοπός είναι να καταγράψουμε πώς συμπεριφέρεται όταν διαχειρίζεται μεγάλο όγκο δορυφορικών δεδομένων, εξετάζοντας δύο συγκεκριμένα σενάρια (ένα με μεγαλύτερο γεωχώρο από το άλλο) για να δούμε τους χρόνους απόκρισης ανάλογα με τον φόρτο του server.

7.15.1.1 Σενάριο A: Μικρός Γεωχώρος (Δήμος Θεσσαλονίκης)

Το πρώτο σενάριο αφορά την εξαγωγή του NDVI για τα γεωγραφικά όρια του Δήμου Θεσσαλονίκης. Με βάση τα αποτελέσματα που εξήγαγε το σύστημα (Εικόνα 7.38) και την καταγραφή των πόρων του συστήματος (Εικόνα 7.39), εξάγονται τα εξής σημαντικά συμπεράσματα:

- **Χρόνος Εκτέλεσης και Επεξεργασία Δεδομένων:** Το σύστημα ολοκλήρωσε την ανάλυση σε μόλις 15.04 δευτερόλεπτα (Εικόνα 7.39). Σε αυτό το σύντομο χρονικό διάστημα, το Hellas

Cube κατάφερε να επεξεργαστεί 4.032.539 συνολικά pixels, εκ των οποίων τα 3.633.498 αξιολογήθηκαν ως έγκυρα (valid_px), επιτυγχάνοντας ποσοστό κάλυψης 90.1% (Εικόνα 7.39)

- **Χρήση Επεξεργαστή (CPU):** Στο γράφημα παρακολούθησης (Εικόνα 7.40), παρατηρείται μια ταυτόχρονη, κατακόρυφη αύξηση της δραστηριότητας και στους 4 πυρήνες του συστήματος (από περίπου 25% σε επίπεδα μεταξύ 50% και 75%) κατά τη διάρκεια των ~15 δευτερολέπτων της ανάλυσης. Αυτό επιβεβαιώνει την επιτυχή εφαρμογή της τεχνικής πολλαπλών νημάτων (multithreading). Όπως σχεδιάστηκε στην αρχιτεκτονική του συστήματος, τα δεδομένα Sentinel επιτρέπουν την παράλληλη επεξεργασία, εκμεταλλευόμενα πλήρως την υπολογιστική ισχύ του διακομιστή.
- **Χρήση Μνήμης (RAM):** Το σημαντικότερο ίσως τεχνικό επίτευγμα αποτυπώνεται στη γραμμή της μνήμης (Εικόνα 7.40), η οποία παραμένει απόλυτα σταθερή στα 9.0 GB (53.6%), χωρίς καμία απότομη κορύφωση. Αντί το σύστημα να φορτώσει ολόκληρη την εικόνα των 4+ εκατομμυρίων pixels στη μνήμη (γεγονός που θα οδηγούσε σε εξάντληση των πόρων), η "τεμπέλικη" φόρτωση της βιβλιοθήκης Dask τμηματοποίησε τα δεδομένα σε μικρότερα chunks διαστάσεων 1024x1024. Αυτό διατηρεί το αποτύπωμα μνήμης σταθερό, καθιστώντας το Hellas Cube εξαιρετικά ανθεκτικό σε αιτήματα μεγάλου όγκου (Big Data).

```

1  {
2    "STATUS": "OK",
3    "analyzation": "NDVI",
4    "place": "Municipality of Thessaloniki",
5    "result": {
6      "index": "NDVI",
7      "mean": 0.104,
8      "median": 0.072,
9      "min": -0.98,
10     "max": 0.999,
11     "std": 0.264,
12     "p10": -0.197,
13     "p25": -0.129,
14     "p75": 0.281,
15     "p90": 0.511,
16     "valid_px": 3633498,
17     "total_px": 4032539,
18     "coverage_pct": "90.1 %"
19   },
20   "time": "15.04s"
21 }
```

Εικόνα 7.39: Αποτελέσματα Σεναρίου Α



Εικόνα 7.40: Χρήση CPU και RAM κατά το πρώτο σενάριο

7.15.1.2 Σενάριο B: Μεγαλύτερος Γεωχώρος (Δήμος Πολυγύρου)

Το δεύτερο σενάριο εστιάζει στην αξιολόγηση της συμπεριφοράς του συστήματος υπό μεγαλύτερο υπολογιστικό φόρτο, εξετάζοντας την ικανότητα της αρχιτεκτονικής να διαχειριστεί μια μεγαλύτερη γεωγραφική περιοχή. Βάσει του παραγόμενου αρχείου αποτελεσμάτων (Εικόνα 7.41) και της παρακολούθησης πόρων κατά την ενδιάμεση (Εικόνα 7.42) και τελική φάση (Εικόνα 7.43) της εκτέλεσης, εξάγονται σημαντικά συμπεράσματα για τη συμπεριφορά του λογισμικού:

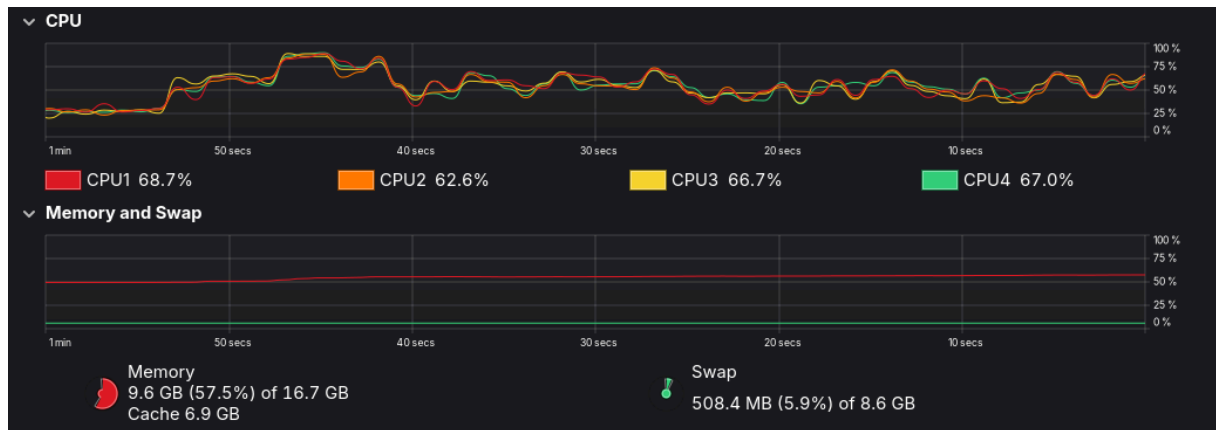
- **Όγκος Δεδομένων:** Το σύστημα κλήθηκε να επεξεργαστεί έναν εξαιρετικά μεγάλο όγκο δεδομένων, αγγίζοντας τα 238.420.104 συνολικά pixels (Εικόνα 7.41). Αυτός ο όγκος είναι σχεδόν 60 φορές μεγαλύτερος από το Σενάριο A. Ο χρόνος απόκρισης ανήλθε στα 126.85 δευτερόλεπτα, με τα έγκυρα pixels (μετά την εφαρμογή της μάσκας SCL) να ανέρχονται σε 32.137.508, επιτυγχάνοντας ποσοστό κάλυψης 13.48% (μη-έμπιστο αποτέλεσμα) (Εικόνα 7.41).
- **Συμπεριφορά Επεξεργαστή:** Κατά την ενδιάμεση φάση της ανάλυσης, η CPU παρουσίασε σταθερά υψηλό φόρτο, με τους τέσσερις πυρήνες να δουλεύουν παράλληλα και αποδοτικά μεταξύ 62.6% και 68.7% (Εικόνα 7.42). Αντίθετα, στην τελική φάση της εκτέλεσης, η εικόνα άλλαξε αισθητά: η απόδοση των πυρήνων έπεσε σε επίπεδα μεταξύ 51.0% και 63.6% (Εικόνα 7.43), παρουσιάζοντας έντονες διακυμάνσεις και αποδεικνύοντας ότι ο επεξεργαστής δεν μπορούσε πλέον να λειτουργήσει στο μέγιστο.
- **Εξάντληση Μνήμης (RAM) με Αντίκτυπο στην Απόδοση:** Η ασταθής συμπεριφορά του επεξεργαστή στο τέλος της εκτέλεσης εξηγείται πλήρως από τη διαχείριση της μνήμης. Παρόλο που το σύστημα δεν φορτώνει ολόκληρη την εικόνα κατευθείαν (χρησιμοποιώντας chunks), ο τεράστιος αριθμός αυτών των τμημάτων οδήγησε τη φυσική μνήμη (16.7 GB) στα όριά της κατά τη φάση συγκέντρωσης των τελικών στατιστικών. Ενώ ενδιάμεσα η RAM βρισκόταν στα 9.6 GB με το Swap στο ελάχιστο (508.4 MB), το λειτουργικό σύστημα αναγκάστηκε τελικά να δεσμεύσει 2.3 GB (26.3%) εικονικής μνήμης (Εικόνα 7.43). Αυτή ακριβώς η ανάγκη για μεταφορά δεδομένων στον πιο αργό σκληρό δίσκο προκάλεσε τις καθυστερήσεις στα τελευταία 20 δευτερόλεπτα, αναγκάζοντας τους πυρήνες της CPU να περιμένουν την ανάγνωση και εγγραφή των δεδομένων.

```

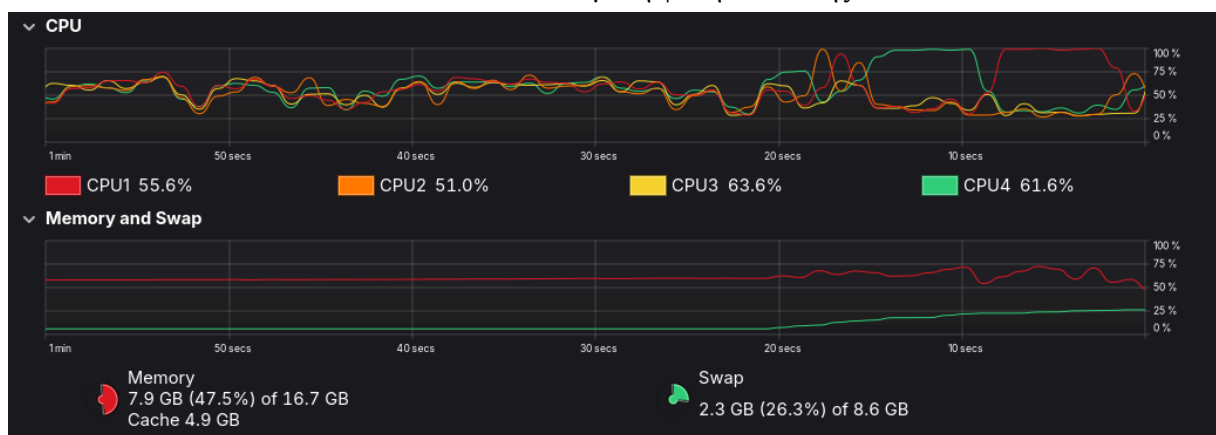
1  {
2      "STATUS": "OK",
3      "analyzation": "NDVI",
4      "place": "Polygyros Municipality",
5      "result": {
6          "index": "NDVI",
7          "mean": 0.553,
8          "median": 0.554,
9          "min": -0.987,
10         "max": 0.999,
11         "std": 0.248,
12         "p10": 0.222,
13         "p25": 0.325,
14         "p75": 0.796,
15         "p90": 0.875,
16         "valid_px": 32137508,
17         "total_px": 238420104,
18         "coverage_pct": "13.48 %"
19     },
20     "time": "126.85s"
21 }

```

Εικόνα 7.41: Αποτελέσματα Σεναρίου B



Εικόνα 7.42: Ενδιάμεση φάση εκτέλεσης



Εικόνα 7.43: Τελική φάση εκτέλεσης

7.15.2. Συμπεράσματα

Όπως επιβεβαιώθηκε και από τα σενάρια αξιολόγησης (Α και Β), η ανάπτυξη του υποσυστήματος ανάλυσης ανέδειξε σημαντικούς αρχιτεκτονικούς συμβιβασμούς αναφορικά με τη διαχείριση των υπολογιστικών πόρων και την ταχύτητα εκτέλεσης. Αφενός, η χρήση της βιβλιοθήκης Dask στα δεδομένα Sentinel επιτρέπει την παραλληλοποιημένη και lazy φόρτωση, διατηρώντας το αποτύπωμα μνήμης σταθερό και αποτρέποντας την εξάντληση της RAM (Σενάριο Α). Αφετέρου, αναλύσεις σε τεράστιους γεωχώρους αυξάνουν δραματικά τις απαιτήσεις σε κύρια μνήμη, οδηγώντας αναπόφευκτα σε χρήση εικονικής μνήμης (Swap) και καθυστερήσεις στην CPU (Σενάριο Β). Παράλληλα, η στρατηγική απόφαση να ενθυλακωθεί όλη η επιστημονική λογική (υπολογισμοί `chartay`, εφαρμογή τύπων ,αλγόριθμος WOfS) πίσω από ένα εσωτερικό API προσφέρει επιπλέον ταχύτητα απόκρισης, καθώς εξαλείφει τις καθυστερήσεις αρχικοποίησης σε κάθε νέο αίτημα. Εντούτοις, η αξιοπιστία αυτού του πυρήνα ανάλυσης καθιστά αναγκαία την ύπαρξη ενός εξίσου στιβαρού μηχανισμού διαχείρισης και ασφαλούς καταγραφής των αποτελεσμάτων. Αυτός ο ρόλος ανατίθεται στη σχεσιακή βάση δεδομένων του συστήματος.

Κεφάλαιο 8ο: Η Βάση Δεδομένων του Hellas Cube

8.1 RDBMS και Περιβάλλον Ανάπτυξης Βάσης

Η διαχείριση των χρηστών, η καταγραφή των αιτημάτων και η αποθήκευση των αποτελεσμάτων του συστήματος Hellas Cube απαιτούν μια απαραίτητη υποδομή. Για τον σκοπό αυτό, υλοποιήθηκε μια βάση δεδομένων με το σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων (RDBMS) PostgreSQL, η οποία εξασφαλίζει τη μόνιμη αποθήκευση και την ακεραιότητα των συνδέσεων του συστήματος. Ο σχεδιασμός της υποστηρίζει την ταυτόχρονη εξυπηρέτηση πολλαπλών αιτημάτων από το API, διασφαλίζοντας την ομαλή λειτουργία ακόμη και υπό συνθήκες υψηλού φόρτου. Παράλληλα, παρέχει την απαραίτητη ευελιξία για την ασφαλή διαχείριση τόσο των κρυπτογραφημένων διαπιστευτηρίων όσο και των προσαρμοσμένων γεωχωρικών δεδομένων (GeoJSON) των χρηστών. Για την μέγιστη αποτελεσματικότητα, τα ερωτήματα δημιουργίας της βάσης δεδομένων και των πινάκων της εκτελέστηκαν μέσω της εφαρμογής DBeaver, ενός ελεύθερου λογισμικού ανοιχτού κώδικα αφιερωμένου στην διαχείριση βάσεων δεδομένων. Το DBeaver υποστηρίζει ένα ευρύ φάσμα συστημάτων διαχείρισης βάσεων δεδομένων μέσω ενός ενιαίου γραφικού περιβάλλοντος. Το ενσωματωμένο περιβάλλον συγγραφής SQL (SQL Editor) και η οπτικοποίηση της δομής της βάσης δεδομένων μέσω διαγραμμάτων Entity-Relationship (ER diagrams), τα οποία δημιουργούνται αυτόματα με βάση τις σχέσεις (foreign keys) μεταξύ των πινάκων προσέφεραν σημαντικά πλεονεκτήματα κατά την φάση σχεδίασης και υλοποίησης της βάσης.

8.2 Δομή Πινάκων

Η αρχιτεκτονική της βάσης στηρίζεται στο σχεσιακό μοντέλο, όπου κυριαρχούν οι σχέσεις πρωτεύοντος και ξένου κλειδιού (Primary / Foreign Keys). Για την υλοποίηση του σχήματος, επιλέχθηκαν σύγχρονοι τύποι δεδομένων που βελτιστοποιούν την απόδοση του συστήματος.

8.2.1. Πίνακας user

Για τις ανάγκες διαχείρισης και ασφαλούς αποθήκευσης των λογαριασμών του συστήματος, υλοποιήθηκε ο κεντρικός πίνακας users, ο οποίος αποτελεί τον πυρήνα ταυτοποίησης των χρηστών. Όπως αποτυπώνεται στο διάγραμμα σχεσιακής διασύνδεσης (Εικόνα 8.1), ένα από τα πλέον κρίσιμα πεδία του πίνακα είναι το email, το οποίο δέχεται έναν εξειδικευμένο τύπο δεδομένων προσαρμοσμένο αποκλειστικά για ηλεκτρονικές διευθύνσεις. Ο συγκεκριμένος μηχανισμός επιβάλλει αυστηρό έλεγχο εγκυρότητας (Εικόνα 8.1 DOMAIN email) απευθείας στο επίπεδο της βάσης δεδομένων, αποκλείοντας έτσι οποιαδήποτε απόπειρα καταχώρησης μη έγκυρων μορφών. Παράλληλα, το user_id δηλώνεται ως Primary Key και αξιοποιώντας τον τύπο δεδομένων uuid, ένα πρότυπο αναγνωριστικό μήκους 128-bit, το οποίο παράγεται αλγοριθμικά, εξασφαλίζεται πρακτικά αμελητέα πιθανότητα σύγκρουσης UUID.

```
CREATE TABLE public.users (
    user_id uuid NOT NULL,
    "password" varchar NOT NULL,
    declared_geo_json json NULL,
    "email" public."email" NOT NULL,
    CONSTRAINT user_email_key UNIQUE (email),
    CONSTRAINT user_email_not_null NOT NULL email,
    CONSTRAINT user_id_pk PRIMARY KEY (user_id),
    CONSTRAINT user_password_not_null NOT NULL password,
    CONSTRAINT user_user_id_not_null NOT NULL user_id
);

CREATE DOMAIN public."email" AS character varying(255)
COLLATE "default"
CONSTRAINT email_check CHECK (VALUE::text ~* '^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$'::text);
```

Εικόνα 8.1: Πίνακας user και έλεγχος μορφής email

8.2.2. Πίνακας request_log_file

Η παρακολούθηση κάθε ανάλυσης εξασφαλίζονται με απόλυτη ακρίβεια μέσω του πίνακα request_log_file, ο οποίος λειτουργεί ως η κεντρική οντότητα καταγραφής του συστήματος. Δεδομένου ότι η επεξεργασία ογκωδών γεωχωρικών δεδομένων αποτελεί συχνά μια χρονοβόρα διαδικασία, η παρακολούθηση του κύκλου ζωής κάθε αιτήματος είναι αναγκαία. Για τον λόγο αυτό, κάθε νέο αίτημα που καταφθάνει στο σύστημα λαμβάνει αυτόματα ένα μοναδικό αναγνωριστικό request_id, το οποίο ορίζεται ρητά ως Primary Key. Η εξέλιξη της διεργασίας παρακολουθείται δυναμικά μέσω του πεδίου status, στο οποίο καταγράφεται κάθε μεταβολή της κατάστασης του αιτήματος. Παράλληλα, διατηρείται απόλυτος έλεγχος μέσω ακριβών χρονοσημάνσεων (timestamps), αποτυπώνοντας τόσο την ακριβή στιγμή της αρχικής υποδοχής του αιτήματος (req_timestamp), όσο και την ώρα της τελευταίας ενημέρωσης της κατάστασής του (status_timestamp). Αυτή η δομή επιτρέπει τη διαφανή και ασφαλή διαχείριση της ροής, αποτρέποντας την απώλεια πληροφορίας σε περίπτωση σφάλματος.

```
CREATE TABLE public.request_log_file (
    request_id uuid NOT NULL,
    req_timestamp timestampz DEFAULT now() NOT NULL,
    status varchar NOT NULL,
    status_timestamp timestampz DEFAULT now() NOT NULL,
```

Εικόνα 8.2: Πεδία πίνακα καταγραφής αιτημάτων

8.2.3. Πίνακες user_results και general_results

Ο πίνακας αποτελεσμάτων χρηστών (user_results) είναι ο πίνακας που είναι αφιερωμένος στους χρήστες του συστήματος και συνδέεται με τον πίνακα users μέσω του ξένου κλειδιού του user_id, καθώς και με τον πίνακα request_log_file. Η διπλή αυτή σχεσιακή σύνδεση διασφαλίζει σύνδεση όπου: κάθε καταχωρημένο αποτέλεσμα δεν ανήκει απλώς σε έναν συγκεκριμένο χρήστη, αλλά ταυτοποιείται χρονικά και εκτελεστικά με το ακριβές αίτημα που το παρήγαγε. Στους δύο αυτούς πίνακες αποθηκεύονται τα στοιχεία της απάντησης του κάθε αιτήματος. Για την στήλη res_json επιλέχθηκε ο τύπος δεδομένων json για την καλύτερη διαχείριση των JSON που αποθηκεύονται. Λαμβάνοντας υπόψη ότι οι διαφορετικοί περιβαλλοντικοί δείκτες και αλγόριθμοι (όπως NDVI, NDCI, ή WOfS) ενδέχεται να παράγουν ετερογενείς δομές πληροφορίας ή διαφορετικά σείτ μετρήσεων, η χρήση JSON προσφέρει τεράστια ευελιξία. Επιτρέπει την ενιαία αποθήκευση των πολλαπλών στατιστικών μεγεθών σε μία μόνο στήλη, εξαλείφοντας την ανάγκη για ένα άκαμπτο, εκτενές σχήμα με δεκάδες κενές στήλες. Παράλληλα, η χρήση του εξειδικευμένου τύπου daterange της PostgreSQL βελτιστοποιεί την αναπαράσταση των χρονικών παραθύρων των δορυφορικών αναλύσεων. Αντί να

χρησιμοποιούνται δύο ξεχωριστές στήλες (έναρξης και λήξης), το `daterange` επιτρέπει την εγγενή διαχείριση της περιόδου ως ενιαίας οντότητας, καθιστώντας ταχύτερες τις χωροχρονικές αναζητήσεις. Ακριβώς παρόμοια δομή ακολουθεί και ο πίνακας των γενικών αποτελεσμάτων (`general_results`), ο οποίος εξυπηρετεί τα αποτελέσματα αναλύσεων που έγιναν από μη χρήστες. Η βασική του διαφορά είναι ότι ο γενικός πίνακας αποθηκεύει το όνομα της περιοχής (`area_name`) αντί για το αναγνωριστικό κάποιου χρήστη. Αυτή η δομή είναι στρατηγικής σημασίας για την αρχιτεκτονική μας, καθώς εξυπηρετεί τα αιτήματα τύπου `TARGET` που αφορούν καθορισμένες περιοχές, επιτρέποντας στο σύστημα να πραγματοποιεί έναν έλεγχο τύπου `cache` προκειμένου να ελέγξει αν υπάρχει ήδη καταχωρημένη η απάντηση στη βάση δεδομένων. Σε περίπτωση που τα δεδομένα υπάρχουν, το σύστημα ανασύρει τα ήδη υπολογισμένα στατιστικά και τα επιστρέφει ακαριαία στον πελάτη, παρακάμπτοντας εντελώς την κοστοβόρα και χρονοβόρα κλήση προς το υποσύστημα.

```
CREATE TABLE public.general_results (
    res_id uuid NOT NULL,
    analysis varchar NOT NULL,
    area_name varchar NOT NULL,
    res_json json NULL,
    date_range daterange NOT NULL,
    request_id uuid NULL,
    CONSTRAINT g_res_analysis_not_null NOT NULL analysis,
    CONSTRAINT g_res_area_id_not_null NOT NULL area_name,
    CONSTRAINT g_res_date_range_not_null NOT NULL date_range,
    CONSTRAINT g_res_pk PRIMARY KEY (res_id),
    CONSTRAINT g_res_res_id_not_null NOT NULL res_id,
    CONSTRAINT general_results_request_log_file_fk FOREIGN KEY (request_id) REFERENCES public.request_log_file(request_id);
);

CREATE TABLE public.user_results (
    res_id uuid NOT NULL,
    analysis varchar NOT NULL,
    date_range daterange NOT NULL,
    res_json json NULL,
    request_id uuid NOT NULL,
    user_id uuid NULL,
    CONSTRAINT default_res_analysis_not_null NOT NULL analysis,
    CONSTRAINT default_res_date_range_not_null NOT NULL date_range,
    CONSTRAINT default_res_pk PRIMARY KEY (res_id),
    CONSTRAINT default_res_res_id_not_null NOT NULL res_id,
    CONSTRAINT user_results_request_id_not_null NOT NULL request_id,
    CONSTRAINT user_results_request_log_file_fk FOREIGN KEY (request_id) REFERENCES public.request_log_file(request_id),
    CONSTRAINT user_results_users_fk FOREIGN KEY (user_id) REFERENCES public.users(user_id);
);
```

Εικόνα 8.3: Πίνακες αποθήκευσης αποτελεσμάτων

8.3 API keys

Η διαχείριση της ασφαλούς πρόσβασης και της ταυτοποίησης των χρηστών στο API του συστήματος αποτελεί θεμελιώδη άξονα της αρχιτεκτονικής του και βασίζεται οργανικά στον πίνακα `api_k`. Προκειμένου να διασφαλιστεί η μέγιστη αποδοτικότητα και να ελαχιστοποιηθεί ο υπολογιστικός φόρτος στο επίπεδο της εφαρμογής, ολόκληρος ο κύκλος ζωής των διαπιστευτηρίων διαχειρίζεται πλήρως και αυτοματοποιημένα μέσω ενσωματωμένων διεργασιών απευθείας στο επίπεδο της βάσης δεδομένων.

8.3.1. Πίνακας

Προκειμένου να επιτευχθεί υψηλός βαθμός αυτοματοποίησης στη διαδικασία απόδοσης κλειδιού και ταυτόχρονα να αποσυμφορηθεί το επίπεδο της εφαρμογής (`backend`) από περιττή προγραμματιστική λογική, υιοθετήθηκε μια στρατηγική αρχιτεκτονική επιλογή: η ενσωμάτωση ενός μηχανισμού `Trigger`

απευθείας στο επίπεδο του Συστήματος Διαχείρισης Βάσης Δεδομένων. Η προσέγγιση αυτή μεταφέρει το βάρος της αρχικοποίησης των δεδομένων στον πυρήνα της βάσης, εξασφαλίζοντας ταχύτερη εκτέλεση και διασφαλίζοντας την ακεραιότητα των συνδέσεων. Ειδικότερα, μόλις πραγματοποιηθεί μια επιτυχής εγγραφή (INSERT) στον κεντρικό πίνακα users, το σύστημα πυροδοτεί αυτόματα την εκτέλεση της αποθηκευμένης συνάρτησης create_api_key_slot() (Εικόνα 8.4). Η συγκεκριμένη διεργασία λειτουργεί προληπτικά, δεσμεύοντας ακαριαία μια κενή εγγραφή (slot) στον πίνακα api_k, η οποία δια συνδέεται άμεσα με τον νέο εγγραφέντα χρήστη μέσω της σχέσης ξένου κλειδιού (user_id). Η δεσμευμένη αυτή εγγραφή παραμένει σε κατάσταση αναμονής, απολύτως έτοιμη να ενημερωθεί δυναμικά (μέσω εντολής UPDATE) με το πραγματικό, μοναδικό κλειδί διεπαφής (API key) και την αυστηρά καθορισμένη ημερομηνία λήξης του (exp_date), κατά την πρώτη επιτυχή προσπάθεια σύνδεσης (login) του χρήστη.

```
CREATE TABLE public.api_k (  
    api_key varchar NOT NULL,  
    exp_date timestampz DEFAULT now() + '24:00:00'::interval NOT NULL,  
    user_id varchar NOT NULL,  
    CONSTRAINT api_k_pk PRIMARY KEY (api_key),  
    CONSTRAINT owner_id_fk FOREIGN KEY (user_id) REFERENCES public.users(user_id)  
);  
  
--trigger to insert a slot in the api_k table for every user  
CREATE OR REPLACE FUNCTION public.create_api_key_slot()  
    RETURNS trigger  
    LANGUAGE plpgsql  
AS $function$  
BEGIN  
    -- Now we can insert a truly blank slot  
    INSERT INTO public.api_k (user_id, api_key, exp_date)  
    VALUES (NEW.user_id, NULL, NULL);  
  
    RETURN NEW;  
END;  
$function$  
;  
  
CREATE TRIGGER trg_create_api_key_slot  
AFTER INSERT ON public.users  
FOR EACH ROW  
EXECUTE FUNCTION public.create_api_key_slot();
```

Εικόνα 8.4: Πίνακας api_k, συνάρτηση και Trigger της συνάρτησης

8.3.2. Αυτόματη Διαχείριση

Ιδιαίτερη έμφαση δόθηκε στην ασφάλεια και τον κύκλο ζωής των API keys, τα οποία αποθηκεύονται στον πίνακα api_k. Η διαχείριση γίνεται μέσω μιας επέκτασης της PostgreSQL, το pg_cron ο οποίος αναλαμβάνει να εκτελέσει SQL κώδικα σε συγκεκριμένη χρονική στιγμή που έχουμε ορίσει εμείς. Ο cron scheduler αποτελεί ένα καθιερωμένο εργαλείο στα συστήματα τύπου Unix και η επέκταση pg_cron να ενσωματώνει την λογική του απευθείας στην PostgreSQL, δηλαδή με το pg_cron μπορούμε να ορίσουμε κάθε πότε εκτελούνται SQL queries π.χ. κάθε 2 ώρες ενημερώνεται ένας πίνακας. Με αυτήν την λογική οι δύο αυτοματοποιημένες ενέργειες οι οποίες γίνονται στην βάση Hellas Cube είναι (Εικόνες 8.5 & 8.6):

1. clear-expired-keys (0 * * * *): Το cron expression εδώ σημαίνει ότι τρέχει κάθε ώρα, σαρώνει τον πίνακα και απενεργοποιεί το κλειδί, θέτοντας ως false την is_active και μηδενίζοντας την exp_date, αν έχει περάσει το χρονικό όριο των 8 ωρών.

2. `sweep_expired-keys(0 1 * * *)`: Το expression τρέχει κάθε βράδυ στη 1:00 το βράδυ ακριβώς. Αυτό είναι το delete των API keys αφού προφανώς έχουν απενεργοποιηθεί.

123 jobid	AZ schedule	AZ command
2	0 * * * *	UPDATE api_k SET is_active = false, exp_date = NULL WHERE exp_date
7	12 0 * * *	DELETE FROM api_k WHERE is_active=false

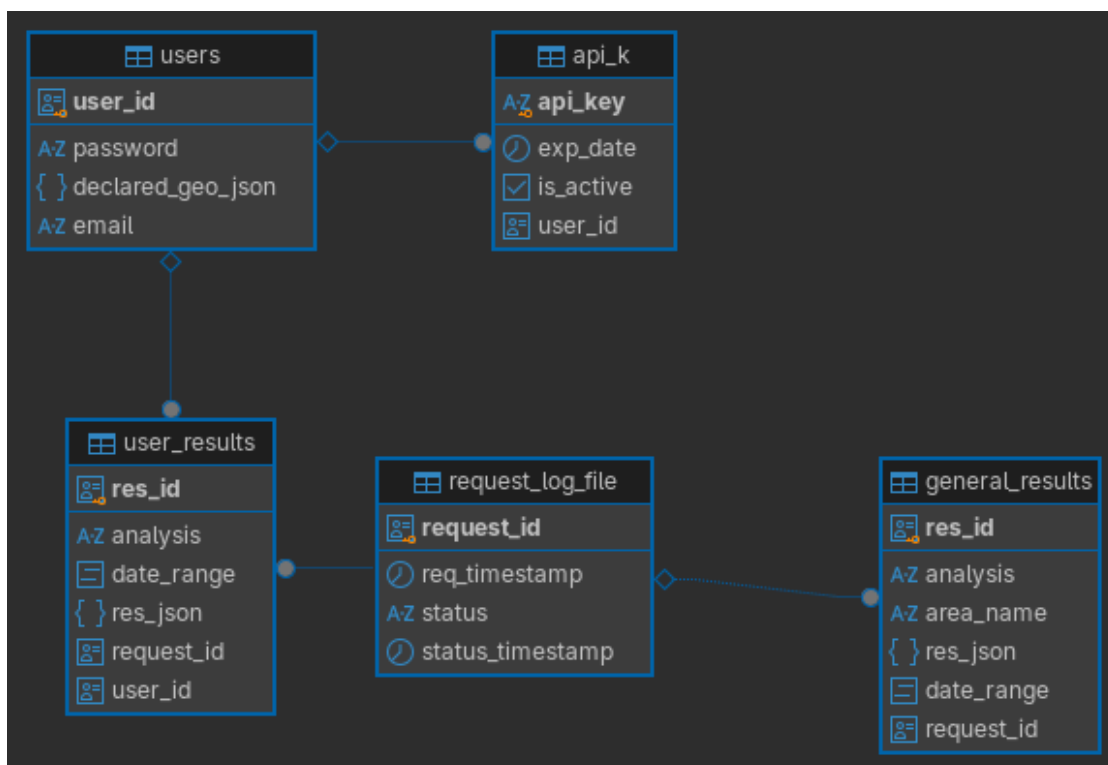
Εικόνα 8.5: Πίνακας χρονοπρογραμματισμένων διεργασιών του pgcron (Query και ώρα επανάληψης)

AZ nodename	123 nodeport	AZ database	AZ username	☒ active	AZ jobname
localhost	5,432	hellas_cube	chsap	[v]	clear-expired-
localhost	5,432	hellas_cube	chsap	[v]	sweep-expire-

Εικόνα 8.6: Πίνακας χρονοπρογραμματισμένων διεργασιών του pgcron (Επιπλέον Πληροφορίες)

8.4 Τελικό Schema

Όπως αποτυπώνεται στην Εικόνα 8.7, το τελικό σχεσιακό σχήμα (schema) αναδεικνύει τον κεντρικό ρόλο των ξένων κλειδιών και τη ροή της πληροφορίας μεταξύ των υποσυστημάτων, επιβεβαιώνοντας τη θεωρητική σχεδίαση στην πράξη. Πιο συγκεκριμένα, παρατηρούμε τους πίνακες με τα αποτελέσματα (`user_results` και `general_results`) να διασυνδέονται με τον πίνακα των αιτημάτων (`request_log_file`) με άμεση σύνδεσή τους το αναγνωριστικό του αιτήματος (`request_id`). Η διασύνδεση των αποτελεσμάτων με τον πίνακα αιτημάτων εξασφαλίζει ιχνηλασιμότητα στο σύστημα. Παράλληλα, η στρατηγική απουσία του `user_id` από τον πίνακα `general_results` όπου διατηρείται αποκλειστικά το όνομα της περιοχής (`area_name`) επιτρέπει στο API να τον αξιοποιεί ως προσωρινή μνήμη (`cache`) για τα αιτήματα τύπου TARGET. Μέσω αυτής της "επιλεκτικής αποσύνδεσης", το σύστημα ανασύρει ακαριαία δεδομένα για ήδη αναλυμένες περιοχές, παρακάμπτοντας τις περιττές κλήσεις προς το υποσύστημα της Python και εξοικονομώντας δραστηκά υπολογιστικούς πόρους.



Εικόνα 8.7: Διάγραμμα Συνδέσεων Πινάκων Hellas Cube

Κεφάλαιο 9ο: Ανάπτυξη του Rust API

9.1 Ρόλος και Τεχνολογίες του API

Όπως έχει ήδη αναφερθεί, η επιλογή της γλώσσας Rust αποτελεί μια μη συμβατική, αλλά απολύτως συνειδητή προσέγγιση. Δεδομένου ότι η συγκεκριμένη τεχνολογία δεν αποτελεί ευρέως διαδεδομένη επιλογή για την ανάπτυξη Web APIs, η υλοποίηση του υποσυστήματος αναμενόταν εξ αρχής ιδιαίτερα απαιτητική. Για την ανάπτυξη του API χρησιμοποιήθηκαν οι βιβλιοθήκες όπως `axum` η οποία είναι μία βιβλιοθήκη δρομολόγησης και διαχείρισης HTTP αιτημάτων, η οποία εστιάζει στην προγραμματιστική εργονομία και αρθρωτική σχεδίαση [37]. Παράλληλα με την βιβλιοθήκη `axum` επιλέχθηκαν και βιβλιοθήκες όπως η `sqlx` για τα ερωτήματα (queries) στην βάση δεδομένων του Hellas Cube καθώς και το γενικό dependency `tokio` για την ανάπτυξη του ασύγχρονου μας συστήματος και υποστηρίζεται από τις παραπάνω βιβλιοθήκες.

Ο ρόλος του υποσυστήματος είναι τριπλός:

1. Διαχείριση ασφαλούς εισόδου και ταυτοποίηση των χρηστών
2. Επικοινωνία διαρκώς με τη βάση δεδομένων.
3. Δρομολόγηση των αιτημάτων στο υποσύστημα ανάλυσης δεδομένων, και επιστρέφει της περιβαλλοντική πληροφορία στον χρήστη.

Η ανάλυση της αρχιτεκτονικής του κεντρική API θα πραγματοποιηθεί σταδιακά στα επόμενα κεφάλαια εξετάζοντας αρχικά την εκκίνηση του διακομιστή, προχωρώντας σε μηχανισμούς διασύνδεσης και εκτέλεσης ερωτημάτων στην βάση δεδομένων και καταλήγοντας στον πυρήνα εξυπηρέτησης των περιβαλλοντικών δεικτών.

9.2 Αρχιτεκτονική Rust API

Όπως και στο επίπεδο του Python υποσυστήματος η δημιουργία μιας προσχεδίου αρχιτεκτονικής είναι απαραίτητη για την οραμάτηση των λειτουργιών και την ροή ενός αιτήματος από την υποδοχή του έως την διεκπεραίωση του.

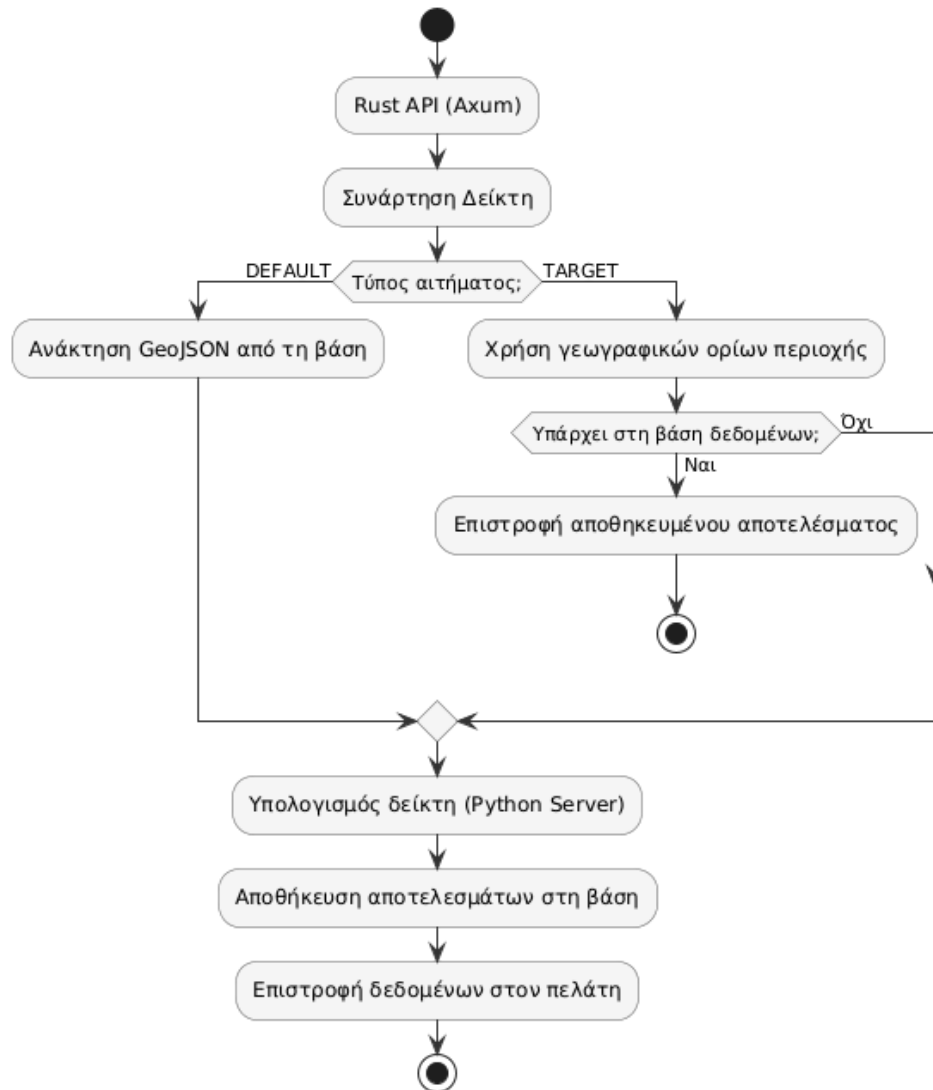
9.2.1. Περιβαλλοντικά αιτήματα

Σύμφωνα με την Εικόνα 9.1 όταν ο χρήστης ζητά τον υπολογισμό ενός περιβαλλοντικού δείκτη, το σύστημα εκτελεί τον παρακάτω κύκλο ζωής:

- Υποδοχή Αιτήματος (με την βιβλιοθήκη `Axum`): Το αίτημα HTTP καταφθάνει στο εξωτερικό API. Το πλαίσιο `Axum` αναλαμβάνει την παραλαβή και δρομολόγηση απαντήσεων.
- Τύπος αιτήματος: Ανάλογα με την φύση της περιοχής ενδιαφέροντος έτσι και διακλαδίζεται η ροή:
 - **DEFAULT:** Το αίτημα αφορά μία προσαρμοσμένη περιοχή του χρήστη. Το σύστημα ανακτά το αποθηκευμένο `Geojson` από την βάση δεδομένων και δρομολογεί το αίτημα απευθείας προς υπολογισμό.
 - **TARGET:** Το αίτημα αφορά μία προκαθορισμένη περιοχή. **TARGET:** Αφορά καθορισμένες περιοχές, όπου το σύστημα αξιοποιεί τα προϋπάρχοντα γεωγραφικά όρια και πραγματοποιεί έναν έλεγχο τύπου `cache`: «Υπάρχει ήδη καταχωρημένη η απάντηση στη βάση δεδομένων;». Σε περίπτωση που υπάρχει το σύστημα ανασύρει τα αποθηκευμένα αποτελέσματα και τα επιστρέφει ακαριαία στον πελάτη, αλλιώς

συνεχίζει κανονικά το σύστημα.

- Ανάθεση στο Υπολογιστικό Υποσύστημα: Εφόσον τα δεδομένα δεν βρέθηκαν έτοιμα το αίτημα προωθείτε στο Υποσύστημα.
- Αποθήκευση και Επιστροφή (Τερματισμός ροής): Με την ολοκλήρωση των υπολογισμών, τα νέα αποτελέσματα επιστρέφουν, αποθηκεύονται στη βάση δεδομένων και προωθούνται στον χρήστη.



Εικόνα 9.1: Ροή Αρχιτεκτονικής αιτημάτων ανάλυσης

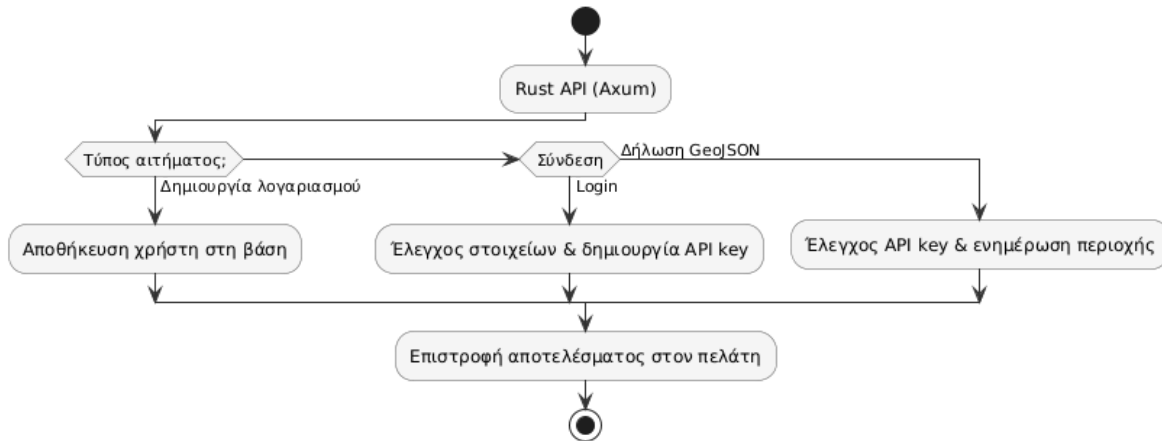
9.2.2. Αιτήματα Χρήστη

Όπως φαίνεται στην Εικόνα 9.2, η υποβολή αιτήματος από τον χρήστη για λειτουργίες χρήστη ακολουθεί τις εξής διαδικασίες:

- Δημιουργία Λογαριασμού: Το σύστημα λαμβάνει τα στοιχεία του χρήστη και τα αποθηκεύει στον πίνακα των χρηστών.
- Σύνδεση Λογαριασμού: Όταν ένας υφιστάμενος χρήστης προσπαθεί να αποκτήσει πρόσβαση, η ροή κατευθύνεται στον έλεγχο ταυτοποίησης. Πραγματοποιείται ένας έλεγχος στοιχείων και, εφόσον η αυθεντικοποίηση είναι επιτυχής, προχωρά στη δημιουργία έκδοση ενός νέου API_Key. Το κλειδί αυτό δημιουργείται, αποθηκεύεται στην βάση και επιστρέφεται

στον χρήστη για περαιτέρω χρήση.

- Δήλωση Γεωχωρικών Ορίων: Εφόσον η πλατφόρμα παρέχει τη δυνατότητα στους εγγεγραμμένους χρήστες να ορίσουν τη δική τους προεπιλεγμένη περιοχή ενδιαφέροντος. Καθώς πρόκειται για ενέργεια τροποποίησης δεδομένων, το σύστημα εκτελεί πρώτα έλεγχο του API key. Αν το κλειδί είναι έγκυρο τότε αποθηκεύεται το geojson της περιοχής.



Εικόνα 9.2: Ροή Αρχιτεκτονικής αιτημάτων χρήστη

9.3 Εκκίνηση του API

Η αφητηρία εκτέλεσης του κεντρικού υποσυστήματος σε Rust εντοπίζεται στο αρχείο [main.rs](#). Ο κύριος ρόλος είναι η αρχικοποίηση των βασικών λειτουργιών του API όπως τα endpoints, διατήρησης μία σύνδεσης με την βάση δεδομένων και του συστήματος καταγραφής αιτημάτων. Για να εξασφαλίσουμε ότι η εξυπηρέτηση των εισερχόμενων αιτημάτων πραγματοποιείται χωρίς καθυστερήσεις και εμπλοκές, ορίζουμε το σύστημα ως ασύγχρονο πριν από την υλοποίηση των βασικών του λειτουργιών, με την χρήση της εντολής `#[tokio::main]` (Εικόνα 9.3) η οποία μετατρέπει την παραδοσιακή συγχρονισμένη `main` σε μία ασύγχρονη διεργασία. Το πλαίσιο Tokio αναλαμβάνει να διαχειριστεί την ταυτόχρονη εκτέλεση πολλαπλών αιτημάτων HTTP χωρίς να μπλοκάρει το κεντρικό νήμα, εξασφαλίζοντας την υψηλή απόδοση του API.

```

#[tokio::main]
async fn main() {

    //log
    tracing_subscriber::fmt::init();

    //connecting to the database
    let pool=db::ping_database().await.expect("Failed connection to Database");

    //calling the listening function in analysis to start the API
    analysis::listening(analysis::pathing(pool)).await;
}
  
```

Εικόνα 9.3: Εκκίνηση API

Η πρώτη αρχικοποίηση που γίνεται είναι ο μηχανισμός καταγραφής συμβάντων μέσω της βιβλιοθήκης `tracing` για την παρακολούθηση υγείας του API καθώς και για μελλοντική αποσφαλμάτωση. Για την δημιουργία σύνδεσης καλείται η συνάρτηση `db::ping_database().await` (Εικόνα 9.3). Αυτή η κλήση

είναι υπεύθυνη για την επικοινωνία με την PostgreSQL και τη δημιουργία μιας κεντρικής μονάδας συνδέσεων (pool), καθώς η εντολή expect λειτουργεί ως ασφάλεια (fail fast) εάν το API δεν καταφέρνει να συνδεθεί με την βάση. Επιπλέον, η ενσωμάτωση των endpoints επιτυγχάνεται μεταβιβάζοντας το pool συνδέσεων της βάσης ως όρισμα στη συνάρτηση δρομολόγησης `analysis::pathing(pool)`. Αυτό το dependency injection επιτρέπει σε όλα τα τελικά σημεία να έχουν πρόσβαση στη βάση. Το τελικό αποτέλεσμα προωθείται στην συνάρτηση `analysis::listening`, η οποία δεσμεύει τη θύρα του server και ξεκινάει να δέχεται αιτήματα HTTP.

9.3.1. Σύνδεση Στην βάση δεδομένων

Όπως προαναφέρθηκε με την συνάρτηση `ping_database()` δημιουργούμε μία σταθερή με την βάση δεδομένων και επιστρέφουμε την σύνδεση ώστε να παραχωρείται σε όλες τις διεργασίες του API. Για την σύνδεση αυτή αρχικά παίρνουμε το URL της βάσης με το οποίο γίνεται δυνατή η σύνδεση. Με την χρήση του URL και του εργαλείου `PgPoolOptions::new().connect()` της βιβλιοθήκης `sqlx::postgres` γίνεται δυνατή η σύνδεση (Εικόνα 9.4) (με μέγιστο 5 συνδέσεις).

```
dotenv::from_filename("/home/christossapounas/.hellascube/dbconfig.env").ok(); //  
  
//getting the database url from the config file  
let database_url = std::env::var("DATABASE_URL")  
    .expect("DATABASE_URL not found in config");  
  
//making the connection and putting it inside a variable  
let pool = PgPoolOptions::new()  
    .max_connections(5)  
    .connect(&database_url)  
    .await?;
```

Εικόνα 9.4: Σύνδεση με βάση δεδομένων

9.3.2. Εκκίνηση server και τα Endpoints του API

Από την συνάρτηση της `main` καλείται πρώτα η συνάρτηση `pathing()` στην οποία δηλώνονται τα σημεία τα οποία μπορεί να στείλει αιτήματα ο χρήστης. Συγκεκριμένα, όπως φαίνεται στην Εικόνα 9.5, η διαδρομή (path) `/ndvi` δέχεται κλήσεις τύπου POST. Η επιλογή της συγκεκριμένης μεθόδου κρίνεται απαραίτητη, καθώς η διεργασία συνεπάγεται την καταχώρηση νέων εγγράφων στη βάση δεδομένων. Κατά την παραλαβή του αιτήματος, ενεργοποιείται η συνάρτηση `analysis_request::run`, η οποία διαχειρίζεται τα αιτήματα ανάλυσης δεδομένων. Επιπλέον, από τη δομή της Εικόνας 9.5 παρατηρούμε ότι το «σώμα» (body) του αιτήματος ακολουθεί συγκεκριμένο σχηματισμό, ο οποίος αποτυπώνεται προγραμματιστικά μέσω της δομής `struct IndexRequest`. Το Structure λειτουργεί ως αναπαράσταση του αιτήματος (DTO) που υποβάλλει ο χρήστης μέσω της διεπαφής γραμμής εντολών. Άλλα routes όπως οι `/cacc`, `/login`, `/history` και `/declare_geojson` χρησιμοποιούν τα δικά τους DTO όπως το structure `UserData` αποκλειστικά για το email και το password (για τα `/cacc` και `/login`), το `GeoJsonREQ` το οποίο δέχεται υποχρεωτικά το API key και τα γεωχωρικά όρια σε μορφής string (για το `/declare_geojson`) και επιπλέον το `HistoryRequest` για τα αιτήματα ιστορίας χρήστη (`/history`). Με τον τρόπο αυτό, εξασφαλίζεται η τυποποιημένη και ασφαλής μεταβίβαση των παραμέτρων εισόδου προς τη συνάρτηση επεξεργασίας.

```
pub fn pathing(pool: PgPool) -> Router {
  Router::new()
    .route("/api", get(test::run))
    .route("/ndvi", post(|State(pool): State<PgPool>,
      Extension(reporter): Extension<StatusReporter>,
      body: Json<IndexRequest>| {
        analysis_request::run(pool, reporter, body, "http://localhost:8080/analyzation/ndvi")
      })))
}
```

Εικόνα 9.5: Συνάρτηση μονοπατιών του API

Επιπλέον σε κάθε endpoint μεταβιβάζεται η μεταβλητή reporter. Η μεταβλητές pool, η σύνδεση με την βάση, και η μεταβλητή reporter, η οποία δίνει την δυνατότητα να ενημερώσουμε την κατάσταση ενός αιτήματος μέσα στην εφαρμογή του API. Ο κάθε reporter δηλώνεται ως Status report που παρέχει στην μεταβλητή την συνάρτηση update (Εικόνα 9.6) καλώντας την κάθε φορά που θέλουμε να ανανεώσουμε το status.

```
impl StatusReporter {
  pub async fn update(&self, status: &str, result: Option<Value>, payload: Option<IndexRequest>) {
    self.tx.send((status.to_string(), result, payload, place_or_userid, query)).await.ok()
  }
}
```

Εικόνα 9.6: Αρχικοποίηση συνάρτησης ενημέρωσης του reporter κατάστασης

Μαζί με κάθε endpoint της εφαρμογής, πέραν των συνδέσεων pool όπου η κατάσταση κληρονομείται σε όλα τα επίπεδα του Router του API, περνάμε την συνάρτηση log_request, μία εξειδικευμένη συνάρτηση για την ενημέρωση της κατάστασης του αιτήματος. (Εικόνα 9.7)

```
.with_state(pool.clone())
.layer(middleware::from_fn_with_state(pool.clone(), log_request))
```

Εικόνα 9.7: Πρόσθετα επίπεδα στα μονοπάτια του API

Η συνάρτηση log_request όπως παρουσιάζεται και στην Εικόνα 9.8 όταν δέχεται ένα νέο αίτημα αρχικοποιεί (για το κάθε αίτημα) μεταβλητές όπως το id, το οποίο παράγεται με την χρήση της βιβλιοθήκης uuid. Επιπλέον δημιουργείται ένα κανάλι ασύγχρονης επικοινωνίας με τον reporter το οποίο δέχεται συγκεκριμένες μεταβλητές όπως την κατάσταση, αποτέλεσμα, περιοχή και ερώτημα.

```
async fn log_request(State(pool): State<PgPool>, mut req: Request<Body>, next: Next) -> impl axum::response::IntoResponse {
  //Reporter logging status
  //getting a new request
  let request_id = Uuid::new_v4().to_string();
  let method = req.method().clone();
  let uri = req.uri().clone();
  println!("→ {} {}", method, uri);

  //initializing the reporter waiter
  let (tx, mut rx) = mpsc::channel::(String, Option<Value>, Option<IndexRequest>, Option<String>, Option<String>);
  req.extensions_mut().insert(StatusReporter { tx });
}
```

Εικόνα 9.8: Συνάρτηση για την ενημέρωση κατάστασης του αιτήματος

Η συγκέντρωση όλων αυτών των δεδομένων είναι απαραίτητη, διότι μόλις ο reporter σηματοδοτήσει την ολοκλήρωση του αιτήματος, η διεργασία log_request αναλαμβάνει την καταχώρηση των τελικών αποτελεσμάτων στη βάση δεδομένων (Εικόνα 9.9). Στην περίπτωση που το αίτημα βρίσκεται ακόμη σε εξέλιξη αλλά ο reporter επιστρέφει κάποια απόκριση μέσω της rx.recv(), ενημερώνεται άμεσα η

τρέχουσα κατάσταση του αιτήματος, καθώς και η χρονική στιγμή της τελευταίας τροποποίησης της. Τέλος, μόλις το αίτημα ολοκληρωθεί επιτυχώς ή τερματιστεί λόγω σφάλματος, ο listener του reporter απενεργοποιείται, σηματοδοτώντας την ολοκλήρωση της διεργασίας `log_request` (Εικόνα 9.9). Η εκτέλεση ερωτημάτων στη βάση γίνεται μέσω της `sqlx::query` με χρήση prepared statements και παραμετροποιημένων τιμών (`bind`), διασφαλίζοντας το σύστημα από SQL injection στο επίπεδο των δεδομένων εισόδου.

```
//waiting for response from the reporter to update request status
while let Some((status, result, payload, shared_variable, query)) = rx.recv().await {
    println!("> {} -> {}", uri, status);
    if status.starts_with("DONE") {
        if let Some(result_data) = result {
            let result_id=Uuid::new_v4();
            let payload=payload.unwrap();
            let date_range = format!("{}", &payload.from, &payload.till);
            match sqlx::query(&query.unwrap())
                .bind(result_id)
                .bind(payload.index)
                .bind(shared_variable)
                .bind(date_range)
                .bind(result_data)
                .bind(&request_id)
                .execute(&pool)
                .await {
                Ok(_) => println!("Result saved"),
                Err(e) => println!("Failed to save result, database error: {}", e),
            }
        }
    }
    sqlx::query("UPDATE request_log_file SET status = $1, status_timestamp = now() WHERE request_id = $2")
        .bind(&status)
        .bind(&request_id)
        .execute(&pool)
        .await
        .ok();
    if status.starts_with("DONE") || status.starts_with("FAILED") {
        println!("BREAKING");
        break;
    }
}
```

Εικόνα 9.9: Ασύγχρονες ενημερώσεις κατάστασης στην βάση

Η συνάρτηση `listening` αναλαμβάνει την εκκίνηση του διακομιστή στη θύρα 3000, αξιοποιώντας τις δυνατότητες ασύγχρονης εκτέλεσης του `tokio`. Στο πλαίσιο αυτό, καλείται η συνάρτηση `pathing`, η οποία έχει ήδη περιγραφεί, ενώ παράλληλα μέσω των μηχανισμών του `axum` μεταβιβάζονται στον διακομιστή ο listener και ο router που περιλαμβάνει τα endpoints με τις αντίστοιχες συναρτήσεις τους.

```
//!Starting the server
pub async fn listening(app: Router) {
    let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
    println!("Server started successfully at 0.0.0.0:3000");
    axum::serve(listener, app).await.unwrap();
}
```

Εικόνα 9.10: Εκκίνηση του server

9.4 Τομέας Αιτημάτων Ανάλυσης

Όταν ο διακομιστής λάβει ένα αίτημα ανάλυσης δεδομένων, τότε την πλήρη διαχείριση ενός αιτήματος αναλαμβάνει η συνάρτηση `analysis_requests::run()`. Η συγκεκριμένη συνάρτηση λειτουργεί ως γέφυρα μεταξύ του τελικού χρήστη, της βάσης δεδομένων και του υποσυστήματος ανάλυσης, εκτελώντας μια καθορισμένη ακολουθία.

9.4.1. Τύπος Αιτήματος

Το πρώτο βήμα την συνάρτησης είναι η κατηγοριοποίηση του αιτήματος (req_type), όπως φαίνεται και στην Εικόνα 9.11:

- Εάν το αίτημα είναι DEFAULT τότε το σύστημα απαιτεί την ταυτοποίηση του api_key. Για την ταυτοποίηση καλείται η βοηθητική συνάρτηση check_api η οποία θα αναλυθεί σε επόμενα κεφάλαια. Εφόσον ο έλεγχος ήταν επιτυχής τότε ανακτάται το geojson αρχείο του χρήστη από την βάση δεδομένων ώστε να προωθηθεί στο σύστημα ανάλυσης.
- Εάν το αίτημα είναι τύπου TARGET, τότε η διαδικασία απλοποιείται καθώς το σύστημα χρησιμοποιεί το όνομα την προκαθορισμένης περιοχής.

```
match check_api(&pool, api_key).await {
  Ok(uid) => {
    user_id=Some(uid.clone());
    reporter.update("PROCESSING: Account verification", None, None, None, None).await;
    let result = sqlx::query("SELECT declared_geo_json FROM users WHERE user_id = $1")
      .bind(&uid)
      .fetch_one(&pool)
      .await
      .map_err(|_| StatusCode::INTERNAL_SERVER_ERROR)?;
    let geo: Value = result.try_get("declared_geo_json")
      .map_err(|_| StatusCode::INTERNAL_SERVER_ERROR)?;
    geo
  }
  Err(_) => {
    reporter.update("FAILED: Authorization failure", None, None, None, None).await;
    return Err(StatusCode::UNAUTHORIZED)}
}

]else if payload.req_type == "TARGET"{
  let city = payload.city.clone();
  Value::String(city)
}else{
  Value::Null
};
```

Εικόνα 9.11: Έλεγχος εγκυρότητας API

9.4.2. Επαναχρησιμοποίηση δεδομένων

Για την βελτιστοποίηση των υπολογιστικών πόρων, μέσα στην συνάρτηση run των αναλύσεων ενσωματώθηκε ένας μηχανισμός ελέγχου ιστορικού αποκλειστικά για τα αιτήματα TARGET. Εφόσον τα αιτήματα TARGET αφορούν χρήστες που κάνουν κλήσεις για προκαθορισμένα γεωχωρικά δεδομένα τότε με την συνάρτηση check_general_cache, η οποία αποτελεί ερώτημα εύρεσης δεδομένων με την ίδια ανάλυση, περιοχή και διάστημα περιοχών. Αν η συνάρτηση επιστρέφει δεδομένα τότε έχουμε cache hit και η συνάρτηση τερματίζεται νωρίς επιστρέφοντας την απάντηση απευθείας. Διαφορετικά συνεχίζει κανονικά στην κλήση του υποσυστήματος.

```

//!db cache lookup
if payload.req_type == "TARGET" {
    if let Some(area_name) = place.as_str() {
        match check_general_cache(&pool, area_name, &payload.index, &payload.from, &payload.till) {
            Ok(Some(cached)) => {
                reporter.update("DONE: Cache hit", None, None, None, None).await;
                return Ok(Json(cached));
            }
            Ok(None) => {}
            Err(e) => {eprintln!("Cache lookup failed: {e}");}
        }
    }
}
}

```

Εικόνα 9.12: Έλεγχος για υπάρχοντα δεδομένα στην βάση δεδομένων

9.4.3. Κλήση Υποσυστήματος Ανάλυσης

Στην περίπτωση που τα απαιτούμενα δεδομένα δεν είναι διαθέσιμα στη βάση, το αίτημα υπολογίζεται εκ νέου. Το API αξιοποιεί τον HTTP client `reqwest::Client` (Εικόνα 9.13) προκειμένου να συνθέσει και να αποστείλει το αίτημα προς το υποσύστημα. Μόλις ληφθεί η απόκριση, υφίσταται δυναμική σειριοποίηση μέσω την βιβλιοθήκης `serde_json` για την ορθή μορφοποίηση την απάντησης του json αρχείου στο cli. Η κλήση εκτελείται ασύγχρονα, γεγονός που επιτρέπει το API να παραμένει διαθέσιμο προς εξυπηρέτηση.

```

//call python for analyzation
let to_send: serde_json::Value = serde_json::json!({
    "req_type": payload.req_type.clone(),
    "place": place_value,
    "index": payload.index.clone(),
    "date1": payload.from.clone(),
    "date2": payload.till.clone(),
    "source": payload.source.clone()
});

//send request
let client = Client::new();
let response = match client.post(req_url).json(&to_send).send().await {

```

Εικόνα 9.13: Συγγραφή και αποστολή του json στο υποσύστημα

Κατά τη διάρκεια εκτέλεσης του αιτήματος ανάλυσης από το υποσύστημα, ο μηχανισμός `reporter` παρακολουθεί τη διεργασία και αποστέλλει συνεχώς ενημερώσεις καταστάσεων και αλλαγών στη συνάρτηση `log_request`, ώστε αυτές να καταγράφονται στον αντίστοιχο πίνακα της βάσης δεδομένων. Μόλις το υποσύστημα ολοκληρώσει την απόκρισή του, ο `reporter` διαχειρίζεται το αποτέλεσμα: σε περίπτωση αποτυχίας ενημερώνει για το σφάλμα, ενώ σε περίπτωση επιτυχίας επιβεβαιώνει την ολοκλήρωση και τροφοδοτεί το κατάλληλο ερώτημα για την αποθήκευση των δεδομένων στον σωστό πίνακα. Συγκεκριμένα, τα αποτελέσματα των ερωτημάτων τύπου `DEFAULT` καταχωρούνται στον πίνακα `user_results`, ενώ τα ερωτήματα τύπου `TARGET` διοχετεύονται στον πίνακα `general_results`.

```

// Some Request
let client = Client::new();
let response = match client.post(req_url).json(&to_send).send().await {
    Ok(res) => {
        res
    },
    Err(e) => {
        reporter.update("FAILED: Request to python failed", None, None, None, None).await;
        eprintln!("Request failed: {e}");
        return Err(StatusCode::BAD_GATEWAY);
    }
};
let resp: Value = match response.json::<Value>().await {
    Ok(val) => {
        if payload.req_type == "DEFAULT" {
            reporter.update("DONE: Alanysis successfull", Some(val.clone()), Some(payload), user_id.clone(),
                Some("INSERT INTO user_results (res_id, analysis, user_id, date_range, res_json, request_id) VALUES
            ) else {
                reporter.update("DONE: Alanysis successfull", Some(val.clone()), Some(payload), place.as_str().map(
                Some("INSERT INTO general_results (res_id, analysis, area_name, date_range, res_json, request_id) V
            )
        }
        val
    }
    Err(e) => {
        eprintln!("Failed to parse response JSON: {e}");
        reporter.update("FAILED", None, None, None, None).await;
    }
};

```

Εικόνα 9.14: Διαχείριση απάντησης του υποσυστήματος

9.5 Τομέας Εκτέλεσης Αιτημάτων Χρηστών

Το module το οποίο καλείται σε endpoints που αφορούν επεξεργασίας χρήστη εντοπίζεται το αρχείο `user.rs`. Είναι επιφορτισμένο με την ασφαλή διαχείριση των λογαριασμών, την κρυπτογραφική προστασία των στοιχείων του χρήστη καθώς και την έκδοση των κλειδιών για το API. Αναλύοντας τον πηγαίο κώδικα, η λειτουργικότητα του συστήματος δομείται γύρω από τους εξής άξονες:

9.5.1. Δημιουργία λογαριασμού

Η συνάρτηση `cacc` (Create Account) αναλαμβάνει την εγγραφή νέων χρηστών. Για την εξασφάλιση της μοναδικότητας κάθε εγγραφής σε κατανεμημένα περιβάλλοντα, το σύστημα δεν βασίζεται σε απλούς αύξοντες αριθμούς, αλλά παράγει ένα μοναδικό αναγνωριστικό τύπου UUID (από την βιβλιοθήκη που αναφέρθηκε στο κεφάλαιο 9.3.2) για κάθε νέο χρήστη. Στη συνέχεια, τα δεδομένα εγγράφονται με ασφάλεια στον πίνακα `users` μέσω ερωτημάτων (Ο κωδικός λαμβάνεται ήδη hashed για μέγιστη προστασία).

9.5.2. Διαδικασία Ταυτοποίησης

Η είσοδος στο σύστημα ελέγχεται από τον συνδυασμό συναρτήσεων `login` και `check_cred` (Εικόνα 9.16). Η διαδικασία έχει ως εξής:

- Προστασία Κωδικών (Argon2): Η συνάρτηση `check_cred` δεν συγκρίνει ποτέ κωδικούς σε απλό κείμενο. Ανακτά τον αποθηκευμένο hashed κωδικό από την βάση δεδομένων και τον επαληθεύει αξιοποιώντας τον αλγόριθμο Argon2, οποίος αποτελεί το σύγχρονο πρότυπο για την ασφαλή αποθήκευση κωδικών πρόσβασης, καθώς είναι ανθεκτικών σε επιθέσεις.
- Έκδοση API key: Εφόσον η ταυτοποίηση είναι επιτυχής, η συνάρτηση `gen_api_key` (Εικόνα 9.16) παράγει ένα ισχυρό, τυχαίο κλειδί 48 αλφαριθμητικών χαρακτήρων (με το αρχή `hc_`). Το κλειδί αυτό καταχωρείται στον πίνακα `api_k` με ρητή ημερομηνία λήξης (`exp_date`) ορισμένη στις 8 ώρες, εφαρμόζοντας την αρχή του ελάχιστου χρόνου προνομιών.

```
match check_cred(pool.clone(), &payload.email, &payload.password).await {
    Ok(user_id) => {
        api_key = gen_api_key();
        let query = "INSERT INTO api_k (api_key, exp_date, user_id) VALUES ($1, NOW() +
                    INTERVAL '8 hours', $2)";
```

Εικόνα 9.15: Έλεγχος στοιχείων, δημιουργία API key και ενημέρωση βάσης

```
//Generate a new API KEY
pub fn gen_api_key() -> String {
    let key: String = rand::thread_rng()
        .sample_iter(&rand::distributions::Alphanumeric)
        .take(48)
        .map(char::from)
        .collect();

    return format!("hc_{}", key);
}
```

Εικόνα 9.16: Δημιουργία API key

9.5.3. Βελτιστοποιημένη Παραγωγή κλειδιών

Μία από τις πιο σημαντικές συναρτήσεις ασφάλειας είναι η `api_key_hash` (Εικόνα 9.17). Ενώ οι κωδικοί πρόσβασης προστατεύονται με τον βαρύ αλγόριθμο Argon2, για τα API keys επιλέχθηκε ο αλγόριθμος SHA-256. Καθώς το API Key ελέγχεται σε κάθε μεμονωμένο αίτημα (π.χ. στην ανάλυση δορυφορικών εικόνων), η χρήση του ταχύτερου SHA-256 εξασφαλίζει ότι το σύστημα δεν χρησιμοποιήσει υπερβολικούς υπολογιστικούς πόρους για κάθε διεκπεραίωση αιτήματος με API key. Ταυτόχρονα, τα κλειδιά αποθηκεύονται στη βάση αποκλειστικά ως hashes, διασφαλίζοντας ότι ακόμη και σε περίπτωση υποκλοπής των δεδομένων της βάσης, τα πραγματικά κλειδιά παραμένουν κρυφά.

```
fn api_key_hash(hash_object: &str) -> String {
    let mut hasher = Sha256::new();
    hasher.update(hash_object.as_bytes());
    hex::encode(hasher.finalize())
}
```

Εικόνα 9.17: Κωδικοποίηση του api key με SHA-256

9.5.4. Ενημέρωση Γεωχωρικών Δεδομένων

Η συνάρτηση `initialize_geo_json` παρέχει το τελικό endpoint μέσω του οποίου ένας εξουσιοδοτημένος χρήστης (εφόσον το API Key του επικυρωθεί από τη συνάρτηση `check_api`) μπορεί να επισυνάψει ή να ανανεώσει την προεπιλεγμένη περιοχή ενδιαφέροντός του (GeoJSON) στον λογαριασμό του.

9.5.5. Ιστορία χρήστη

Καθώς τα αποτελέσματα του χρήστη αποθηκεύονται στον πίνακα `user_results`, δίνεται η δυνατότητα ανάκτησής τους. Με την κλήση του endpoint `/history` καλείται η συνάρτηση `history()`. Η συνάρτηση δέχεται το API key από τον χρήστη και αφού επικυρώνεται ως έγκυρο, με την βοήθεια της `check_api` που προαναφέρθηκε. Αφού ταυτοποιηθεί το API key με την εκτέλεση query παίρνουμε τα δεδομένα του χρήστη σε φθίνουσα σειρά (Εικόνα 9.18). Όταν τελειώσει η συλλογή των αναλύσεων του χρήστη, επιστρέφονται ως json στον χρήστη.

```
Ok(user_id) => {
  let query = r#"
    SELECT res_id, analysis, date_range::text as date_range, res_json, request_id
    FROM user_results
    WHERE user_id = $1
    ORDER BY date_range DESC
  "#;
```

Εικόνα 9.18: Ερώτημα ανάκτησης ιστορικού χρήστη.

9.6 Αξιολόγηση API

Η ανάπτυξη του κεντρικού API εξόπλισε το σύστημα με ένα API Gateway, ικανό να διαχειρίζεται ταυτόχρονα αιτήματα μέσω της βιβλιοθήκης Tokio, να επιλύει ζητήματα ταυτοποίησης με κρυπτογραφία βιομηχανικού προτύπου και να ελαχιστοποιεί τον χρόνο απόκρισης αξιοποιώντας την έξυπνη προσωρινή αποθήκευση (caching). Βάσει των αρχών της σύγχρονης Μηχανικής Λογισμικού, η τεράστια αυτή τεχνική πολυπλοκότητα οφείλει να παραμένει κρυμμένη στον τελικό χρήστη [40]. Η επιτακτική ανάγκη για γεφύρωση αυτής της απαιτητικής backend αρχιτεκτονικής με μια διαισθητική, απλή και άμεση εμπειρία χρήσης καλύπτεται από την ανάπτυξη του CLI.

Κεφάλαιο 10ο: Ανάπτυξη του Command Line Interface

10.1 Ρόλος CLI

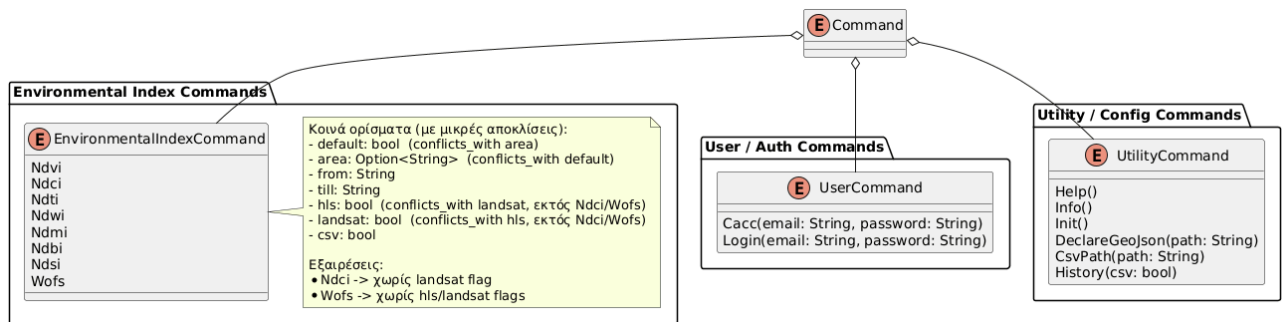
Η αλληλεπίδραση του τελικού χρήστη με το ευρύτερο οικοσύστημα του Hellas Cube πραγματοποιείται μέσω της εξειδικευμένης Διεπαφής Γραμμής Εντολών. Στον τομέα της ανάλυσης γεωχωρικών και περιβαλλοντικών δεδομένων όπου συχνά απαιτείται η διαχείριση τεράστιου όγκου δορυφορικών εικόνων (όπως Sentinel-2, Landsat, HLS) χρειάζεται μεγάλη υπολογιστική δύναμη με αποτέλεσμα η διεπαφή να είναι μόνο το σημείο επαφής με τα δεδομένα και την ανάλυση που προσφέρει ο server. Ο θεμελιώδης ρόλος αυτού του υποσυστήματος είναι να λειτουργεί ως ένας μεταφραστής στον άνθρωπο και στην μηχανή. Στην πράξη, το CLI καλύπτει πλήρως την εσωτερική πολυπλοκότητα της αρχιτεκτονικής, όπως τη σύνθεση των HTTP αιτημάτων, τη διαχείριση των κεφαλίδων εξουσιοδότησης (API Keys) και τη σειριοποίηση των γεωχωρικών δεδομένων σε μορφή JSON/GeoJSON. Ο χρήστης αρκεί να εισάγει μια δομημένη εντολή με τα επιθυμητά flags στο τερματικό του, και το σύστημα αναλαμβάνει την ασφαλή δρομολόγηση του αιτήματος στο API, καθώς και την ορθή παρουσίαση των τελικών αποτελεσμάτων στην οθόνη.

10.2 Αρχιτεκτονική των Εντολών του CLI

Η αρχιτεκτονική του Command Line Interface σχεδιάστηκε με σκοπό την κατηγοριοποίηση των εντολών σε τρεις διακριτές ομάδες. Οι ομάδες αυτές οργανώνουν τις λειτουργίες του συστήματος με τρόπο ώστε να διαχωρίζονται διακριτά σε εντολές: χρήστη, βοηθητικές συστήματος και αναλύσεων.

10.2.1. Εντολές

Το CLI του Hellas Cube οργανώνεται γύρω από το enum Command με την πρώτη κατηγορία να είναι οι User/Auth Commands, περιλαμβάνει τις εντολές Cacc και Login, που είναι υπεύθυνες για τη δημιουργία λογαριασμού και την αυθεντικοποίηση του χρήστη μέσω email και κωδικού πρόσβασης. Η δεύτερη κατηγορία, Utility/Config Commands, συγκεντρώνει βοηθητικές εντολές όπως οι Help, Info και Init για γενική πληροφόρηση και αρχικοποίηση, καθώς και οι DeclareGeoJson και CsvPath, που ορίζουν διαδρομές αρχείων στη ρύθμιση του συστήματος, μαζί με την History, η οποία επιτρέπει την ανάκτηση του ιστορικού αιτημάτων προαιρετικά σε μορφή CSV. Η τρίτη και πιο εκτενής κατηγορία, Environmental Index Commands, περιλαμβάνει τις εντολές υπολογισμού περιβαλλοντικών δεικτών, οι οποίες μοιράζονται σε μεγάλο βαθμό κοινό σχήμα ορισμάτων: την επιλογή μεταξύ προκαθορισμένης περιοχής (default) ή συγκεκριμένης περιοχής μέσω GeoJSON (target), οι οποίες είναι αμοιβαία αποκλειόμενες, το χρονικό εύρος ανάλυσης (from, till), την επιλογή πηγής δεδομένων μεταξύ HLS και Landsat και τη δυνατότητα εξαγωγής των αποτελεσμάτων σε CSV. Εξαίρεση αποτελεί εντολή NDCI, που δεν υποστηρίζει την επιλογή Landsat, και Wofs, που δεν διαθέτει καθόλου τις σημαίες hls/landsat, καθώς ο υπολογισμός πλημμυρικής έκτασης βασίζεται αποκλειστικά σε δεδομένα Landsat. Κατά την εκτέλεση των παραπάνω εντολών ανάλυσης στο τερματικό, οι επιλογές του χρήστη όσον αφορά την πηγή των δεδομένων περιορίζονται αυστηρά στα υποστηριζόμενα υποσύνολα (Landsat, Sentinel, HLS). Το εργαλείο Hellas Cube δεν ζητά δεδομένα από τον χρήστη, παρά μόνο την επιλογή της πηγής δεδομένων μεταξύ αυτών που υποστηρίζονται από το σύστημα. Λόγω του μεγάλου πλήθους των εντολών και για λόγους παραδειγματισμού θα εξεταστούν ενδεικτικά αποκλειστικά οι εντολές ndvi (ως μία εντολή) και cacc.



Εικόνα 10.1: Σχεδιάγραμμα Εντολών CLI

Με χρήση των macro attributes της βιβλιοθήκης clap (`#[derive(Subcommand)]`) γίνεται αυτόματη η παραγωγή του CLI parsing. Στην περίπτωση της NDVI, κάθε πεδίο επισημαίνεται με `#[arg(long)]` ώστε να αντιστοιχεί σε named flag (π.χ. `--from`, `--till`), ενώ τα πεδία `default` και `area` φέρουν επιπλέον το attribute `conflicts_with`, το οποίο δηλώνει στο clap ότι τα δύο ορίσματα είναι αμοιβαία αποκλειόμενα αν δοθούν και τα δύο ταυτόχρονα, το πρόγραμμα επιστρέφει σφάλμα πριν καν εκτελεστεί η λογική της εντολής. Ομοίως, τα HLS και Landsat δηλώνονται με αμφίδρομο `conflicts_with` μεταξύ τους. Το πεδίο `area` χρησιμοποιεί επιπλέον `value_parser = validate_area`, που σημαίνει ότι η τιμή του περνάει από συνάρτηση επικύρωσης (`validate_area`) πριν γίνει αποδεκτή, επιτρέποντας έλεγχο μορφής ή εγκυρότητας της περιοχής κατά το parsing και όχι αργότερα στο runtime. Τα υπόλοιπα πεδία (`from`, `till`, `csv`) είναι απλά υποχρεωτικά (ή boolean flags στην περίπτωση του `csv`) χωρίς πρόσθετους περιορισμούς.

Η `Cacc`, αντίθετα, είναι πολύ πιο απλή δήλωση: περιέχει μόνο δύο υποχρεωτικά πεδία, `email` και `password`, και τα δύο τύπου `String`, επισημασμένα απλώς με `#[arg(long)]` χωρίς κανέναν περιορισμό `conflicts_with` ή `value_parser` καθώς πρόκειται για απευθείας εντολή δημιουργίας λογαριασμού που απλά προωθεί τα δύο credentials στο backend για επεξεργασία (π.χ. hashing μέσω Argon2, όπως αναφέρεται και στην υλοποίηση του auth συστήματος).

```

Cacc{
    #[arg(long, value_parser = parse_email)]
    email: String,
    #[arg(long)]
    password: String
},
Ndvi{
    #[arg(long, conflicts_with = "area")]
    default: bool,
    #[arg(long, conflicts_with = "default", value_parser = validate_area)]
    area: Option<String>,
    #[arg(long, value_parser = parse_date)]
    from: String,
    #[arg(long, value_parser = parse_date)]
    till: String,
    #[arg(long, conflicts_with = "landsat")]
    hls: bool,
    #[arg(long, conflicts_with = "hls")]
    landsat: bool,
    #[arg(long)]
    csv: bool,
}
  
```

Εικόνα 10.2: Enum εντολών CLI

10.2.2. Fail Fast εντολών

Ο μηχανισμός fail-fast του CLI βασίζεται στην αρχή ότι κάθε λανθασμένη είσοδος πρέπει να εντοπίζεται και να απορρίπτεται κατά τη φάση του parsing από το clap, πριν καν την εκτέλεση της αντίστοιχης εντολής, αποφεύγοντας έτσι περιττές κλήσεις προς το backend ή σφάλματα βαθύτερα στη ροή εκτέλεσης. Αυτό επιτυγχάνεται μέσω τεσσάρων συναρτήσεων επικύρωσης που συνδέονται στα ορίσματα με το attribute value_parser (Εικόνα 10.2). Η parse_email (Παράδειγμα Εικόνας 10.3) ελέγχει τη δομική εγκυρότητα ενός email (ύπαρξη @, μη κενό local part, ύπαρξη τελείας στο domain, απουσία διπλών τελειών ή τελείας στην αρχή/τέλος του domain) και χρησιμοποιείται στα πεδία email των εντολών Login και Cacc. Η parse_date επιβάλλει αυστηρή μορφή ημερομηνίας DD-MM-YYYY, ελέγχοντας μήκος, θέση παυλών, εύρος μήνα/ημέρας και ακόμη και έτη δίσεκτα για τον Φεβρουάριο, και εφαρμόζεται σε όλα τα πεδία from/till των εντολών δεικτών. Η validate_path με την βοήθεια της βιβλιοθήκης PathBuf, ελέγχει αν το path που δόθηκε υπάρχει πραγματικά στο filesystem (path.exists()) και αξιοποιείται στις DeclareGeoJson και CsvPath. Τέλος, η validate_area απορρίπτει δεσμευμένες/άκυρες τιμές όπως "null", "none", "all" στο πεδίο area των εντολών δεικτών, ώστε ο χρήστης να μην περνά ψευδοτιμές αντί για πραγματικό όνομα περιοχής. Παράλληλα με τους value_parsers, το fail-fast ενισχύεται και από τους δηλωτικούς περιορισμούς conflicts_with του clap, οι οποίοι εγγυώνται στατικά (πριν από οποιαδήποτε λογική) ότι δεν μπορούν να δοθούν ταυτόχρονα default και area, ούτε hls και landsat (με τις γνωστές εξαιρέσεις στις Ndcι και Wofs). Το αποτέλεσμα είναι ότι κάθε λανθασμένη είσοδος εντοπίζεται αποκλειστικά στο επίπεδο του command-line parsing, με το πρόγραμμα να τερματίζει άμεσα με περιγραφικό μήνυμα σφάλματος αντί να προχωρήσει σε κλήσεις προς το API με άκυρα δεδομένα.

```
fn parse_email(s: &str) -> Result<String, String> {
    if s.is_empty() {
        return Err("Email cannot be empty".to_string());
    }
    if !s.contains('@') {
        return Err("Email must contain @".to_string());
    }
    let parts: Vec<&str> = s.split('@').collect();
    if parts.len() != 2 {
        return Err("Email must contain exactly one @".to_string());
    }
    if parts[0].is_empty() {
        return Err("Email local part cannot be empty".to_string());
    }
    if !parts[1].contains('.') {
        return Err("Email domain must contain a dot".to_string());
    }
    if parts[1].starts_with('.') || parts[1].ends_with('.') || parts[1].contains("..") {
        return Err("Email domain is invalid".to_string());
    }
    Ok(s.to_string())
}
```

Εικόνα 10.3: Παράδειγμα Fail-fast μεθοδού

10.3 Εκτέλεση Εντολών

Το module commands, μέσω της συνάρτησης matching, αποτελεί το κεντρικό σημείο δρομολόγησης (dispatch) του CLI: παραλαμβάνει το parsed Command enum και, με χρήση pattern matching, το προωθεί στην κατάλληλη υπο-συνάρτηση, χωρίζοντας εννοιολογικά την εκτέλεση σε τρεις κατηγορίες αιτημάτων.

10.3.1. Εκτέλεση Αιτημάτων Ανάλυσης

Στην Εκτέλεση Αιτημάτων Ανάλυσης ανήκουν οι εντολές δεικτών, οι οποίες ακολουθούν ένα ενιαίο μοτίβο στο matching: πρώτα προσδιορίζεται η πηγή δεδομένων (`get_dataset_source` για `hls/landsat/sentinel`, ή `get_hls_source` για τα δύο-επιλογών indices όπως το NDCI), έπειτα διακλαδίζεται η λογική ανάλογα με το αν χρησιμοποιείται default (οπότε ανακτάται το api key και ο τύπος αιτήματος ορίζεται "DEFAULT") ή area (όπου ο τύπος γίνεται "TARGET" χωρίς api key)(Εικόνα 10.4).

```
Command::Ndv{default,area, from, till, hls, landsat, csv}>={
  let source = get_dataset_source(hls, landsat);
  if default {
    let api_key = get_api_key().map_err(|_| "Failed to get API key"?; //getting the api key that is
    ndi::run("http://localhost:3000/ndvi",
      Some("".to_string()),
      &from,
      &till,
      "NDVI",
      "DEFAULT",
      Some(api_key),
      &source,
      &csv)
  } else {
    let area = area.ok_or("Provide --area or use --default"?;
    ndi::run("http://localhost:3000/ndvi",
      Some(area),
      &from,
      &till,
      "NDVI",
      "TARGET",
      Some("".to_string()),
      &source,
      &csv)]
```

Εικόνα 10.4: Εκτέλεση εντολής ndvi

Τέλος καλείται η κοινή συνάρτηση `ndi::run`, η οποία χτίζει το JSON payload με όλα τα παραμέτρους (`index`, `req_type`, `api_key`, `city`, `from`, `till`, `source`) και το στέλνει στο αντίστοιχο endpoint. Όταν η απάντηση είναι επιτυχής (`Ok(body)`), η ροή συνεχίζει σε δύο βήματα: πρώτα καλείται η `export(&body)`, η οποία αναλαμβάνει να τυπώσει ή να παρουσιάσει το αποτέλεσμα στον χρήστη (π.χ. στο τερματικό). Στη συνέχεια ελέγχεται η τιμή της μεταβλητής `csv`: αν είναι `true`, καλείται η `export_to_csv(&body, None)`, η οποία παίρνει το ίδιο `body` και το μετατρέπει σε αρχείο CSV. Το αποτέλεσμα αυτής της εγγραφής ελέγχεται ξεχωριστά με δικό του `match`, τυπώνοντας είτε μήνυμα επιτυχίας είτε το συγκεκριμένο σφάλμα CSV. Τέλος, ανεξάρτητα από το αν ζητήθηκε CSV export ή όχι, η συνάρτηση επιστρέφει `Ok()`, σηματοδοτώντας πλήρη επιτυχία της εντολής στο επίπεδο του caller (matching). (Οι μέθοδοι εξαγωγής δεδομένων θα αναλυθούν στα επόμενα κεφάλαια)

```
let res = send(path, json);

match res {
  Ok(body) => {
    export(&body);
    if *csv {
      match export_to_csv(&body) {
        Ok(_) => println!("Successfully appended to CSV"),
        Err(e) => println!("CSV Error: {}", e),
      }
    }
  }
}
```

Εικόνα 10.5: Μέθοδος εξαγωγής

10.3.2. Εκτέλεση Αιτημάτων Χρήστη

Στην Εκτέλεση Αιτημάτων Χρήστη ανήκουν οι Cacc, Login, DeclareGeoJson, History και CsvPath, όλες υλοποιημένες στο module user.

10.3.2.1 cacc

Η συνάρτηση cacc αναλαμβάνει τη διεργασία δημιουργίας νέου λογαριασμού. Αρχικά, πριν τη διαμόρφωση του JSON payload, ο κωδικός πρόσβασης διοχετεύεται στην τοπική συνάρτηση κατακερματισμού (hash) που βασίζεται στον αλγόριθμο Argon2 (Εικόνα 10.6): δημιουργείται ένα τυχαίο salt μέσω της συνάρτησης SaltString::generate(&mut OsRng) (κρυπτογραφικά ασφαλής γεννήτρια τυχαίων αριθμών) και κατόπιν εκτελείται η argon2.hash_password(). Με αυτή τη διαδικασία, ο κωδικός πρόσβασης δεν μεταδίδεται ποτέ σε μορφή καθαρού κειμένου μέσω του δικτύου, αλλά διακινείται αποκλειστικά το hash του. Στη συνέχεια, το αίτημα διαβιβάζεται στο endpoint /cacc και η ληφθείσα απάντηση προβάλλεται στην έξοδο (println!) εντός της match δομής, χωρίς περαιτέρω προώθηση. Η συνάρτηση επιστρέφει σταθερά Ok(()), γεγονός που σημαίνει ότι ακόμη και σε περίπτωση αποτυχίας του request, η καλούσα συνάρτηση (matching) λαμβάνει ένδειξη επιτυχίας, ενώ η εκάστοτε αποτυχία γνωστοποιείται με το ανάλογο μήνυμα.

```
fn hash(hash_object: &str) -> String {
    let salt = SaltString::generate(&mut OsRng);
    let argon2 = Argon2::default();
    argon2.hash_password(hash_object.as_bytes(), &salt)
        .unwrap()
        .to_string()
}
```

Εικόνα 10.6: Hashing απο το CLI

10.3.2.2 login

Η συνάρτηση login ξεκινά με έναν τοπικό προληπτικό έλεγχο (fail-fast): εάν το αρχείο ρυθμίσεων hc_config.toml δεν εντοπιστεί στον κατάλογο ~/.hellascube, η διαδικασία διακόπτεται άμεσα. Παράλληλα, επιστρέφεται ενημερωτικό μήνυμα που προτρέπει τον χρήστη να εκτελέσει αρχικά την εντολή init (Εικόνα 10.6), αποφεύγοντας έτσι την περιττή αποστολή δικτυακού αιτήματος. Εφόσον το αρχείο εντοπιστεί κανονικά, το συνηματικό αποστέλλεται σε μορφή καθαρού κειμένου, καθώς η επαλήθευση του hash πραγματοποιείται στον διακομιστή. Με την παραλαβή της επιτυχούς απάντησης, το σώμα της (JSON string) αναλύεται σε δομή serde_json::Value και εξάγεται το api_key (εφόσον εμπεριέχεται). Ακολούθως, το τοπικό αρχείο TOML διαβάζεται, μετατρέπεται μέσω deserialization στη δομή Config, ενημερώνεται με τα νέα διαπιστευτήρια (email και api_key) μέσω της συνάρτησης set_creds, και αποθηκεύεται εκ νέου στο δίσκο με πλήρη διαχείριση πιθανών σφαλμάτων.

```
pub fn login(email: &str, password: &str) -> Result<String, &'static str> {
    let config_file = dirs::home_dir().expect("Could not find home directory").join(".hellascube")
        .join("hc_config.toml");
    if !config_file.exists() {
        return Ok("Hellascube is not initialized. Run `hellascube init` first.".to_string());
    }
}
```

Εικόνα 10.7: Συνάρτηση login

10.3.2.3 Δήλωση Geojson στον server

Η `declare_geojson` διαβάζει πρώτα το τοπικό αρχείο GeoJSON μέσω `fs::read_to_string` ανακτώντας στην συνέχεια το `api_key` μέσω `get_api_key()` και χτίζεται το payload με το περιεχόμενο του GeoJSON. Το payload αποστέλλεται στο αφιερωμένο endpoint δήλωσης του geojson στην βάση του server. Στο τέλος αποθηκεύεται το μονοπάτι του geojson που αποθηκεύτηκε την τελευταία φορά.

```
let res= send("http://localhost:3000/declare_geojson", json); //build the api request
match res {
  Ok(body) => {
    if let Err(e) = save_path_to_config(geojson_path) {
      eprintln!("Warning: geojson declared successfully but failed to save path to config: {e}");
    }
    Ok(body)
  }
  Err(_e) => Err("Error declaring geojson, check API KEY.")
}
```

Εικόνα 10.8: Αποστολή geojson αποθήκευση του μονοπατιού στο τοπικό config

10.3.2.4 Αποθήκευση μονοπατιού στο config

Οι `save_path_to_config` και `save_csv_path_to_config` εκτελούν επεξεργασία και ανάγνωση πάνω στο TOML config, με κύρια διαφορά τη builder μέθοδο του Config καλούν (`save_gj_path` έναντι `save_csv_path`).

```
fn save_path_to_config(path: &str)-> Result<(), &'static str>{
  let config_file = dirs::home_dir().expect("Could not find home directory").join(".hellas");
  let existing = fs::read_to_string(&config_file).map_err(|_| "Failed to read config");
  let config: Config = toml::from_str(&existing).map_err(|_| "Failed to parse config");
  let updated = config.save_gj_path(path.to_string());
  let toml_str = toml::to_string(&updated).map_err(|_| "Failed to serialize config");
  fs::write(&config_file, toml_str).map_err(|_| "Failed to write config");
  Ok(())
}
```

Εικόνα 10.9: Συνάρτηση αποθήκευσης μονοπατιού στο config

10.3.2.5 Ανάκτηση Ιστορικού χρήστη

Η συνάρτηση `get_history` λειτουργεί με παρόμοιο τρόπο όπως η `ndi::run`: αφού κάνει κλήση στο endpoint `/history` και λάβει με επιτυχία την απάντηση του ιστορικού του χρήστη, καλεί τη μέθοδο `export(&body)` για την προβολή των αποτελεσμάτων. Στη συνέχεια, εξετάζει τη σημαία `csv` και, εάν αυτή έχει ενεργοποιηθεί, προχωρά στην κλήση της `export_to_csv(&body)`.

10.3.2.6 Ανάκτηση στοιχείων από config

Συνοψίζοντας, οι συναρτήσεις `get_api_key` και `get_creds` αποτελούν απλούς μηχανισμούς ανάγνωσης του αρχείου ρυθμίσεων (config): η πρώτη απομονώνει και επιστρέφει αποκλειστικά την τιμή του `api_key`, ενώ η δεύτερη παρέχει ολόκληρη τη δομή `Config struct`. Και οι δύο ακολουθούν την ίδια ακολουθία διεργασιών, η οποία περιλαμβάνει τον εντοπισμό του αρχικού καταλόγου (home directory), την ανάγνωση του αρχείου και τη μετατροπή των δεδομένων μέσω TOML αποσειριοποίησης (deserialization).

```
//get api_key so that we can take it on each request
pub fn get_api_key()-> Result<String, &'static str>{
    let config_file = dirs::home_dir().expect("Could not find home directory").join(".hellascube");
    let existing = fs::read_to_string(&config_file).map_err(|_| "Failed to read config"?);
    let config: Config = toml::from_str(&existing).map_err(|_| "Failed to parse config"?);
    Ok(config.api_key)
}
```

Εικόνα 10.10: Λήψη του API key απο το config

10.3.3. Εκτέλεση εντολών συστήματος

Στην Εκτέλεση Αιτημάτων Συστήματος ανήκουν οι εντολές Help, Info και Init, οι οποίες δεν επικοινωνούν με το backend αλλά διαχειρίζονται τοπική κατάσταση: η Init δημιουργεί τον φάκελο ~/.hellascube και το αρχείο hc_config.toml με προεπιλεγμένες τιμές τις τιμές του Struct Config (Εικόνα 10.) (μέσω Config::default() και serialization σε TOML), ελέγχοντας πρώτα αν υπάρχει ήδη για να αποφύγει επανεγγραφή, ενώ η Info διαβάζει και τυπώνει το τρέχον περιεχόμενο του config (email, api key, paths) μέσω της user::get_creds().

```
Command::Init {}=>{
    let mut dir = dirs::home_dir().expect("Could not find home directory");
    dir.push(".hellascube");
    let config_file = dir.join("hc_config.toml");
    if config_file.exists() {
        println!("HellasCube is already initialized.");
        return Ok(());
    }
    fs::create_dir_all(&dir)
        .map_err(|_| "Failed to create config directory"?);
    let default_config = Config::default();
    let toml_str = toml::to_string(&default_config)
        .map_err(|_| "Failed to serialize config"?);
    fs::write(&config_file, toml_str)
        .map_err(|_| "Failed to write config file"?);
    println!("Initialized HellasCube at ~/.hellascube/hc_config.toml");
    Ok(())
}
```

Εικόνα 10.11: Εντολή Init

10.4 Εξαγωγή δεδομένων

Καθώς το CLI αποτελεί τη βασική οπτική διεπαφή του χρήστη, η ακατέργαστη εμφάνιση των απαντήσεων σε μορφή JSON θα δυσκόλευε την ανάγνωση των περιβαλλοντικών δεικτών. Για αυτό τον λόγο αναπτύχθηκαν δύο τρόποι εξαγωγής και οπτικοποίησης των δεδομένων, μορφοποίηση στο τερματικό του χρήστη καθώς και εξαγωγή σε csv. Επιπλέον, η δυνατότητα εξαγωγής των αποτελεσμάτων σε μορφή αρχείων CSV είναι ένα ιδιαίτερα ωφέλιμο εργαλείο για τους ερευνητές, καθώς διευκολύνει την περαιτέρω επεξεργασία, ανάλυση και αξιοποίηση των δεδομένων από εξωτερικές επιστημονικές εφαρμογές.

10.4.1. Εξαγωγή στο τερματικό

Η συνάρτηση export(body: &str) δέχεται το ακατέργαστο σώμα της απόκρισης από τον διακομιστή,

αναλαμβάνοντας τη διοχέτευσή του στον κατάλληλο μηχανισμό μορφοποίησης. Για τη δομημένη και ευθυγραμμισμένη προβολή των δεδομένων σε περιβάλλον κονσόλας, αξιοποιείται η εξειδικευμένη βιβλιοθήκη `prettytable-rs` [38]. Για τη δομημένη, καθαρή και ευθυγραμμισμένη προβολή των δεδομένων σε περιβάλλον κονσόλας, αξιοποιείται η εξειδικευμένη βιβλιοθήκη `prettytable-rs` (`phsym/prettytable-rs`). Η συγκεκριμένη βιβλιοθήκη παρέχει όλα τα απαραίτητα δομικά στοιχεία (`Table`, `Row`, `Cell` και μακροεντολές μορφοποίησης όπως το `row!`[38]), επιτρέποντας τον δυναμικό σχηματισμό πινάκων απευθείας στο τερματικό. (Παράδειγμα χρήσης της βιβλιοθήκης στην Εικόνα 10.12).

```

+-----+-----+
| Title 1 | Title 2 |
+-----+-----+
| This is | foo     |
| a multiline |      |
| cell    |      |
+-----+-----+
| Yo dawg ;) You can even | +-----+-----+-----+ | | | | | |
| print tables            | | ABC   | DEFG | HIJKLMN | |
| into tables            | | +-----+-----+-----+ |
|                        | | | foobar | bar  | foo    | |
|                        | | +-----+-----+-----+ |
|                        | | | foobar2 | bar2 | foo2    | |
|                        | | +-----+-----+-----+ |
+-----+-----+

```

Εικόνα 10.12: Παράδειγμα εξαγωγής με την `prettytable-rs`

Αρχικά, η συνάρτηση πραγματοποιεί `deserialization` του κειμένου σε ένα γενικό αντικείμενο `serde_json::Value`. Στη συνέχεια, μέσω ειδικών συναρτήσεων ανίχνευσης, το σύστημα προσδιορίζει αυτόματα τον τύπο της απάντησης για την ανάλογη μορφοποίηση:

- `is_analysis`: Εντοπίζει εάν η απάντηση αφορά αποτέλεσμα επιστημονικής ανάλυσης (περιέχει πεδία όπως `STATUS`, `analyzation`, `result`).
- `is_history`: Εντοπίζει εάν η απάντηση περιέχει τη λίστα προηγούμενων αναλύσεων (πεδίο `history`).
- Γενικά αντικείμενα: Καλύπτουν περιπτώσεις όπως η επιτυχής σύνδεση (`/login`) ή η δημιουργία λογαριασμού (`/cacc`), καθώς και απλά μηνύματα σφάλματος (`format_plain`).

```

pub fn export(body: &str) {
    match serde_json::from_str::<Value>(body) {
        Ok(Value::Object(map)) => format_object(&map),
        Ok(Value::Array(arr)) => {
            let mut table = Table::new();
            table.add_row(row![b->"Array List"]);
            table.add_row(row![json_to_table(&Value::Array(arr))]);
            table.printstd();
        }
        Ok(other) => println!["{}", format_value(&other)],
        Err(_) => format_plain(body),
    }
}

```

Εικόνα 10.13: Παράδειγμα εξαγωγής τερματικό

10.4.1.1 Παρουσίαση Αποτελεσμάτων Ανάλυσης (print_analysis)

Όταν πρόκειται για την απάντηση μιας περιβαλλοντικής ανάλυσης (π.χ. NDVI), η συνάρτηση `print_analysis` κατασκευάζει έναν πίνακα δύο στηλών (Field και Value). Τα βασικά μεταδεδομένα (κατάσταση, τύπος δείκτη, περιοχή, χρονικό παράθυρο) τοποθετούνται σε ξεχωριστές γραμμές, ενώ τα πολυδιάστατα στατιστικά αποτελέσματα ενσωματώνονται δυναμικά μέσω της αναδρομής (Εικόνα 10.).

```
fn print_analysis(map: &serde_json::Map<String, Value>) {
    let mut table = Table::new();
    table.set_titles(row![b->"Field", b->"Value"]);

    add_kv_row(&mut table, "Status", map.get("STATUS"));
    add_kv_row(&mut table, "Index", map.get("analyzation"));

    if let Some(place) = map.get("place") {
        let val_str = truncate(format_value(place), 60);
        table.add_row(row![b->"Area", val_str]);
    }

    add_kv_row(&mut table, "Time", map.get("time"));

    if let Some(result) = map.get("result") {
        table.add_row(row![b->"Result", json_to_table(result)]);
    }

    println!("\n#### HellasCube Analysis Result ####");
    table.printstd();
}
```

Εικόνα 10.14: Τύπωση μοναδιαίας ανάλυσης στο τερματικό

10.4.1.2 Παρουσίαση Ιστορικού Χρήστη (print_history)

Για τη δομημένη παρουσίαση του ιστορικού των αναλύσεων, η συνάρτηση `print_history` επεξεργάζεται διαδοχικά τις διαθέσιμες εγγραφές, συνθέτοντας έναν αναλυτικό πίνακα. Ο πίνακας αυτός περιλαμβάνει τον αύξοντα αριθμό, τον τύπο της ανάλυσης, το χρονικό εύρος, το αναγνωριστικό του αιτήματος, καθώς και τα παραγόμενα αποτελέσματα. Επειδή τα δεδομένα αποθηκεύονται στη βάση ως συμβολοσειρές JSON, έχει αναπτυχθεί ένας ειδικός μηχανισμός που επιτρέπει την προβολή των αποτελεσμάτων με τη μορφή εμφωλευμένων πινάκων (Εικόνα 10.15).

```

if let Some(res) = obj.get("res_json") {
    // Unwrap stringified JSON dynamically
    let mut parsed = res.clone();
    while let Some(s) = parsed.as_str() {
        if let Ok(v) = serde_json::from_str::<Value>(s) {
            parsed = v;
        } else {
            break;
        }
    }
    table.add_row(row![
        (i + 1).to_string(),
        analysis,
        range,
        req_id,
        json_to_table(&parsed)
    ]);
} else {
    table.add_row(row![
        (i + 1).to_string(),
        analysis,
        range,
        req_id,
        "(no result)"
    ]);
}

```

Εικόνα 10.15: Λογική εμφωλευμένου json σε πίνακα του prettytable

10.4.2. Εξαγωγή σε CSV

Η κεντρική συνάρτηση `export_to_csv` (`json_body: &str`) αναλαμβάνει την διαδικασία της μόνιμης αποθήκευσης και εξαγωγής των δεδομένων σε μορφή πίνακα μέσω αρχείων τιμών διαχωρισμένων με κόμμα (CSV). Η διαδικασία αυτή υλοποιείται αξιοποιώντας την ισχυρή βιβλιοθήκη csv της Rust, εξασφαλίζοντας την απρόσκοπτη ενσωμάτωση των αποτελεσμάτων σε εξωτερικά εργαλεία ανάλυσης (όπως λογιστικά φύλλα ή προγράμματα Python/R).

10.4.2.1 Αποθήκευση αρχείου

Η τοποθεσία αποθήκευσης του αρχείου CSV ακολουθεί μια ιεραρχική σειρά επίλυσης (path resolution order):

1. Προσαρμοσμένη διαδρομή: Εάν έχει οριστεί ρητά διαδρομή από τον χρήστη.
2. Διαδρομή από τις Ρυθμίσεις (config): Το σύστημα ελέγχει τα αποθηκευμένα διαπιστευτήρια και τις ρυθμίσεις μέσω της συνάρτησης `get_creds()`. Εάν υπάρχει αποθηκευμένη διαδρομή στο πεδίο `csv_path`, αξιοποιείται άμεσα.
3. Εξ' ορισμού τοποθεσία (Home Directory): Εάν δεν βρεθεί ενεργή ρύθμιση ή καταχώρηση στο αρχείο ρυθμίσεων, η συνάρτηση `default_csv_path()` εντοπίζει αυτόματα τον κεντρικό φάκελο χρήστη (home directory) του λειτουργικού συστήματος και δημιουργεί εξ' ορισμού ένα αρχείο με την ονομασία `export.csv`.

```
// Determine the export path
let mut file_path: PathBuf = if let Ok(config) = get_creds() {
    if !config.csv_path.is_empty() {
        PathBuf::from(config.csv_path)
    } else {
        default_csv_path()?
    }
} else {
    default_csv_path()?
};

if file_path.is_dir()
    || file_path.to_string_lossy().ends_with(std::path::MAIN_SEPARATOR)
    || file_path.extension().is_none()
{
    file_path = file_path.join("hellas_cube_export.csv");
}
if let Some(parent) = file_path.parent() {
    if !parent.as_os_str().is_empty() && !parent.exists() {
        std::fs::create_dir_all(parent)
            .map_err(|e| format!("Failed to create directory {}: {}", parent.display(), e))
    }
}

let file_exists = file_path.exists();
```

Εικόνα 10.16: Καθορισμός export μονοπατιού

10.4.2.2 Λειτουργία Εισχωρίσης δεδομένων

Ένα σημαντικό χαρακτηριστικό του μηχανισμού εξαγωγής είναι ότι δεν αντικαθιστά τα προηγούμενα δεδομένα, αλλά λειτουργεί αθροιστικά.

- Χρησιμοποιώντας το `OpenOptions::new().create(true).append(true)` (Εικόνα 10.17), το σύστημα ανοίγει το αρχείο σε λειτουργία προσάρτησης (append mode). Τα νέα δεδομένα προστίθενται διαρκώς στο τέλος του ίδιου αρχείου, επιτρέποντας στον χρήστη να διατηρεί ενιαίο ιστορικό εξαγωγών.
- Μέσω της βιβλιοθήκης `csv::WriterBuilder`, ο κώδικας ελέγχει εάν το αρχείο υφίσταται ήδη (`file_path.exists()`) (Εικόνα 10.17). Εάν το αρχείο είναι ολοκαίνουργιο, εγγράφει αυτόματα την πρώτη γραμμή με τις επικεφαλίδες (Analysis, Range, Result Data), ενώ στις επόμενες κλήσεις παραλείπει τις επικεφαλίδες αποτρέποντας τη διπλή εγγραφή τους.

10.4.2.3 Εισαγωγή τύπων δεδομένων

Η συνάρτηση αξιοποιεί συνεχή match patterns για να διαχειριστεί τρεις διαφορετικές δομές δεδομένων εισόδου: λίστες ιστορικού (που περιέχουν το κλειδί history), απλούς πίνακες (Array), ή μεμονωμένα αντικείμενα αποτελεσμάτων (Object) (Εικόνα 10.17). Για κάθε εγγραφή, η βοηθητική συνάρτηση `write_csv_row` απομονώνει έξυπνα τα βασικά πεδία:

- Εντοπίζει τον τύπο της ανάλυσης (analyzation ή analysis).
- Ανακτά το χρονικό διάστημα (`date_range`).
- Επεξεργάζεται τα nested json αποτελεσμάτων, μετατρέποντάς τα σε μια καθαρή, ενιαία συμβολοσειρά JSON, η οποία αποθηκεύεται με ασφάλεια σε ένα κελί του αρχείου CSV.

```

let file_exists = file_path.exists();

let file = OpenOptions::new()
    .create(true)
    .append(true)
    .open(&file_path)
    .map_err(|e| format!("Failed to open CSV file: {}", e))?;

let mut wtr = WriterBuilder::new()
    .has_headers(!file_exists)
    .from_writer(file);

if !file_exists {
    wtr.write_record(&["Analysis", "Range", "Result Data"])
        .map_err(|e| format!("Failed to write CSV headers: {}", e))?;
}

match parsed {
    Value::Object(map) if map.contains_key("history") => {
        if let Some(arr) = map.get("history").and_then(|v| v.as_array()) {
            for item in arr {
                write_csv_row(&mut wtr, item)?;
            }
        }
    }
    Value::Array(arr) => {
        for item in arr {
            write_csv_row(&mut wtr, &item)?;
        }
    }
    Value::Object(_) => {
        write_csv_row(&mut wtr, &parsed)?;
    }
    _ => return Err("Invalid JSON structure for CSV export".to_string()),
}

wtr.flush().map_err(|e| format!("Failed to flush CSV writer: {}", e))?;

```

Εικόνα 10.17: Εξαγωγή δεδομένων ανάλογα την τιμή που πήρε ως δεδομένο

Με την ενσωμάτωση του αυτοματοποιημένου μηχανισμού εξαγωγής (CSV export) και την αυστηρή επικύρωση των δεδομένων εισόδου στο επίπεδο του πελάτη (Fail-Fast), ο κύκλος ανάπτυξης του λογισμικού Hellas Cube ολοκληρώνεται. Όλα τα προαναφερθέντα υποσυστήματα η σχεσιακή βάση δεδομένων, το υποσύστημα ανάλυσης, το API και το CLI συνενώνονται πλέον σε μια ενιαία επιχειρησιακή οντότητα, η πραγματική συμπεριφορά και ευχρηστία της οποίας επαληθεύεται μέσω πρακτικών σεναρίων χρήσης.

Κεφάλαιο 11ο: Χρήση του συστήματος

11.1 Εγκατάσταση εργαλείου

Έχοντας αναλύσει διεξοδικά το θεωρητικό υπόβαθρο, τις απαιτήσεις, την αρχιτεκτονική και την εσωτερική υλοποίηση του κεντρικού διακομιστή (Rust API) και του υποσυστήματος ανάλυσης (Python/Open Data Cube), το παρόν κεφάλαιο εστιάζει στην πρακτική αξιοποίηση του συστήματος από την οπτική γωνία του τελικού χρήστη. Ο στόχος αυτού του οδηγού βήμα προς βήμα (walkthrough) είναι να γεφυρώσει τη θεωρία λογισμικού με την πραγματική εμπειρία χρήσης αποδεικνύοντας πώς το Hellas Cube CLI εκτελείται και χειρίζεται.

Προτού ο χρήστης μπορεί να αλληλεπιδράσει με την πλατφόρμα και να εκτελέσει τις επιθυμητές περιβαλλοντικές αναλύσεις, απαιτείται η τοπική μεταγλώττιση (build) και εγκατάσταση του εργαλείου Hellas Cube CLI στον υπολογιστή του. Αξιοποιώντας τις εγγενείς δυνατότητες της γλώσσας Rust και του επίσημου διαχειριστή πακέτων της (Cargo). Η διαδικασία προετοιμασίας και εγκατάστασης ολοκληρώνεται σε τρία βασικά βήματα:

1. Βελτιστοποιημένη Μεταγλώττιση (Release Build): Αρχικά, εντός του φακέλου του πηγαίου κώδικα, εκτελείται η εντολή μεταγλώττισης παραγωγής και παράγει ένα αυτόνομο, δυαδικό αρχείο υψηλής ταχύτητας στον φάκελο `./target/release/`: `cargo build --release`
2. Εγκατάσταση στο Σύστημα (cargo install): Για να καταστεί το εργαλείο προσβάσιμο από οποιοδήποτε σημείο του τερματικού, ο χρήστης το εγκαθιστά τοπικά στο σύστημά του εκτελώντας: `cargo install --path ~/Projects/Hellas_Cube/cli`
3. Επαλήθευση και Δοκιμή Λειτουργίας: Για να επιβεβαιωθεί η ορθή εγκατάσταση και η επικοινωνία με το εργαλείο, ο χρήστης μπορεί να ζητήσει την προβολή των διαθέσιμων εντολών και της βοήθειας:



Εικόνα 11.1: Εκτέλεση Help εντολής, logo συστήματος

11.2 Σενάριο Α : Χρήση Χωρίς Λογαριασμό

Το πρώτο και πιο άμεσο σενάριο αφορά την αλληλεπίδραση με την πλατφόρμα χωρίς να απαιτείται η προηγούμενη εγγραφή. Το συγκεκριμένο σενάριο απευθύνεται σε ερευνητές ή πολίτες που επιθυμούν να λάβουν γρήγορα και στοχευμένα αποτελέσματα για γνωστές γεωγραφικές οντότητες (όπως δήμους, λίμνες ή περιοχές ενδιαφέροντος) αξιοποιώντας τον μηχανισμό προκαθορισμένων στόχων (TARGET). Για την εκκίνηση του υπολογισμού ενός περιβαλλοντικού δείκτη όπως ο δείκτης βλάστησης NDVI μιας συγκεκριμένης περιοχής (π.χ. τη Θεσσαλονίκη) σε ένα συγκεκριμένο χρονικό παράθυρο, εκτελεί την αντίστοιχη εντολή από το τερματικό:

`HellasCube ndvi --area "Municipality of Thessaloniki" --from "10-05-2022" --till "10-07-2022"`

Η παράμετρος `--area` ορίζει τη γεωγραφική περιοχή ενδιαφέροντος μέσω των ενσωματωμένων διανυσματικών ορίων (geojson), ενώ οι παράμετροι `--from` και `--till` καθορίζουν το χρονικό εύρος της αναζήτησης (από 10 Μαΐου 2022 έως 10 Ιουλίου 2022) για την ανάκτηση και την επεξεργασία των δορυφορικών δεδομένων, παράγοντας το ακόλουθο αποτέλεσμα:

```
christossapounas@192:~$ HellasCube ndvi --area "Municipality of Thessaloniki" --from "10-05-2022" --till "10-07-2022"

#### HellasCube Analysis Result ####
+-----+-----+
| Field | Value |
+-----+-----+
| Status | OK |
+-----+-----+
| Index | NDVI |
+-----+-----+
| Area | Municipality of Thessaloniki |
+-----+-----+
| Time | 11.03s |
+-----+-----+
| Result | coverage_pct | 81.91 % |
|         | index         | NDVI    |
|         | max           | 0.929   |
|         | mean          | 0.121   |
|         | median        | 0.071   |
|         | min           | -0.545  |
|         | p10           | -0.164  |
|         | p25           | -0.064  |
|         | p75           | 0.27    |
|         | p90           | 0.526   |
|         | std           | 0.251   |
|         | total_px      | 3456462 |
|         | valid_px      | 2831071 |
+-----+-----+
```

Εικόνα 11.2: Εκτέλεση εντολής για αίτημα περιβαλλοντικής ανάλυσης

Με βάση τα αποτελέσματα της εκτέλεσης και τις κλίμακες του δείκτη NDVI (Πίνακας 3.7):

- Ο μέσος όρος (mean: 0.121) και η διάμεσος (median: 0.071) κυμαίνονται στο διάστημα [0, 0.2], γεγονός που αποτυπώνει χαρακτηριστικά τον αστικό χαρακτήρα του Δήμου Θεσσαλονίκης (επικράτηση δομημένου περιβάλλοντος, ασφάλτου και άγονου εδάφους).
- Η τιμή του 75ου εκατοστημορίου (p75: 0.27) και η μέγιστη τιμή (max: 0.929) ξεπερνούν το όριο του 0.2, αποκαλύπτοντας την παρουσία αστικής βλάστησης.

11.3 Σενάριο Β : Διαχείριση Λογαριασμού

Το δεύτερο σενάριο αφορά την προηγμένη χρήση της πλατφόρμας, όπου ο ερευνητής μεταβαίνει από τον ρόλο του ανώνυμου επισκέπτη σε χρήστη. Αυτή η ροή επιτρέπει τη δημιουργία λογαριασμού, την ασφαλή σύνδεση και τη δήλωση προσαρμοσμένων γεωχωρικών ορίων.

Αρχικά ο χρήστης εγγράφεται στο σύστημα παρέχοντας το email και τον κωδικό πρόσβασής του (Εικόνα 11.3) εμφανίζονται τα δεδομένα που έδωσε. Στην συνέχεια για να αποκτήσει δικαιώματα πρόσβασης, ο χρήστης συνδέεται με τα στοιχεία του και ο διακομιστής επικυρώνει τα στοιχεία, δημιουργεί ένα μοναδικό κλειδί (API Key) και το αποθηκεύει τοπικά στον υπολογιστή του χρήστη για μελλοντική χρήση. Τέλος Ο χρήστης δηλώνει τα όρια μιας δικής του περιοχής ενδιαφέροντος μέσω ενός τοπικού αρχείου GeoJSON και το αρχείο επικυρώνεται και καταχωρείται στη βάση δεδομένων, συνδεδεμένο πλέον αποκλειστικά με το προφίλ του χρήστη. Ο χρήστης μπορεί να επιβεβαιώσει την κατάσταση του ανά πάσα στιγμή με την εντολή HellasCube info. (Εικόνα 11.3)

```
christossapounas@192:~$ HellasCube cacc --email example@gmail.com --password thesisexample
User-id: {"user_id":"1518bb7f-1d0d-4df9-b4c2-00ceb5ed91f0"}
User created!
Email: example@gmail.com
Password: thesisexample
christossapounas@192:~$ HellasCube login --email example@gmail.com --password thesisexample
Credentials saved.
Api key generated: "{\"api_key\":\"hc_QKGhbatRwpPwcgeXXunTqCj8rdl1l3w9APKSywbYYJ8XG7d\"}"
(always store API KEY, in case of emergency)
christossapounas@192:~$ HellasCube declare-geo-json --path /home/christossapounas/specific_square.geojson
GeoJson declared in our Database: /home/christossapounas/specific_square.geojson
christossapounas@192:~$ HellasCube info
Email: example@gmail.com
API key: hc_QKGhbatRwpPwcgeXXunTqCj8rdl1l3w9APKSywbYYJ8XG7d
GeoJSON path: /home/christossapounas/specific_square.geojson
```

Εικόνα 11.3: Εντολές χρήστη

11.4 Σενάριο Γ : Χρήση με Λογαριασμό

Μετά την ολοκλήρωση της εγγραφής, της σύνδεσης και της δήλωσης του προσωπικού αρχείου GeoJSON (όπως περιγράφηκε στο προηγούμενο σενάριο), ο χρήστης έχει πλέον τη δυνατότητα να εκτελέσει προηγμένες περιβαλλοντικές αναλύσεις βασισμένες αποκλειστικά στον ιδιωτικό του γεώχωρο, χωρίς να χρειάζεται να ορίζει εκ νέου γεωγραφικά όρια ή πόλεις. Αξιοποιώντας το τοπικά αποθηκευμένο API Key και ενεργοποιώντας τη σημαία --default, ο χρήστης ζητά τον υπολογισμό του δείκτη βλάστησης NDVI για συγκεκριμένο χρονικό διάστημα (Εικόνα 11.4).

```
christossapounas@192:~$ HellasCube ndvi --default --from "10-05-2022" --till "10-06-2022"

##### HellasCube Analysis Result #####
+-----+-----+
| Field | Value |
+-----+-----+
| Status | OK |
+-----+-----+
| Index | NDVI |
+-----+-----+
| Area | DEFAULT |
+-----+-----+
| Time | 30.57s |
+-----+-----+
| Result | coverage_pct | 78.47 % |
|         | index         | NDVI |
|         | max           | 0.988 |
|         | mean          | -0.093 |
|         | median        | -0.035 |
|         | min           | -0.991 |
|         | p10           | -0.217 |
|         | p25           | -0.079 |
|         | p75           | -0.002 |
|         | p90           | 0.026 |
|         | std           | 0.212 |
|         | total_px      | 17604132 |
|         | valid_px      | 13813569 |
+-----+-----+
```

Εικόνα 11.4: Εκτέλεση εντολής με προσωπική περιοχή ενδιαφέροντος

11.5 Σενάριο Δ : Εξαγωγή Ιστορικού

Το τελευταίο σενάριο αφορά την ανάκτηση του ιστορικού των προηγούμενων αναλύσεων του χρήστη, καθώς και τον μηχανισμό μόνιμης αποθήκευσης και μαζικής εξαγωγής δεδομένων.

Προτού ενεργοποιηθεί η εξαγωγή, ο χρήστης μπορεί να ορίσει ρητά την προσαρμοσμένη διαδρομή αποθήκευσης των αρχείων CSV στο σύστημά του (Εικόνα 11.5). Στην συνέχεια με την προσθήκη της ειδικής σημαίας --csv, ο χρήστης μπορεί να ζητήσει την προβολή του ιστορικού του και ταυτόχρονα την αυτόματη εξαγωγή των αποτελεσμάτων στο αρχείο CSV (Εικόνα 11.6).

```
christossapounas@192:~$ HellasCube csv-path --path /home/christossapounas/Documents/csv_exports_hellas_cube
CSV export path saved as: /home/christossapounas/Documents/csv_exports_hellas_cube
christossapounas@192:~$ HellasCube history --csv

#### History (2 record(s)) ####
+-----+-----+-----+-----+-----+
| # | Analysis | Range | Request ID | Result |
+-----+-----+-----+-----+-----+
| 1 | NDVI | [2022-10-05,2022-10-06) | 5582db06-88f5-4... | STATUS OK  
analyzation NDVI  
place DEFAULT  
result coverage_pct 78.47 %  
index NDVI  
max 0.988  
mean -0.093  
median -0.035  
min -0.991  
p10 -0.217  
p25 -0.079  
p75 -0.002  
p90 0.026  
std 0.212  
total_px 17604132  
valid_px 13813569  
time 30.57s |
+-----+-----+-----+-----+-----+
| 2 | NDVI | [2022-10-05,2022-10-06) | 30281962-d2e1-4... | STATUS OK  
analyzation NDVI  
place DEFAULT  
result coverage_pct 78.47 %  
index NDVI  
max 0.988  
mean -0.093  
median -0.035  
min -0.991  
p10 -0.217  
p25 -0.079  
p75 -0.002  
p90 0.026  
std 0.212  
total_px 17604132  
valid_px 13813569  
time 32.51s |
+-----+-----+-----+-----+-----+

Successfully appended to CSV
```

Εικόνα 11.5: Αποτέλεσμα ιστορικού στο τερματικό

	A	B	
1	Analysis	Range	Result Data
2	NDVI	[2022-10-05,2022-10-06)	{"STATUS":"OK","analyzation":"NDVI","place":"DEFAULT","result":{"coverage_pct":"78.47 %","index":"NDVI","max":"0.988","mean":"-0.093","median":"-0.035","min":"-0.991","p10":"-0.217","p25":"-0.079","p75":"-0.002","p90":"0.026","std":"0.212","total_px":"17604132","valid_px":"13813569","time":"30.57s"}}
3	NDVI	[2022-10-05,2022-10-06)	{"STATUS":"OK","analyzation":"NDVI","place":"DEFAULT","result":{"coverage_pct":"78.47 %","index":"NDVI","max":"0.988","mean":"-0.093","median":"-0.035","min":"-0.991","p10":"-0.217","p25":"-0.079","p75":"-0.002","p90":"0.026","std":"0.212","total_px":"17604132","valid_px":"13813569","time":"32.51s"}}
4			
5			
6			
7			
8			
9			
10			

Εικόνα 11.6: Αποτέλεσμα ιστορικού σε csv

Η επιτυχής ολοκλήρωση της μαζικής εξαγωγής του ιστορικού σε αρχείο CSV επιβεβαιώνει την λειτουργικότητα ολόκληρης της αλυσίδας επεξεργασίας δεδομένων, από την υποβολή της απλής εντολής στο τερματικό του χρήστη, έως την πολύπλοκη ανάκτηση και ανάλυση της φασματικής πληροφορίας από το νέφος της AWS. Έτσι το σύστημα αποδεικνύει την χρησιμότητά του

Κεφάλαιο 12ο: Συμπεράσματα και Μελλοντικές Επεκτάσεις

12.1 Συμπεράσματα

Η παρούσα διπλωματική εργασία ξεκίνησε με την κεντρική ιδέα να συνδυαστεί η Περιβαλλοντική Επιστήμη με τη Μηχανική Λογισμικού, ώστε να μην σταματάμε απλώς στην παραγωγή των δεδομένων, αλλά να προχωράμε στην εξαγωγή ουσιαστικών πορισμάτων και πληροφοριών. Η ανάπτυξη του συστήματος Hellas Cube αποτέλεσε ένα εξαιρετικά απαιτητικό και περίπλοκο εγχείρημα, μέσα από το οποίο αποκόμισα τεράστια εμπειρία και γνώση. Στο πλαίσιο της παρούσας διπλωματικής εργασίας, μελετήθηκαν και υλοποιήθηκαν σε βάθος ζητήματα που αφορούν την επεξεργασία περιβαλλοντικών τύπων και υπολογιστικών δεικτών (όπως NDVI, NBR κ.ά.), καθώς και την ανάλυση δορυφορικών δεδομένων μεγάλου όγκου από αποστολές όπως ο Sentinel-2 και ο Landsat.

Ένα από τα μεγαλύτερα επιτεύγματα της εργασίας ήταν η επιτυχής δημιουργία ενός πλήρως συγχρονισμένου συστήματος, αξιοποιώντας και συνδυάζοντας αρμονικά πολλές διαφορετικές γλώσσες προγραμματισμού. Αναπτύχθηκε μια αρχιτεκτονική τριών επιπέδων: μια ταχύτατη και απλή διεπαφή γραμμής εντολών γραμμένη σε Rust, η οποία επικοινωνεί με το API του διακομιστή. Το API με τη σειρά του διαχειρίζεται και καλεί τα κατάλληλα scripts στην Python, αξιοποιώντας το ισχυρό framework του Open Data Cube (ODC) για την επεξεργασία, σε συνεργασία με τη PostgreSQL για την αποθήκευση των απαραίτητων ευρετηρίων (indexes) των δορυφορικών δεδομένων. Οι προκλήσεις που κλήθηκα να αντιμετωπίσω ήταν πολλές, με κυριότερη την ανάπτυξη ενός server ικανού να ανταπεξέλθει στις τεράστιες τεχνικές απαιτήσεις. Η διαχείριση των requests μέσω του API και η ελαχιστοποίηση του χρόνου επεξεργασίας απαιτούσαν την άριστη χρήση εργαλείων και τεχνικών, ώστε η τελική εμπειρία του χρήστη να βρίσκεται στο υψηλότερο δυνατό επίπεδο.

Εν κατακλείδι, οι στόχοι που τέθηκαν στο ξεκίνημα της Διπλωματικής Εργασίας επιτεύχθηκαν με επιτυχία. Δημιουργήθηκε ένα λογισμικό ανοιχτού κώδικα, ειδικά εστιασμένο στην πραγματικότητα και τις ανάγκες του Ελλαδικού χώρου.

12.2 Μελλοντικές Επεκτάσεις

Παρά το γεγονός ότι το σύστημα αναπτύχθηκε επιτυχώς και εκπληρώνει τους αρχικούς του στόχους, η φύση της τεχνολογίας και των περιβαλλοντικών επιστημών απαιτεί συνεχή εξέλιξη. Με βάση την αρχιτεκτονική του Hellas Cube και τις ανάγκες της ερευνητικής κοινότητας, προτείνονται οι παρακάτω μελλοντικές επεκτάσεις και βελτιώσεις του συστήματος:

1. Φιλοξενία του Συστήματος σε VPS (Virtual Private Server): Μια άμεση μελλοντική επέκταση είναι η μεταφορά και φιλοξενία (hosting) του backend API και της βάσης δεδομένων σε έναν απομακρυσμένο διακομιστή (VPS) ή σε υποδομές cloud. Αυτό θα απελευθερώσει τον χρήστη από την ανάγκη να τρέχει τον server τοπικά στον δικό του υπολογιστή και θα εξασφαλίσει την αδιάλειπτη (24/7) διαθεσιμότητα της εφαρμογής για πολλαπλούς ταυτόχρονους χρήστες.
2. Επέκταση Λειτουργιών στο Σύστημα Χρηστών: Το τρέχον σύστημα αυθεντικοποίησης και διαχείρισης γεωχώρου (GeoJSON) μπορεί να εμπλουτιστεί με επιπλέον λειτουργίες.

Προβλέπεται η ανάπτυξη δυνατοτήτων όπως η δημιουργία ομάδων (teams) για κοινή χρήση γεωχωρικών δεδομένων μεταξύ ερευνητών, η αποθήκευση πολλαπλών προσαρμοσμένων περιοχών (custom areas) ανά χρήστη, καθώς και η ανάπτυξη συστήματος ειδοποιήσεων όταν ολοκληρώνονται χρονοβόρες δορυφορικές αναλύσεις.

3. Εξαγωγή Δεδομένων σε Πολλαπλά Μορφότυπα (Multi-format Export): Πέραν της προβολής σε πίνακες κονσόλας και της εξαγωγής σε μορφή CSV που υποστηρίζεται ήδη, το εργαλείο μπορεί να επεκταθεί ώστε να εξάγει δεδομένα σε όσο το δυνατόν περισσότερα format. Η υποστήριξη μορφοτύπων όπως GeoTIFF, NetCDF και προηγμένων JSON δομών θα επιτρέψει την άμεση εισαγωγή των αποτελεσμάτων σε εξωτερικά λογισμικά Γεωγραφικών Συστημάτων Πληροφοριών (GIS) για περαιτέρω χαρτογράφηση.
4. Ενσωμάτωση Περισσότερων Περιβαλλοντικών Τύπων και Αναλύσεων: Καθώς το framework του Open Data Cube είναι εξαιρετικά ευέλικτο, ο υπολογιστικός πυρήνας της Python μπορεί να εμπλουτιστεί. Στόχος είναι η ενσωμάτωση όσο το δυνατόν περισσότερων περιβαλλοντικών δεικτών (π.χ. δείκτες ξηρασίας, υγρασίας εδάφους, δείκτες ανίχνευσης υδάτινων σωμάτων ή καμένων εκτάσεων) και σύνθετων αναλύσεων, ώστε το Hellas Cube να καλύπτει μια ακόμα ευρύτερη γκάμα σεναρίων για την παρακολούθηση του περιβάλλοντος.
5. Ενσωμάτωση Περισσότερων Περιβαλλοντικών Τύπων: Καθώς το framework του Open Data Cube είναι εξαιρετικά ευέλικτο, ο υπολογιστικός πυρήνας του υποσυστήματος μπορεί να εμπλουτιστεί. Στόχος είναι η εγγενής ενσωμάτωση όσο το δυνατόν περισσότερων έτοιμων περιβαλλοντικών δεικτών (π.χ. δείκτες ξηρασίας, υγρασίας εδάφους, ανίχνευσης υδάτινων σωμάτων) και σύνθετων αναλύσεων.
6. Δυναμικές Αναλύσεις: Στην τρέχουσα υλοποίηση του Hellas Cube, αποτελεί λειτουργικό περιορισμό το γεγονός ότι ο χρήστης περιορίζεται αυστηρά στην επιλογή μιας προκαθορισμένης λίστας δεικτών. Ως εκ τούτου, κρίνεται ιδιαίτερα χρήσιμη η ανάπτυξη ενός μηχανισμού που θα επιτρέπει στους προχωρημένους ερευνητές να εισάγουν δυναμικά τους δικούς τους, προσαρμοσμένους μαθηματικούς τύπους μέσω του CLI, αλληλεπιδρώντας ελεύθερα με τα διαθέσιμα φασματικά κανάλια.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

[40] T. C. Lethbridge και R. Laganière, *Μηχανική Αντικειμενοστραφούς Λογισμικού: Πρακτική Ανάπτυξη Λογισμικού με τις Γλώσσες UML και Java*, 2η Έκδ. Εκδόσεις Τζιόλα, 2016, ISBN: 978-960-418-598-6

Journal Articles

[1] A. Lewis et al., “The Australian Geoscience Data Cube — Foundations and lessons learned,” *Remote Sensing of Environment*, vol. 202, pp. 276–292, Apr. 2017, doi: 10.1016/j.rse.2017.03.015.

[4] A. Munteanu, “Data Cubes and Cloud-Native Environments for Earth Observation: An Overview,” *Scalable Computing Practice and Experience*, vol. 25, no. 6, Oct. 2024, doi: 10.12694/scpe.v25i6.4999.

[8] V. Fotakidis, T. Roustanis, K. Panayiotou, I. Chrysafis, E. Fitoka, and G. Mallinis, “The EL-BIOS Earth Observation Data Cube for supporting biodiversity monitoring in Greece,” *Remote Sensing*, vol. 16, no. 20, p. 3771, Oct. 2024, doi: 10.3390/rs16203771.

[9] Q. Zhao et al., “An overview of the applications of Earth observation satellite data: Impacts and future trends,” *Remote Sensing*, vol. 14, no. 8, p. 1863, Apr. 2022, doi: 10.3390/rs14081863.

[10] D. DKovács, J. Musial, J. Bojanowski, D. Clarijs, J. De La Mar, and A. Zlinszky, “Copernicus Data Space Ecosystem establishes public cloud processing for earth observation data,” *Scientific Data*, vol. 13, no. 1, Feb. 2026, doi: 10.1038/s41597-026-06765-8.

[11] M. Drusch et al., “Sentinel-2: ESA’s Optical High-Resolution Mission for GMES Operational Services,” *Remote Sensing of Environment*, vol. 120, pp. 25–36, Feb. 2012, doi: 10.1016/j.rse.2011.11.026.

[13] M. A. Wulder et al., “Fifty years of Landsat science and impacts,” *Remote Sensing of Environment*, vol. 280, p. 113195, Jul. 2022, doi: 10.1016/j.rse.2022.113195.

[16] M. Berger, J. Moreno, J. A. Johannessen, P. F. Levelt, and R. F. Hanssen, “ESA’s sentinel missions in support of Earth system science,” *Remote Sensing of Environment*, vol. 120, pp. 84–90, Feb. 2012, doi: 10.1016/j.rse.2011.07.023.

[18] K. Tarrío et al., “Comparison of cloud detection algorithms for Sentinel-2 imagery,” *Science of Remote Sensing*, vol. 2, p. 100010, Oct. 2020, doi: 10.1016/j.srs.2020.100010.

[26] N. Mueller et al., “Water observations from space: Mapping surface water from 25 years of Landsat imagery across Australia,” *Remote Sensing of Environment*, vol. 174, pp. 341–352, Nov. 2015, doi: 10.1016/j.rse.2015.11.003.

[28] N. D. Matsakis and F. S. Klock, “The rust language,” *The Rust Language*, pp. 103–104, Oct. 2014, doi: 10.1145/2663171.2663188.

[29] J. G. Politz et al., “Python,” *ACM SIGPLAN Notices*, vol. 48, no. 10, pp. 217–232, Oct. 2013, doi: 10.1145/2544173.2509536.

Paper in Conference Proceedings

[2] B. Killough, "Overview of the Open Data Cube Initiative," *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, Valencia, Spain, 2018, pp. 8629-8632, doi: 10.1109/IGARSS.2018.8517694.

[3] B. Killough, A. Siqueira and G. Dyke, "Advancements in the Open Data Cube and Analysis Ready Data — Past, Present and Future," *IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium*, Waikoloa, HI, USA, 2020, pp. 3373-3375, doi: 10.1109/IGARSS39084.2020.9324712.

[5] D. Gavin et al., "Digital Earth Australia - from Satellite Data to Better Decisions," *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, Valencia, Spain, 2018, pp. 8633-8635, doi: 10.1109/IGARSS.2018.8518160.

[6] F. Yuan et al., "Advancing Application Development with Analysis-Ready Data in the Digital Earth Africa Program," *IGARSS 2024 - 2024 IEEE International Geoscience and Remote Sensing Symposium*, Athens, Greece, 2024, pp. 2487-2489, doi: 10.1109/IGARSS53475.2024.10642683.

[7] “The Euro Data Cube - from algorithm prototyping to large scale processing,” *IEEE Conference Publication | IEEE Xplore*. <https://ieeexplore.ieee.org/document/10282602>.

[15] “The Copernicus Programme and its Climate Change Service,” *IEEE Conference Publication | IEEE Xplore*. <https://ieeexplore.ieee.org/abstract/document/8518067>

[17] M. Main-Knorn, B. Pflug, J. Louis, V. Debaecker, U. Müller-Wilm, and F. Gascon, “Sen2Cor for Sentinel-2,” *Sen2Cor for Sentinel-2*, p. 3, Oct. 2017, doi: 10.1117/12.2278218.

Internet Site

[12] H. J. Kramer, “SPOT-6 and SPOT-7 (Azersky),” *eoPortal*.
<https://www.eoportal.org/satellite-missions/spot-6-7#spot-6-and-spot-7-azersky-commercial-imaging-constellation>

[14] B. L. Markham et al., “Landsat program,” *NASA Technical Reports Server (NTRS)*, Sep. 01,

2016. <https://ntrs.nasa.gov/citations/20170005836>

[19] G. Team, “Red Edge in Remote Sensing: applications and advantages,” Geopera, Mar. 03, 2025. <https://geopera.com/blog/red-edge-remote-sensing>

[20] E. C. Alex, K. V. Ramesh, and S. Hari, “Quantification and understanding the observed changes in land cover patterns in Bangalore,” ResearchGate, May 2017, [Online]. Available: https://www.researchgate.net/publication/316471841_Quantification_and_understanding_the_observed_changes_in_land_cover_patterns_in_Bangalore

[21] “Normalized difference chlorophyll Index - ClimateEngine.org.” <https://climateengine.org/datasets/vegetation/normalized-difference-chlorophyll-index/>

[22] “NDWI: Index Formula, value range, and uses in agriculture,” EOS Data Analytics, Nov. 07, 2023. <https://eos.com/make-an-analysis/ndwi/>

[23] Geopera, “NDBI — Normalized Difference Built-up Index,” Geopera, Jun. 01, 2025. <https://docs.geopera.com/spectral-indices/ndbi>

[24] S.-H. B. Sinergise, “Normalised Difference Snow Index, NDSI,” Sentinel Hub Custom Scripts. <https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/ndsi/>

[25] “Normalized Burn Ratio (NBR) | UN-SPIDER Knowledge Portal.” <https://un-spider.org/advisory-support/recommended-practices/recommended-practice-burn-severity/in-detail/normalized-burn-ratio>

[27] GeeksforGeeks, “Fundamentals of Software architecture Design,” GeeksforGeeks, Jul. 23, 2025. <https://www.geeksforgeeks.org/software-engineering/fundamentals-of-software-architecture-design/>

[30] Codecademy, “Understanding Polymorphism in Python (With Examples),” Codecademy. <https://www.codecademy.com/article/understanding-polymorphism-in-python>

[31] “Overview — Open Data Cube 1.9.22.dev47+gf96f0c590 documentation.” <https://opendatacube.readthedocs.io/en/latest/about-core-concepts/index.html#high-level-odc-overview>

[32] “The Open Data Cube — Open Data Cube 1.9.22.dev50+g8051b943b documentation.” <https://opendatacube.readthedocs.io/en/latest/index.html>

[33] “conda-forge/miniforge: A conda-forge distribution.,” GitHub. <https://github.com/conda-forge/miniforge>

[34] Digitalearthafrika, “config/products at master · digitalearthafrika/config,” GitHub. <https://github.com/digitalearthafrika/config/tree/master/products>

[35] “subprocess — Subprocess management,” Python Documentation.

<https://docs.python.org/3/library/subprocess.html>

[36] “AWS Console - signup.” https://signin.aws.amazon.com/signup?request_type=register

[37] “axum - Rust.” <https://docs.rs/axum/latest/axum/>

[38] “phsym/prettytable-rs: A rust library to print aligned and formatted tables,” GitHub.

<https://github.com/phsym/prettytable-rs>

[39] “overpass turbo.” <https://overpass-turbo.eu/>

[41] “Normalized difference moisture index,” *USGS*.
<https://www.usgs.gov/landsat-missions/normalized-difference-moisture-index>

[42] “Turbidity in wetlands — Digital Earth Africa 2026 documentation.”
https://docs.digitalearthafrika.org/en/latest/sandbox/notebooks/Real_world_examples/Wetland_turbidity.html

ΠΑΡΑΡΤΗΜΑ Α: Πηγαίος Κώδικας του Συστήματος Hellas Cube

Το σύνολο του πηγαίου κώδικα που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας φιλοξενείται στην πλατφόρμα GitHub. Το αποθετήριο είναι διαθέσιμο στον ακόλουθο σύνδεσμο:

[Hellas Cube Github Repository](#)