ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάκτηση Δεδομένων με μεθόδους Web Scrapping

Του φοιτητή
Χατσατουριάν Αλέξανδρου
Αρ. Μητρώου: 154565

Επιβλέπων
Ονοματεπώνυμο Μιχαήλ
Σαλαμπασης
Βαθμίδα Καθηγητή

Ημερομηνία 19-06-2025

Τίτλος Δ.Ε. Web Scrapping με χρηση Node.js
Κωδικός Δ.Ε. 23152
Ονοματεπώνυμο φοιτητή/τών Αλέξανδρος Χατσατουριάν
Ονοματεπώνυμο εισηγητή Μιχαήλ Σαλαμπασης
Ημερομηνία ανάληψης Δ.Ε. 16-03-2023
Ημερομηνία περάτωσης Δ.Ε. …

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

Αφιέρωση

Αφιερώνω την πτυχιακή μου εργασία στους γονείς μου.

Πρόλογος

Η παρούσα πτυχιακή εργασία εκπονήθηκε στο πλαίσιο των σπουδών μου στο Διεθνές Πανεπιστήμιο της Ελλάδος, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων, με σκοπό την απόκτηση του πτυχίου μου.

Το διαδίκτυο αποτελεί σήμερα το κυρίαρχο μέσο επικοινωνίας και πληροφόρησης. Ο όγκος των δεδομένων που διακινούνται καθημερινά είναι τεράστιος και πολλές φορές περιέχει πληροφορίες μεγάλης αξίας. Σε αυτό το πλαίσιο, η τεχνική του web scraping επιτρέπει την αυτοματοποιημένη συλλογή αυτών των πληροφοριών, με στοχευμένο τρόπο.

Ο στόχος της πτυχιακής μου εργασίας ήταν η επιτυχής συλλογή δεδομένων από συγκεκριμένες πηγές. Για την επίτευξη αυτού του στόχου, κρίθηκε απαραίτητη η ανάπτυξη δύο εργαλείων scraping . Το πρώτο εργαλειο αφορά την συλλογή δεδομένων για όλα τα φυσικά πρόσωπα του ordino.gr (όπως ηθοποιοί, σκηνοθέτες, παραγωγοί και άλλοι συντελεστές) που συμμετέχουν σε θεατρικές παραστάσεις μέσω της αντιστοχης ιστοσελίδας. Το δεύτερο εργαλείο αφορούσε την εξόρυξη πληροφοριών για επιχειρήσεις που δραστηριοποιούνται στην Ελλάδα, από τον ιστότοπο businessportal.gr.

Επέλεξα το συγκεκριμένο θέμα γιατί θεωρώ πως μία εφαρμογή χωρίς δεδομένα είναι σαν ένα κτίριο ”γυμνό” μονο με τα θεμέλια δηλαδή. Επιπλέον, ήθελα πολύ να ενισχύσω τις γνώσεις μου πάνω στην JavaScript, καθώς εργάζομαι ως full stack developer και ήθελα να εξοικειωθώ περισσότερο με βιβλιοθήκες και εργαλεία που σχετίζονται με το web scraping αλλά και με το σχεδιασμό και ανάπτυξη του οπτικού περιβάλλοντος που αντικρύ ο χρήστης που θα χρησιμοποιήσει την εφαρμογή μας (UI representation).

Κατά την υλοποίηση της εργασίας αντιμετώπισα αρκετές τεχνικές δυσκολίες, ειδικά όσον αφορά τη διαχείριση της δομής των ιστοσελίδων, τον εντοπισμό των απαραίτητων στοιχείων στο DOM και την αντιμετώπιση πιθανών αποκλεισμών από τα sites. Παρ’ όλα αυτά, η διαδικασία αυτή με βοήθησε να εξελιχθώ και να αποκτήσω ακόμα περισσότερη εμπειρία στον τομέα της ανάπτυξης web εφαρμογών.

Περίληψη

Σκοπός αυτής της πτυχιακής εργασίας ήταν να δημιουργηθεί ένα σύστημα που να μπορεί να συλλέγει αυτόματα δεδομένα (μέσω του web scraping) από δύο διαφορετικές ιστοσελίδες: το **ordino.gr** και το **businessportal.gr**. Οι πληροφορίες που αντλούνται σχετίζονται είτε με θεατρικές παραστάσεις και τους συντελεστές τους, είτε με επιχειρήσεις που δραστηριοποιούνται στην Ελλάδα.

Για την υλοποίηση του συστήματος χρησιμοποιήθηκε η τεχνολογία **Node.js** στο backend και η **React.js** στο frontend, ώστε να υπάρχει πλήρης λειτουργικότητα τόσο στην επεξεργασία όσο και στην προβολή των δεδομένων.Επειδή κάθε ιστοσελίδα παρουσίαζε τις πληροφορίες με διαφορετικό τρόπο, χρειάστηκε να εφαρμόσουμε δύο ξεχωριστές μεθόδους για να μπορέσουμε να συλλέξουμε σωστά τα δεδομένα. Όπου τα δεδομένα ήταν προσβάσιμα απευθείας μέσα στον HTML κώδικα,χρησιμοποιήθηκε η βιβλιοθήκη **Cheerio** για parsing του DOM και εξαγωγή των στοιχείων.Σε περιπτώσεις που χρειαζόταν να γίνει πλοήγηση, αναζήτηση ή κλικ (όπως θα έκανε ένας πραγματικός χρήστης), εφαρμόστηκε η χρήση του **Puppeteer**, που επιτρέπει την εξομοίωση πραγματικής αλληλεπίδρασης με τη σελίδα.

Τα δεδομένα που συλλέγονται αποθηκεύονται σε μια **βάση MongoDB**, λόγω της ευελιξίας που παρέχει στην αποθήκευση και αναζήτηση ημιδομημένων πληροφοριών, και επειδή συνεργάζεται άμεσα με το Node.js.

Για τη διαχείριση των δεδομένων αναπτύχθηκε ένα απλό αλλά λειτουργικό γραφικό περιβάλλον, ενώ το σύστημα μπορεί να λειτουργεί και "σιωπηλά" στο παρασκήνιο, ως υπηρεσία συστήματος, χωρίς να απαιτείται να ανοίγει ο χρήστης την εφαρμογή.

Data Retrieval Using Web Scraping Methods

Alexandros Chatsatourian

Abstract

The purpose of this thesis is the development of an automated data collection system (web scraping) that extracts information from two different websites: **ordino.gr** and **businessportal.gr**. The collected information relates either to theatrical performances and their contributors or to businesses operating in Greece.

The system was implemented using **Node.js** for the backend and **React.js** for the frontend, providing complete functionality for both processing and displaying the collected data.

Since each website presents its information in a different structure, two separate scraping methods were used to ensure accurate data retrieval. In cases where the data was directly accessible in the HTML code, the **Cheerio** library was used for DOM parsing and data extraction. For more complex scenarios where navigation, search, or clicks were required—similar to a real user's behavior—the **Puppeteer** library was used, allowing simulation of real-time interaction with the page.

The collected data is stored in a **MongoDB** database, chosen for its flexibility in handling semi-structured information and its seamless integration with Node.js.

A simple yet functional **graphical interface** was developed to manage and view the data. Additionally, the system is capable of running **silently in the background** as a system service, without requiring direct user interaction.

This thesis demonstrates how tasks that would otherwise be performed manually can now be fully automated using web scraping technologies. The results show that the system successfully identifies and stores all relevant data, providing a solid foundation for further extension and use.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την οικογένεια μου για την συνεχής υποστήριξη κατά την εκπόνηση της διπλωματικής εργασίας και τον επιβλέποντα καθηγητή Μιχαήλ Σαλαμπάση για την εύκαιρα που μου δόθηκε να επιλέξω αυτό το θέμα μαζί με την καθοδήγηση και την συμπαράστασή του σε όλη την διάρκεια της διπλωματικής.

Περιεχόμενα

Πρόλογος..... v
Περίληψη..... vi
Abstract..... vii
Ευχαριστίες..... viii
Περιεχόμενα..... ix
Κατάλογος Σχημάτων..... xi
Κατάλογος Πινάκων..... xi
Συντομογραφίες..... xii
Κεφάλαιο 1ο: Εισαγωγή στο Web Scraping..... 1
1.1 Εισαγωγή..... 1
1.2 Web scraping..... 1
1.3 Web crawlers..... 1
1.4 Γιατι Web scraping..... 1
1.5 API..... 2
1.6 Μεγαλα Δεδομενα..... 2
1.7 Αναλυση Δεδομενων..... 2
1.8 Νομιμότητα και ηθικη του Scraping..... 2
1.9 Document Object Model (DOM)..... 2
1.10 ordino.gr..... 2
1.11 businessportal.gr..... 2
1.12 Επιλογος..... 2
Κεφάλαιο 2ο: Υλοποιηση του scraping με Node.js..... 3
2.1 Εισαγωγή..... 3
2.2 Η γλωσσα Node.js..... 3
2.3 Βιβλιοθηκη Puppeteer..... 3
2.4 Βιβλιοθηκη Cheerio..... 4
2.5 Βιβλιοθηκη Express..... 5
2.6 Regular Expresions..... 5
2.7 Error Handling..... 5
2.8 Anti-Detection Techniques..... 5
2.9 Single Thread Application..... 5
2.10 Event Driven Architecture..... 5

2.12 Introduction Log System....5
2.12 A Declarative Architecture....5
2.13 Designing for Horizontal Scalability....5
2.14 Docker....5
2.15 Github....5
2.16 Επιλογος....5
Κεφάλαιο 3ο: Αποθηκευση πληροφοριας....7
3.1 Εισαγωγή....7
3.2 MongoDb....7
3.2.1 Δυνατοτητες και πλεονεκτηματα της MongoDB....2
3.2.2 Εσωτερική Λειτουργία της MongoDB (Behind the Scenes)....2
3.2.3 Αρχιτεκτονική Πελάτη (Client Architecture)....2
3.2.4 Αρχιτεκτονική Sharding (Sharded Cluster Architecture)....2
3.2.5 Αρχιτεκτονική Συλλογών (Collection-Level Architecture)....2
3.2.6 Cursors και Μηχανισμός Ανάκτησης Δεδομένων (Cursors & Data Retrieval)....2
3.3 Μαζικη εισαγωγη δεδομενων(Bulk-write)....7
3.4 Document Based Database....8
3.5 Η βαση δεδομενων του scraper....8
3.5.1 Το collection business....2
3.5.2 Το collection businessessids....2
3.5.3 Το collection logs....2
3.6 Επιλογος....8
Κεφάλαιο 4ο: Συστημα scraper....9
4.1 Εισαγωγή....7
4.2 Χρηση React για τη Δημιουργία του UI....7
4.2.1 Τι Είναι το React....2
4.2.2 Γιατί React....2
4.3 Υλοποίηση User-Friendly UI....7
4.4 Real-Time Progress και Ενημερώσεις....7
4.5 Φίλτρα για τη Στοχοποιημένη Αναζήτηση....7
4.6 Ροή Προγράμματος....7
4.7 Στοχος Προγράμματος....7
4.8 Επιλογος....7
Κεφάλαιο 5ο: Συμπερασματα....9

5.1 Ελεγχος αποτελεσματων.....7
5.2 Προτασεις για μελλοντικη βελτιωση.....7
ΒΙΒΛΙΟΓΡΑΦΙΑ..... 11
ΠΑΡΑΡΤΗΜΑ Α : ΤΙΤΛΟΣ ΠΑΡΑΡΤΗΜΑΤΟΣ.....13

Κατάλογος Σχημάτων

2.3: Βιβλιοθηκη Puppeteer....23
Σχήμα 2.1: Puppeteer code example....23
2.4: Βιβλιοθηκη Cheerio....23
Σχήμα 2.2: Cheerio code example....23
2.5: Βιβλιοθηκη Express....25
Σχήμα 2.3: Express code example....26
Σχήμα 2.4: Express code example....27
2.6: Regular Expresions....29
Σχήμα 2.5: Regex....29
2.8: Anti-Detection Techniques....30
Σχήμα 2.6: Delay....31
Σχήμα 2.7: Delay....31
2.14: Docker....33
Σχήμα 2.8: Docker UI....34
3.2: MongoDb....37
Σχήμα 3.1: JSON stracture....37
3.3: Μαζικη εισαγωγη δεδομενων(Bulk-write)....40
Σχήμα 3.2: Bulk Write Actors....40
Σχήμα 3.3: Bulk Write Businessids....41
3.5: Η βαση δεδομενων του scraper....43
Σχήμα 3.4: Actors Collection....43
Σχήμα 3.5: Business Collection....44
Σχήμα 3.6: Businessesids Collection....44

4.3: Υλοποίηση User-Friendly UI....48
Σχήμα 4.1:User UI....49
4.3: Real-Time Progress και Ενημερώσεις....49
Σχήμα 4.2:Real Time Progress UI....50
Σχήμα 4.3:Socket IO....51
4.6: Ροή Προγράμματος....52
Σχήμα 4.4:Scraper Browser....52

5.1: Ελεγχος αποτελεσματων....54
Σχήμα 5.1: Filters Page....55
Σχήμα 5.2: Results Page....55
Σχήμα 5.3: Scraper Results Page....56
Σχήμα 5.4: Business Info....56
Σχήμα 5.5: Business Info....57
Σχήμα 5.6: Saved Data....57

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία

Κεφάλαιο 1ο: Εισαγωγή στο Web Scraping

1.1 Εισαγωγή

Στο πρώτο κεφάλαιο θα γίνει μια εισαγωγή με τις βασικές έννοιες και τεχνολογίες που χρησιμοποιεί το web scraping, για ποιον λόγο να υπάρχει το web scraping, σύγκριση με εναλλακτικές τεχνολογίες, τις τεχνικές του και ανάλυση της δομής θεατρικών πρωταγωνιστών από το ordino.gr και των επιχειρήσεων από το businessportal.gr.

1.2 Web scraping.

Το **web scraping**, γνωστό και ως εξαγωγή δεδομένων από το διαδίκτυο, είναι η **αυτοματοποιημένη διαδικασία συλλογής πληροφοριών από ιστοσελίδες**. Η τεχνική αυτή περιλαμβάνει την αποστολή αιτήματος σε έναν ιστότοπο, την παραλαβή της απάντησης (συνήθως σε μορφή HTML) και την εξαγωγή συγκεκριμένων στοιχείων που μας ενδιαφέρουν. Τα δεδομένα που συλλέγονται μπορούν να αποθηκευτούν σε μια βάση δεδομένων ή σε αρχείο, ώστε να αξιοποιηθούν για ανάλυση ή προβολή των δεδομένων.

Οι χρήσεις του web scraping είναι πολλές και πρακτικές: από την παρακολούθηση τιμών προϊόντων και την ανάλυση της αγοράς, μέχρι τη συλλογή δεδομένων από δημόσιες βάσεις, ειδησεογραφικές ιστοσελίδες ή πολιτιστικές πλατφόρμες. Στο πλαίσιο της παρούσας εργασίας, το web scraping χρησιμοποιείται για την καταγραφή πληροφοριών σχετικά με **θεατρικές παραστάσεις** και **επιχειρήσεις στην Ελλάδα**.

Καθώς το διαδίκτυο παράγει τεράστιους όγκους δεδομένων καθημερινά, το web scraping αποτελεί πλέον **σημαντικό εργαλείο για τη μαζική και γρήγορη εξαγωγή πληροφορίας**. Η τεχνολογία αυτή αντικαθιστά χρονοβόρες χειροκίνητες διαδικασίες και επιτρέπει την άντληση δεδομένων με ακρίβεια και συνέπεια.

Η διαδικασία περιλαμβάνει συνήθως δύο βασικά στάδια:

1. Την **ανάκτηση του περιεχομένου** της ιστοσελίδας μέσω HTTP ή browser.
2. Την **εξαγωγή των χρήσιμων δεδομένων** με τεχνικές parsing του HTML (ή άλλων μορφών όπως JSON, XML).

Μόλις η πληροφορία συλλεχθεί, μπορεί να οργανωθεί σε σύνολα δεδομένων για περαιτέρω επεξεργασία, αποθήκευση ή και οπτικοποίηση.

1.3 Web Crawlers

Οι **web crawlers**, γνωστοί και ως "ρομποτάκια" αναζήτησης, είναι αυτοματοποιημένα προγράμματα που «σαρώνουν» το διαδίκτυο με σκοπό να εντοπίσουν και να καταγράψουν το περιεχόμενο ιστοσελίδων. Βασικός στόχος τους είναι να καταλάβουν για τι πράγμα μιλάει κάθε ιστοσελίδα, ώστε να μπορέσει αργότερα μια μηχανή αναζήτησης να εμφανίσει τις κατάλληλες πληροφορίες όταν τις ζητήσει κάποιος χρήστης.

Συνήθως, οι crawlers λειτουργούν για λογαριασμό μηχανών αναζήτησης, όπως η Google. Όταν ένας χρήστης πληκτρολογεί κάτι στη γραμμή αναζήτησης, οι μηχανές χρησιμοποιούν τα δεδομένα που έχουν συλλέξει οι crawlers, τα περνούν από ειδικούς αλγορίθμους και εμφανίζουν τις πιο σχετικές ιστοσελίδες στα αποτελέσματα.

Σε αντίθεση με το **web scraping**, που έχει πιο συγκεκριμένο σκοπό, οι crawlers κινούνται από σελίδα σε σελίδα, ακολουθώντας συνδέσμους και συλλέγοντας γενικά δεδομένα από όλο το διαδίκτυο. Ένας **scraper** μπορεί να έχει σχεδιαστεί για να τραβά δεδομένα από συγκεκριμένες σελίδες μόνο (π.χ. πληροφορίες επιχειρήσεων ή ονόματα ηθοποιών), χωρίς να συνεχίζει την πλοήγηση σε άλλα links.

1.4 Γιατί Web Scraping;

Αν ο μόνος τρόπος να έχουμε πρόσβαση στο διαδίκτυο ήταν μέσω ενός browser (όπως ο Chrome ή ο Firefox), θα χάναμε πολλές δυνατότητες που μπορούν να προσφέρουν πιο εξειδικευμένα εργαλεία. Οι browsers είναι πολύ καλοί για καθημερινή χρήση: δείχνουν τις σελίδες οπτικά όμορφα διαμορφωμένες, εμφανίζουν εικόνες, εκτελούν JavaScript και γενικά κάνουν την πλοήγηση πιο ευχάριστη και κατανοητή για τον άνθρωπο.

Ωστόσο, όταν μιλάμε για **μεγάλο όγκο δεδομένων** και ανάγκη για **ταχύτητα και αυτοματισμό**, τότε οι **web scrapers** έχουν το πάνω χέρι. Αυτά τα εργαλεία είναι σχεδιασμένα για να εντοπίζουν, να συλλέγουν και να επεξεργάζονται μαζικά δεδομένα από πολλές ιστοσελίδες με τρόπο που ένας απλός browser δεν μπορεί .

Επιπλέον, οι scrapers μπορούν να αποκτήσουν πρόσβαση σε περιεχόμενο που **δεν φαίνεται πάντα σε μια αναζήτηση στο Google**, όπως πληροφορίες που βρίσκονται βαθύτερα σε έναν ιστότοπο ή δεδομένα που απαιτούν κάποιου είδους πλοήγηση ή αλληλεπίδραση για να εμφανιστούν. Με άλλα λόγια, μπορούν να φτάσουν εκεί που οι παραδοσιακές μηχανές αναζήτησης σταματούν.

1.5 API

Μια διεπαφή προγραμματισμού εφαρμογών ή API, επιτρέπει στις εταιρείες να ανοίγουν τα δεδομένα και τη λειτουργικότητα των εφαρμογών τους σε εξωτερικούς τρίτους προγραμματιστές, επιχειρηματικούς συνεργάτες και εσωτερικά τμήματα των εταιρειών τους. Αυτό επιτρέπει στις υπηρεσίες και τις εφαρμογές να επικοινωνούν μεταξύ τους και να αξιοποιούν τα δεδομένα και τη λειτουργικότητα του άλλου μέσω μιας τεκμηριωμένης διεπαφής. Οι προγραμματιστές δεν χρειάζεται να γνωρίζουν πώς υλοποιείται ένα API. χρησιμοποιούν απλώς τη διεπαφή για να επικοινωνούν με άλλες εφαρμογές και υπηρεσίες. Οι εφαρμογές, ονομάζονται ως «API endpoints».

Γενικά, είναι προτιμότερο να χρησιμοποιείτε ένα API εάν υπάρχει η δυνατότητα αντί να δημιουργήσετε ένα bot για να λαμβάνετε τα ίδια δεδομένα. Ωστόσο, υπάρχουν λόγοι για τους οποίους ένα API ενδέχεται να μην υπάρχει:

- Συγκεντρώνετε δεδομένα σε μια συλλογή τοποθεσιών που δεν διαθέτουν public API.
- Το σύνολο των δεδομένων που θέλετε είναι λίγα, οπού ο ιδιοκτήτης της ιστοσελίδας δεν πιστεύει ότι δικαιολογεί ένα API.
- Η πηγή δεν έχει την υποδομή ή την τεχνική ικανότητα να δημιουργήσει ένα API.
- Επίσης μπορεί να υπάρχουν και λόγοι για την προτίμηση του scraping ενώ υπάρχει API:
- Περιορισμένη διαθεσιμότητα στα δεδομένα ενός API.
- Το API υπάρχει ωστόσο υπάρχει τιμή για να δοθεί πρόσβαση.
- Η έλλειψη προσαρμοστικότητας του API.
- Ανωνυμία στην συλλογή πληροφορίας.

Αντιθέτως κάποιοί λόγοι που το web scraping δεν είναι εφικτό

- Δεν επιτρέπουν όλοι οι ιστότοποι το web scraping (π.χ. Facebook)
- Είναι παράνομο ή ανήθικο
- Είναι πάρα πολύ δύσκολο ανάλογα την ιστοσελίδα(πχ .στοιχηματικές)

Εν ολίγοις, το scraping και η χρήση του API έχουν τον ίδιο στόχο – την πρόσβαση στα απαιτούμενα δεδομένα. Ωστόσο, το scraping είναι μια προτιμότερη επιλογή όταν πρόκειται για συλλογή τεράστιων ποσοτήτων δεδομένων από διαφορετικούς πόρους. Η επιλογή μεταξύ web scraping από API εξαρτάται από τους επιχειρηματικούς στόχους. Εάν χρειάζεται να συλλέγετε δεδομένα από τον ίδιο ιστότοπο συνεχώς, το API είναι η κατάλληλη επιλογή. Δεδομένα που scraper κάνει μεγάλο χρονικό διάστημα να συλλέξει, ένα API μπορεί να τα εμφανίσει με μια κλήση. Είναι ένας από τους πιο δημοφιλείς τρόπους ανταλλαγής δεδομένων μεταξύ επιχειρήσεων

1.6 Μεγαλα Δεδομενα

Τα μεγάλα δεδομένα (big data) που είναι διαθέσιμα στον Παγκόσμιο ιστό αποτελούνται από δομημένα, ημι-δομημένα, μη δομημένα ποσοτικά και ποιοτικά δεδομένα που διανέμονται με τη μορφή ιστοσελίδων, πινάκων HTML, βάσεων δεδομένων, email και άλλες μορφές δεδομένων. Η αξιοποίηση των μεγάλων δεδομένων απαιτεί την αντιμετώπιση ορισμένων τεχνικών ζητημάτων που σχετίζονται με τον όγκο, την ποικιλία, την ταχύτητα και την ακρίβεια των δεδομένων. Πρώτον, τα δεδομένα στον Ιστό συχνά χαρακτηρίζονται από τεράστιο όγκο που μετράτε σε Zettabytes. Δεύτερον, αυτά τα τεράστια αποθετήρια δεδομένων που είναι διαθέσιμα στον Ιστό διατίθενται σε διάφορες μορφές και βασίζονται σε μια ποικιλία τεχνολογικών και κανονιστικών προτύπων. Τρίτον, τα δεδομένα στον Ιστό δεν είναι στατικά. Αλλάζουν με εξαιρετική ταχύτητα. Το τελευταίο χαρακτηριστικό των μεγάλων δεδομένων είναι η αξιοπιστία τους σαν πληροφορία. Λόγω των ανοιχτών, εθελοντικών και συχνά ανώνυμων αλληλεπιδράσεων στον Ιστό, υπάρχει μια αβεβαιότητα που σχετίζεται με τη διαθεσιμότητα και την ποιότητα των δεδομένων.

1.7 Αναλυση Δεδομενων

Η ανάλυση δεδομένων είναι η διεξοδική μελέτη ενός συνόλου πληροφοριών, στόχος του οποίου είναι η εξαγωγή συμπερασμάτων που επιτρέπουν σε μια εταιρεία ή οντότητα να λάβει απόφαση. Δηλαδή, αναφερόμαστε στην εξέταση και την ερμηνεία μιας βάσης δεδομένων. Αυτό, για να επιτευχθεί η επίλυση ενός προβλήματος ή ερώτησης. Κατά τη διάρκεια αυτής της ανάλυσης, τα δεδομένα μπορούν να ανταλλαχθούν, για παράδειγμα, για τη λήψη στατιστικών δεικτών. Πρέπει να σημειωθεί ότι πρόκειται για μια διαδικασία επιστήμης δεδομένων που λαμβάνει χώρα μετά τη συλλογή των πληροφοριών. Δηλαδή, αυτή η ανάλυση περιλαμβάνει όλα τα εργαλεία που μπορούμε να χρησιμοποιήσουμε για τη μελέτη μιας βάσης δεδομένων, συμπεριλαμβανομένων οπτικών όπως το ιστόγραμμα, το διάγραμμα ράβδων, το γράφημα πίτας, μεταξύ άλλων. Η ανάλυση δεδομένων μπορεί να είναι δύο τύπων:

- Ποσοτικός: Οι πληροφορίες είναι αριθμητικές από τις οποίες μπορούν νασυγκεντρωθούν ακριβή στατιστικά στοιχεία. Για παράδειγμα, οι βαθμοί που αποκτήθηκαν από τους μαθητές μιας τάξης κατά το τελευταίο εξάμηνο.
- Ποιοτικός: Είναι πληροφορίες που λαμβάνονται από μια βάση δεδομένων που συνήθως παρουσιάζεται σε μορφή κειμένου. Για παράδειγμα, μια ομάδα-στόχος όπου οι συμμετέχοντες έχουν ζητήσει τη γνώμη τους για ένα νέο προϊόν.

Για την ανάλυση δεδομένων υπάρχουν διαφορετικά εργαλεία που προέρχονται από τομείς σπουδών όπως στατιστικά, οικονομετρικά ή μαθηματικά. Έτσι, για παράδειγμα, θα μπορούσαμε να χρησιμοποιήσουμε στατιστικές μετρήσεις όπως ο μέσος όρος, η τυπική απόκλιση ή ο διάμεσος για να λάβουμε πληροφορίες σχετικά με τη συμπεριφορά μιας μεταβλητής. Από την πλευρά της, η οικονομετρία μας παρέχει βασικά εργαλεία όπως η ανάλυση παλινδρόμησης. Σε αυτές τις γραμμές, μπορούμε επίσης να χρησιμοποιήσουμε γραφικά που παρέχουν οπτικές πληροφορίες. Για παράδειγμα, από ένα ιστόγραμμα. Ωστόσο, αξίζει να σημειωθεί ότι η ανάλυση δεδομένων δεν είναι χωρίς τους περιορισμούς της. Αυτό καθώς υπάρχουν μεταβλητές που είναι δύσκολο να ποσοτικοποιηθούν με ακρίβεια. Αυτός είναι ο λόγος για τον οποίο στην ανάλυση δεδομένων είναι συνηθισμένο να μιλάμε ως προς την πιθανότητα. Η ανάλυση δεδομένων μπορεί να έχει διαφορετικές εφαρμογές, τόσο για εταιρείες όσο και για κρατικούς οργανισμούς ή για εκείνους με μη κερδοσκοπικούς στόχους. Για παράδειγμα, μια οντότητα που επιδιώκει να μειώσει τον παιδικό υποσιτισμό σε μια χώρα θα αξιολογεί συνεχώς τα ποσοστά αναιμίας των παιδιών σε ένα συγκεκριμένο ηλικιακό εύρος. Ομοίως, μια εταιρεία μπορεί να αναλύσει τα δεδομένα ικανοποίησης που δείχνουν οι πελάτες της. Αυτό, αφού πραγματοποίησε μια έρευνα για όλα τα άτομα που προσέλαβαν τις υπηρεσίες τους τον προηγούμενο μήνα. Με αυτόν τον τρόπο, μπορείτε να λάβετε αποφάσεις για την επιχειρηματική σας στρατηγική. Η ανάλυση δεδομένων γίνεται σχετική σε περιόδους Big Data, που είναι τόσο μεγάλα σύνολα δεδομένων που υπερβαίνουν την ικανότητα των παραδοσιακών εφαρμογών υπολογιστών να τις χειρίζονται σε εύλογο χρονικό διάστημα. Σήμερα, οι εταιρείες μπορούν να έχουν τεράστιες βάσεις δεδομένων, για παράδειγμα, όταν δημιουργούν εφαρμογές στις οποίες έχουν πρόσβαση όλοι οι πελάτες και το κοινό-στόχο τους

1.8 Νομιμότητα και ηθική του Scraping

Ενώ πολλά εργαλεία και τεχνολογίες είναι διαθέσιμα για να βοηθήσουν τους ερευνητές με το Web Scraping, η νομιμότητα του Web Scraping εξακολουθεί να είναι μια «γκρίζα περιοχή» στο νομικό πεδίο.

Εδώ ορίζουμε τη νομιμότητα ως συμμόρφωση με τους ισχύοντες νόμους και τις νομικές θεωρίες. Προς το παρόν δεν υπάρχει νομοθετικό σώμα που απευθύνεται απευθείας στο Web Scraping. Καθοδηγείται από ένα σύνολο σχετικών, θεμελιωδών νομικών θεωριών και νόμων, όπως «παραβίαση πνευματικών δικαιωμάτων». Μερικές συγκεκριμένες λεπτομέρειες για το πώς αυτές οι θεμελιώδεις νομικές θεωρίες εφαρμόζονται στο Web Scraping παρέχονται παρακάτω.

- Όροι χρήσης Έχει υποστηριχθεί συχνά στο νομικό πεδίο ότι ένας κάτοχος ιστότοπου μπορεί να αποτρέψει αποτελεσματικά την πρόσβαση μέσω προγραμματισμού σε έναν ιστότοπο, απαγορεύοντας ρητά κάτι τέτοιο στην πολιτική «όρων χρήσης» που δημοσιεύεται στον ιστότοπο. Η μη συμμόρφωση με αυτούς τους όρους μπορεί να οδηγήσει σε «παραβίαση της σύμβασης» από την πλευρά του χρήστη του ιστότοπου. Για να διώξει κάποιον για παραβίαση των «όρων χρήσης», ο χρήστης του ιστότοπου πρέπει να συνάψει μια ρητή συμφωνία με τον κάτοχο του ιστότοπου για τη συμμόρφωση με την πολιτική «όρων χρήσης», συνήθως γίνεται όταν ο χρήστης επισκέπτεται την ιστοσελίδα και μια φόρμα αποδοχής ορών χρήσης εμφανίζεται. Επομένως, η απλή απαγόρευση της ανίχνευσης και της απόσυρσης ιστού στον ιστότοπο δεν μπορεί να αποκλείσει κάποιον από την ανίχνευση του ιστότοπου από νομική άποψη. • Υλικό που προστατεύεται από πνευματικά δικαιώματα Το scraping και η αναδημοσίευση δεδομένων ή πληροφοριών που ανήκουν και προστατεύονται ρητά από πνευματικά δικαιώματα από τον κάτοχο του ιστότοπου μπορεί να οδηγήσει σε «παραβίαση πνευματικών δικαιωμάτων». Ωστόσο, ένας ιστότοπος δεν κατέχει απαραίτητα τα δεδομένα που δημιουργούνται από τους χρήστες του. Έτσι, μπορεί κανείς να χρησιμοποιήσει δεδομένα που προστατεύονται από πνευματικά δικαιώματα για να δημιουργήσει περιλήψεις δεδομένων που προστατεύονται από πνευματικά δικαιώματα. Τέλος, οποιοσδήποτε μπορεί να χρησιμοποιήσει υλικό που προστατεύεται από πνευματικά δικαιώματα σε περιορισμένη κλίμακα σύμφωνα με την αρχή της «δίκαιης χρήσης».

- Σκοπός Web Scraping Οποιαδήποτε παράνομη ή δόλια χρήση δεδομένων που λαμβάνονται μέσω Web Scraping απαγορεύεται από διάφορους νόμους. Στον διαδίκτυο, αυτό συμβαίνει συχνά όταν κάποιος αποκτά εν γνώσει του πρόσβαση σε "περιεχόμενο premium" και στη συνέχεια το μεταπωλεί ή συνεχίζει να έχει πρόσβαση στο περιεχόμενο μέσω μη εξουσιοδοτημένου καναλιού αφού λάβει μια επιστολή "παύσης και διακοπής" από τον ιδιοκτήτη του ιστότοπου. • Βλάβη στον Ιστότοπο Εάν το Web Scraping υπερφορτώσει ή καταστρέψει έναν ιστότοπο ή έναν διακομιστή, τότε το άτομο που ευθύνεται για τη ζημιά μπορεί να διωχθεί σύμφωνα με τον νόμο. Ωστόσο, η ζημιά πρέπει να είναι υλική και να αποδεικνύεται εύκολα στο δικαστήριο, προκειμένου ο κάτοχος του διακομιστή να δικαιούται οικονομική αποζημίωση.
- Ατομικό Απόρρητο Ένα ερευνητικό έργο που βασίζεται σε δεδομένα που συλλέγονται από έναν ιστότοπο ενδέχεται να θέσει σε κίνδυνο άθελα το απόρρητο των ατόμων που συμμετέχουν στις δραστηριότητες που παρέχονται από τον ιστότοπο. Ακόμη και αν δεν παραβιαστεί το ατομικό απόρρητο, το πρόβλημα είναι ότι οι πελάτες ενός ιστότοπου ενδέχεται να μην έχουν συναινέσει σε χρήση των δεδομένων τους από τρίτους. Επομένως, η χρήση αυτών των δεδομένων χωρίς συναίνεση αποτελεί παραβίαση των δικαιωμάτων των υποκειμένων της έρευνας. Αυτές οι παραβιάσεις απορρήτου και δικαιωμάτων μπορούν να οδηγήσουν σε σοβαρές συνέπειες για τον κάτοχο του ιστότοπου, δεδομένης της έντονης ανησυχίας για το απόρρητο στο διαδίκτυο υπό το φως των σκανδάλων απορρήτου που αφορούν τέτοιους οργανισμούς. • Απόρρητο και εμπορικά μυστικά Ακριβώς όπως τα άτομα έχουν δικαίωμα στην ιδιωτική ζωή, έτσι και οι οργανισμοί έχουν επίσης το δικαίωμα να διατηρήσουν εμπιστευτικές ορισμένες πτυχές των εργασιών τους. Το Web Scraping μπορεί να αποκαλύψει άθελος εμπορικά μυστικά ή απλώς εμπιστευτικές πληροφορίες σχετικά με τον οργανισμό που κατέχει έναν ιστότοπο. Μπορεί να αποκαλύψει ορισμένες λεπτομέρειες και, ενδεχομένως, ελαττώματα στον τρόπο αποθήκευσης των δεδομένων από τον ιστότοπο. Όλα αυτά μπορούν να βλάψουν τη φήμη της εταιρείας πίσω από τον ιστότοπο και να οδηγήσουν σε οικονομικές απώλειες. • Μείωση της κερδοφορίας για την εταιρία Εάν κάποιος αποκτήσει πρόσβαση στον ιστότοπο παραλείποντας τη διεπαφή της ιστοσελίδας που είναι κατασκευασμένη για ανθρώπους, τότε το άτομο δεν θα εκτεθεί στις διαφημίσεις που χρησιμοποιεί ο ιστότοπος για τη δημιουργία εσόδων από το περιεχόμενό του. Επιπλέον, η παρουσίαση των δεδομένων που δημιουργείται με τη βοήθεια του Web Scraping, μπορεί άμεσα ή έμμεσα να ανταγωνιστεί την επιχείρηση του ιδιοκτήτη του ιστότοπου. Όλα αυτά μπορεί να οδηγήσουν σε οικονομικές απώλειες στον ιδιοκτήτη του ιστότοπου ή, τουλάχιστον, σε άδικη διανομή της αξίας από την ιδιοκτησία των δεδομένων.

1.9 DOM

Το Document Object Model (DOM) είναι μια διεπαφή προγραμματισμού εφαρμογών (API) για έγκυρη HTML και XML. Καθορίζει τη λογική δομή των σελίδων και τον τρόπο που γίνεται η πρόσβαση και χειραγώγηση. Στην προδιαγραφή DOM, ο όρος "document" χρησιμοποιείται με την ευρεία έννοια, η XML χρησιμοποιείται ως τρόπος αναπαράστασης πολλών διαφορετικών ειδών πληροφοριών που μπορεί να είναι αποθηκευμένα σε διαφορετικά συστήματα και πολλά από αυτά θα θεωρούνταν παραδοσιακά ως δεδομένα και όχι ως έγγραφα(σελίδες). Ωστόσο, η XML παρουσιάζει αυτά τα δεδομένα ως έγγραφα και το DOM μπορεί να χρησιμοποιηθεί για τη διαχείριση αυτών των δεδομένων. Με το DOM, οι προγραμματιστές μπορούν να δημιουργήσουν έγγραφα, να περιηγηθούν στη δομή τους, να προσθέσουν, να τροποποιήσουν ή να διαγράψουν στοιχεία και περιεχόμενο. Εισαγωγή στο Web scraping 7 Οτιδήποτε βρίσκεται σε ένα έγγραφο HTML ή XML μπορεί να προσπελαστεί, να αλλάξει, να διαγραφεί ή να προστεθεί χρησιμοποιώντας το DOM με λίγες εξαιρέσεις. Ως προδιαγραφή του W3C, ένας σημαντικός στόχος για το DOM είναι να παρέχει ένα τυπικό API που μπορεί να χρησιμοποιηθεί σε μεγάλη ποικιλία περιβαλλόντων και εφαρμογών. Το DOM έχει σχεδιαστεί για να χρησιμοποιείται με οποιαδήποτε γλώσσα προγραμματισμού.

Στο DOM, τα έγγραφα έχουν μια λογική δομή που μοιάζει πολύ με ένα δέντρο. Κάθε έγγραφο περιέχει μηδέν ή έναν κόμβους τύπου doctype, έναν κόμβο ριζικού στοιχείου και μηδέν ή περισσότερα σχόλια ή οδηγίες επεξεργασίας. το ριζικό στοιχείο χρησιμεύει ως η ρίζα του δέντρου στοιχείων για το έγγραφο. Ωστόσο, το DOM δεν διευκρινίζει ότι τα έγγραφα πρέπει να υλοποιούνται ως δέντρο, ούτε διευκρινίζει πώς θα υλοποιηθούν οι σχέσεις μεταξύ των αντικειμένων. Το DOM είναι ένα λογικό μοντέλο που μπορεί να εφαρμοστεί με οποιονδήποτε βολικό τρόπο. Το όνομα "Μοντέλο αντικειμένου εγγράφου" επιλέχθηκε επειδή είναι ένα "μοντέλο αντικειμένου" με την παραδοσιακή αντικειμενοστραφή σχεδίαση: τα έγγραφα μοντελοποιούνται χρησιμοποιώντας αντικείμενα και το μοντέλο περιλαμβάνει όχι μόνο τη δομή ενός εγγράφου, αλλά και τη συμπεριφορά ενός εγγράφου και τα αντικείμενα από τα οποία αποτελείται. Με άλλα λόγια, οι κόμβοι στο παραπάνω διάγραμμα δεν αντιπροσωπεύουν δομή δεδομένων, αντιπροσωπεύουν αντικείμενα, τα οποία έχουν συναρτήσεις και ταυτότητα. ο DOM προσδιορίζει: Τι είναι το μοντέλο αντικειμένου εγγράφου οι διεπαφές και τα αντικείμενα που χρησιμοποιούνται για την αναπαράσταση και τον χειρισμό ενός εγγράφου τη σημασιολογία αυτών των διεπαφών και αντικειμένων - συμπεριλαμβανομένης της συμπεριφοράς και των χαρακτηριστικών των σχέσεων και συνεργασιών μεταξύ αυτών των διεπαφών και αντικειμένων. Η δομή των εγγράφων SGML παραδοσιακά αντιπροσωπεύεται από ένα αφηρημένο μοντέλο δεδομένων, όχι από ένα μοντέλο αντικειμένου. Σε ένα αφηρημένο μοντέλο δεδομένων, το μοντέλο επικεντρώνεται γύρω από τα δεδομένα. Στις αντικειμενοστραφείς γλώσσες προγραμματισμού, τα ίδια τα δεδομένα ενσωματώνονται σε αντικείμενα που κρύβουν τα δεδομένα, προστατεύοντάς τα από άμεσο εξωτερικό χειρισμό. Οι συναρτήσεις που σχετίζονται με αυτά τα αντικείμενα καθορίζουν τον τρόπο χειρισμού των αντικειμένων και αποτελούν μέρος του μοντέλου αντικειμένου.

1.10 Ordino.gr

Το ordino.gr είναι μια ιστοσελίδα που παρουσιάζει θεατρικές παραστάσεις και πολιτιστικές εκδηλώσεις, προσφέροντας στους επισκέπτες της τη δυνατότητα να δουν συγκεντρωμένες πληροφορίες για κάθε έργο. Στην κεντρική σελίδα προβάλλεται μια λίστα με όλες τις διαθέσιμες παραστάσεις, ενώ για κάθε μία υπάρχει ξεχωριστή σελίδα που περιλαμβάνει τον τίτλο του έργου, τις ημερομηνίες και ώρες διεξαγωγής, την τοποθεσία, την περιγραφή του περιεχομένου, τη διάρκειά του, τους συντελεστές που συμμετέχουν, οπτικοακουστικό υλικό, τον διοργανωτή και τις διαθέσιμες επιλογές για κράτηση εισιτηρίων. Η πλατφόρμα λειτουργεί ως σημείο πληροφόρησης και διευκόλυνσης για το κοινό που αναζητά θεατρικές παραγωγές και σχετικές δραστηριότητες.

1.11 Businessportal.gr

Το **businessportals.gr** είναι μια διαδικτυακή πλατφόρμα που παρέχει πληροφορίες για όλες τις επιχειρήσεις στην Ελλάδα. Λειτουργεί ως ένας online επιχειρηματικός κατάλογος, συγκεντρώνοντας και παρουσιάζοντας βασικά στοιχεία κάθε επιχείρησης, όπως επωνυμία, διεύθυνση, τηλέφωνο, δραστηριότητα, ΑΦΜ, ΓΕΜΗ και άλλα σχετικά δεδομένα. Στόχος του είναι να προσφέρει εύκολη αναζήτηση και πρόσβαση σε εμπορικές και νομικές πληροφορίες για κάθε καταχωρημένη επιχείρηση στη χώρα, εξυπηρετώντας επαγγελματίες, ερευνητές, αλλά και απλούς πολίτες που θέλουν να μάθουν περισσότερα για μία εταιρεία.

1.12 Επίλογος

Στο πρώτο κεφάλαιο έγινε η απαραίτητη ανάλυση της έννοιας του web scraping, παρουσιάστηκε ο σωστός τρόπος υλοποίησης της διαδικασίας και έγινε σύγκριση με άλλες παρόμοιες τεχνολογίες συλλογής δεδομένων. Αφού αποκτήθηκαν οι βασικές γνώσεις και τέθηκαν τα θεωρητικά θεμέλια, στα

επόμενα κεφάλαια θα δούμε αναλυτικά πώς εφαρμόστηκαν αυτές οι αρχές στην πράξη για την ανάπτυξη ενός πλήρους συστήματος scraping με χρήση της τεχνολογίας **Node.js**. Μέσα από την ανάλυση της υλοποίησης, θα κατανοήσουμε τη ροή της πληροφορίας, τις τεχνικές που χρησιμοποιήθηκαν για την ανάκτηση, ανάλυση και αποθήκευση των δεδομένων, καθώς και τις προκλήσεις που προέκυψαν κατά τη διαδικασία.

Κεφάλαιο 2ο: Υλοποίηση του scraping με Node.js

2.1 Εισαγωγή

Στο δεύτερο κεφάλαιο περιγράφεται η γλώσσα JavaScript, οι βιβλιοθήκες που χρησιμοποιήθηκαν για την υλοποίηση του συστήματος scraper, καθώς και οι αρχιτεκτονικές προσεγγίσεις της εφαρμογής μας. Επιπλέον, παρουσιάζονται οι βελτιστοποιήσεις που έγιναν με γνώμονα τη μελλοντική εξέλιξη της εφαρμογής, τη δομή του κώδικα και τη διαχείριση σφαλμάτων (error handling).

2.2 Η γλώσσα Node.js

Η **Node.js** είναι ένα περιβάλλον εκτέλεσης JavaScript υψηλής απόδοσης, το οποίο επιτρέπει την εκτέλεση κώδικα JavaScript εκτός του προγράμματος περιήγησης, στο server ή σε οποιοδήποτε περιβάλλον υπολογιστή. Η Node.js βασίζεται σε μια αρχιτεκτονική **event-driven** και **ασύγχρονης λειτουργίας (asynchronous I/O)**, η οποία την καθιστά ιδιαίτερα αποδοτική στην εκτέλεση λειτουργιών που απαιτούν μεγάλο αριθμό ταυτόχρονων αιτημάτων, όπως η διαχείριση δικτύου και αρχείων.

Η βασική καινοτομία της Node.js είναι το μηχανισμό **event loop**, που επιτρέπει την εκτέλεση πολλαπλών εργασιών χωρίς να μπλοκάρει η κύρια ροή του προγράμματος. Αντί να περιμένει η εφαρμογή να ολοκληρωθεί μια λειτουργία (π.χ. ανάγνωση αρχείου ή απόκριση σε δίκτυο), η Node.js διαχειρίζεται τις λειτουργίες αυτές με callbacks ή σύγχρονα εργαλεία όπως **Promises** και **async/await**, επιτρέποντας την παράλληλη εκτέλεση άλλων εργασιών.

Η JavaScript, ως γλώσσα που τρέχει στη Node.js, είναι μια γλώσσα δυναμική και ευέλικτη, με συντακτικό που είναι εύκολο να διαβαστεί και να γραφτεί, ενώ η Node.js προσφέρει ένα πλούσιο οικοσύστημα βιβλιοθηκών και εργαλείων μέσα από το **npm (Node Package Manager)**, που υποστηρίζει τη γρήγορη ανάπτυξη εφαρμογών.

Η Node.js είναι ιδιαίτερα δημοφιλής για την ανάπτυξη εφαρμογών που απαιτούν υψηλή απόκριση και χαμηλή καθυστέρηση, όπως web servers, APIs, real-time εφαρμογές και εργαλεία αυτοματισμού. Το γεγονός ότι επιτρέπει την εκτέλεση JavaScript τόσο στο client όσο και στο server καθιστά δυνατή την ομοιογένεια στην ανάπτυξη και διευκολύνει την επικοινωνία μεταξύ των δύο πλευρών.

Όπως και με άλλες τεχνολογίες, η Node.js έχει και περιορισμούς. Δεν είναι ιδανική για εφαρμογές που απαιτούν πολύπλοκους υπολογισμούς CPU-bound, καθώς το μοντέλο event-driven βασίζεται σε ένα single-threaded event loop. Παρόλα αυτά, με τη χρήση πρόσθετων βιβλιοθηκών ή εργαλείων (όπως worker threads ή clustering), μπορεί να υποστηρίξει και τέτοιες απαιτήσεις.

Το σύστημα web-scraping που υλοποιήθηκε χρησιμοποιεί Node.js, αξιοποιώντας την ασύγχρονη λειτουργία της για την αποδοτική ανάκτηση δεδομένων από το διαδίκτυο. Μέσω βιβλιοθηκών όπως το **axios** ή το **puppeteer**, η Node.js επιτρέπει την εύκολη διαχείριση αιτημάτων HTTP, την ανάλυση HTML και την αυτοματοποίηση περιηγητών, καθιστώντας την ιδανική για τέτοιου είδους εφαρμογές.

Εν ολίγοις, θα λέγαμε ότι το Node.js είναι κατάλληλο για εφαρμογές που βασίζονται σε event-driven αρχιτεκτονική και οχι μονο.

2.3 Βιβλιοθηκη Puppeteer

Η **Puppeteer** είναι μια δημοφιλής βιβλιοθήκη για τη γλώσσα προγραμματισμού JavaScript, η οποία επιτρέπει τον έλεγχο του προγράμματος περιήγησης Google Chrome ή Chromium με κώδικα. Με απλά λόγια, είναι ένα εργαλείο που σου επιτρέπει να «οδηγήσεις» τον browser μέσα από τον κώδικα σου, να

ανοίξεις ιστοσελίδες, να αλληλεπιδράσεις με αυτές (π.χ. να κάνεις κλικ, να συμπληρώσεις φόρμες), να πάρεις δεδομένα (web scraping), αλλά και να τραβήξεις screenshots ή να δημιουργήσεις αρχεία PDF από τις συγκεκριμένες ιστοσελίδες.

Η συγκεκριμένη βιβλιοθήκη λειτουργεί σε headless mode, δηλαδή χωρίς να εμφανίζεται γραφικά ο browser στην οθόνη, κάτι που κάνει τις διεργασίες πιο γρήγορες και αποδοτικές. Φυσικά, μπορείς να τη ρυθμίσεις να δουλεύει και με ορατό παράθυρο, για σκοπούς δοκιμών και ανάπτυξης ακομα και για unit testing των εφαρμογών σου.

Είναι πολύ χρήσιμη για εργασίες όπως:

- Αυτόματη περιήγηση σε ιστοσελίδες
- Συλλογή δεδομένων από ιστοσελίδες (web scraping)
- Δοκιμές (testing) εφαρμογών web
- Δημιουργία στιγμιότυπων οθόνης (screenshots) ή PDF
- Ελέγχους και αυτοματισμούς σε πολύπλοκες ιστοσελίδες που βασίζονται σε JavaScript

Η χρήση της Puppeteer είναι σχετικά εύκολη ακόμα και για αρχάριους, γιατί παρέχει έναν απλό και κατανοητό API (σύνολο εντολών) για την εκτέλεση όλων αυτών των λειτουργιών.Επίσης, υπάρχει αρκετό υλικό καθοδήγησης από τους δημιουργούς της.

```
async scrapeForDateRange(startDate, endDate) {        You, 6 months ag
  const browser = await puppeteer.launch({ headless: false });
  const page = await browser.newPage();
  await page.goto(this.url);

  const filterClicked = await this.clickFilterButton(page);
  if (!filterClicked) {
    console.log("Failed to find or click the filter button.");
    await browser.close();
    return [];
  }
  await this.setDates(page, startDate, endDate);
  await this.selectStatusDropdown(page);
  await this.selectRegionAndScrape(page);
  await this.enterPostalCode(page);
  await this.gemi(page);
  const searchClicked = await this.clickSearchButton(page);
  if (!searchClicked) {
    console.log("Failed to find or click the search button.");
    await browser.close();
    return [];
  }
```

Σχήμα 2.1: Puppeteer code example

2.4 Βιβλιοθηκη Cheerio

Η **Cheerio** είναι μια δημοφιλής βιβλιοθήκη για τη γλώσσα JavaScript που χρησιμοποιείται για να «διαβάζει» και να επεξεργάζεται HTML και XML έγγραφα πολύ εύκολα και γρήγορα. Με απλά λόγια, η Cheerio σου επιτρέπει να αναλύεις το περιεχόμενο μιας ιστοσελίδας (όπως το HTML της), να ψάχνεις συγκεκριμένα στοιχεία, να παίρνεις κείμενα, να αλλάζεις ή να διαγράφεις μέρη του κώδικα HTML — όλα αυτά χωρίς να χρειάζεται να ανοίξεις έναν browser.

Η Cheerio είναι πολύ χρήσιμη ειδικά σε εφαρμογές **web scraping**, όπου θέλουμε να «τραβήξουμε» δεδομένα από ιστοσελίδες και να τα επεξεργαστούμε στον κώδικα μας. Λειτουργεί πολύ γρήγορα, γιατί δεν τρέχει πραγματικά έναν browser, αλλά απλώς «διαβάζει» και χειρίζεται το HTML σαν να ήταν ένα απλό κείμενο.

Η σύνταξη της Cheerio είναι εμπνευσμένη από τη δημοφιλή βιβλιοθήκη jQuery, που σημαίνει ότι αν έχεις ασχοληθεί με jQuery, θα σου φανεί πολύ οικεία και εύκολη στη χρήση. Μπορείς δηλαδή να επιλέξεις στοιχεία της σελίδας με CSS selectors, να πάρεις ή να αλλάξεις περιεχόμενο, και να δουλέψεις με τα δεδομένα όπως θα έκανες στο frontend.

Μερικές βασικές χρήσεις της Cheerio είναι:

- Ανάκτηση συγκεκριμένων δεδομένων από ιστοσελίδες
- Φιλτράρισμα και καθαρισμός HTML περιεχομένου
- Εξαγωγή κειμένων, συνδέσμων και εικόνων
- Γρήγορη επεξεργασία HTML εγγράφων χωρίς γραφικό περιβάλλον

Συνεπώς, είναι πολύ χρήσιμη όταν η ιστοσελίδα είναι βασική και δεν περιλαμβάνει έντονη αλληλεπίδραση με τον χρήστη, ώστε να απαιτείται κάτι πιο αυτοματοποιημένο ή πολύπλοκο.

```
async scrapeData(html) {
  const details = ActorStructure();

  const $ = cheerio.load(html);
  const actorname = $(".bodytitle1 > .bodytitle1 > .bodytitle1")
    .text()
    .trim()
    .replace(/[^\w\s]/g, "");

  const firstTdBodyMain = $("td.bodymain").first();
  const bElementsInFirstTd = firstTdBodyMain.find("b");

  const firstRole = firstTdBodyMain.find("b:first-child").text().trim();

  $("a:has(img[src*='photos/small/'])").each((index, element) => {
    const href = $(element).attr("href");
    details.images.push(href.trim().replace(/\s+/g, " "));
  });

  const roles = bElementsInFirstTd
    ? bElementsInFirstTd
        .map((index, element) => $(element).text().trim())
        .get()
    : [];

  const spansAfterLanguages = $(
    'span.bodymain:contains("Languages")'
  ).nextAll("span.bodymain");

  spansAfterLanguages.each((index, element) => {
    const textContent = $(element).text().trim().replace(/\s+/g, " ");
    details.languages.push(textContent);
  });
```

Σχήμα 2.2: Cheerio code example

2.5 Βιβλιοθηκη Express

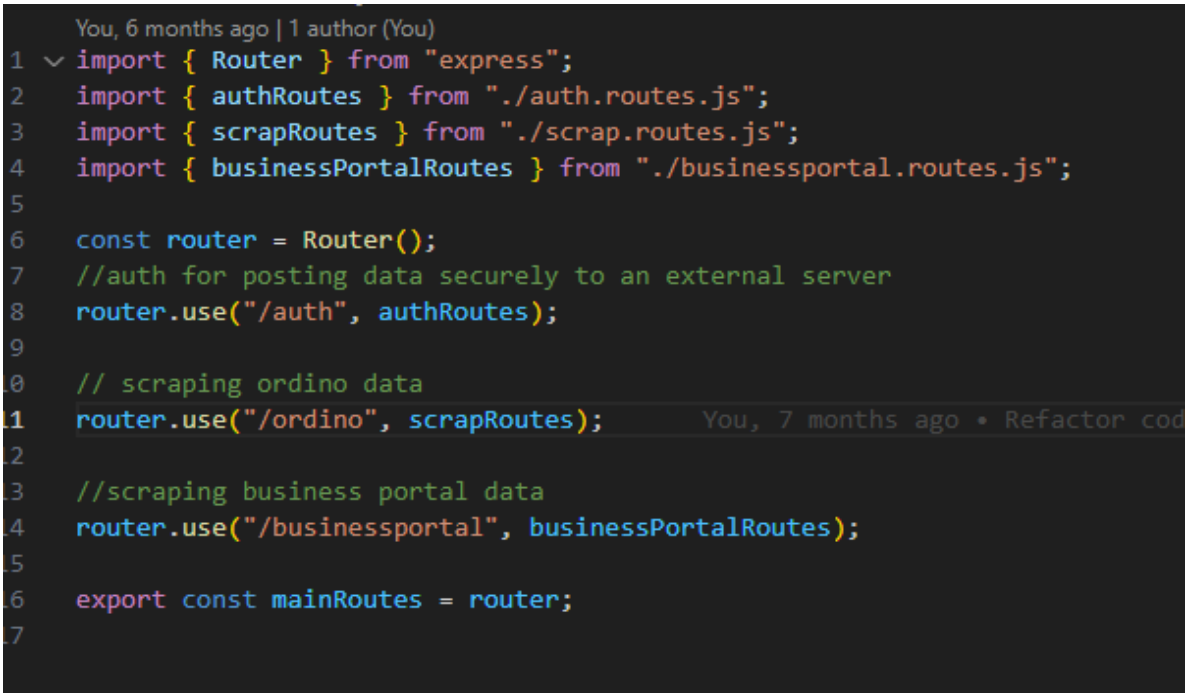
Η **Express** είναι μια δημοφιλής βιβλιοθήκη (framework) για τη Node.js, που χρησιμοποιείται για την ανάπτυξη **web εφαρμογών** και **API** με απλό και γρήγορο τρόπο. Με την Express, μπορείς να φτιάξεις διακομιστές (servers) που ανταποκρίνονται σε αιτήματα από χρήστες, να διαχειριστείς διαδρομές (routes), να στείλεις δεδομένα, να λάβεις φόρμες και γενικά να δημιουργήσεις ολόκληρες ιστοσελίδες ή υπηρεσίες που τρέχουν στο διαδίκτυο.

Η Express είναι **ελαφριά** και **ευέλικτη**, επιτρέποντας στους προγραμματιστές να χτίζουν εφαρμογές τόσο απλές όσο ένα στατικό site, όσο και πολύπλοκες με πολλά επίπεδα λειτουργικότητας. Παρέχει βασικά εργαλεία για να διαχειριστείς HTTP αιτήματα και απαντήσεις, middleware (κόμβοι επεξεργασίας δεδομένων), και υποστήριξη για templates, ώστε να μπορείς να παράγεις δυναμικό περιεχόμενο.

Χαρακτηριστικά που κάνει την Express αγαπητή:

- Απλή και καθαρή σύνταξη
- Υποστήριξη για REST APIs
- Εύκολη διαχείριση routes (δηλαδή ποια διεύθυνση κάνει τι)
- Υποστήριξη middleware για πρόσθετες λειτουργίες όπως ασφάλεια, logging, και parsing δεδομένων
- Μεγάλη κοινότητα και πλήθος διαθέσιμων προσθηκών (plugins)

Με λίγα λόγια, η Express είναι το εργαλείο που επιτρέπει στη Node.js να «μιλήσει» με το διαδίκτυο και να εξυπηρετήσει χρήστες, καθιστώντας την ανάπτυξη server-side εφαρμογών πολύ πιο απλή και γρήγορη καθώς μας παρέχει πολλές build in συναρτήσεις.



Σχήμα 2.3: Express code example

```
const router = Router();
router.post("/actors", async (req, res) => {
  const { range, stop } = req.body;
  try {
    if (stop) {
      res.json({ message: "Scraping stopped." });
    }
    const service = new  const service: ScrapeTool
    const data = await service.scrapeDataForActorRange(range.start, range.end);
    res.json(data);
  } catch (error) {
    console.error("Error while scraping:", error);
    res.status(500).json({ error: "Internal server error" });
  }
});

router.post("/movies", async (req, res) => {
  try {
    const { range, stop } = req.body;
    const service = new ScrapeTool();

    if (stop) {
      return res.json({ message: "Scraping stopped." });
    }
    const result = await service.getVenues(5000);
    res.json({
      message: "Scraping movies completed successfully!",
      data: result,
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Internal server error" });
  }
});


export const scrapRoutes = router;      You, 7 months ago • Refactor code and update dependencies
```

Σχήμα 2.4: Express code example

2.6 Regular Expressions

Οι κανονικές εκφράσεις (regex) είναι ένα πολύ ισχυρό εργαλείο στην JavaScript για την αναζήτηση, ταυτοποίηση και αντικατάσταση μοτίβων μέσα σε κείμενα. Στο web scraping με Puppeteer, χρησιμοποιήσαμε regex για να εντοπίσουμε συγκεκριμένες λέξεις ή φράσεις μέσα στο περιεχόμενο της σελίδας και να φιλτράρουμε μόνο τα δεδομένα που μας ενδιέφεραν, όπως για παράδειγμα διευθύνσεις URL εικόνων προϊόντων. Με αυτό τον τρόπο αποφεύξαμε να πιάσουμε περιττά στοιχεία, όπως λογότυπα ή κενές εικόνες που χρησιμοποιούνται για διάταξη.

Επιπλέον, όταν δουλεύουμε με δεδομένα από τον ιστό, πολλές φορές συναντάμε προβλήματα με σπασμένους χαρακτήρες, ειδικά σε κείμενα με ελληνικά που έχουν θέματα UTF encoding. Για να το λύσουμε αυτό, εφαρμόσαμε decoding στα δεδομένα πριν τα επεξεργαστούμε, ώστε να διαβάζονται σωστά.

Παράλληλα, φροντίσαμε να ελέγχουμε αν κάποια πεδία είναι **null** ή κενά πριν τα χρησιμοποιήσουμε, ώστε να αποφύγουμε σφάλματα κατά την εκτέλεση του κώδικα. Επίσης, κάναμε trim στα κείμενα,

δηλαδή αφαιρούσαμε περιττά κενά και λευκούς χαρακτήρες, για να έχουμε καθαρά και αξιόπιστα δεδομένα.

Τέλος, αφού καθαρίσαμε και προετοιμάσαμε τα δεδομένα, τα στείλαμε σε ένα backend API, όπου γινόταν περαιτέρω επεξεργασία, έλεγχοι και κατηγοριοποίηση. Με αυτό τον τρόπο διασφαλίσαμε ότι τα τελικά αποτελέσματα ήταν όσο το δυνατόν πιο σωστά και χρήσιμα.

Η σωστή χρήση των regex, σε συνδυασμό με τον σχολαστικό καθαρισμό και έλεγχο των δεδομένων, κάνει το scraping με Puppeteer πιο αποτελεσματικό και σταθερό, ενώ μειώνει τα πιθανά λάθη και προβλήματα που μπορεί να προκύψουν από ακατάλληλα ή ελλιπή δεδομέν

```
$("td.bodymain").each((index, element) => {
  const text = $(element).text().trim();
  const [key, value] = text.split(":");

  if (key && value) {
    if (key === "Hair color") {
      details.hairColor = value.trim().replace(/\s+/g, " ");
    } else if (key === "Eye color") {
      details.eyeColor = value.trim().replace(/\s+/g, " ");
    } else if (key === "Height") {
      details.height = value.trim().replace(/\s+/g, " ");
    } else if (key === "Weight") {
      details.weight = value.trim().replace(/\s+/g, " ");
    }
  }
});

return details;
}
```

Σχήμα 2.5: Regex

2.7 Error Handling

Στην εφαρμογή μας, επειδή είχαμε πολλές ασύγχρονες λειτουργίες και αυτοματισμούς, ήταν πολύ σημαντικό να διαχειριζόμαστε σωστά τα σφάλματα ώστε να μην "σπάει" η εφαρμογή μας και να έχουμε πλήρη έλεγχο σε ό,τι πάει στραβά.

Για αυτόν τον λόγο, στα περισσότερα σημεία του κώδικα όπου εκτελούνται λειτουργικές ή λογικές διεργασίες, τις ενσωματώσαμε σε try-catch blocks, ώστε να διαχειριζόμαστε αποτελεσματικά τυχόν σφάλματα. Με αυτόν τον τροπο, μπορούσαμε να πιάσουμε συγκεκριμένα σφάλματα και να αποφύγουμε την ανεξέλεγκτη διακοπή της εφαρμογής.

Επιπλέον, για να έχουμε οργανωμένη διαχείριση σφαλμάτων στο server, φτιάξαμε ένα ειδικό **middleware** που αναλαμβάνει να πιάσει τα σφάλματα που πετάμε μέσα στο πρόγραμμα. Αυτό το middleware χρησιμοποιεί το `next()` της Node.js, το οποίο επιτρέπει να περάσουμε το σφάλμα στο επόμενο στάδιο επεξεργασίας – δηλαδή στο error handler μας.

Παράλληλα, δημιουργήσαμε ένα σύστημα καταγραφής (logging) που αποθηκεύει όλα τα σφάλματα που προκύπτουν κατά τη διάρκεια της εκτέλεσης. Αυτό μας βοηθάει στο στάδιο της ανάπτυξης (development) να εντοπίζουμε εύκολα τα προβλήματα και να τα διορθώνουμε πιο γρήγορα.

Με αυτήν την οργάνωση, η εφαρμογή μας έγινε πιο σταθερή, καθώς ακόμα και αν κάτι πήγαινε στραβά σε κάποια ασύγχρονη ενέργεια, δεν προκαλούσε crash, αλλά το σφάλμα καταγραφόταν και διαχειριζόταν με ασφάλεια.

2.8 Anti-Detection-Patterns

Όταν χρησιμοποιούνται εργαλεία αυτοματισμού για συλλογή δεδομένων (scraping), πολλά συστήματα προσπαθούν να εντοπίσουν σημάδια που δείχνουν ότι δεν πρόκειται για εναν κανονικό χρήστη αλλα καποιο συστημα η bot. Για παράδειγμα, πολύ γρήγορη πληκτρολόγηση, άμεσα κλικ ή το να φορτώνει κάποιος σελίδες ακαριαία μπορεί να θεωρηθούν ύποπτα. Ένας τρόπος να αποφεύγονται τέτοιες υποψίες είναι να προστεθούν μικρές καθυστερήσεις στις ενέργειες του «χρήστη», ώστε να μοιάζουν περισσότερο ανθρώπινες.

```
async scrapePaginationResults(page) {
  const results = [];
  const seenIds = new Set();
  const timeoutDuration = 3000;
  const service = new ScrapeTool();

  let totalDiscovered = 0;
  let successCount = 0;       You, 5 months ago • czxcxz

  const waitForSelectorWithTimeout = async (selector) => {
    const timeoutPromise = new Promise((_, reject) =>
      setTimeout(
        () => reject(new Error("Timeout waiting for selector")),
        timeoutDuration
      )
    );
    return Promise.race([
      page.waitForSelector(selector, { visible: true }),
      timeoutPromise,
    ]);
  };

  this.io.emit("scrape-summary", {
    status: "started",
    date: new Date().toISOString(),
    totalDiscovered: 0,
  });
```

Σχήμα 2.6: Delay

```
await page.waitForSelector('ul[role="listbox"]', { visible: true });
console.log("Dropdown menu opened successfully.");

const options = await page.$$('ul[role="listbox"] li');
let optionSelected = false;
await new Promise((resolve) => setTimeout(resolve, 500));
```

Σχήμα 2.7: Delay

2.9 Single-thread-application

Η JavaScript είναι από τη φύση της μονονηματική γλώσσα — δηλαδή, εκτελεί εντολές σε μια μόνο κύρια ροή (thread) κάθε φορά. Αυτό απλοποιεί την εκτέλεση του κώδικα, αλλά περιορίζει τη δυνατότητα παράλληλης επεξεργασίας πολλαπλών εργασιών.

Ωστόσο, με σύγχρονες τεχνικές, όπως οι **Web Workers**, μπορούμε να εκτελούμε βαριές ή δευτερεύουσες εργασίες σε ξεχωριστά νήματα, χωρίς να εμποδίζεται το κύριο thread. Αν και αυτό δεν αποτελεί «πραγματικό» πολυνηματισμό όπως σε άλλες γλώσσες χαμηλότερου επιπέδου, προσφέρει παρόμοια αποτελέσματα σε πολλές περιπτώσεις.

Ακόμη πιο σημαντικό είναι ότι σε περιβάλλοντα διακομιστών, μπορούμε να ξεπεράσουμε αυτόν τον περιορισμό μέσω **scaling** και **orchestration**. Με εργαλεία όπως το **Kubernetes** και αρχιτεκτονική **μικροϋπηρεσιών (microservices)**, κάθε υπηρεσία μπορεί να εκτελείται σε δικό της pod, συχνά σε ξεχωριστό πυρήνα CPU ή ακόμα και σε διαφορετικό server. Έτσι επιτυγχάνεται οριζόντια κλιμάκωση (horizontal scaling), με τρόπο που προσομοιώνει πολυνηματική εκτέλεση, παρόλο που κάθε node μπορεί να τρέχει μονονηματικά.

Πλεονεκτήματα της μονονηματικής φύσης της JavaScript:

- Πιο απλός και ευκολότερος εντοπισμός σφαλμάτων
- Αποφεύγονται πολλά προβλήματα συγχρονισμού (π.χ. race conditions)
- Πολύ αποδοτική σε I/O-βασισμένες εργασίες λόγω async/await

Μειονεκτήματα:

- Εργασίες που καταναλώνουν CPU μπορεί να μπλοκάρουν την εφαρμογή
- Απαιτείται πρόσθετη αρχιτεκτονική για παράλληλη επεξεργασία
- Δεν υποστηρίζεται φυσικός πολυνηματισμός σε μεγάλο βάθος.

2.10 Event-driven

Στην εφαρμογή μας εφαρμόσαμε μια **αρχιτεκτονική βασισμένη σε γεγονότα (event-driven)**, ώστε να προσφέρουμε μια πιο άμεση και διαδραστική εμπειρία στον χρήστη. Αντί ο χρήστης να περιμένει να ολοκληρωθεί ένα `POST` ή `fetch` για να δει το αποτέλεσμα, χρησιμοποιήσαμε το **Socket.IO** για να παρέχουμε ενημερώσεις προόδου σε πραγματικό χρόνο.

Χρησιμοποιήσαμε τη βιβλιοθήκη **Socket.IO**, η οποία επιτρέπει την αποστολή και λήψη γεγονότων (events) μεταξύ server και client, διατηρώντας **μια ενεργή (live) σύνδεση**. Σε κάθε βήμα μιας διαδικασίας, ο server μπορεί να στείλει ενημέρωση στον client — π.χ. "Ξεκίνησε", "Σε εξέλιξη", "Ολοκληρώθηκε" — χωρίς ο χρήστης να χρειάζεται να ανανεώσει τη σελίδα ή να περιμένει παθητικά.

Ένα από τα πιο ισχυρά χαρακτηριστικά του Socket.IO είναι ότι **υποστηρίζει διπλή επικοινωνία (two-way communication)**. Δηλαδή, **δεν εκπέμπει μόνο ο server — και ο client μπορεί να στείλει events**, με απλές `emit` εντολές απευθείας από το frontend. Αυτό επιτρέπει στον χρήστη να στέλνει ενέργειες, εντολές ή δεδομένα στον διακομιστή σε αληθινό χρόνο, χωρίς την ανάγκη για παραδοσιακά HTTP αιτήματα.

Η σύνδεση παραμένει **ενεργή και συνεχής**, σε αντίθεση με τα συνήθη αιτήματα που ανοίγουν και κλείνουν (όπως στα `fetch`). Έτσι, επιτυγχάνεται άμεση και αμφίδρομη επικοινωνία, η οποία καθιστά την εφαρμογή πιο διαδραστική(reactive), αποτελεσματική και «ζωντανή».

Το Socket.IO βασίζεται τεχνικά πάνω σε WebSockets, αλλά προσφέρει μια πολύ πιο εύκολη υλοποίηση, με έτοιμες λειτουργίες για αποστολή και λήψη μηνυμάτων.

2.11 Introduction to log system

Στην εφαρμογή μας υλοποιήσαμε ένα **σύστημα καταγραφής (log system)**, με στόχο τη βελτίωση της παρακολούθησης, της συντήρησης και της ασφάλειας της εφαρμογής. Δημιουργήσαμε μια συλλογή (collection) όπου αποθηκεύονται όλα τα logs που παράγονται κατά τη λειτουργία της εφαρμογής, ώστε να μπορούμε να εντοπίζουμε σφάλματα, να καταλαβαίνουμε τι συνέβη και να αποτρέπουμε μελλοντικές αστοχίες.

Το σύστημα log λειτουργεί τόσο σε επίπεδο backend όσο και frontend. Από την πλευρά του server, έχουμε ενσωματώσει logs σχεδόν σε κάθε βασικό βήμα, ώστε οι developers να μπορούν να παρακολουθούν σε πραγματικό χρόνο την κατάσταση του συστήματος και να εντοπίζουν έγκαιρα σοβαρά ζητήματα ή ασυνήθιστες συμπεριφορές.

Επιπλέον, έχουμε τη δυνατότητα να προβάλλουμε αυτά τα logs **στο frontend σε μορφή πίνακα (table view)**, με **φίλτρα, αναζητήσεις και άλλες λειτουργίες**, ώστε οι χρήστες (ή οι admin) να βλέπουν τι συμβαίνει στο σύστημα με διαφάνεια και ευκολία

Αν και το logging προσφέρει σημαντικό έλεγχο, στο μέλλον η βέλτιστη πρακτική θα ήταν να ενσωματωθούν **unit tests και backend tests** για πιο αυτοματοποιημένο και προληπτικό έλεγχο. Αυτες ειναι πιο προχωρημένες τεχνικές που μπορούν να εφαρμοστούν σε επόμενα στάδια ανάπτυξης της εκάστοτε εφαρμογης.

2.12 Declarative Architecture

Κατά την ανάπτυξη της εφαρμογής μας, ακολουθήσαμε μια **δηλωτική αρχιτεκτονική (declarative architecture)** με στόχο την καλύτερη **αναγνωσιμότητα**, **επεκτασιμότητα** και **συντηρησιμότητα** του κώδικα. Η βασική ιδέα πίσω από αυτό το μοντέλο είναι ότι κάθε μέρος του συστήματος — είτε πρόκειται για backend είτε για frontend — πρέπει να είναι ξεκάθαρο στο *τι* κάνει, χωρίς να περιγράφει *πώς* το κάνει.

Στην πράξη, αυτό σημαίνει ότι προσπαθήσαμε να διαχωρίσουμε τη λογική από τη ροή του κώδικα. Κάθε συνάρτηση ή module πρέπει να εκτελεί **μια συγκεκριμένη λειτουργία**, χωρίς να εξαρτάται από εσωτερικές λεπτομέρειες άλλων κομματιών. Για παράδειγμα, μια συνάρτηση δεν χρειάζεται να γνωρίζει τα βήματα με τα οποία θα εκτελέσει κάτι — της λέμε απλά *τι* θέλουμε να κάνει, και όχι *πώς*.

Το ίδιο εφαρμόζεται και στο frontend. Στην περίπτωση του React, προσπαθήσαμε να διατηρούμε τα components **καθαρά και δηλωτικά**, δηλαδή να επικεντρώνονται αποκλειστικά στο rendering. Αν, για παράδειγμα, έχουμε ένα component `ShowBusiness`, τότε ο στόχος του είναι μόνο να εμφανίσει τα δεδομένα. Η επιχειρησιακή λογική (όπως fetch, filtering, state handling) τοποθετείται έξω από το component — για παράδειγμα σε **custom hooks** — έτσι ώστε κάθε κομμάτι του UI να κάνει μόνο *μία δουλειά* και να μπορεί να επαναχρησιμοποιηθεί πιο εύκολα.

Αυτό το στυλ προγραμματισμού ενισχύει τη συνεργασία μεταξύ developers, αφού ο κώδικας είναι πιο ξεκάθαρος, modular και εύκολα δοκιμάσιμος στο μέλλον.

2.13 Designing for scaling

Κατά τον σχεδιασμό της εφαρμογής μας, δώσαμε έμφαση στην **οριζόντια επεκτασιμότητα (horizontal scalability)**. Ο στόχος ήταν να μπορούμε να επεκτείνουμε την εφαρμογή με νέα χαρακτηριστικά ή νέα υποσυστήματα (όπως νέες πηγές scraping) **χωρίς να επηρεάζουμε** ή να μπλέκουμε με τον υπάρχοντα κώδικα.

Για να το πετύχουμε αυτό, οργανώσαμε το backend μας με μια **modular προσέγγιση**, χρησιμοποιώντας **Express routes** και **ξεχωριστούς controllers** για κάθε λειτουργικό κομμάτι. Ουσιαστικά ακολουθήσαμε ένα **MVC μοτίβο (Model-Controller)**, χωρίς το View μέρος, καθώς δεν κάνουμε server-side rendering (SSR). Έτσι, κάθε route αντιπροσωπεύει μια "μονάδα" business logic, με τον αντίστοιχο controller να διαχειρίζεται τη λειτουργία της.

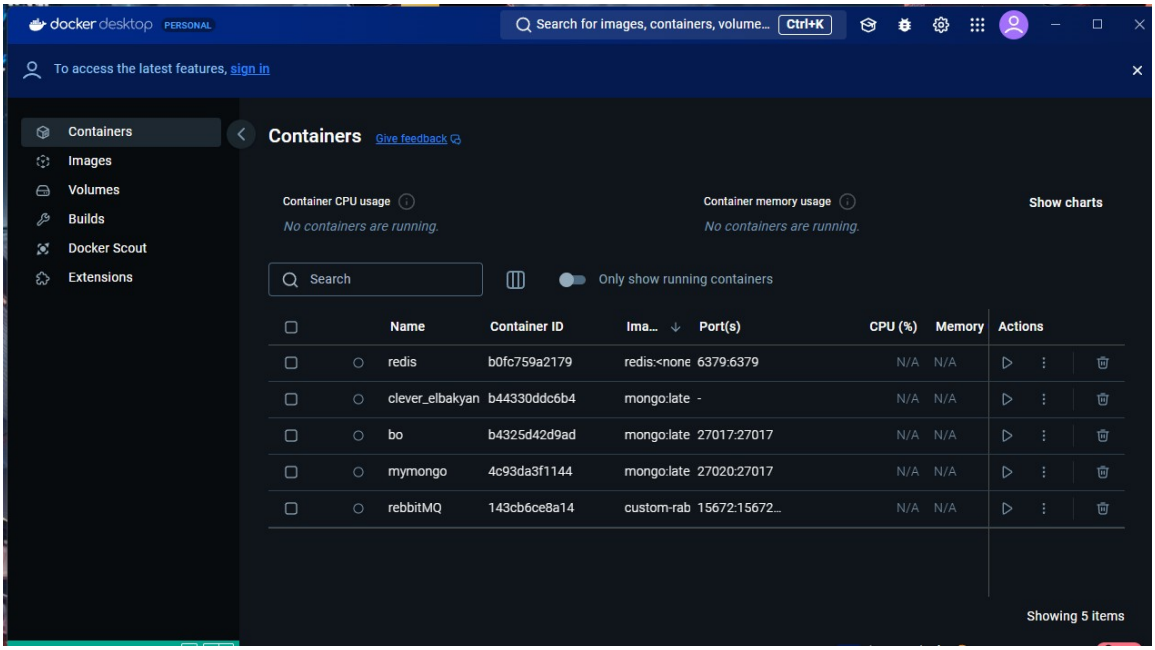
Αυτό μας επιτρέπει να προσθέτουμε εύκολα νέα modules — για παράδειγμα, αν θέλουμε να υποστηρίξουμε scraping από νέα ιστοσελίδα, μπορούμε να δημιουργήσουμε μια νέα route και controller, χωρίς να πειράξουμε τον υπάρχοντα κώδικα. Η αρχιτεκτονική αυτή κάνει την εφαρμογή **εύκολα επεκτάσιμη, οργανωμένη και ευανάγνωστη** για την ομάδα ανάπτυξης.

Η modular αυτή προσέγγιση υποστηρίζει και την **παράλληλη ανάπτυξη** από πολλούς developers, καθώς κάθε κομμάτι είναι απομονωμένο και διαχειρίσιμο, μειώνοντας έτσι την πιθανότητα λαθών ή conflicts.

2.14 Docker

Στην εφαρμογή μας χρησιμοποιήσαμε το **Docker**, που είναι μια πλατφόρμα η οποία επιτρέπει να πακετάρουμε την εφαρμογή μαζί με όλα τα απαραίτητα αρχεία, βιβλιοθήκες, ρυθμίσεις και εξαρτήσεις, μέσα σε ένα **container**. Το container λειτουργεί σαν ένα ελαφρύ, απομονωμένο **εικονικό περιβάλλον (virtual environment)**, που έχει δικό του δίκτυο, πόρτες (ports) και αρχεία, χωρίς να χρειάζεται ολόκληρη η εικονική μηχανή (VM).

Με το Docker, είναι πολύ πιο εύκολο για κάποιον να πάρει τον κώδικά μας και να τον τρέξει στο δικό του σύστημα, χωρίς να χρειάζεται να ανησυχεί για προβλήματα συμβατότητας ή για το αν λείπουν κάποιες εξαρτήσεις. Επιπλέον, το Docker βοηθά στο να διατηρείται η εφαρμογή σταθερή και εύκολα μεταφερόμενη σε διαφορετικά περιβάλλοντα. Αυτό κάνει την ανάπτυξη και τη διανομή της πιο απλή και αποτελεσματική, μειώνοντας τα προβλήματα και επιταχύνοντας τη διαδικασία.



Σχήμα 2.8: Docker UI

2.15 GitHub

Το **GitHub** είναι μια διαδικτυακή πλατφόρμα που χρησιμοποιείται για την αποθήκευση, διαχείριση και συνεργασία σε κώδικα. Βασίζεται στο **Git**, το σύστημα ελέγχου εκδόσεων (version control), και επιτρέπει σε προγραμματιστές να αποθηκεύουν το έργο τους, να παρακολουθούν αλλαγές, να συνεργάζονται με άλλους και να διαχειρίζονται εκδόσεις του κώδικα τους με ευκολία.

Με το GitHub μπορείς να δημιουργήσεις αποθετήρια (repositories), να δουλέψεις σε branches, να κάνεις pull requests, και να ελέγχεις την ιστορία του κώδικα. Επίσης, παρέχει εργαλεία για διαχείριση θεμάτων (issues), wiki, και αυτοματοποιημένες διαδικασίες μέσω GitHub Actions.

2.16 Επίλογος

Κατά την υλοποίηση της εφαρμογής μας, εστιάσαμε σε αρχές και τεχνολογίες που διασφαλίζουν μια σύγχρονη, επεκτάσιμη και ευανάγνωστη λύση. Η επιλογή μιας **δηλωτικής αρχιτεκτονικής** βοήθησε να κρατήσουμε τον κώδικα οργανωμένο, κατανοητό και εύκολα συντηρήσιμο, επιτρέποντας στα μέλη της ομάδας να συνεργάζονται αποτελεσματικά.

Η χρήση της **event-driven αρχιτεκτονικής με Socket.IO** μας έδωσε τη δυνατότητα να προσφέρουμε ζωντανές, αμφίδρομες ενημερώσεις σε πραγματικό χρόνο, βελτιώνοντας την εμπειρία χρήσης και την αλληλεπίδραση με την εφαρμογή.

Η ενσωμάτωση ενός **συστήματος καταγραφής (log system)** μας εξασφαλίζει ότι μπορούμε να παρακολουθούμε και να εντοπίζουμε σφάλματα γρήγορα, κάνοντας την εφαρμογή πιο σταθερή και ασφαλή.

Παράλληλα, η **οριζόντια επεκτασιμότητα** επιτυγχάνεται μέσα από τη διαχωρισμένη οργάνωση routes και controllers, που ακολουθούν ένα MVC-like πρότυπο, καθιστώντας εύκολη την προσθήκη νέων λειτουργιών χωρίς να μπερδεύεται ο κώδικας.

Χρησιμοποιήσαμε το **Docker** για να δημιουργήσουμε φορητά, ελαφριά και απομονωμένα περιβάλλοντα εκτέλεσης, διευκολύνοντας τη διανομή και την ανάπτυξη της εφαρμογής σε διαφορετικά συστήματα.

Τέλος, το **GitHub** αποτέλεσε το κεντρικό σημείο συνεργασίας και διαχείρισης του κώδικα, προσφέροντας εργαλεία για version control, διαχείριση θεμάτων και ομαδική δουλειά.

Όλα αυτά μαζί συνθέτουν μια σύγχρονη, αποδοτική και αξιόπιστη εφαρμογή, έτοιμη να ανταποκριθεί στις ανάγκες τόσο των χρηστών όσο και της ομάδας ανάπτυξης.

Κεφάλαιο 3ο: Αποθήκευση πληροφορίας

3.1 Εισαγωγή

Στο παρόν κεφάλαιο θα περιγράφει η Mongodb, τα collections με όλα τα πεδία της βάσης δεδομένων του scraper, η document-oriented και το σύστημα logger στην βάση μέσω Bulk operation rights. Επιπλέον θα αναλυθούν τα βασικα στοιχεια της document-oriented βαση δεδομεονων

3.2 MongoDb

Η MongoDB είναι ένα είδος βάσης δεδομένων. Δηλαδή, είναι ένα μέρος όπου αποθηκεύουμε και οργανώνουμε πληροφορίες, όπως κάνουν και άλλες βάσεις δεδομένων (π.χ. MySQL). Όμως η MongoDB δουλεύει λίγο διαφορετικά.

Αντί να αποθηκεύει τα δεδομένα σε πίνακες με σειρές και στήλες, όπως οι κλασικές βάσεις, τα αποθηκεύει σε μορφή **εγγράφων** που μοιάζουν με αρχεία **JSON** (σαν τα αντικείμενα στη JavaScript). Δηλαδή, κάθε "κομμάτι" πληροφορίας είναι σαν ένα μικρό αρχείο με ζευγάρια **κλειδί – τιμή**, π.χ.:

```
data_stracture() {
  return {
    "Αριθμός ΓΕΜΗ": "companyNumber",
    EUID: "euid",
    "Διακριτικοί Τίτλοι": "tradeTitles",
    "Επωνυμία με λατινικούς χαρακτήρες": "latinName",
    ΑΦΜ: "vatNumber",
    "Ημ/νία Σύστασης": "establishmentDate",
    "Νομική Μορφή": "legalForm",
    Κατάσταση: "status",
    Διεύθυνση: "address",
    Ιστοσελίδα: "website",
    "e-shop": "eshop",
    "E-mail": "email",
    Τηλέφωνο: "phoneNumber",
    "Αρμόδια Υπηρεσία ΓΕΜΗ": "responsibleService",
    "Τμήμα Επιμελητηρίου": "chamberDepartment",
    "Αριθμός Μητρώου Επιμελητηρίου": "chamberRegistryNumber",
    "Τηλέφωνο Επικοινωνίας Επιμελητηρίου": "chamberPhoneNumber",
    "Ιστοσελίδα Επιμελητηρίου": "chamberWebsite",
    "Ημερομηνία Εγγραφης": "registrationDate",
```

Σχήμα 3.1: JSON Stracture

Αυτό το κάνει πολύ πιο ευέλικτο γιατί:

- Δεν χρειάζεται όλα τα "αρχεία" να έχουν ακριβώς την ίδια δομή.
- Μπορούμε να προσθέτουμε ή να αλλάζουμε πληροφορίες εύκολα χωρίς πολύπλοκες αλλαγές στη βάση.

- Είναι γρήγορη και καλή για μεγάλες εφαρμογές (όπως ιστοσελίδες ή apps) που αλλάζουν συχνά.

Η MongoDB είναι μια μοντέρνα βάση δεδομένων που αποθηκεύει τις πληροφορίες σαν "έγγραφα", είναι εύκολη στη χρήση και πολύ δυνατή για web εφαρμογές.

3.2.1 Δυνατότητες και Πλεονεκτήματα της MongoDB

Η MongoDB έχει πολλά θετικά, ειδικά όταν φτιάχνεις μοντέρνες εφαρμογές:

- **Ευελιξία στα δεδομένα: Δεν χρειάζεται όλα τα "αρχεία" να έχουν την ίδια δομή. Μπορείς να προσθέσεις ή να αλλάξεις πεδία εύκολα.**
- **Γρήγορη απόδοση**: Λειτουργεί πολύ καλά ακόμα και με μεγάλα σύνολα δεδομένων.
- **Εύκολη στην κλίμακα (scalability)**: Αν η εφαρμογή σου μεγαλώσει, μπορείς να προσθέσεις περισσότερους server για να αντέξει την κίνηση.
- **Ανοιχτού κώδικα**: Είναι δωρεάν και υποστηρίζεται από μια μεγάλη κοινότητα.
- **Καλή για εφαρμογές πραγματικού χρόνου**: Όπως chat, games, e-shops κ.λπ.
- **Υποστήριξη γεωγραφικών δεδομένων**: Πολύ χρήσιμο για χάρτες, τοποθεσίες κ.λπ.

3.2.2 Εσωτερική Λειτουργία της MongoDB (Behind the Scenes)

Πίσω από την "κουρτίνα", η MongoDB:

- **Αποθηκεύει τα δεδομένα σε έγγραφα BSON**, μια μορφή που μοιάζει με JSON αλλά είναι πιο γρήγορη για υπολογιστή και ποιο ελαφριά σε όγκο αν και υπάρχει μια μικρή καθυστέρηση στην μετατροπή απο json σε bson.
- Αυτά τα έγγραφα μπαίνουν σε **συλλογές (collections)**.
- Κάθε φορά που κάνεις αναζήτηση ή αποθήκευση, η MongoDB χρησιμοποιεί έναν μηχανισμό που λέγεται **engine** για να διαβάσει και να γράψει τα δεδομένα γρήγορα.
- Χρησιμοποιεί **indexes**, όπως κάνουν και οι βιβλιοθήκες, για να βρίσκει τα δεδομένα γρήγορα

3.2.3 Αρχιτεκτονική Πελάτη (Client Architecture)

Όταν μια εφαρμογή (π.χ. ένα site) θέλει να συνδεθεί στη MongoDB:

- Υπάρχει ένας **πελάτης (client)**, που είναι είτε ένα πρόγραμμα είτε ένα εργαλείο, που στέλνει εντολές (queries) στη MongoDB.
- Ο client επικοινωνεί με τον server της MongoDB μέσω **δικτύου** (TCP/IP).
- Χρησιμοποιεί **drivers**, δηλαδή ειδικά κομμάτια κώδικα για να "μιλάει" με τη MongoDB. Υπάρχουν drivers για Python, JavaScript, Java κ.λπ.
- Η επικοινωνία γίνεται με “ερωτήσεις” και “απαντήσεις” σε μορφή BSON.

3.2.4 Αρχιτεκτονική Sharding (Sharded Cluster Architecture)

Το **sharding** είναι ο τρόπος της MongoDB να διαχειρίζεται **πολύ μεγάλα σύνολα δεδομένων**. Τι κάνει;

- **Χωρίζει τα δεδομένα σε κομμάτια (shards)** και τα αποθηκεύει σε διαφορετικούς servers.
- Ένας ειδικός server που λέγεται **mongos** αναλαμβάνει να στείλει τα queries στον σωστό shard.

- Έτσι, η βάση μπορεί να **εξυπηρετεί πολλά αιτήματα ταυτόχρονα** και να **δουλεύει γρήγορα**, ακόμα και όταν έχει εκατομμύρια εγγραφές.

Με απλά λόγια: Αντί να φορτώνεται ένας μόνο server με όλα, τα "μοιράζει" σε πολλούς.

3.2.5 Αρχιτεκτονική Συλλογών (Collection-Level Architecture)

- Οι **συλλογές** (collections) είναι το σημείο όπου μπαίνουν τα έγγραφα (σαν να λέμε "οι πίνακες" στις άλλες βάσεις).
- Κάθε συλλογή μπορεί να έχει όσα έγγραφα θέλεις.
- Δεν χρειάζεται τα έγγραφα να έχουν την ίδια μορφή.
- Μπορείς να φτιάξεις **indexes** για γρηγορότερη αναζήτηση.
- Οι συλλογές μπορούν να είναι **capped** (δηλαδή με όριο στο μέγεθος, χρήσιμο για logs).

Πλεονέκτημα: Ευκολία στη διαχείριση και μεγάλη ευελιξία.

3.2.6 Cursors και Μηχανισμός Ανάκτησης Δεδομένων (Cursors & Data Retrieval)

Όταν ζητάς δεδομένα από τη MongoDB, δεν στα δίνει όλα μονομιάς (εκτός αν είναι λίγα).

- Δημιουργεί έναν **cursor**, δηλαδή ένα "δειγματολήπτη" που διαβάζει τα δεδομένα λίγα-λίγα.
- Αυτό είναι χρήσιμο γιατί:
  - Γλιτώνει μνήμη.
  - Δεν βαραίνει το σύστημα.
  - Μπορείς να διαχειριστείς μεγάλα αποτελέσματα πιο ομαλά.

Μπορείς να κάνεις **περιορισμούς**, **ταξινόμηση**, **φιλτράρισμα** στα δεδομένα μέσα από τον cursor

3.3 Μαζική Εισαγωγή Δεδομένων (Bulk Write)

Η **μαζική εισαγωγή**, ή αλλιώς *bulk write*, είναι όταν θέλουμε να κάνουμε πολλές ενέργειες στη βάση δεδομένων ταυτόχρονα — για παράδειγμα, να εισάγουμε, να ενημερώσουμε ή να διαγράψουμε πολλά έγγραφα με μία μόνο εντολή.

Χρησιμοποιώντας **JavaScript κώδικα** σε συνδυασμό με τη **φύση της MongoDB**, μπορούμε πολύ εύκολα να δημιουργήσουμε ή να επεξεργαστούμε τα έγγραφα που θέλουμε να εισάγουμε. Με αυτόν τον τρόπο, γεμίζουμε τα δεδομένα που θέλουμε να περάσουμε και τα στέλνουμε όλα μαζί στη βάση.

Η JavaScript ταιριάζει **ιδανικά με τη MongoDB**, γιατί μας επιτρέπει να χειριζόμαστε τα έγγραφα εύκολα (σαν αντικείμενα), να τα φτιάχνουμε, να τα αλλάζουμε, και μετά να τα εισάγουμε ή να τα ενημερώσουμε όλα μαζί, με μία μαζική εντολή. Αυτό κάνει τη διαδικασία πολύ πιο απλή και αποδοτική.

Γι' αυτόν τον λόγο, ο συνδυασμός MongoDB και JavaScript είναι **ιδανικός** για τέτοιες λειτουργίες μαζικής εισαγωγής ή ενημέρωσης δεδομένων.

Η μαζική εισαγωγή στη MongoDB είναι:

- Γρήγορη
- Οικονομική σε πόρους
- Πολύ χρήσιμη όταν δουλεύεις με μεγάλους όγκους δεδομένων

Αν έχεις εισαγωγές από Excel, JSON, ή άλλες βάσεις, το bulk write είναι ο καλύτερος τρόπος να τις περάσεις σωστά και αποδοτικά.

```
const bulkOps = results.map((id) => ({
  insertOne: {
    document: { id: id },
  },
}));

try {
  const result = await businessIdsModel.bulkWrite(bulkOps, {
    ordered: false,
  });
  successCount = result.insertedCount;

  console.log("Scraping process completed successfully");
} catch (err) {
  console.error("Error during bulk write:", err);
  await service.logError(err);
}
```

Σχήμα 3.2: Bulk write BusinessesIds

```
async  saveScrapedData(scrapedData) {
  try {
    const bulkOperations = scrapedData.map((actor) => ({
      updateOne: {
        filter: { fullname: actor.fullname },
        update: { $set: actor },
        upsert: true,
      },
    }));

    const result = await ActorModel.bulkWrite(bulkOperations);
    console.log("Bulk write operation completed:", result);
    return result;
  } catch (error) {
    console.error("Error during bulk write operation:", error);
  }
}
```

Σχήμα 3.3: Bulk write Actors

3.4 Document Based Database

Οι **document-based** βάσεις δεδομένων, όπως η **MongoDB**, αποθηκεύουν τα δεδομένα σε μορφή **εγγράφων** (documents), συνήθως σε μορφή **JSON ή BSON**.

- Κάθε "έγγραφο" είναι σαν ένα **αυτόνομο αντικείμενο** που περιέχει όλα τα στοιχεία για μια οντότητα (π.χ. έναν πελάτη, ένα προϊόν, μια παραγγελία).
- Τα έγγραφα αποθηκεύονται σε **συλλογές (collections)**, που είναι κάτι αντίστοιχο με τους πίνακες σε σχεσιακές βάσεις.

Πλεονεκτήματα (Pros)

Ευελιξία στη δομή των δεδομένων :

Δεν χρειάζεται να έχεις συγκεκριμένο σχήμα (schema). Κάθε έγγραφο μπορεί να έχει διαφορετικά πεδία, και αυτό επιτρέπει γρήγορες αλλαγές χωρίς τροποποίηση της βάσης.

Γρήγορη Ανάγνωση και Εγγραφή :

Επειδή τα έγγραφα είναι αυτοτελή, η βάση διαβάζει ή γράφει δεδομένα γρήγορα, χωρίς πολύπλοκα joins ή σχέσεις.

Καλή για εφαρμογές με JSON :

Αν φτιάχνεις web εφαρμογές με JavaScript (π.χ. Node.js), τα δεδομένα σου είναι ήδη σε JSON, οπότε συνδέονται τέλεια με document databases.

Εύκολο στην κλίμακα (Scalable) :

Οι document βάσεις μπορούν να διαχειριστούν τεράστιους όγκους δεδομένων μέσω **sharding** και **replication**.

Ιδανική για πραγματικού χρόνου εφαρμογές

Όπως chats, e-commerce, παρακολούθηση δεδομένων, social apps κ.λπ.

Μειονεκτήματα (Pros)

Όχι ιδανική για πολύπλοκες σχέσεις **:**

Αν η εφαρμογή σου βασίζεται σε πολλές σχέσεις μεταξύ πινάκων , τότε ίσως μια σχεσιακή βάση (SQL) να είναι καλύτερη .

Δυσκολότερη η τυποποίηση :

Χωρίς schema, μπορεί να καταλήξεις με έγγραφα διαφορετικής δομής, που είναι δύσκολο να διαχειριστείς σε μεγάλο σύστημα.

Κατανάλωση χώρου :

Καθώς κάθε έγγραφο περιέχει όλα τα δεδομένα μέσα του, μπορεί να καταναλώσει περισσότερο αποθηκευτικό χώρο.

Πιο σύνθετη αναζήτηση σε nested δεδομένα :

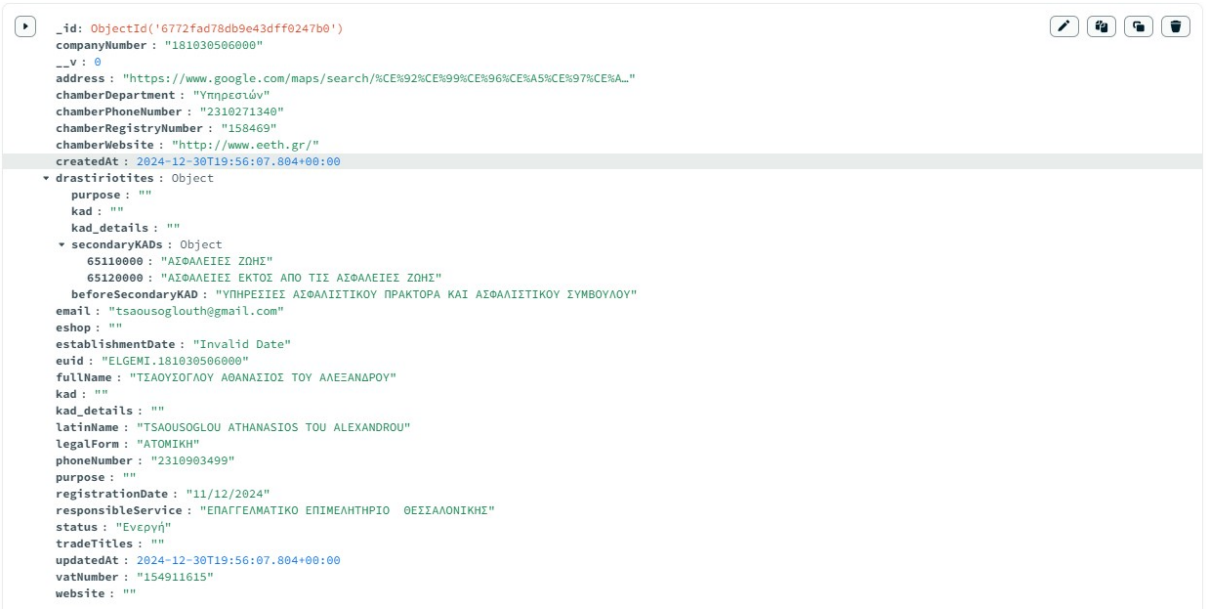
Αν έχεις δεδομένα "μέσα σε άλλα δεδομένα" (όπως arrays ή υπο-αντικείμενα), κάποιες αναζητήσεις μπορεί να είναι πιο δύσκολες ή αργές.

3.5 Βάση Δεδομένων του Scraper

3.5.1 Collection Actors

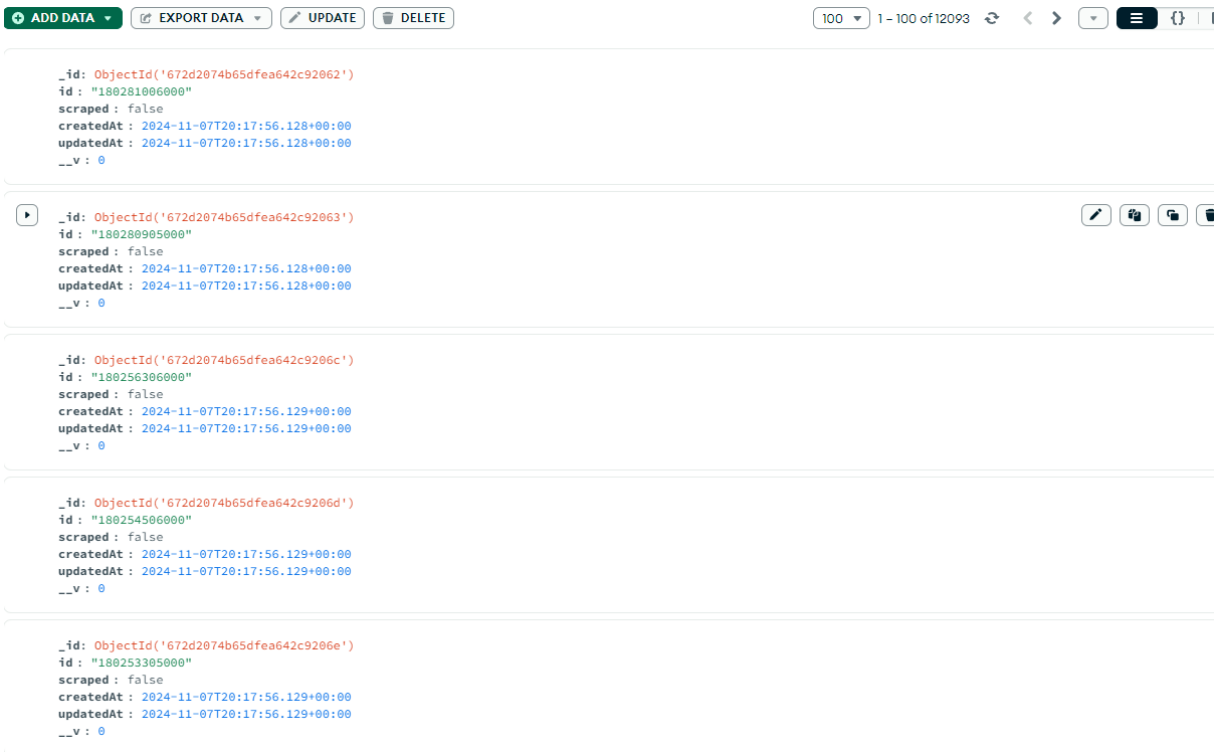
Σχήμα 3.4: Actors Collection

3.5.2 Collection Businesses



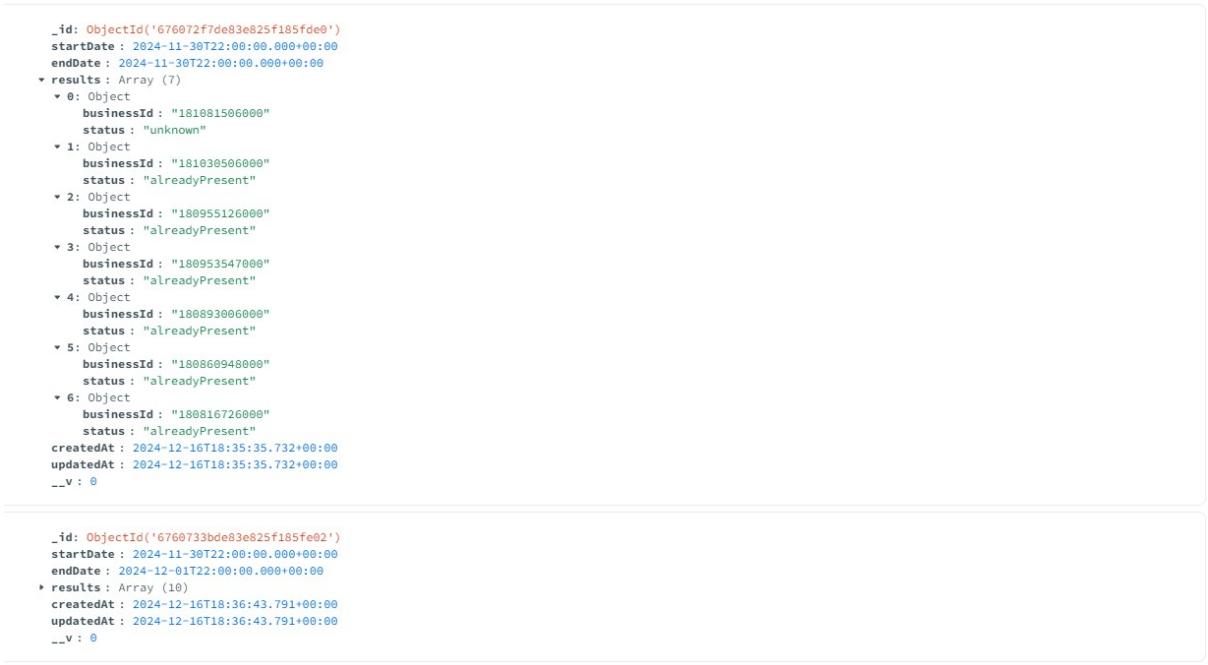
Σχήμα 3.5: Businesses Collection

3.5.3 Collection Businessids



Σχήμα 3.6: Businessids Collection

3.5.4 Collection Logs



Σχήμα 3.7: Logs Collection

3.6 Επίλογος

Η δομή της βάσης δεδομένων που χρησιμοποιείται από τον scraper είναι απλή αλλά αποτελεσματική. Χωρίζοντας τα δεδομένα σε ξεχωριστά collections (`business`, `businessessids`, `logs`), καταφέρνουμε να έχουμε:

- **Οργάνωση** των επιχειρησιακών πληροφοριών με σαφή και δομημένο τρόπο.
- **Αποφυγή διπλοεγγραφών**, χάρη στη χρήση αναγνωριστικών.
- **Ιχνηλασιμότητα και παρακολούθηση της λειτουργίας** του scraper, μέσω των logs.

Αυτό το μοντέλο καθιστά τη MongoDB ιδανική επιλογή για τέτοιου τύπου εφαρμογές, καθώς συνδυάζει ευελιξία, ταχύτητα και ευκολία διαχείρισης δεδομένων, ειδικά όταν αυτά προέρχονται από αυτοματοποιημένες διαδικασίες συλλογής.

Η απλότητα του σχήματος επιτρέπει την εύκολη επεκτασιμότητα στο μέλλον, καλύπτοντας επιπλέον ανάγκες ή νέες λειτουργίες που μπορεί να προκύψουν στην πορεία.

Κεφάλαιο 4ο: Σύστημα scraper

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο περιγράφουμε πώς δημιουργήσαμε το **γραφικό περιβάλλον χρήστη (UI)** και τον τρόπο με τον οποίο **ο scraper αλληλεπιδρά με αυτό**, προσφέροντας μια πλήρως λειτουργική και ευχάριστη εμπειρία στον χρήστη.

Ο scraper μας λειτουργεί ανοίγοντας έναν **headless browser** (δηλαδή έναν browser χωρίς ορατό παράθυρο), και εκτελεί αυτοματοποιημένες ενέργειες μέσα σε μια πραγματική ιστοσελίδα. Από εκεί, αντλεί δεδομένα με βάση **συγκεκριμένα φίλτρα και ημερομηνιακό εύρος** που θέτει ο χρήστης, όπως π.χ. χώρα, ταχυδρομικός κώδικας, αν η επιχείρηση είναι ενεργή, κ.ά.

Ο πρώτος scraper εντοπίζει όλα τα **business IDs (κωδικοί GEMI)** που υπάρχουν μέσα σε συγκεκριμένο χρονικό διάστημα και με τα αντίστοιχα φίλτρα. Ο scraper μιμείται τον χρήστη, "πατώντας" κουμπιά για αλλαγή σελίδας (pagination) και μαζεύει σταδιακά όλα τα IDs.

Μόλις ολοκληρωθεί η συλλογή των IDs, ακολουθεί ένας **δεύτερος scraper**, ο οποίος επισκέπτεται για κάθε ID μια νέα μοναδική σελίδα με όλα τα στοιχεία της επιχείρησης. Εκεί αντλούμε πληροφορίες όπως:

- Τίτλος και δραστηριότητα επιχείρησης
- Πρωτεύουσα και δευτερεύουσα δραστηριότητα
- Στοιχεία επικοινωνίας
- Διεύθυνση και τοποθεσία

Για να γίνει πιο κατανοητή η διαδικασία και πιο **εύχρηστη για τον τελικό χρήστη**, σχεδιάσαμε ένα **φιλικό UI με React**. Μέσα από αυτό, ο χρήστης μπορεί:

- Να ορίσει εύκολα τα φίλτρα που θέλει (π.χ. ημερομηνίες, περιοχές, καταστάσεις)
- Να παρακολουθεί **σε πραγματικό χρόνο** την πρόοδο του scraper, δηλαδή σε ποιο βήμα βρίσκεται, αν κάποια επιχείρηση βρέθηκε, αν έχει ήδη συλλεχθεί, αν υπήρξε κάποιο πρόβλημα κ.λπ.

Το interface έχει σχεδιαστεί έτσι ώστε να δίνει την αίσθηση ότι "ο χρήστης κάνει τη δουλειά", ενώ στην πραγματικότητα **ο scraper αυτοματοποιεί τα πάντα στο παρασκήνιο**.

Επίσης, προσθέσαμε **μηχανισμούς φίλτρων για στοχευμένη αναζήτηση**, όπου ο χρήστης μπορεί να βλέπει live τι κάνει ο scraper, καθώς εκείνος ανοίγει τη σελίδα-στόχο και συλλέγει τις πληροφορίες.

Ο δεύτερος scraper ήταν πιο απλός, γιατί δεν είχε inputs, selectors ή pagination. Εκεί δεν χρειάστηκε να "μιμηθούμε" τη συμπεριφορά του χρήστη. Χρησιμοποιήσαμε απλό JavaScript και όχι React, καθώς το DOM ήταν πιο "καθαρό". Αντίθετα, η πρώτη σελίδα χρησιμοποιούσε React με Tailwind και τυχαία ονόματα κλάσεων, κάνοντάς την πιο δύσκολη στην πλοήγηση και την εξαγωγή δεδομένων.

Ο τελικός στόχος του έργου ήταν να **συλλέξουμε όλους τους κωδικούς GEMI και τα πλήρη στοιχεία τους**, προσφέροντας στον χρήστη έναν απλό αλλά αποδοτικό τρόπο να το κάνει αυτό, μέσα από μια ομαλή και διαδραστική εμπειρία.

4.2 Χρήση React για τη Δημιουργία του UI

Η επιλογή της **React** για την ανάπτυξη του περιβάλλοντος χρήστη (UI) του συστήματος έγινε με στόχο να παρέχουμε μια **διαδραστική(reactive), εύχρηστη και δυναμική εμπειρία** στον τελικό χρήστη. Το UI λειτουργεί ως γέφυρα μεταξύ του χρήστη και του scraper, προσφέροντας δυνατότητα ελέγχου αλλά και ορατότητα σε πραγματικό χρόνο.

4.2.1 Τι είναι η React

Η **React** είναι μια δημοφιλής βιβλιοθήκη(όχι framework) JavaScript που χρησιμοποιείται για τη δημιουργία **μοντέρνων web εφαρμογών με διαδραστικά UI**. Δημιουργήθηκε από τη Facebook και βασίζεται στην έννοια των **(components)** — δηλαδή μικρών, επαναχρησιμοποιούμενων κομματιών κώδικα που χειρίζονται μια συγκεκριμένη λειτουργία της διεπαφής.

Με το React, μπορούμε να:

- Ενημερώνουμε άμεσα το UI όταν αλλάζουν τα δεδομένα (π.χ. όταν ο scraper βρει νέα επιχείρηση).
- Χειριζόμαστε πολύ εύκολα την κατάσταση της εφαρμογής (state).
- Δημιουργούμε επαναχρησιμοποιούμενα, καθαρά και απομονωμένα modules.
- Είναι πολύ καλή για την επαναχρησιμοποίηση ίδιου κώδικα(reusable-components).
- Χρησιμοποιεί συντακτικό που διευκολύνει τη χρήση και ενισχύει την επαναχρησιμοποίηση των components (JSX).

4.2.2 Γιατί React

Επιλέξαμε την React γιατί:

- **Είναι ιδανική για εφαρμογές που αλλάζουν συνεχώς περιεχόμενο** σε πραγματικό χρόνο (όπως όταν τρέχει ο scraper).
- **Επιτρέπει εύκολη διαχείριση των φίλτρων**, των αποτελεσμάτων και της κατάστασης του scraper.
- Είναι **ελαφριά, αποδοτική και συμβατή** με τις περισσότερες μοντέρνες τεχνολογίες.
- Υπάρχει μεγάλη υποστήριξη και κοινότητα, κάτι που βοηθά στη γρήγορη επίλυση προβλημάτων.

Η χρήση της React σε συνδυασμό με άλλες τεχνολογίες, όπως Tailwind CSS για το styling και βιβλιοθήκες για τη διαχείριση της κατάστασης (όπως useState/useEffect), μας έδωσε τη δυνατότητα να φτιάξουμε ένα **μοντέρνο και responsive UI**, που προσαρμόζεται εύκολα σε διάφορες συσκευές και ανάγκες.

4.3 Υλοποίηση User-Friendly UI

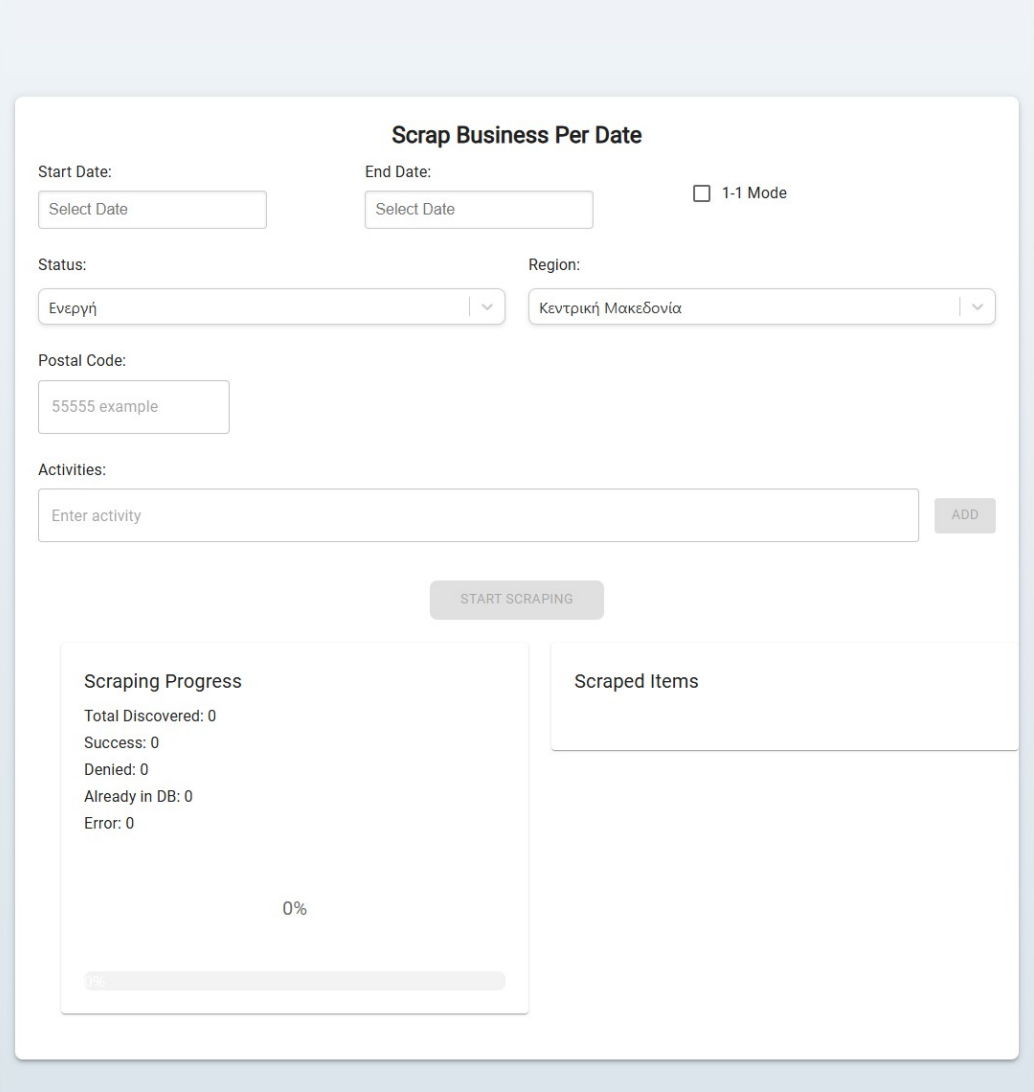
Η βασική προτεραιότητά μας ήταν να δημιουργήσουμε ένα **φιλικό προς τον χρήστη περιβάλλον**, όπου ακόμη και κάποιος χωρίς τεχνικές γνώσεις να μπορεί να χειριστεί εύκολα τον scraper και να παρακολουθεί την πορεία του.

Για να πετύχουμε αυτό το στόχο, δώσαμε έμφαση σε μερικά βασικά σημεία:

- **Απλή και καθαρή διάταξη:** Το UI σχεδιάστηκε ώστε να είναι ξεκάθαρο, με εμφανή κουμπιά και πεδία φίλτρων, χωρίς περιττές πληροφορίες που θα μπερδέψουν τον χρήστη.

- **Εύκολη εισαγωγή φίλτρων:** Ο χρήστης μπορεί να ορίσει εύκολα φίλτρα όπως ημερομηνίες, περιοχή, κατάσταση επιχείρησης και άλλα, χωρίς να χρειάζεται να καταλάβει τη δομή της βάσης δεδομένων.
- **Άμεση ενημέρωση:** Χρησιμοποιώντας τις δυνατότητες του React, το UI ενημερώνεται αυτόματα όταν αλλάζει η κατάσταση του scraper, ώστε ο χρήστης να βλέπει ακριβώς τι συμβαίνει κάθε στιγμή.
- **Διαδραστικά στοιχεία:** Χρησιμοποιήσαμε progress bars, μηνύματα κατάστασης και πίνακες για να δώσουμε οπτική πληροφόρηση για την πρόοδο, τα αποτελέσματα και τυχόν σφάλματα.
- **Ευκολία χειρισμού:** Το UI είναι responsive και λειτουργεί καλά σε υπολογιστές, tablets και κινητά, ώστε ο χρήστης να μπορεί να το χρησιμοποιεί από όποια συσκευή επιθυμεί

Με αυτήν την προσέγγιση, το περιβάλλον δεν είναι απλά μια “φόρμα”, αλλά ένα εργαλείο που βοηθά τον χρήστη να κατανοήσει και να ελέγξει όλη τη διαδικασία του scraping με ασφάλεια και άνεση.



Σχήμα 4.1:User UI

4.4 Real-Time Progress και Ενημερώσεις

Μία από τις πιο σημαντικές λειτουργίες του UI είναι η δυνατότητα να παρακολουθεί ο χρήστης σε πραγματικό χρόνο την πορεία του scraper. Αυτό προσφέρει διαφάνεια και τον κρατάει ενήμερο για το τι συμβαίνει κάθε στιγμή.

Πώς λειτουργεί:

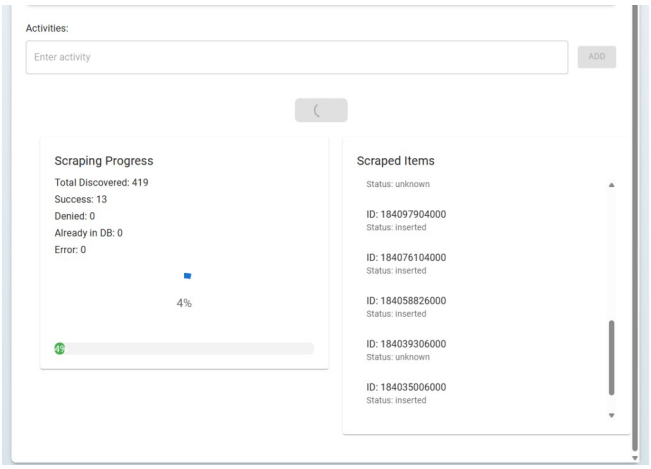
Καθώς ο scraper εκτελεί κάθε βήμα του, ο server στέλνει άμεσα ενημερώσεις προς το UI μέσω **Socket.IO**, που επιτρέπει την επικοινωνία σε πραγματικό χρόνο. Στο React, χρησιμοποιούμε ένα ειδικό hook, το `useEffect`, που "ακούει" αυτές τις αλλαγές και ενημερώνει την οθόνη με τα νέα δεδομένα.

Έτσι, το UI εμφανίζει συνεχώς πληροφορίες όπως:

- Πόσες επιχειρήσεις έχουν βρεθεί μέχρι στιγμής
- Αν κάποια επιχείρηση έχει ήδη συλλεχθεί προηγουμένως
- Ποια σελίδα βρίσκεται τώρα σε επεξεργασία
- Αν υπάρχουν σφάλματα ή προβλήματα κατά την εκτέλεση

Χρησιμοποιούνται επίσης οπτικά στοιχεία, όπως progress bars και μηνύματα κατάστασης, που δείχνουν την πρόοδο είτε σε ποσοστό είτε βήμα-βήμα.

Αυτή η τεχνική επιτρέπει στο χρήστη να βλέπει άμεσα τι συμβαίνει χωρίς να χρειάζεται να ανανεώσει τη σελίδα, αυξάνοντας έτσι την εμπιστοσύνη του και επιτρέποντάς του να αντιδρά γρήγορα αν παρουσιαστεί κάποιο πρόβλημα.



Σχήμα 4.2:Real Time Progress UI

4.5 Φίλτρα για τη Στοχοποιημένη Αναζήτηση

```
useEffect(() => {
  socket.on('discovered', (data) => {
    setProgress((prev) => ({
      ...prev,
      discovered: prev.discovered + data.discovered,
    }));
  });

  socket.on("scrape-progress-item", (data) => {
    console.log("scrape-progress-item", data);

    setProgress((prev) => {
      const updatedItems = prev.items.map((item) =>
        item.id === data.id ? { ...item, status: data.status, stats: data.stats } : item
      );

      updatedItems.push({ id: data.id, status: data.status, stats: data.stats });

      let { alreadyPresent, failed, inserted, scraped } = data.stat || {};

      if (data.stat) {
        alreadyPresent = data.stat.alreadyPresent;
        failed = data.stat.failed;
        inserted = data.stat.inserted;
        scraped = data.stat.scraped;
      }

      return {
        ...prev,
        items: updatedItems,
        startDate: data.startDate,
        endDate: data.endDate,
        alreadyPresent: prev.alreadyPresent + (alreadyPresent || 0),
        failed: prev.failed + (failed || 0),
        inserted: prev.inserted + (inserted || 0),
        scraped: prev.scraped + (scraped || 0),
      };
    });
  });

  return () => {
    socket.disconnect();
  };
}, []);
```

Σχήμα 4.3: Socket IO

Για να γίνει η αναζήτηση πιο **στοχευμένη και αποδοτική**, σχεδιάσαμε και υλοποιήσαμε στο UI μια σειρά από φίλτρα που επιτρέπουν στον χρήστη να περιορίσει τα αποτελέσματα σύμφωνα με συγκεκριμένα κριτήρια.

Τα βασικά χαρακτηριστικά των φίλτρων είναι:

- **Ευκολία στη χρήση:** Οι επιλογές φίλτρων είναι ξεκάθαρες και προσβάσιμες, ώστε ο χρήστης να μπορεί να επιλέξει χωρίς δυσκολία παραμέτρους όπως περιοχή, ημερομηνίες, κατάσταση επιχείρησης (ενεργή ή μη), και άλλα.
- **Δυναμική ενημέρωση:** Με την εφαρμογή ή αλλαγή ενός φίλτρου, το UI ενημερώνεται άμεσα, ώστε ο χρήστης να βλέπει αμέσως τα αντίστοιχα αποτελέσματα.

- **Συνδυασμός φίλτρων:** Επιτρέπεται η χρήση πολλαπλών φίλτρων ταυτόχρονα, για πιο ακριβή και ειδική αναζήτηση.
- **Ορατότητα στην πρόοδο:** Καθώς το scraper τρέχει με τα επιλεγμένα φίλτρα, ο χρήστης βλέπει ζωντανά πώς τα κριτήρια επηρεάζουν την πρόοδο και τα δεδομένα που συλλέγονται.

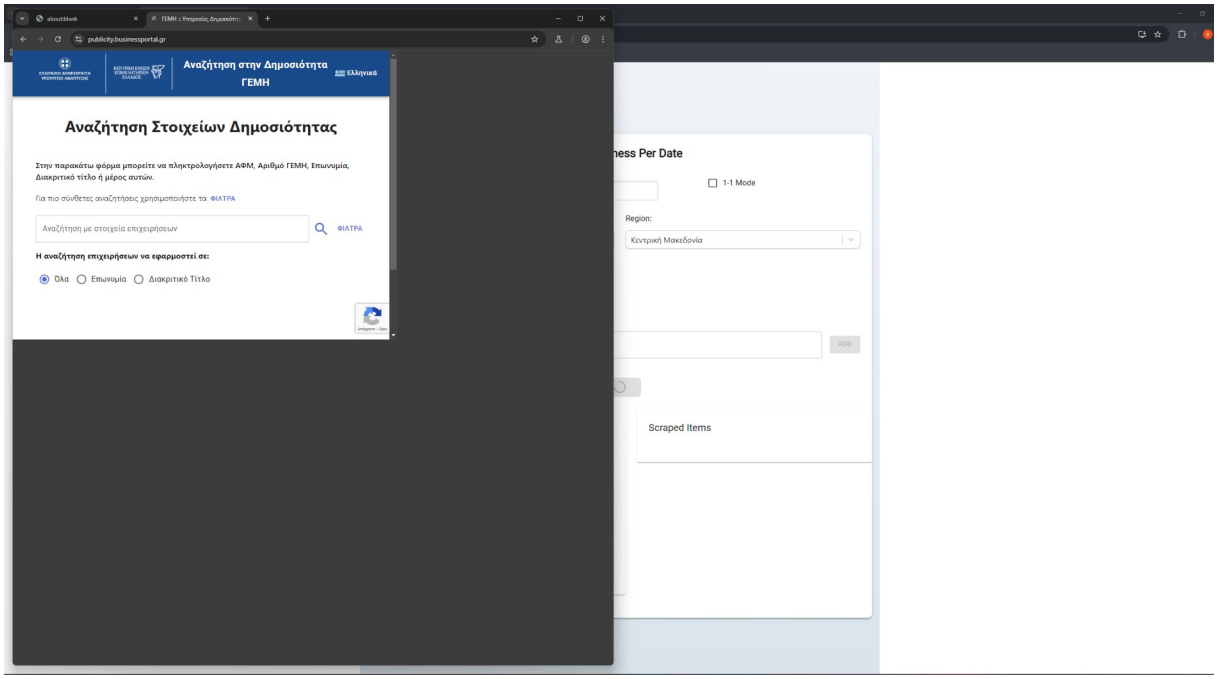
Αυτή η λειτουργικότητα βοηθά στο να γίνει η διαδικασία πιο αποδοτική, μειώνοντας το χρόνο αναζήτησης και επιτρέποντας στο χρήστη να εστιάσει σε συγκεκριμένες επιχειρήσεις που τον ενδιαφέρουν περισσότερο.

4.6 Ροή Προγράμματος

Η ροή λειτουργίας του προγράμματος έχει σχεδιαστεί έτσι ώστε να είναι απλή, γραμμική και ευέλικτη, επιτρέποντας στον χρήστη να εκτελέσει το scraping με ελάχιστα βήματα και μέγιστη αποτελεσματικότητα.

1. Εισαγωγή φίλτρων: Ο χρήστης επιλέγει τα φίλτρα που επιθυμεί στο UI, όπως ημερομηνιακό εύρος, περιοχή, κατάσταση επιχείρησης κτλ.
2. Εκκίνηση scraping: Με ένα κλικ ξεκινά η διαδικασία του πρώτου scraper, που συλλέγει όλα τα business IDs που αντιστοιχούν στα φίλτρα.
3. Παρακολούθηση προόδου: Καθ' όλη τη διάρκεια, ο χρήστης ενημερώνεται σε πραγματικό χρόνο για την πορεία του scraper.
4. Συλλογή λεπτομερειών: Μετά την ολοκλήρωση του πρώτου βήματος, ο δεύτερος scraper επισκέπτεται τις σελίδες κάθε επιχείρησης και συλλέγει τις αναλυτικές πληροφορίες.
5. Ολοκλήρωση και αποθήκευση: Τα δεδομένα αποθηκεύονται στη βάση, και ο χρήστης λαμβάνει ενημέρωση για το τέλος της διαδικασίας.

Η ροή αυτή επιτρέπει την εύκολη επέκταση και βελτίωση του προγράμματος στο μέλλον, ενώ παράλληλα προσφέρει μια ξεκάθαρη εμπειρία χρήσης.



Σχήμα 4.4: Scraper Browser

4.7 Στόχος Προγράμματος

Ο κύριος στόχος του προγράμματος είναι να **αυτοματοποιήσει τη διαδικασία συλλογής δεδομένων επιχειρήσεων (business data)** από δημόσιες πλατφόρμες, με τρόπο αξιόπιστο, αποδοτικό και εύχρηστο για τον τελικό χρήστη.

- **Αυτόματη συλλογή όλων των GEMI (Γ.Ε.ΜΗ.) IDs** που αντιστοιχούν στα φίλτρα που επιλέγει ο χρήστης (π.χ. τοποθεσία, ημερομηνίες, κατάσταση επιχείρησης).
- Για κάθε GEMI ID, **εξαγωγή αναλυτικών πληροφοριών** όπως:
- Επωνυμία, ΑΦΜ, έδρα
- Κύρια και δευτερεύουσα δραστηριότητα
- Στοιχεία επικοινωνίας (τηλέφωνο, email, κ.λπ.)
- Κατάσταση επιχείρησης
- **Αποθήκευση όλων των δεδομένων** σε MongoDB, ώστε να είναι εύκολα προσβάσιμα και επεξεργάσιμα.
- Παροχή ενός **φιλικού UI**, που επιτρέπει στον χρήστη να έχει ενεργή συμμετοχή στη διαδικασία και να παρακολουθεί την πρόοδο βήμα-βήμα.

Ουσιαστικά, η εφαρμογή αυτή λειτουργεί ως ένα **εργαλείο scraping και παρακολούθησης**, που συνδυάζει τεχνική ακρίβεια με ευχρηστία, και μπορεί εύκολα να προσαρμοστεί για άλλες πηγές ή τύπους δεδομένων στο μέλλον.

4.8 Επίλογος

Με την ανάπτυξη του προγράμματος αυτού, καταφέραμε να δημιουργήσουμε ένα ολοκληρωμένο σύστημα scraping, το οποίο είναι ταυτόχρονα **τεχνικά αποτελεσματικό** και **εύκολο στη χρήση**.

Συνδυάσαμε:

- **Τη δύναμη της αυτοματοποίησης** μέσω headless browser και έξυπνου DOM parsing
- **Την ευελιξία του React** για τη δημιουργία ενός διαδραστικού και user-friendly περιβάλλοντος
- **Την ταχύτητα και απλότητα της MongoDB** για αποθήκευση και ανάκτηση των δεδομένων
- **Ενημέρωση σε πραγματικό χρόνο** με χρήση Socket.IO για καλύτερη εμπειρία χρήστη

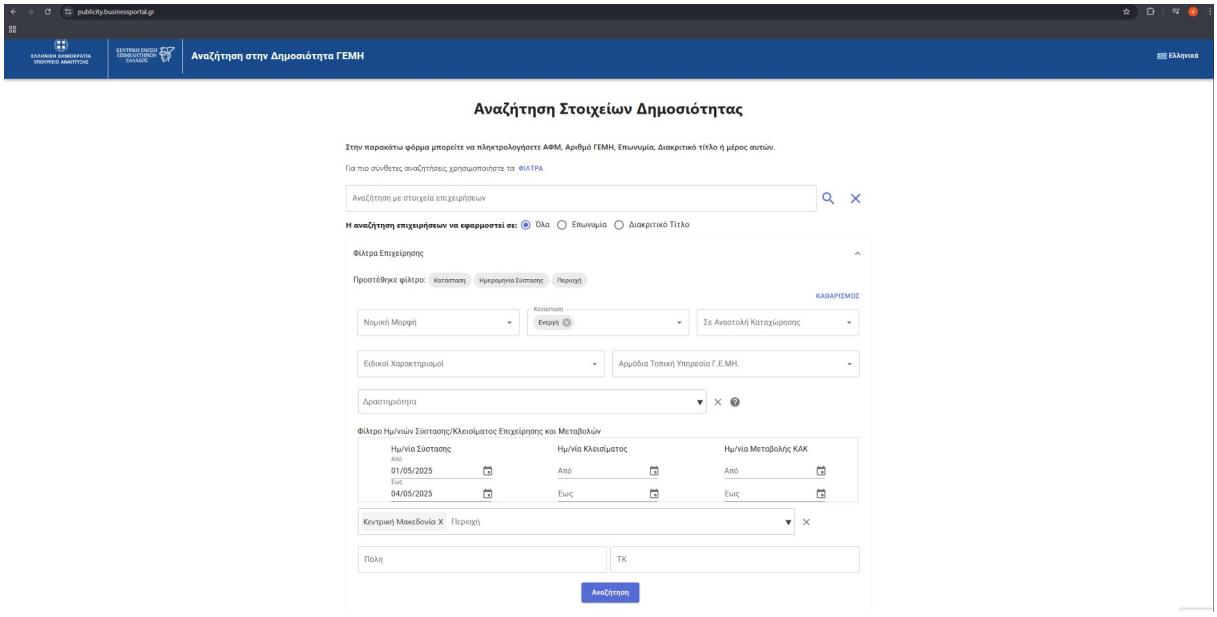
Ο χρήστης έχει πλέον τη δυνατότητα να ορίσει ακριβή φίλτρα, να βλέπει την πορεία του scraper σε πραγματικό χρόνο και να έχει άμεση πρόσβαση στα αποτελέσματα — χωρίς να χρειάζεται τεχνικές γνώσεις.

Η υλοποίηση αυτή δεν είναι μόνο ένα τεχνικό εργαλείο, αλλά ένα παράδειγμα για το πώς η τεχνολογία μπορεί να **απλοποιήσει σύνθετες διαδικασίες** και να φέρει τη δύναμη των δεδομένων στα χέρια του τελικού χρήστη.

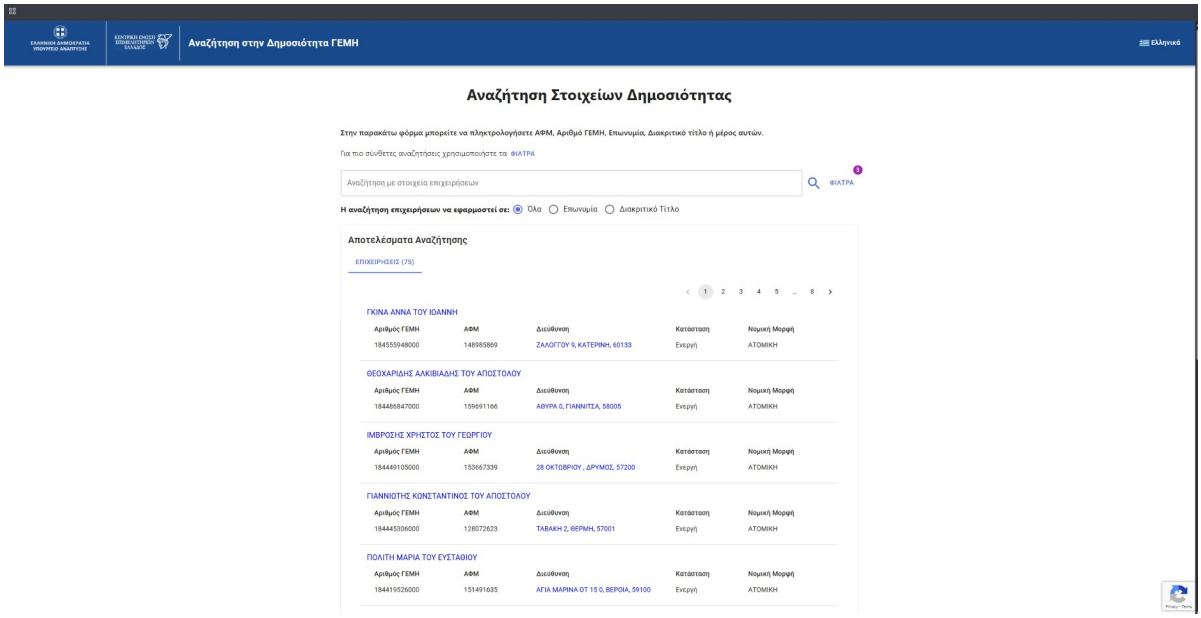
Κεφάλαιο 5ο: Συμπεράσματα

5.1 Έλεγχος αποτελεσμάτων

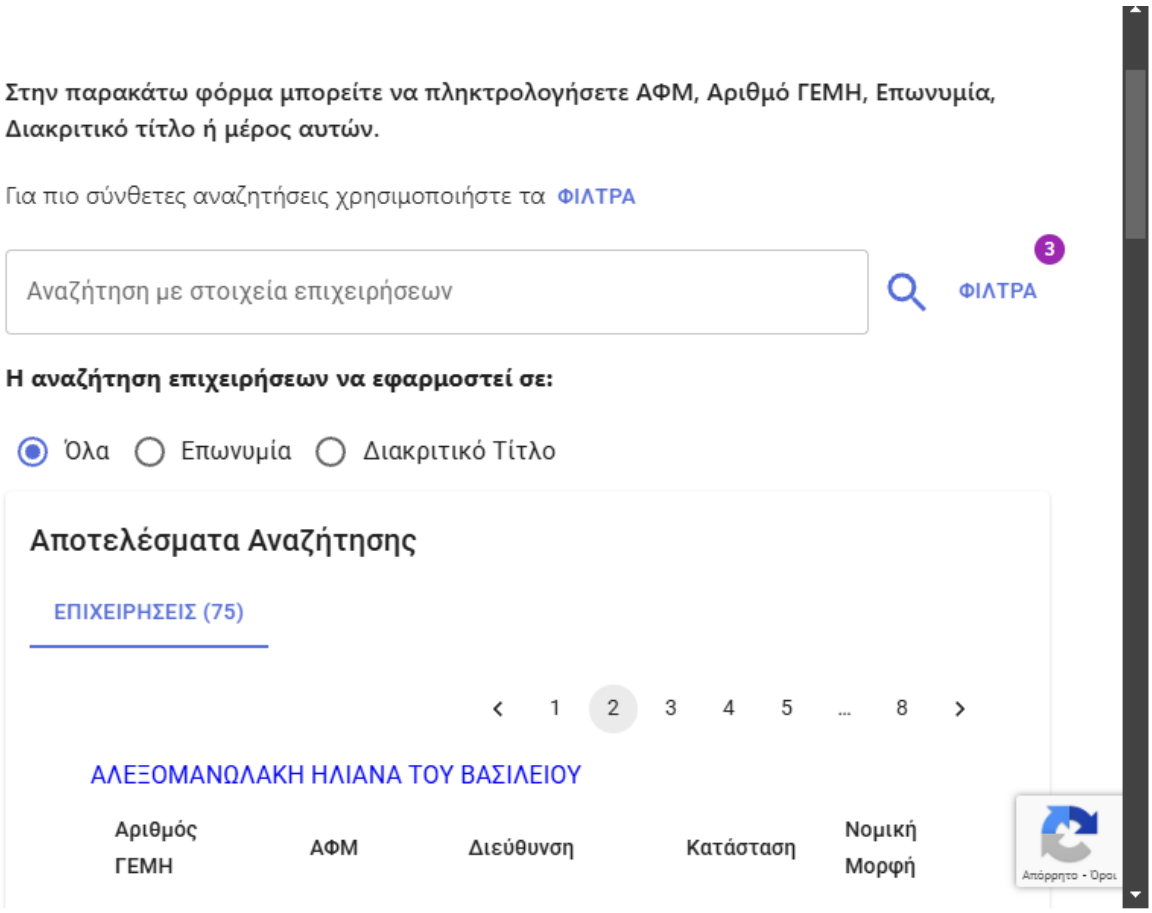
1. Στην πρώτη εικόνα εμφανίζεται η αρχική σελίαναζήτησης(https://publicity.businessportal.gr/), όπου εφαρμόζουμε τα επιθυμητά φίλτρα (π.χ. δραστηριότητα, γεωγραφική περιοχή, νομική μορφή κ.ά.), ώστε να εντοπίσουμε τις επιχειρήσεις που πληρούν τα συγκεκριμένα κριτήρια.
2. Στη δεύτερη εικόνα φαίνονται τα αποτελέσματα που προκύπτουν από την εφαρμογή των φίλτρων. Είναι σημαντικό να προσέξουμε τον αριθμό των επιχειρήσεων που επιστρέφονται, γιατί θα τον συγκρίνουμε με τον αριθμό αποτελεσμάτων που επιστρέφει ο scraper μας. Αυτό είναι ένα κρίσιμο σημείο επαλήθευσης της ορθής λειτουργίας του scraper.
3. Στην τρίτη εικόνα παρατηρούμε ότι ο scraper επιστρέφει ακριβώς τον ίδιο αριθμό επιχειρήσεων (π.χ. 75), κάτι που υποδηλώνει ότι η συλλογή δεδομένων έγινε με ακρίβεια. Επιλέγουμε τυχαία μία επιχείρηση (με συγκεκριμένο αριθμό ΓΕΜΗ) και προχωρούμε στο δεύτερο στάδιο του scraper.
4. Ο scraper ανακτά αναλυτικές πληροφορίες για τη συγκεκριμένη επιχείρηση: τίτλος, ΚΑΔ, δευτερεύοντες ΚΑΔ, διεύθυνση, νομική μορφή, ημερομηνία σύστασης κ.λπ. Ελέγχουμε στη βάση δεδομένων μας αν έχουν αποθηκευτεί όλες αυτές οι πληροφορίες – ακόμα και οι πιο δύσκολα εντοπίσιμες όπως οι δευτερεύοντες ΚΑΔ.
5. Τα δεδομένα αποθηκεύονται στη MongoDB χρησιμοποιώντας **embedded documents** (ενσωματωμένα έγγραφα), **χωρίς να απαιτείται η προσθήκη επιπλέον συλλογών ή σχέσεων μεταξύ των αντικειμένων**, όπως συμβαίνει στις σχεσιακές βάσεις (relational databases).
6. Η Mongo προτείνει σε γενικές γραμμές τη χρήση embedded documents όταν τα δεδομένα είναι άμεσα συνδεδεμένα και η αποθήκευσή τους μαζί προσφέρει ταχύτερη και ευκολότερη αναζήτηση μέσω queries.



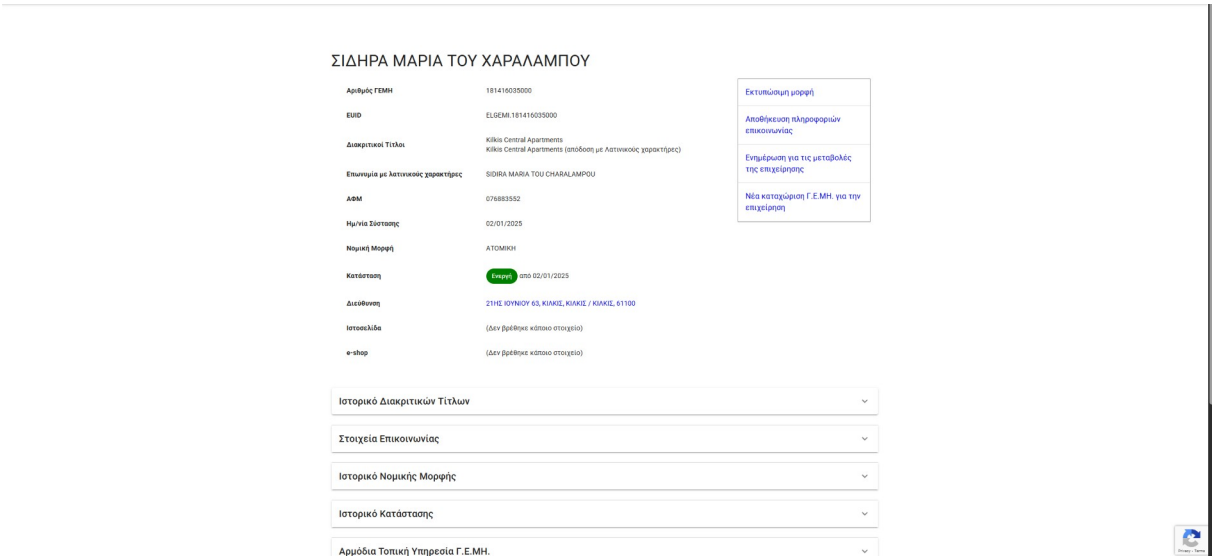
Σχήμα 5.1: Filters Page



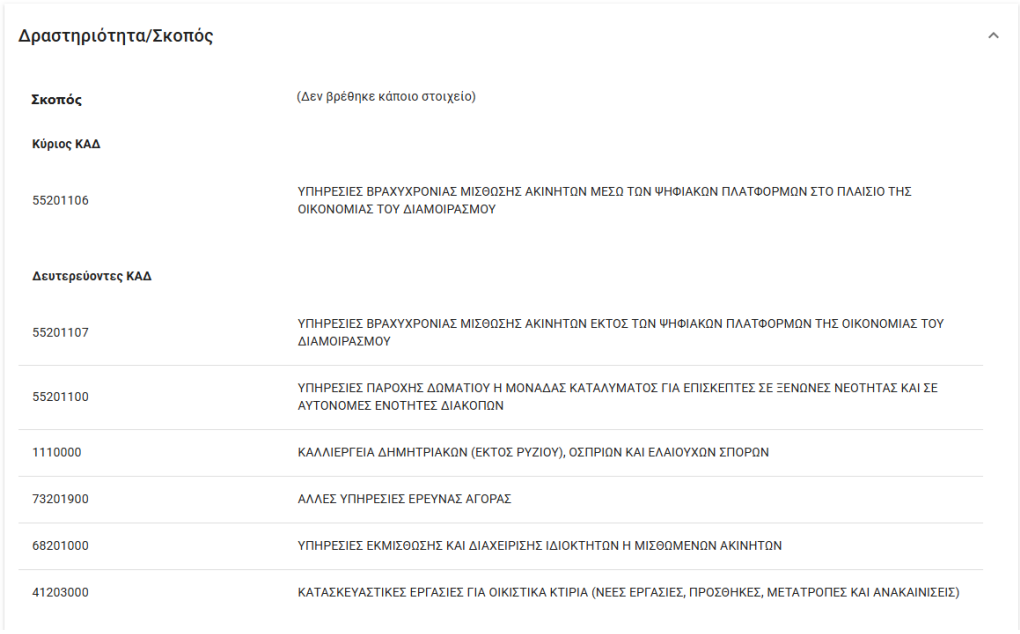
Σχήμα 5.2: Results Page



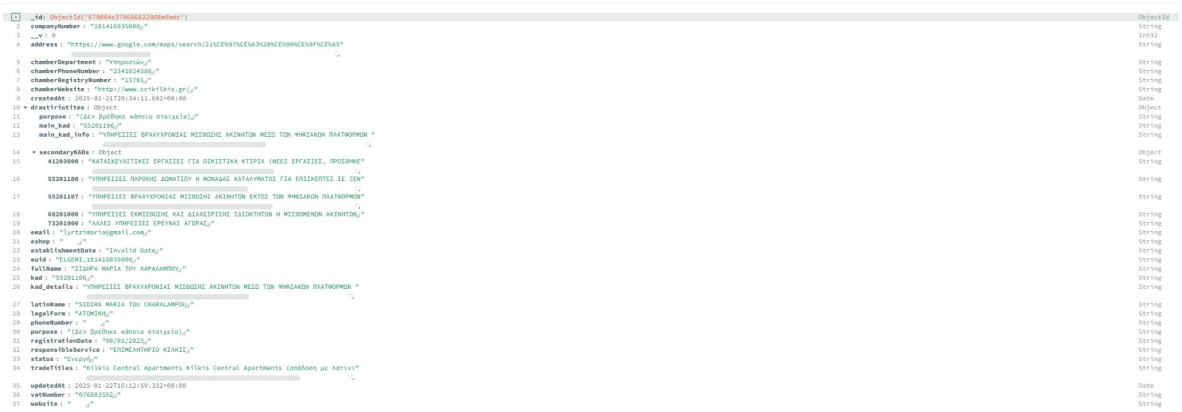
Σχήμα 5.3: Scraper Results Page



Σχήμα 5.4: Business Info



Σχήμα 5.5: Business Info



Σχήμα 5.6: Saved Data

5.2 Αποτελέσματα και συμπεράσματα

Κατά τη διάρκεια της ανάπτυξης και εκτέλεσης των scrapers σε διαφορετικούς ιστότοπους, προέκυψαν σημαντικά και αξιοσημείωτα αποτελέσματα.

Συγκεκριμένα, στο **businessportal.gr**, η διαδικασία συλλογής δεδομένων ήταν ιδιαίτερα επιτυχημένη. Ο scraper κατάφερε να εντοπίσει και να καταγράψει περίπου το **98% των διαθέσιμων εγγράφων**, γεγονός που υποδεικνύει υψηλή ακρίβεια και αποτελεσματικότητα στην εξαγωγή πληροφορίας. Ο ιστότοπος αποδείχθηκε πλούσιος σε πληροφορίες, με σωστή δομή και σχεδόν καθολική κάλυψη των δεδομένων.

Αντίθετα, στον ιστότοπο **ordino.gr**, παρόλο που ο scraper λειτούργησε σωστά και κατάφερε να εντοπίσει όλους τους **ηθοποιούς και χώρους παραστάσεων**, το περιεχόμενο του ίδιου του ιστότοπου ήταν **πολύ περιορισμένο**. Από τα περίπου 15 πεδία δεδομένων που είχαν καθοριστεί, συχνά ήταν συμπληρωμένα μόνο τα 2–3. Στις καλύτερες περιπτώσεις υπήρχαν 6–7 πεδία με περιεχόμενο, και σπανιότερα 8–9, χωρίς να υπάρχει πλήρης και σταθερή πληροφόρηση.

Ως εκ τούτου, για μελλοντικές προσπάθειες συλλογής δεδομένων θεατρικού ή καλλιτεχνικού περιεχομένου, θα ήταν προτιμότερο να εντοπιστεί κάποια **εναλλακτική ιστοσελίδα με πιο πλήρη και δομημένη πληροφορία** για τους ηθοποιούς και τις παραστάσεις.

5.3 Προτάσεις και βελτίωση

Κατά την ανάπτυξη της εφαρμογής, εντοπίστηκαν ορισμένα σημεία που θα μπορούσαν να βελτιωθούν τόσο σε επίπεδο εμπειρίας χρήστη όσο και σε επίπεδο αρχιτεκτονικής και απόδοσης:

Βελτίωση περιβάλλοντος χρήστη(UI)

Το γραφικό περιβάλλον χρήστη θα μπορούσε να ανασχεδιαστεί με σκοπό να είναι πιο φιλικό και λειτουργικό, ενόψει και της μελλοντικής ενσωμάτωσης ενός **backoffice** συστήματος διαχείρισης.

Εισαγωγή TypeScript

Καθώς η JavaScript δεν παρέχει ενσωματωμένο έλεγχο τύπων (type safety), η μετάβαση σε **TypeScript** θα προσφέρει αυξημένη ασφάλεια στον κώδικα, καλύτερη συντηρησιμότητα και καλύτερη εμπειρία για την ομάδα ανάπτυξης.

Αναδιοργάνωση αρχιτεκτονικής κώδικα

Η υπάρχουσα δομή του κώδικα μπορεί να οργανωθεί καλύτερα για να εξασφαλιστεί η **επεκτασιμότητα** της εφαρμογής, ώστε να μπορεί να προσαρμοστεί ευκολότερα από νέα μέλη της ομάδας και να υποστηρίξει μελλοντικές δυνατότητες.

Πιο δηλωτική προσέγγιση στην React

Η εφαρμογή μπορεί να γίνει πιο **δηλωτική** και πιο λειτουργική με τη χρήση περισσότερων **React Hooks**, ακολουθώντας καλύτερες πρακτικές του οικοσυστήματος.

Caching με Redis

Για βελτίωση των χρόνων απόκρισης, μπορούν να αποθηκεύονται προσωρινά (cache) συχνά χρησιμοποιούμενα ερωτήματα βάσης δεδομένων στη **μνήμη με Redis**, το οποίο έχει πολύ χαμηλή καθυστέρηση και υψηλή ταχύτητα ανάκτησης.

Σύστημα διαχείρισης περιβαλλόντων(Branching & Environments)

Η υιοθέτηση συστήματος με διακριτά περιβάλλοντα (π.χ. `development`, `staging`, `production`) και ξεκάθαρη δομή κλάδων (branching) στο Git, θα επιτρέψει σε πολλούς προγραμματιστές να συνεργάζονται ομαλά και να δοκιμάζουν αλλαγές πριν από την παραγωγή.

Επεξεργασία σφαλμάτων(ErrorHandling)

Στην πλευρά του backend, μπορεί να ενσωματωθεί **κεντρικό middleware χειρισμού σφαλμάτων**, με περισσότερα σημεία ελέγχου για την καλύτερη κατανόηση και καταγραφή των προβλημάτων.

Σύστημα καταγραφής (Logging System)

Η ανάπτυξη ενός πιο εκτενούς συστήματος καταγραφής (logs) θα βοηθήσει στην παρακολούθηση της απόδοσης, της λειτουργίας των scrapers και στην ευκολότερη εντοπισμό προβλημάτων.

Εμφάνιση στατιστικών και γραφημάτων στο UI

Στην πλευρά του χρήστη, μπορούν να προστεθούν **στατιστικά και γραφήματα** που να εμφανίζουν πιο λεπτομερή δεδομένα σχετικά με τις ενέργειες scraping, προσφέροντας καλύτερη εποπτεία της λειτουργίας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Internet Sites

[1] Docker Documentation. Available at: https://docs.docker.com/

[2] MongoDB Documentation. Available at: https://www.mongodb.com/docs/

[3] Node.js Documentation. Available at: https://nodejs.org/en/docs .

[4] React Documentation. Available at: https://react.dev/learn

[5] Puppeteer Documentation. Available at: https://pptr.dev/

[6] Cheerio Documentation. Available at: https://cheerio.js.org/

[7] Web Scraping. Available at: https://en.wikipedia.org/wiki/Web_scraping

Βιβλία

[8] https://www.besanttechnologies.com/what-is-expressjs

[9] https://www.netguru.com/glossary/node-js

[10] https://www.zyte.com/learn/what-is-web-scraping/

[11] https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs

[12] **https://www.effectussoftware.com/blog/what-is-react-js**

[13] **https://www.mongodb.com/resources/languages/mern-stack**

[14] **https://www.geeksforgeeks.org/mern/understand-mern-stack/**

[15] **https://www.ibm.com/think/topics/mongodb**

[16] **https://aws.amazon.com/documentdb/what-is-mongodb/**