Διεθνές Πανεπιστήμιο Ελλάδος
Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη Συστήματος Συλλογής, Οργάνωσης και Διάθεσης Τιμών Υγρών Καυσίμων στην Ελλάδα με API και Front-End Παρουσίασης

Φοιτητής:
Γεώργιος Κουτσιαμάνης
Αριθμός Μητρώου: 174858

Επιβλέπων:
Στέφανος Ουγιάρογλου

29 Μαΐου 2026

Title of Dissertation Development of a System for Collecting, Organizing, and Publishing Liquid Fuel Prices in Greece with an API and a Front-End Interface
Code of Dissertation 23101
Student's full name Georgios Koutsiamanis
Supervisor's full name Stefanos Ougiaroglou
Date of undertaking 16-11-2025
Date of completion 29-05-2026

We hereby affirm the authorship of this paper as well as the acknowledgement and credit of whichever assistance We received in its composition. We have, furthermore, noted the various sources from which We extracted data, ideas, visual or written material, in paraphrase or exact quotation. Moreover, we affirm the exclusive composition of this paper by myself only, for the purpose of it being a dissertation, in the Department of Information and Electronic Engineering of the I.H.U.

The approval of this dissertation by the Department of Information and Electronic Engineering of the International Hellenic University, does not necessarily entail the adoption of the author's views, on behalf of the Department.

Αφιέρωση

Στην οικογένεια μου.

Abstract

The Greek Ministry of Development publishes daily fuel price bulletins for all 51 prefectures of the country through its `fuelprices.gr` portal. Although the information is publicly available, it is distributed in PDF form which makes automated processing extremely difficult. In the context of the open data scoring scheme proposed by Tim Berners-Lee these data sit at the first level (1) and remain largely inaccessible to researchers, journalists and citizens who could otherwise benefit from them.

This thesis presents an end-to-end system that transforms this dataset into a genuinely open and reusable resource. The system consists of three parts. First, an automated Python pipeline scrapes the ministry's website, downloads each daily bulletin and extracts the prices into a relational database. The pipeline uses a two-tier extraction approach combining the pdfplumber library for the standardised modern bulletins (2023 onwards) and the Google Gemini API for the older non-standardised PDFs. Validation rules, automatic LLM fallback, weekly email summaries and a daily cron job make the pipeline self-sustaining. Second, an open REST API written in PHP exposes the data through documented JSON endpoints with stable URIs, satisfying the FAIR principles and bringing the dataset to the third level (3) of the Berners-Lee scheme with characteristics that approach the fourth. Third, a vanilla HTML/CSS/JavaScript web application built on Chart.js visualises the data through interactive charts that allow side-by-side comparisons across prefectures and fuel types. The usability of the final application is evaluated through the System Usability Scale questionnaire.

Περίληψη

Το Υπουργείο Ανάπτυξης δημοσιεύει καθημερινά τα δελτία τιμών καυσίμων για όλους τους 51 νομούς της Ελλάδας μέσω του ιστότοπου `fuelprices.gr`. Παρότι η πληροφορία είναι δημόσια διαθέσιμη, παρέχεται σε μορφή PDF που καθιστά την αυτόματη επεξεργασία της εξαιρετικά δύσκολη. Στο σχήμα ανοιχτών δεδομένων του Tim Berners-Lee τα δεδομένα αυτά βρίσκονται στο πρώτο επίπεδο (1) και παραμένουν πρακτικά μη αξιοποιήσιμα για ερευνητές, δημοσιογράφους και πολίτες που θα μπορούσαν να επωφεληθούν από αυτά.

Η παρούσα εργασία παρουσιάζει ένα ολοκληρωμένο σύστημα που μετατρέπει αυτό το dataset σε έναν πραγματικά ανοιχτό και επαναχρησιμοποιήσιμο πόρο. Το σύστημα αποτελείται από τρία μέρη. Πρώτο, ένα αυτοματοποιημένο pipeline σε Python που κατεβάζει τα ημερήσια δελτία από τον ιστότοπο του Υπουργείου και εξάγει τις τιμές σε μια σχεσιακή βάση δεδομένων. Το pipeline συνδυάζει δύο τρόπους εξαγωγής: τη βιβλιοθήκη pdfplumber για τα τυποποιημένα νέα δελτία (από το 2023 και έπειτα) και το Google Gemini API για τα παλαιότερα PDF χωρίς σταθερή μορφή. Έλεγχοι εγκυρότητας, αυτόματο fallback στο LLM, εβδομαδιαίες αναφορές μέσω email και ένα cron job κάνουν το pipeline αυτόνομο. Δεύτερο, ένα ανοιχτό REST API σε PHP εκθέτει τα δεδομένα μέσω τεκμηριωμένων JSON endpoints με σταθερά URIs, ικανοποιώντας τις αρχές FAIR και ανεβάζοντας το dataset στο τρίτο επίπεδο (3) του σχήματος Berners-Lee με χαρακτηριστικά που αγγίζουν το τέταρτο. Τρίτο, μια εφαρμογή ιστού σε vanilla HTML, CSS και JavaScript με Chart.js οπτικοποιεί τα δεδομένα μέσω διαδραστικών διαγραμμάτων που επιτρέπουν παράλληλη σύγκριση μεταξύ νομών και τύπων καυσίμων. Η ευχρηστία της τελικής εφαρμογής αξιολογείται μέσω του ερωτηματολογίου System Usability Scale.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Τιμές Υγρών Καυσίμων

Τα υγρά καύσιμα είναι ένα από τα βασικότερα προϊόντα της καθημερινότητας μας . Σε αυτά ανήκουν η αμόλυβδη βενζίνη, το πετρέλαιο κίνησης, το πετρέλαιο θέρμανσης και το υγραέριο κίνησης. Τα χρησιμοποιούμε στα αυτοκίνητά μας, στην μεταφορά εμπορευμάτων, στην θέρμανση των σπιτιών και στη λειτουργία πολλών επιχειρήσεων. Η τιμή τους επηρεάζει άμεσα τους καταναλωτές αλλά και την οικονομία γενικότερα. Όταν αυξάνονται απότομα γίνονται από τα βασικά θέματα συζήτησης.
Η λιανική τιμή των καυσίμων εξαρτάται από αρκετούς παράγοντες αλλά ίσως ο πιο σημαντικός είναι η διεθνής τιμή του αργού πετρελαίου. Παίζει ρόλο επίσης η ισοτιμία του ευρώ με το δολάριο. Πάνω σε αυτά προστίθενται ο ειδικός φόρος κατανάλωσης και ο ΦΠΑ. Τέλος στην τελική τιμή υπολογίζονται και τα κέρδη των διυλιστηρίων και των πρατηρίων. Η Ελλάδα δεν παράγει δικό της πετρέλαιο και το εισάγει σχεδόν εξ ολοκλήρου από άλλες χώρες [1, 2]. Έτσι ό,τι γίνεται διεθνώς επηρεάζει πολύ γρήγορα την τιμή που πληρώνει ο καταναλωτής εδώ.
Το κράτος έχει σημαντικό ρόλο στην παρακολούθηση αυτών των τιμών. Με αυτόν τον τρόπο μπορεί να διασφαλίσει κάποια διαφάνεια στην αγορά και να προστατέψει τον καταναλωτή. Για τον λόγο αυτό το Υπουργείο Ανάπτυξης συγκεντρώνει κάθε μέρα τις τιμές των καυσίμων από όλους τους νομούς της χώρας και τις δημοσιεύει σε ένα ημερήσιο δελτίο. Έτσι πολίτες, ερευνητές και δημόσιες υπηρεσίες μπορούν να βλέπουν πως κινείται η αγορά. Όπως όμως θα δούμε παρακάτω η μορφή με την οποία ανεβαίνουν τα δεδομένα δεν διευκολύνει καθόλου την περαιτέρω επεξεργασία τους.

1.2 Ανοιχτά Δεδομένα

Τα ανοιχτά δεδομένα είναι δεδομένα που μπορεί ο καθένας να χρησιμοποιήσει χωρίς περιορισμούς. Δεν αρκεί όμως απλά να είναι ανεβασμένα κάπου στο διαδίκτυο. Για να θεωρηθούν πραγματικά ανοιχτά πρέπει να ικανοποιούν συγκεκριμένες προϋποθέσεις. Ο πιο διαδεδομένος ορισμός είναι αυτός του Open Knowledge Foundation [3]. Σύμφωνα με αυτόν δεδομένα θεωρούνται ανοιχτά όταν παρέχονται με ανοιχτή άδεια χρήσης που επιτρέπει την επεξεργασία και την αναδιανομή τους. Πρέπει επίσης να είναι σε μηχαναγνώσιμη μορφή ώστε να μπορούν να χρησιμοποιηθούν εύκολα από διαφορετικά προγράμματα.

Για να μετρηθεί πόσο ανοιχτά είναι κάποια δεδομένα έχουν προταθεί διάφοροι δείκτες. Ο πιο γνωστός είναι το σχήμα πέντε αστέρων του Tim Berners-Lee [4]. Σε αυτό τα δεδομένα ταξινομούνται σε πέντε επίπεδα ανάλογα με τον τρόπο που δημοσιεύονται. Το πρώτο αστέρι δίνεται όταν τα δεδομένα είναι διαθέσιμα στο διαδίκτυο σε οποιαδήποτε μορφή ακόμη και ως PDF. Το δεύτερο όταν δίνονται σε μορφή που μπορεί να επεξεργαστεί ένας υπολογιστής όπως ένα αρχείο Excel. Το τρίτο όταν αντί για κλειστές μορφές χρησιμοποιείται κάποια ανοιχτή όπως το CSV. Το τέταρτο όταν χρησιμοποιούνται πρότυπα του παγκόσμιου ιστού ώστε να μπορούν τα δεδομένα να αναφέρονται μέσα από URLs. Και το πέμπτο όταν τα δεδομένα είναι ήδη συνδεδεμένα με άλλα ανοιχτά δεδομένα δημιουργώντας ένα ευρύτερο δίκτυο πληροφορίας.
Ένας άλλος δείκτης που χρησιμοποιείται ευρέως είναι οι αρχές FAIR. Το ακρωνύμιο σημαίνει Findable, Accessible, Interoperable και Reusable [5]. Δηλαδή τα δεδομένα πρέπει να μπορούν εύκολα να βρεθούν, να είναι προσβάσιμα, να μπορούν να συνδυαστούν με άλλα δεδομένα και τέλος να μπορούν να επαναχρησιμοποιηθούν. Οι αρχές αυτές προτάθηκαν αρχικά για επιστημονικά δεδομένα αλλά πλέον χρησιμοποιούνται και γενικότερα.
Σε επίπεδο χωρών η Ευρωπαϊκή Ένωση εκδίδει κάθε χρόνο την έκθεση Open Data Maturity [6]. Σε αυτή αξιολογείται η πρόοδος κάθε κράτους μέλους στα ανοιχτά δεδομένα. Στη δική μας περίπτωση όμως μας ενδιαφέρει ο χαρακτηρισμός ενός συγκεκριμένου dataset για το οποίο τα κατάλληλα εργαλεία είναι το σχήμα πέντε αστέρων και οι αρχές FAIR. Αυτούς τους δύο δείκτες θα εφαρμόσουμε στην επόμενη ενότητα για να αξιολογήσουμε τη μορφή με την οποία δημοσιεύονται σήμερα οι τιμές των καυσίμων.

1.3 Διανομή Δεδομένων Τιμών Υγρών Καυσίμων

Όπως αναφέραμε στην εισαγωγή το Υπουργείο Ανάπτυξης δημοσιεύει κάθε μέρα τα δελτία τιμών μέσω του ιστότοπου `fuelprices.gr`. Συγκεκριμένα τα δελτία διατίθενται από τη σελίδα `/deltia_dn.view` ως σύνδεσμοι σε αρχεία PDF. Κάθε σύνδεσμος αντιστοιχεί σε μία ημερομηνία και ο επισκέπτης μπορεί να κατεβάσει τα δελτία που τον ενδιαφέρουν. Μέσα σε κάθε PDF περιέχεται ένας πίνακας με τις τιμές των βασικών καυσίμων ανά νομό για τη συγκεκριμένη ημέρα.
Με βάση τους δείκτες που παρουσιάστηκαν στην προηγούμενη ενότητα μπορούμε να αξιολογήσουμε πόσο ανοιχτά είναι αυτά τα δεδομένα. Ξεκινώντας από το σχήμα πέντε αστέρων τα δεδομένα του `fuelprices.gr` βρίσκονται στο πρώτο επίπεδο. Είναι δηλαδή 1★. Πληρούν την πρώτη προϋπόθεση γιατί ανεβαίνουν δωρεάν στο διαδίκτυο και είναι δημόσια διαθέσιμα. Δεν φτάνουν όμως στο δεύτερο αστέρι. Η μορφή PDF δεν θεωρείται μηχαναγνώσιμη με την έννοια που απαιτεί το σχήμα. Τα δεδομένα μέσα σε ένα PDF είναι δύσκολο να εξαχθούν αυτόματα και να επεξεργαστούν από κάποιο πρόγραμμα. Επομένως όλα τα παραπάνω αστέρια από 2 έως 5 χάνονται.
Παρόμοια εικόνα προκύπτει και με τις αρχές FAIR. Από τις τέσσερις αρχές μόνο η μία ικανοποιείται πλήρως. Συγκεκριμένα η αρχή Accessible ισχύει αφού οποιοσδήποτε μπορεί να κατεβάσει τα δελτία ελεύθερα από το διαδίκτυο. Η αρχή Findable ισχύει εν μέρει. Το αρχικό σημείο πρόσβασης είναι γνωστό και ευρετηριάζεται από μηχανές αναζήτησης. Δεν υπάρχει όμως καμία τυποποιημένη μεταπληροφορία που να περιγράφει τα δεδομένα ούτε κάποιο μόνιμο αναγνωριστικό. Η αρχή Interoperable αποτυγχάνει γιατί τα PDF δεν χρησιμοποιούν κάποιο κοινό σχήμα ή λεξιλόγιο. Τέλος η αρχή Reusable δεν πληρείται γιατί για να ξαναχρησιμοποιήσει κα-

νείς τα δεδομένα πρέπει πρώτα να τα εξάγει χειροκίνητα και χωρίς ξεκάθαρους όρους άδειας χρήσης.
Συνολικά τα δεδομένα του `fuelprices.gr` είναι μεν δημόσια αλλά δύσκολα στη χρήση. Η πληροφορία υπάρχει και είναι ενημερωμένη. Όμως για να αξιοποιηθεί πρέπει κάποιος να ασχοληθεί με την εξαγωγή των τιμών από κάθε PDF ξεχωριστά. Αυτή ακριβώς η δυσκολία ήταν και το αρχικό κίνητρο για την παρούσα εργασία.

1.4 Κίνητρο

Όπως είδαμε στην προηγούμενη ενότητα τα δεδομένα του `fuelprices.gr` βρίσκονται στο πρώτο επίπεδο του σχήματος πέντε αστέρων. Η πληροφορία υπάρχει και ανανεώνεται καθημερινά. Το πρόβλημα είναι η μορφή της. Κάποιος που θέλει να δει πώς κινήθηκαν οι τιμές τους τελευταίους μήνες πρέπει να κατεβάσει δεκάδες PDF ένα ένα και να τα διαβάσει χειροκίνητα. Αυτό δεν είναι πρακτικό.
Η ανάγκη για εύκολη πρόσβαση σε ιστορικά δεδομένα τιμών υπάρχει από πολλές πλευρές. Ένας δημοσιογράφος που θέλει να γράψει για τις αυξήσεις στα καύσιμα χρειάζεται να συγκρίνει τιμές διαφορετικών περιόδων. Ένας ερευνητής που μελετά την αγορά ενέργειας χρειάζεται δεδομένα σε μορφή που μπορεί να επεξεργαστεί. Ακόμα και ένας απλός καταναλωτής θα ήθελε να βλέπει αν στον νομό του οι τιμές είναι υψηλότερες από τον εθνικό μέσο όρο. Κανένα από αυτά δεν είναι εύκολο με την τρέχουσα μορφή δημοσίευσης.
Το κίνητρο για αυτή την εργασία ήταν να λυθεί αυτό το πρόβλημα. Ένα dataset που βρίσκεται στο 1★ μπορεί να γίνει πολύ πιο χρήσιμο αν τα δεδομένα του εξαχθούν αυτόματα και εκτεθούν μέσω ενός σύγχρονου API. Με αυτόν τον τρόπο το ίδιο dataset μπορεί να ανεβεί σταθερά στο 3★ του σχήματος Berners-Lee με χαρακτηριστικά που αγγίζουν το 4★, και παράλληλα να ικανοποιεί σε μεγαλύτερο βαθμό τις αρχές FAIR.

1.5 Συνεισφορά

Για να αντιμετωπιστεί το πρόβλημα που περιγράφηκε παραπάνω αναπτύχθηκε ένα ολοκληρωμένο σύστημα από τρία μέρη που λειτουργούν μαζί.

1. **Ανάπτυξη bot** για την αυτόματη ανάκτηση των ημερήσιων δελτίων από τον ιστότοπο του Υπουργείου Ανάπτυξης. Το bot κατεβάζει κάθε PDF εξάγει τις τιμές και τις αποθηκεύει σε σχεσιακή βάση δεδομένων. Τρέχει αυτόματα κάθε μέρα ώστε η βάση να παραμένει ενημερωμένη.

2. **Δημιουργία ανοιχτού API** που εκθέτει τα δεδομένα της βάσης σε μορφή JSON μέσω HTTP. Οποιοσδήποτε μπορεί να το χρησιμοποιήσει για να πάρει τιμές ανά νομό, καύσιμο και χρονική περίοδο χωρίς να χρειαστεί να ασχοληθεί με PDF.

3. **Δημιουργία εφαρμογής ιστού** που χρησιμοποιεί το API και παρουσιάζει τα δεδομένα οπτικά στον τελικό χρήστη μέσω διαδραστικών διαγραμμάτων και συγκριτικών πινάκων.

1.6 Οργάνωση της Εργασίας

Η υπόλοιπη εργασία είναι οργανωμένη ως εξής.

- Στο **Κεφάλαιο 2** παρουσιάζονται οι γλώσσες προγραμματισμού και τα εργαλεία που χρησιμοποιήθηκαν.
- Στο **Κεφάλαιο 3** περιγράφεται η διαδικασία ανάκτησης δεδομένων και η σχεδίαση της βάσης.
- Στο **Κεφάλαιο 4** αναλύεται το ανοιχτό WEB API με τα διαθέσιμα endpoints.
- Στο **Κεφάλαιο 5** παρουσιάζεται η εφαρμογή ιστού.
- Στο **Κεφάλαιο 6** εκτιμάται η εμπειρία χρήσης μέσω του ερωτηματολογίου SUS.
- Στο **Κεφάλαιο 7** συνοψίζονται τα συμπεράσματα και προτείνονται μελλοντικές επεκτάσεις.

Κεφάλαιο 2

Γλώσσες και Τεχνολογίες

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζονται οι γλώσσες προγραμματισμού και τα εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση της εργασίας. Επιλέχθηκαν τεχνολογίες ώριμες και ευρέως διαδεδομένες ώστε ο κώδικας να είναι εύκολο να συντηρηθεί και να επεκταθεί στο μέλλον. Για κάθε τεχνολογία δίνεται μια σύντομη γενική περιγραφή και εξηγείται για ποιο λόγο επιλέχθηκε.

2.2 Python

Η Python είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου που σχεδιάστηκε από τον Guido van Rossum στα τέλη της δεκαετίας του 1980 και κυκλοφόρησε δημόσια το 1991 [7]. Είναι interpreted γλώσσα με δυναμικούς τύπους και δίνει έμφαση στην αναγνωσιμότητα του κώδικα. Η έκδοση που χρησιμοποιήθηκε στην εργασία είναι η Python 3.
Η Python έχει ένα από τα μεγαλύτερα οικοσυστήματα βιβλιοθηκών μέσω του Python Package Index (PyPI). Σε αυτό υπάρχουν χιλιάδες έτοιμα πακέτα για επεξεργασία δεδομένων, αιτήματα HTTP, ανάλυση HTML και άλλες εργασίες που συναντώνται συχνά σε ένα data pipeline. Αυτό κάνει την Python δημοφιλή επιλογή για scripts ανάκτησης και επεξεργασίας δεδομένων.
Στην παρούσα εργασία η Python χρησιμοποιήθηκε για την ανάπτυξη του bot που κατεβάζει και επεξεργάζεται τα PDF δελτία. Η επιλογή της βασίστηκε στη διαθεσιμότητα κατάλληλων βιβλιοθηκών για όλα τα βήματα της διαδικασίας. Συγκεκριμένα χρησιμοποιήθηκαν τα πακέτα `requests` για τα HTTP αιτήματα, `beautifulsoup4` για την ανάλυση του HTML, `pdfplumber` για την εξαγωγή πινάκων από τα PDF και `pymysql` για τη σύνδεση με τη βάση.

2.3 PHP

Η PHP είναι μια γλώσσα σεναρίων που τρέχει στον web server και χρησιμοποιείται κυρίως για την ανάπτυξη δυναμικών ιστοσελίδων και υπηρεσιών ιστού [8]. Δημιουργήθηκε από τον Rasmus Lerdorf το 1994 και έκτοτε έχει εξελιχθεί σε μία από τις πιο διαδεδομένες server side γλώσσες παγκοσμίως. Η έκδοση που χρησιμοποιήθηκε εδώ είναι η PHP 8.

Ένα από τα βασικά πλεονεκτήματά της είναι ότι έρχεται εγκατεστημένη σχεδόν σε κάθε shared hosting περιβάλλον. Συνεργάζεται άμεσα με τον Apache HTTP Server και προσφέρει ενσωματωμένη υποστήριξη για αιτήματα HTTP χωρίς να χρειάζεται κάποιο επιπλέον framework. Επίσης η σύνταξή της είναι αρκετά κοντά σε αυτή της C και της Java που κάνει την εκμάθηση εύκολη για κάποιον με προγραμματιστικό υπόβαθρο.
Στην εργασία η PHP χρησιμοποιήθηκε για την υλοποίηση του ανοιχτού API. Επιλέχθηκε γιατί ο server του τμήματος έχει ήδη εγκατεστημένο τον Apache με PHP support και επομένως δεν χρειαζόταν επιπλέον setup. Για την επικοινωνία με τη βάση χρησιμοποιήθηκε το PHP Data Objects (PDO) που είναι ο επίσημος τρόπος για ασφαλή ερωτήματα με prepared statements.

2.4 MariaDB

Η MariaDB είναι ένα σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων που προέκυψε ως fork της MySQL το 2009 από τον αρχικό δημιουργό της MySQL Michael Widenius [9]. Στόχος του fork ήταν να παραμείνει το λογισμικό πραγματικά ανοιχτό μετά την εξαγορά της MySQL από την Oracle. Η MariaDB διατηρεί συμβατότητα σε επίπεδο πρωτοκόλλου και SQL σύνταξης με τη MySQL.
Όπως κάθε σχεσιακή βάση η MariaDB αποθηκεύει τα δεδομένα σε πίνακες με προκαθορισμένο σχήμα και υποστηρίζει σχέσεις μεταξύ τους μέσω foreign keys. Υποστηρίζει επίσης transactions, indexes για γρήγορη αναζήτηση και views για απλοποίηση σύνθετων ερωτημάτων. Η αλληλεπίδραση γίνεται μέσω της γλώσσας SQL.
Για την εργασία η MariaDB ήταν φυσική επιλογή γιατί είναι ήδη εγκατεστημένη και διαθέσιμη στον server του τμήματος. Η φύση των δεδομένων μας ταιριάζει στο σχεσιακό μοντέλο αφού έχουμε σταθερές οντότητες όπως νομοί καύσιμα και ημερήσιες τιμές με ξεκάθαρες σχέσεις μεταξύ τους. Το σχήμα της βάσης περιγράφεται αναλυτικά στο Κεφάλαιο 3.

2.5 HTML, CSS και JavaScript

Η HTML είναι η γλώσσα σήμανσης που χρησιμοποιείται για την περιγραφή της δομής μιας ιστοσελίδας. Η CSS είναι η γλώσσα που ορίζει την εμφάνιση και την παρουσίαση των στοιχείων της σελίδας. Η JavaScript είναι η γλώσσα προγραμματισμού που τρέχει μέσα στον φυλλομετρητή και προσθέτει διαδραστική συμπεριφορά στις σελίδες. Και οι τρεις τυποποιούνται από το W3C και το WHATWG και υποστηρίζονται από όλους τους σύγχρονους φυλλομετρητές [10].
Μαζί αποτελούν το λεγόμενο vanilla web stack. Με αυτό μπορεί κανείς να φτιάξει μια ολόκληρη εφαρμογή ιστού χωρίς να χρησιμοποιήσει κάποιο framework ή κάποιο εργαλείο μεταγλώττισης. Τα αρχεία στέλνονται απευθείας από τον server στον browser και εκτελούνται χωρίς ενδιάμεσα βήματα.
Στην εργασία επιλέχθηκε αυτή η προσέγγιση για το frontend γιατί η εφαρμογή δεν είναι ιδιαίτερα σύνθετη και δεν δικαιολογείται η χρήση ενός framework όπως το React ή το Vue. Η αποφυγή ενός build step σημαίνει επίσης ότι το deployment είναι πιο απλό και ότι ο κώδικας μπορεί να διαβαστεί και να τροποποιηθεί από οποιονδήποτε ξέρει βασικά HTML CSS και JavaScript.

2.6 BeautifulSoup4

Το BeautifulSoup είναι μια βιβλιοθήκη της Python για την ανάλυση HTML και XML εγγράφων. Δημιουργήθηκε από τον Leonard Richardson και η τέταρτη έκδοσή του είναι η πιο διαδεδομένη [11]. Δέχεται ως είσοδο το raw HTML μιας σελίδας και επιστρέφει ένα δομημένο αντικείμενο που μπορεί να διασχιστεί όπως ένα δέντρο.
Με το BeautifulSoup μπορεί κανείς να βρει στοιχεία της σελίδας με βάση το tag το class ή κάποιο attribute. Αυτό βοηθάει όταν θέλουμε να εξάγουμε συγκεκριμένα δεδομένα από μια ιστοσελίδα χωρίς να γράψουμε δικό μας parser. Η βιβλιοθήκη είναι ανεκτική σε κακογραμμένο HTML και αυτό την κάνει ιδανική για scraping.
Στο bot της εργασίας χρησιμοποιείται για την ανάλυση της σελίδας `deltia_dn.view` του ιστότοπου του Υπουργείου Ανάπτυξης. Από εκεί εξάγονται οι σύνδεσμοι προς τα PDF αρχεία μαζί με την ημερομηνία κάθε δελτίου ώστε στη συνέχεια να γίνει η λήψη τους.

2.7 pdfplumber

Το pdfplumber είναι μια βιβλιοθήκη της Python που χτίζεται πάνω από το pdfminer.six και προσφέρει υψηλότερου επιπέδου API για την εξαγωγή κειμένου και πινάκων από αρχεία PDF [12]. Δίνει πρόσβαση στις γραμμές και στα ορθογώνια που υπάρχουν στις σελίδες ενός PDF και προσπαθεί να αναγνωρίσει αυτόματα τη δομή των πινάκων.
Σε αντίθεση με άλλες βιβλιοθήκες που επιστρέφουν μόνο σκέτο κείμενο το pdfplumber διατηρεί την πληροφορία της θέσης και της δομής. Αυτό είναι χρήσιμο όταν τα δεδομένα είναι οργανωμένα σε πίνακες όπως συμβαίνει στα ημερήσια δελτία του Υπουργείου. Το API είναι αρκετά απλό και επιτρέπει την εξαγωγή ενός πίνακα με μία μόνο κλήση μεθόδου.
Στην εργασία το pdfplumber χρησιμοποιείται ως ο κύριος parser για τα δελτία. Για κάθε PDF που κατεβάζει το bot το pdfplumber εξάγει τον πίνακα τιμών και τα δεδομένα μετά αποθηκεύονται στη βάση. Όπως θα δούμε στο Κεφάλαιο 3 ορισμένα παλιά PDF δεν έχουν αναγνώσιμη δομή πίνακα και για αυτά χρησιμοποιείται διαφορετική προσέγγιση.

2.8 Google Gemini API

Το Gemini είναι μια οικογένεια μεγάλων γλωσσικών μοντέλων που ανέπτυξε η Google DeepMind και ανακοινώθηκε τον Δεκέμβριο του 2023 [13]. Πρόκειται για multimodal μοντέλα που μπορούν να επεξεργαστούν κείμενο εικόνα ήχο και βίντεο. Η Google προσφέρει πρόσβαση στα μοντέλα μέσω ενός δημόσιου REST API.
Στο πλαίσιο data extraction τα LLM μπορούν να χρησιμοποιηθούν για να διαβάσουν ένα ημιδομημένο έγγραφο και να επιστρέψουν τα δεδομένα σε δομημένη μορφή όπως JSON. Αυτή η προσέγγιση είναι ιδιαίτερα χρήσιμη όταν η μορφή του εγγράφου διαφέρει από αρχείο σε αρχείο και ένας παραδοσιακός parser θα απαιτούσε πολλούς ειδικούς κανόνες.
Στην παρούσα εργασία το Gemini API χρησιμοποιείται ως fallback για την εξαγωγή δεδομένων από παλαιότερα PDF δελτία. Αυτά τα παλιά δελτία είναι κυρίως σκαναρισμένες σελίδες χωρίς αναγνώσιμη δομή και το pdfplumber αποτυγχάνει να εξάγει σωστά τους πίνακες. Σε αυτές τις περιπτώσεις η σελίδα στέλνεται στο Gemini μαζί με ένα prompt που του ζητά να επιστρέψει τις τιμές σε JSON μορφή.

2.9 Chart.js

Το Chart.js είναι μια βιβλιοθήκη JavaScript ανοιχτού κώδικα για τη δημιουργία διαδραστικών διαγραμμάτων μέσα σε ιστοσελίδες [14]. Σχεδιάστηκε αρχικά το 2013 και αυτή τη στιγμή βρίσκεται στην τέταρτη μεγάλη έκδοση. Σχεδιάζει τα διαγράμματα πάνω σε ένα HTML5 canvas και υποστηρίζει μεγάλη ποικιλία τύπων όπως γραμμικά διαγράμματα ραβδογράμματα και διαγράμματα τύρτας.
Σε σύγκριση με άλλες βιβλιοθήκες όπως το D3.js το Chart.js είναι πιο απλό στη χρήση. Δεν απαιτεί από τον χρήστη να ασχοληθεί με χαμηλού επιπέδου SVG ή με transformations. Αρκεί να δοθεί ένα αντικείμενο διαμόρφωσης με τα δεδομένα και τις επιλογές και η βιβλιοθήκη αναλαμβάνει την υπόλοιπη δουλειά.
Στην εφαρμογή ιστού της εργασίας το Chart.js χρησιμοποιείται για την οπτικοποίηση των ιστορικών τιμών των καυσίμων. Ο χρήστης επιλέγει νομούς και καύσιμα από custom dropdowns και η σελίδα εμφανίζει σε ένα γραμμικό διάγραμμα την εξέλιξη των τιμών στον χρόνο. Χρησιμοποιείται επίσης ο time scale του Chart.js ώστε ο άξονας x να ερμηνεύεται σωστά ως ημερομηνία.

2.10 Apache HTTP Server

Ο Apache HTTP Server είναι ένας από τους πιο διαδεδομένους web servers παγκοσμίως και αναπτύσσεται από το Apache Software Foundation [15]. Η πρώτη του έκδοση κυκλοφόρησε το 1995 και εδώ και πολλά χρόνια είναι από τις πιο δημοφιλείς επιλογές σε Linux servers. Είναι ελεύθερο λογισμικό και διαθέτει αρθρωτή αρχιτεκτονική με πολλά modules για επέκταση της λειτουργικότητάς του.
Δύο βασικά χαρακτηριστικά του Apache που χρησιμοποιούνται στην εργασία είναι το `mod_rewrite` και τα αρχεία `.htaccess`. Το πρώτο επιτρέπει την επανεγγραφή URLs με βάση κανόνες ώστε καθαρές διευθύνσεις της μορφής `/api/v1/prices` να αντιστοιχίζονται εσωτερικά στο κατάλληλο PHP script. Τα αρχεία `.htaccess` επιτρέπουν τη ρύθμιση του server σε επίπεδο φακέλου χωρίς να χρειάζεται πρόσβαση στο κεντρικό αρχείο διαμόρφωσης. Στην εργασία ο Apache είναι ο web server που εξυπηρετεί τόσο το API όσο και το frontend. Επιλέχθηκε γιατί είναι ήδη εγκατεστημένος στον server του τμήματος και γιατί συνεργάζεται καλά με την PHP. Επίσης τα `.htaccess` αρχεία κάνουν εύκολη τη ρύθμιση του pretty URL routing για το API χωρίς αλλαγές σε επίπεδο administrator.

2.11 cron

Το cron είναι το χρονικά προγραμματιζόμενο σύστημα εκτέλεσης εντολών των Unix συστημάτων [16]. Υπάρχει από τα τέλη της δεκαετίας του 1970 και αποτελεί έναν από τους πιο σταθερούς και απλούς τρόπους για να τρέξει κανείς ένα script σε τακτά χρονικά διαστήματα. Οι ρυθμίσεις γίνονται μέσω του αρχείου crontab όπου κάθε γραμμή περιγράφει μια εργασία και πότε αυτή πρέπει να εκτελείται.
Η σύνταξη του crontab είναι αρκετά απλή. Πέντε πεδία ορίζουν λεπτό ώρα ημέρα του μήνα μήνα και ημέρα της εβδομάδας και ακολουθεί η εντολή που θα εκτελεστεί. Το cron τρέχει στο παρασκήνιο ως daemon και ελέγχει διαρκώς ποιες εργασίες πρέπει να ξεκινήσουν.

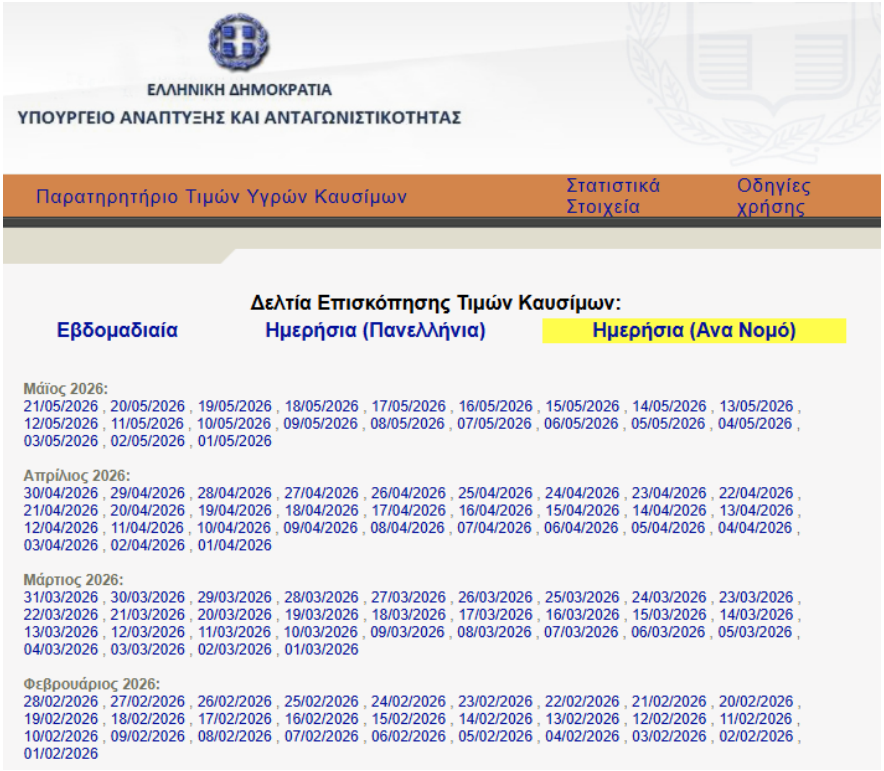
Στην εργασία το cron χρησιμοποιείται για την αυτόματη εκτέλεση του bot. Έχει οριστεί μια εργασία που τρέχει το script κάθε μέρα σε συγκεκριμένη ώρα ώστε να κατεβάζει το νέο δελτίο τιμών αμέσως μόλις γίνει διαθέσιμο και να ενημερώνει τη βάση. Με αυτόν τον τρόπο το σύστημα παραμένει ενημερωμένο χωρίς ανθρώπινη παρέμβαση.

Κεφάλαιο 3

Ανάκτηση Δεδομένων Τιμών και Δημιουργία ΒΔ

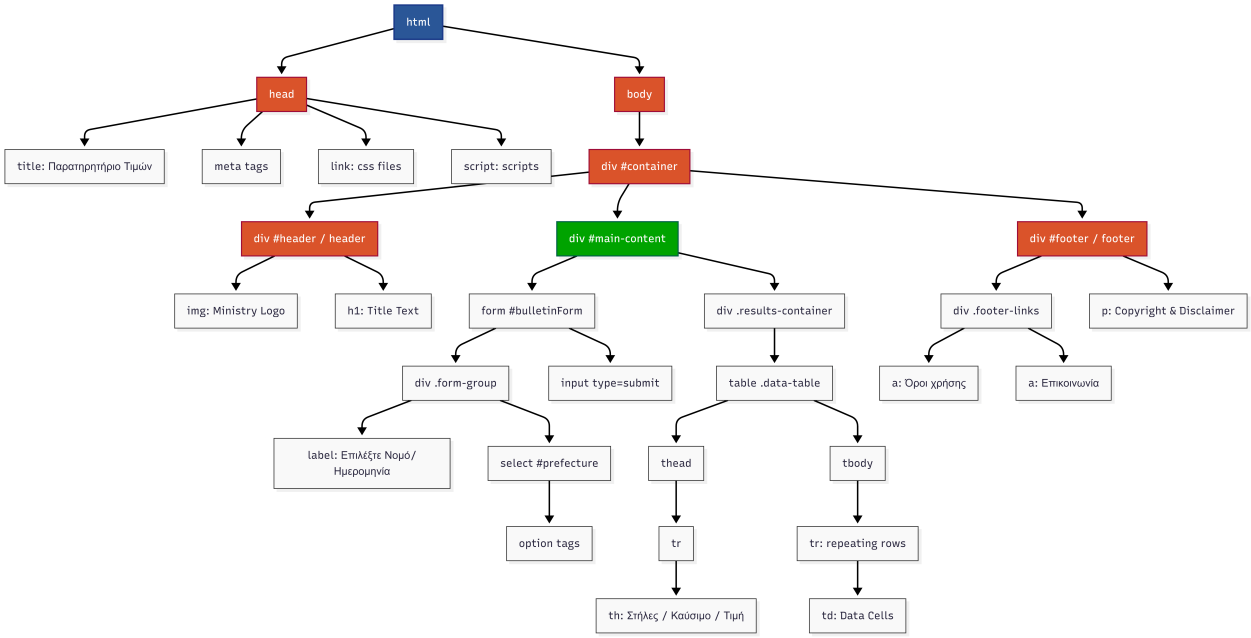
3.1 Ο Ιστότοπος του Υπουργείου

Το πρώτο βήμα για την ανάκτηση των δεδομένων είναι να καταλάβουμε πώς ακριβώς δημοσιεύονται τα ημερήσια δελτία τιμών. Ο ιστότοπος του Υπουργείου Ανάπτυξης που τα φιλοξενεί είναι ο `fuelprices.gr` και η σελίδα από την οποία διατίθενται είναι η `https://www.fuelprices.gr/deltia_dn.view`. Πρόκειται για μια στατική σελίδα HTML που ανανεώνεται καθημερινά μόλις δημοσιευτεί το νέο δελτίο.



Σχήμα 3.1: Η σελίδα `deltia_dn.view` του ιστότοπου `fuelprices.gr`.

Η δομή της σελίδας είναι αρκετά απλή. Στο κύριο περιεχόμενο υπάρχει μια λίστα από συνδέσμους και κάθε σύνδεσμος αντιστοιχεί σε μία ημερομηνία. Το κείμενο κάθε συνδέσμου είναι η ίδια η ημερομηνία σε μορφή `HH/MM/EEEE` και το `href` attribute δείχνει σε ένα αρχείο PDF που βρίσκεται σε υποφάκελο του ίδιου server. Οι σύνδεσμοι είναι ταξινομημένοι σε αντίστροφη χρονολογική σειρά ώστε το πιο πρόσφατο δελτίο να βρίσκεται πάντα στην κορυφή.



Σχήμα 3.2: DOM tree της σελίδας `deltia_dn.view`.

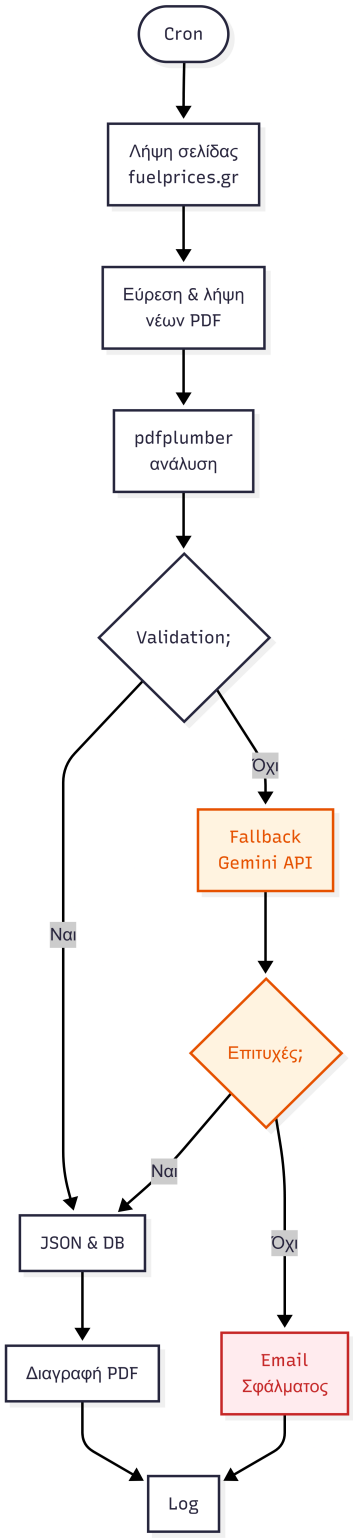
Δεν υπάρχει κάποιο επίσημο API ή κάποια άλλη μηχαναγνώσιμη μορφή που να εκθέτει τα δελτία. Επίσης η σελίδα δεν έχει σταθερά IDs ή classes στα στοιχεία της που θα μπορούσαν να χρησιμοποιηθούν ως σταθερά σημεία αναφοράς. Αυτό που παραμένει σταθερό είναι μόνο η μορφή του κειμένου των συνδέσμων η οποία είναι πάντα μια ημερομηνία. Σε αυτό το χαρακτηριστικό βασίζεται η διαδικασία εξαγωγής όπως θα δούμε στις επόμενες ενότητες όπου εντοπίζονται οι σύνδεσμοι με ένα regular expression που ταιριάζει σε ημερομηνίες.
Δεν υπάρχει κάποιο επίσημο API ή κάποια άλλη μηχαναγνώσιμη μορφή που να εκθέτει τα δελτία. Επίσης η σελίδα δεν έχει σταθερά IDs ή classes στα στοιχεία της που θα μπορούσαν να χρησιμοποιηθούν ως σταθερά σημεία αναφοράς. Αυτό που παραμένει σταθερό είναι μόνο η μορφή του κειμένου των συνδέσμων η οποία είναι πάντα μια ημερομηνία. Σε αυτό το χαρακτηριστικό βασίζεται η διαδικασία εξαγωγής όπως θα δούμε στις επόμενες ενότητες όπου εντοπίζονται οι σύνδεσμοι με ένα regular expression που ταιριάζει σε ημερομηνίες.

3.2 Pipeline Ανάκτησης και Αποθήκευσης Δεδομένων

Για την αυτόματη ανάκτηση και αποθήκευση των τιμών αναπτύχθηκε ένα pipeline σε Python. Το pipeline συνδυάζει όλα τα βήματα που χρειάζονται από τη στιγμή που θα κατέβει ένα PDF μέχρι να αποθηκευτούν τα δεδομένα του στη βάση. Σχεδιάστηκε με τέτοιο τρόπο ώστε να μπορεί να τρέξει χωρίς ανθρώπινη παρέμβαση και να ανακάμπτει από προσωρινά προβλήματα.

Η συνολική ροή έχει τα εξής στάδια. Πρώτα γίνεται λήψη της σελίδας `deltia_dn.view` και εντοπίζονται οι σύνδεσμοι προς τα PDF δελτία. Μετά κατεβαίνουν όσα δελτία δεν υπάρχουν ήδη τοπικά. Για κάθε νέο PDF γίνεται προσπάθεια εξαγωγής των τιμών με τη βιβλιοθήκη pdfplumber. Το αποτέλεσμα περνάει από ένα στάδιο validation που ελέγχει αν τα δεδομένα είναι λογικά. Αν το pdfplumber αποτύχει ενεργοποιείται ένας εναλλακτικός parser που χρησιμοποιεί το Gemini API. Τα δεδομένα που προκύπτουν αποθηκεύονται πρώτα ως JSON και μετά εισάγονται στη βάση. Τέλος καταγράφεται το αποτέλεσμα της εκτέλεσης σε log αρχείο και αν χρειαστεί στέλνεται ειδοποίηση μέσω email.

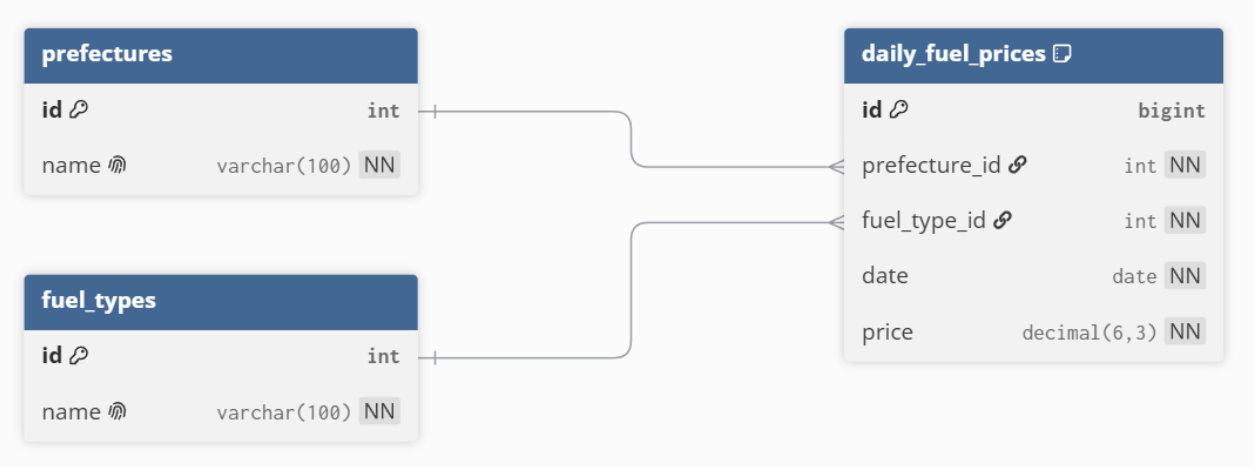
Το pipeline είναι σχεδιασμένο να είναι idempotent. Αυτό σημαίνει ότι αν τρέξει δύο φορές για την ίδια ημερομηνία δεν θα δημιουργήσει διπλές εγγραφές ούτε θα κατεβάσει ξανά το ίδιο αρχείο. Ο έλεγχος για το αν ένα δελτίο έχει ήδη επεξεργαστεί γίνεται μέσω της ύπαρξης του αντίστοιχου αρχείου JSON στον φάκελο `json_output`. Αν το JSON υπάρχει το δελτίο παραλείπεται. Αυτή η σχεδίαση επιτρέπει στο pipeline να ανακάμψει από αποτυχίες προηγούμενων εκτελέσεων απλά τρέχοντας ξανά.



Σχήμα 3.3: Αρχιτεκτονικό σχήμα του pipeline ανάκτησης δεδομένων.

3.3 Δημιουργία Σχεσιακής Βάσης Δεδομένων

Για την αποθήκευση των τιμών σχεδιάστηκε μια σχεσιακή βάση δεδομένων με τρεις πίνακες. Η βάση τρέχει σε MariaDB στον server του τμήματος και χρησιμοποιεί την κωδικοποίηση `utf8mb4` ώστε να υποστηρίζει σωστά τους ελληνικούς χαρακτήρες των ονομάτων των νομών και των καυσίμων.
Ο πίνακας `prefectures` αποθηκεύει τους 51 νομούς της Ελλάδας μαζί με την εγγραφή για τον Πανελλήνιο Σταθμισμένο Μέσο Όρο. Κάθε νομός έχει ένα μοναδικό `id` και ένα όνομα. Αντίστοιχα ο πίνακας `fuel_types` αποθηκεύει τα πέντε είδη καυσίμων που παρακολουθεί το Υπουργείο. Αυτά είναι η Αμόλυβδη 95 οκτανίων, η Αμόλυβδη 100 οκτανίων, το Diesel Κίνησης, το Υγραέριο Κίνησης (Autogas) και το Diesel Θέρμανσης Κατ' οίκον. Και οι δύο πίνακες έχουν περιορισμό `UNIQUE` στο όνομα ώστε να μην μπορούν να εισαχθούν διπλές εγγραφές.



Σχήμα 3.4: ER διάγραμμα της βάσης δεδομένων.

Ο κεντρικός πίνακας είναι ο `daily_fuel_prices`. Κάθε γραμμή αντιστοιχεί σε μία τιμή ενός συγκεκριμένου καυσίμου σε έναν συγκεκριμένο νομό για μία συγκεκριμένη ημερομηνία. Έχει foreign keys προς τους πίνακες `prefectures` και `fuel_types` και η τιμή αποθηκεύεται ως `DECIMAL(6,3)` για ακρίβεια στα τρία δεκαδικά ψηφία. Ορίστηκε επίσης ένας περιορισμός `UNIQUE` πάνω στον συνδυασμό `(prefecture_id, fuel_type_id, date)` ο οποίος εγγυάται ότι δεν θα υπάρξουν διπλές εγγραφές για την ίδια τριάδα. Τέλος υπάρχει ένα index πάνω στη στήλη `date` για να επιταχύνει τα ερωτήματα που ζητούν τιμές για συγκεκριμένο εύρος ημερομηνιών.
Η σχεδίαση αυτή ακολουθεί τους κανόνες της κανονικοποίησης. Τα ονόματα των νομών και των καυσίμων δεν επαναλαμβάνονται σε κάθε γραμμή τιμών αλλά αποθηκεύονται μία φορά στους πίνακες αναφοράς. Αυτό μειώνει τον χώρο που χρειάζεται η βάση και αποτρέπει ανωμαλίες κατά την ενημέρωση αφού μια αλλαγή σε ένα όνομα γίνεται σε ένα μόνο σημείο.
Παρακάτω δίνονται μερικά παραδείγματα ερωτημάτων SQL που μπορούν να χρησιμοποιηθούν απευθείας στη βάση. Το πρώτο επιστρέφει τις τιμές όλων των νομών για ένα συγκεκριμένο καύσιμο την πιο πρόσφατη ημερομηνία που υπάρχουν δεδομένα:

```
SELECT p.name AS prefecture, dfp.price
```

```
FROM daily_fuel_prices dfp
JOIN prefectures p ON p.id = dfp.prefecture_id
JOIN fuel_types ft ON ft.id = dfp.fuel_type_id
WHERE ft.name = 'Diesel Κίνησης'
  AND dfp.date = (SELECT MAX(date) FROM daily_fuel_prices)
ORDER BY dfp.price ASC;
```

Το δεύτερο παράδειγμα επιστρέφει την ιστορική εξέλιξη της τιμής ενός καυσίμου σε έναν συγκεκριμένο νομό για ένα εύρος ημερομηνιών:

```
SELECT dfp.date, dfp.price
FROM daily_fuel_prices dfp
JOIN prefectures p ON p.id = dfp.prefecture_id
JOIN fuel_types ft ON ft.id = dfp.fuel_type_id
WHERE p.name = 'ΝΟΜΟΣ ΘΕΣΣΑΛΟΝΙΚΗΣ'
  AND ft.name = 'Αμόλυβδη 95 οκτ.'
  AND dfp.date BETWEEN '2025-01-01' AND '2025-12-31'
ORDER BY dfp.date ASC;
```

Το τρίτο παράδειγμα υπολογίζει τη μέση τιμή ανά μήνα για ένα καύσιμο σε εθνικό επίπεδο χρησιμοποιώντας μόνο τις εγγραφές του Πανελλήνιου Σταθμισμένου Μέσου Όρου:

```
SELECT DATE_FORMAT(dfp.date, '%Y-%m') AS month,
       AVG(dfp.price) AS avg_price
FROM daily_fuel_prices dfp
JOIN prefectures p ON p.id = dfp.prefecture_id
JOIN fuel_types ft ON ft.id = dfp.fuel_type_id
WHERE p.name = 'ΠΑΝΕΛΛΗΝΙΟΣ ΣΤΑΘΜΙΣΜΕΝΟΣ Μ.Ο.'
  AND ft.name = 'Αμόλυβδη 95 οκτ.'
GROUP BY month
ORDER BY month ASC;
```

3.4 Ανάκτηση των PDF με Web Scraping

Η ανάκτηση των PDF έγινε σε δύο διαφορετικές φάσεις που αντιστοιχούν σε δύο διαφορετικές ανάγκες. Η πρώτη φάση ήταν η αρχική εισαγωγή δεδομένων όπου χρειάστηκε να κατεβούν όλα τα διαθέσιμα ιστορικά δελτία από την πρώτη μέρα που αυτά υπάρχουν online. Η δεύτερη φάση είναι η καθημερινή αυτόματη ανάκτηση που τρέχει μέσω cron και κατεβάζει μόνο τα νέα δελτία κάθε μέρα. Παρόλο που οι δύο φάσεις χρησιμοποιούν την ίδια βασική λογική scraping υπάρχουν σημαντικές διαφορές στις απαιτήσεις και στον τρόπο υλοποίησής τους.

Φάση 1: Μαζική Αρχική Λήψη

Η πρώτη φάση υλοποιήθηκε στο script `downloadALL.py`. Στόχος ήταν να κατέβουν όλα τα PDF που υπάρχουν στη σελίδα `deltia_dn.view` ώστε να έχουμε ένα πλήρες ιστορικό αρχείο από το οποίο θα τρέχαμε τον parser. Η σελίδα περιέχει συνδέσμους για εκατοντάδες ημέρες

οπότε η διαδικασία έπρεπε να είναι αξιόπιστη και να μπορεί να συνεχιστεί από εκεί που σταμάτησε αν κάτι πήγαινε στραβά.
Το πρώτο πρόβλημα που παρουσιάστηκε ήταν ότι ο server του Υπουργείου επιστρέφει σφάλμα `403 Forbidden` όταν δεχτεί αιτήματα χωρίς `User-Agent` header. Αυτό είναι μια απλή άμυνα που χρησιμοποιούν πολλοί servers για να απομακρύνουν bots. Η λύση ήταν να προσθέσουμε ένα header που μιμείται τον Chrome browser:

```
HEADERS = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
                  "AppleWebKit/537.36 (KHTML, like Gecko) "
                  "Chrome/91.0.4472.124 Safari/537.36"
}
```

Μόλις έρθει η απάντηση το HTML της σελίδας δίνεται στο BeautifulSoup. Για να βρούμε τους σωστούς συνδέσμους δεν μπορούσαμε να βασιστούμε σε CSS classes ή σε IDs αφού αυτά δεν είναι σταθερά. Αντίθετα χρησιμοποιήσαμε ένα regular expression που ταιριάζει σε ημερομηνίες της μορφής `HH/MM/EEEE` ή παραλλαγές με τελεία και παύλα:

```
date_pattern = re.compile(r'\d{1,2}[/.-]\d{1,2}[/.-]\d{4}')
links = soup.find_all('a', string=date_pattern)
```

Αυτή η προσέγγιση βρίσκει μόνο τα <a> tags που έχουν ακριβώς μια ημερομηνία ως κείμενο αποφεύγοντας άλλους τυχαίους συνδέσμους της σελίδας.
Ένα άλλο πρόβλημα ήταν η ονομασία των αρχείων. Τα PDF ονομάζονται από τον server με τυχαία εσωτερικά IDs που δεν έχουν καμία σχέση με την ημερομηνία τους. Για να μπορούμε να τα διαχειριστούμε εύκολα χρησιμοποιήσαμε ως όνομα αρχείου την ημερομηνία σε μορφή ISO (`EEEE-MM-HH`). Αυτό γίνεται από τη συνάρτηση `sanitize_filename` που δέχεται το κείμενο του συνδέσμου και επιστρέφει ένα ασφαλές όνομα αρχείου:

```
def sanitize_filename(date_str):
    parts = re.split(r'[/\.-]', date_str.strip())
    if len(parts) == 3:
        day, month, year = parts
        return f"{year}-{month.zfill(2)}-{day.zfill(2)}.pdf"
    return f"{date_str.strip().replace('/', '_')}.pdf"
```

Η μετατροπή σε `EEEE-MM-HH` είναι σκόπιμη γιατί αυτή η μορφή ταξινομείται αλφαβητικά σωστά. Έτσι ένα απλό `ls` ή `sorted()` δίνει τα αρχεία σε χρονολογική σειρά χωρίς επιπλέον λογική.
Τέλος για να μην επιβαρύνουμε τον server του Υπουργείου με εκατοντάδες αιτήματα στη σειρά το script περιμένει μισό δευτερόλεπτο ανάμεσα σε κάθε λήψη (`time.sleep(0.5)`). Το script ελέγχει επίσης αν κάθε αρχείο υπάρχει ήδη τοπικά πριν το κατεβάσει ώστε να μπορεί να σταματήσει και να συνεχιστεί χωρίς να ξαναλαμβάνει αρχεία που έχουν ήδη κατεβεί.

Φάση 2: Καθημερινή Αυτόματη Ανάκτηση

Η δεύτερη φάση υλοποιήθηκε μέσα στο daily_pipeline.py στη συνάρτηση get_pending_pdfs. Εδώ οι απαιτήσεις είναι διαφορετικές. Δεν χρειάζεται να κατεβάσουμε όλα τα διαθέσιμα δελτία αλλά μόνο τα νέα που δεν έχουν επεξεργαστεί ακόμα. Παράλληλα το script πρέπει να είναι αρκετά έξυπνο ώστε να ανακάμπτει από αποτυχίες χωρίς ανθρώπινη παρέμβαση.
Αρχικά κρατούνται μόνο τα 30 πιο πρόσφατα δελτία από τη σελίδα. Αυτό εξασφαλίζει ότι αν το pipeline δεν τρέξει για μερικές μέρες θα μπορέσει να ανακτήσει τα χαμένα δελτία στην επόμενη εκτέλεση. Για κάθε ημερομηνία γίνεται πρώτα έλεγχος αν υπάρχει ήδη το αντίστοιχο JSON στον φάκελο `json_output`. Αν υπάρχει το δελτίο παραλείπεται εντελώς χωρίς καν να γίνει λήψη. Αν δεν υπάρχει ελέγχεται αν υπάρχει το PDF τοπικά και αν δεν υπάρχει κατεβαίνει:

```
for dt, date_str, link in dated_links:
    if (JSON_DIR / f"{date_str}.json").exists():
        continue  # already processed

    pdf_path = PDF_DIR / f"{date_str}.pdf"
    if not pdf_path.exists():
        file_resp = requests.get(
            urljoin(BASE_URL, href), headers=HEADERS, timeout=60
        )
        file_resp.raise_for_status()
        pdf_path.write_bytes(file_resp.content)
```

Η λογική αυτή κάνει ολόκληρη τη λήψη idempotent. Αν το pipeline τρέξει δύο φορές την ίδια μέρα δεν θα ξανακατεβάσει κάτι ούτε θα κάνει διπλή επεξεργασία. Η ύπαρξη του JSON είναι το μοναδικό κριτήριο που αποφαίνεται αν μια ημερομηνία έχει ολοκληρωθεί επιτυχώς.

3.5 Εξαγωγή Δεδομένων από το PDF

Η εξαγωγή των τιμών από τα PDF αποδείχθηκε το πιο δύσκολο κομμάτι ολόκληρης της εργασίας. Στη θεωρία ένας πίνακας με τιμές μέσα σε ένα PDF φαίνεται κάτι απλό για να εξαχθεί αυτόματα. Στην πράξη όμως τα δελτία του Υπουργείου παρουσιάζουν μια σειρά από προβλήματα που εμφανίζονται με διαφορετικό τρόπο ανάλογα με το έτος δημοσίευσης. Για να εξαχθούν όλα τα ιστορικά δεδομένα χρειάστηκε να ακολουθηθεί διαφορετική προσέγγιση ανάλογα με την περίοδο.

Ανεξάρτητη Αρχή Ελέγχου Αγοράς & Προστασίας Καταναλωτή
Διατομεακή Μονάδα Ελέγχου Αγοράς
Διεύθυνση Διαχείρισης Δεδομένων, Στατιστικής Επεξεργασίας και Εποπτείας Ηλεκτρονικού Εμπορίου

Ταχ. Δ/νση : Πλ. Κάνιγγος, Αθήνα
Ταχ. Κώδικας : 106 77
Πληροφορίες : Χρονά Παναγιώτα
Τηλέφωνο : 213 1514 284
Fax : 210 3815 317
Email : fuelprices@mindev.gov.gr
Ιστοσελιδα : www.mindev.gov.gr.

Αθήνα, **12-05-2026**

Αριθ. Πρωτ. : **38160 - 12-05-2026**

ΠΡΟΣ : ΑΝΑΡΤΗΤΕΟ **ΣΤΟ FUELPRICES**

ΘΕΜΑ: ΚΑΘΗΜΕΡΙΝΟ ΔΕΛΤΙΟ ΕΠΙΣΚΟΠΗΣΗΣ ΤΙΜΩΝ ΥΓΡΩΝ ΚΑΥΣΙΜΩΝ

Μέσες τιμές λιανικής πώλησης καυσίμων ανά Νομό, για την

Κυριακή, 10 Μαΐου 2026

(τιμές σε €/λτ, συμπ. ΦΠΑ)

ΝΟΜΟΣ	Αμόλυβδη 95 οκτ.	Αμόλυβδη 100 οκτ.	Diesel Κίνησης	Υγραέριο κίνησης (Autogas)
ΝΟΜΟΣ ΑΤΤΙΚΗΣ	2,075	2,269	1,829	1,192
ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ	2,107	2,271	1,873	1,291
ΝΟΜΟΣ ΑΡΓΟΛΙΔΟΣ	2,099	2,280	1,864	1,176
ΝΟΜΟΣ ΑΡΚΑΔΙΑΣ	2,092	2,283	1,864	1,230
ΝΟΜΟΣ ΑΡΤΗΣ	2,079	2,253	1,846	1,240
ΝΟΜΟΣ ΑΧΑΪΑΣ	2,085	2,270	1,863	1,305
ΝΟΜΟΣ ΒΟΙΩΤΙΑΣ	2,086	2,282	1,870	1,286
ΝΟΜΟΣ ΓΡΕΒΕΝΩΝ	2,101	2,330	1,874	1,286
ΝΟΜΟΣ ΔΡΑΜΑΣ	2,096	2,271	1,847	1,212
ΝΟΜΟΣ ΔΩΔΕΚΑΝΗΣΟΥ	2,177	2,361	1,967	1,430
ΝΟΜΟΣ ΕΒΡΟΥ	2,089	2,262	1,874	1,317
ΝΟΜΟΣ ΕΥΒΟΙΑΣ	2,102	2,279	1,865	1,267
ΝΟΜΟΣ ΕΥΡΥΤΑΝΙΑΣ	2,106	2,356	1,945	1,400
ΝΟΜΟΣ ΖΑΚΥΝΘΟΥ	2,127	2,271	1,910	1,399
ΝΟΜΟΣ ΗΛΕΙΑΣ	2,094	2,264	1,853	1,198
ΝΟΜΟΣ ΗΜΑΘΙΑΣ	2,070	2,239	1,843	1,136
ΝΟΜΟΣ ΗΡΑΚΛΕΙΟΥ	2,141	2,273	1,908	1,407
ΝΟΜΟΣ ΘΕΣΠΡΩΤΙΑΣ	2,082	2,265	1,879	1,280

Σχήμα 3.5: Παράδειγμα ενός σύγχρονου δελτίου με τυποποιημένη μορφή.

Πρώτες προσπάθειες και προβλήματα

Η αρχική ιδέα ήταν να χρησιμοποιηθούν παραδοσιακά εργαλεία εξαγωγής δεδομένων από PDF όπως το pdfplumber, το tabula-py και άλλες παρόμοιες βιβλιοθήκες. Δοκιμάστηκαν επίσης διάφορες προσεγγίσεις με χρήση του BeautifulSoup σε μετατροπές PDF προς HTML αλλά κανένα από αυτά δεν λειτούργησε αξιόπιστα σε όλο το φάσμα των δελτίων. Όλες οι αυτοματοποιημένες προσπάθειες κατέληγαν σε λάθος δεδομένα για ένα σημαντικό ποσοστό αρχείων. Τα προβλήματα που εντοπίστηκαν στα παλιά PDF περιλαμβάνουν τα εξής:

- **Σπασμένη στήλη ονόματος νομού**. Σε αρκετά PDF το όνομα του νομού δεν χωρούσε σε

μία γραμμή του πίνακα και ήταν κομμένο στη μέση. Για παράδειγμα ο "ΝΟΜΟΣ ΘΕΣΣΑΛΟΝΙΚΗΣ" εμφανιζόταν ως "ΝΟΜΟΣ" σε μία γραμμή και "ΘΕΣΣΑΛΟΝΙΚΗΣ" στην επόμενη. Αυτό κάνει αδύνατη την αυτόματη αντιστοίχιση των τιμών με τους νομούς αφού ο parser δεν ξέρει σε ποια από τις δύο γραμμές βρίσκονται οι αριθμοί.

- **Προβλήματα κωδικοποίησης**. Σε αρκετά έγγραφα οι ελληνικοί χαρακτήρες ήταν αναμεμειγμένοι με λατινικούς που μοιάζουν οπτικά. Το ελληνικό κεφαλαίο Α (U+0391) και το λατινικό A (U+0041) είναι αόρατα διαφορετικά αλλά για έναν parser είναι δύο εντελώς διαφορετικοί χαρακτήρες. Αντίστοιχα προβλήματα εμφανίζονταν με τους χαρακτήρες Β, Ε, Η, Κ, Μ, Ν, Ο, Ρ, Τ, Χ και Υ.

- **Σκαναρισμένα PDF**. Πολλά παλαιότερα δελτία ήταν εικόνες σκαναρισμένων εντύπων χωρίς αναγνώσιμη ψηφιακή δομή. Σε αυτές τις περιπτώσεις το pdfplumber δεν μπορεί να εξαγάγει τίποτα γιατί δεν υπάρχει επιλέξιμο κείμενο.

- **Αλλαγή μορφής ανά έτος**. Η σειρά και ο αριθμός των στηλών αλλάζει μεταξύ ετών. Παλιότερα δελτία είχαν επιπλέον στήλες όπως η "Super" αμόλυβδη η οποία έχει πλέον καταργηθεί. Σε άλλα δελτία οι στήλες είναι σε εντελώς διαφορετική σειρά. Ένας parser που υποθέτει σταθερή θέση στηλών αποτυγχάνει αμέσως.

- **Εποχικά καύσιμα**. Το Diesel Θέρμανσης Κατ' οίκον είναι εποχικό προϊόν και δεν εμφανίζεται σε όλα τα δελτία. Στα δελτία των ανοιξιάτικων και των καλοκαιρινών μηνών η αντίστοιχη στήλη απουσιάζει εντελώς. Ένας απλός parser θα κατέληγε να εκχωρήσει τις τιμές των άλλων καυσίμων στη λάθος στήλη.

- **Παραλλαγές γραφής ονομάτων**. Σε παλιότερα δελτία ο νομός Αιτωλοακαρνανίας εμφανίζεται ως "ΝΟΜΟΣ ΑΙΤΩΛΟΑΚΑΡΝΑΝΙΑΣ" αντί για την κανονική γραφή "ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ". Παρόμοιες παραλλαγές εντοπίστηκαν και σε άλλους νομούς.

Μετά από αρκετές αποτυχημένες προσπάθειες παρατηρήθηκε ένα σημαντικό μοτίβο. Τα δελτία από το 2023 και έπειτα έχουν σταθερή και τυποποιημένη μορφή. Η δομή του πίνακα είναι ίδια σε όλα, το κείμενο είναι ψηφιακό αντί για σκαναρισμένη εικόνα και τα ονόματα των νομών εμφανίζονται ολόκληρα σε μία γραμμή. Τα προβλήματα που περιγράφηκαν παραπάνω σχεδόν εξαφανίζονται. Η απόφαση που πάρθηκε ήταν να χωριστεί η εξαγωγή σε δύο μονοπάτια: τα παλιά δελτία (πριν το 2023) θα τα διαχειριζόταν ένα LLM ικανό για ανάλυση εικόνας ενώ τα νέα θα τα διαχειριζόταν ένας κλασικός parser βασισμένος στο pdfplumber.

Παλιά Δελτία: Εξαγωγή με Gemini

Για όλα τα δελτία πριν το 2023 επιλέχθηκε ένα μεγάλο γλωσσικό μοντέλο με δυνατότητες ανάλυσης εικόνας (vision). Τα μοντέλα αυτού του τύπου μπορούν να επεξεργαστούν ένα PDF ως οπτικό περιεχόμενο και να εξαγάγουν δομημένη πληροφορία αντιμετωπίζοντας με φυσικότητα τις παραλλαγές που μπερδεύουν τους παραδοσιακούς parsers. Από τα διαθέσιμα μοντέλα επιλέχθηκε το `gemini-2.5-flash` της Google. Η επιλογή βασίστηκε κυρίως στη σχέση ποιότητας vision προς κόστος. Σε σύγκριση με άλλα μοντέλα της ίδιας κατηγορίας το Gemini Flash

έδωσε σαφώς καλύτερα αποτελέσματα στα σκαναρισμένα PDF διατηρώντας ταυτόχρονα χαμηλό κόστος ανά κλήση.
Η υλοποίηση βρίσκεται στο `parceALLpdfs.py`. Ο πυρήνας της είναι ένα αναλυτικό system prompt που εξηγεί στο μοντέλο τι ακριβώς θέλουμε. Το prompt γράφτηκε στα αγγλικά γιατί τα LLM έχουν εμφανώς καλύτερη απόδοση σε αγγλικές οδηγίες:

```
You are a precise data extraction assistant. You will receive a
PDF of a Greek government daily fuel price bulletin. Extract ALL
data from the fuel price table(s) and return ONLY a valid JSON
object with this exact structure:
{
  "date": "YYYY-MM-DD",
  "entries": [
    {
      "prefecture": "<prefecture name in Greek>",
      "prices": {
        "Αμόλυβδη 95 οκτ.": <float or null>,
        "Αμόλυβδη 100 οκτ.": <float or null>,
        "Diesel Κίνησης": <float or null>,
        "Υγραέριο κίνησης (Autogas)": <float or null>,
        "Diesel Θέρμανσης Κατ΄ οίκον": <float or null>
      }
    }
  ]
}

CRITICAL RULES:

1. COLUMN HEADERS: The column order in the table changes between
   years. You MUST identify each column by reading its actual
   header text, then map each price value to the correct fuel
    key. Do NOT assume a fixed column position.

2. HEATING DIESEL — SEASONAL COLUMN: "Diesel Θέρμανσης Κατ΄ οίκον"
   is sold only in winter months. Many PDFs from spring and summer
   do NOT have this column. If you cannot find it in the table,
   set every entry's value to null. Never invent or borrow values
   from another column to fill it.

3. NULL VALUES: Use null for "-", "0,000", or blank cells.

4. DECIMAL SEPARATOR: Convert comma to dot ("1,743" -> 1.743).

5. Include every row, including "ΠΑΝΕΛΛΗΝΙΟΣ ΣΤΑΘΜΙΣΜΕΝΟΣ Μ.Ο.".
```

```
6. Return ONLY the JSON object — no markdown, no explanation.
```

Οι κρίσιμοι κανόνες (CRITICAL RULES) στο prompt είναι αυτοί που λύνουν τα προβλήματα που είχαμε εντοπίσει. Ο πρώτος κανόνας αντιμετωπίζει την αλλαγή σειράς στηλών. Ο δεύτερος αντιμετωπίζει το εποχικό Diesel Θέρμανσης. Ο τέταρτος αντιμετωπίζει το ευρωπαϊκό κόμμα ως δεκαδικό διαχωριστή. Όλα αυτά είναι θέματα που σε έναν παραδοσιακό parser θα απαιτούσαν πολύπλοκη ειδική λογική.
Το PDF στέλνεται απευθείας στο API ως binary content χωρίς προ-επεξεργασία. Η κλήση γίνεται με `temperature=0` ώστε οι απαντήσεις να είναι ντετερμινιστικές και με `response_mime_type='application/json'` ώστε το API να εγγυάται έγκυρο JSON ως απάντηση:

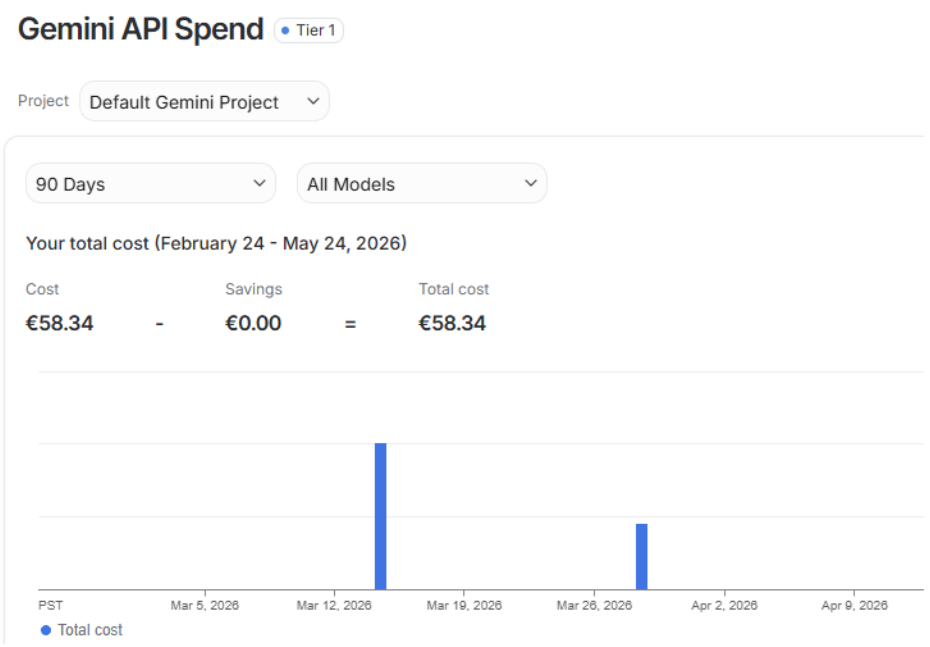
```
response = client.models.generate_content(
    model='gemini-2.5-flash',
    contents=[
        types.Part.from_bytes(
            data=pdf_bytes,
            mime_type='application/pdf'
        ),
        'Extract all fuel price data from this PDF
         and return as JSON.',
    ],
    config=types.GenerateContentConfig(
        system_instruction=SYSTEM_PROMPT,
        temperature=0,
        response_mime_type='application/json',
        max_output_tokens=65536,
    ),
)
data = json.loads(response.text)
```

Παρότι το prompt είναι αναλυτικό και ζητάει συγκεκριμένα ονόματα νομών, το μοντέλο επιστρέφει συχνά παραλλαγές. Σε κάποιες περιπτώσεις γράφει “Νομός Αττικής” αντί για “ΝΟΜΟΣ ΑΤΤΙΚΗΣ”, σε άλλες χρησιμοποιεί την ονομαστική αντί για τη γενική (π.χ. “Πέλλα” αντί για “ΠΕΛΛΗΣ”). Για να διορθωθούν αυτές οι αποκλίσεις υλοποιήθηκε η συνάρτηση `normalize_prefecture` που μετατρέπει το όνομα που επιστρέφει το LLM στην κανονική του μορφή. Η συνάρτηση εφαρμόζει διαδοχικά τρεις στρατηγικές:

```
def normalize_prefecture(llm_name):
    norm = normalize_text(llm_name)

    # 1. Exact match after normalization
    if norm in _NORM_TO_CANONICAL:
        return _NORM_TO_CANONICAL[norm]

    # 2. Substring containment
```

```
    for canon_norm, canon in _NORM_TO_CANONICAL.items():
        if canon_norm in norm or norm in canon_norm:
            return canon

    # 3. Levenshtein distance fallback
    best_dist = float('inf')
    best_canon = llm_name
    for canon_norm, canon in _NORM_TO_CANONICAL.items():
        d = _levenshtein(norm, canon_norm)
        if d < best_dist:
            best_dist = d
            best_canon = canon

    if best_dist <= 4:
        return best_canon
    return llm_name
```

Πρώτα γίνεται ένα ακριβές match μετά από κανονικοποίηση. Αν αποτύχει αυτό δοκιμάζεται έλεγχος υπο-συμβολοσειράς ώστε να πιαστούν μερικά ονόματα. Τέλος γίνεται υπολογισμός της απόστασης Levenshtein με κάθε κανονικό όνομα και κρατείται το πλησιέστερο εφόσον η απόσταση δεν ξεπερνά τους 4 χαρακτήρες. Με αυτή την τρίτη στρατηγική πιάνονται περιπτώσεις όπως “ΠΕΛΛΑΣ” προς “ΠΕΛΛΗΣ” όπου η διαφορά είναι μόνο ένα γράμμα.
Επειδή υπήρχαν εκατοντάδες PDF προς επεξεργασία και κάθε κλήση στο API είχε κόστος υλοποιήθηκε ένα σύστημα παρακολούθησης προόδου. Στο αρχείο `processed_log.json` αποθηκεύονται τα ονόματα των PDF που έχουν ήδη επεξεργαστεί. Αν το script σταματήσει για κάποιο λόγο μπορεί να ξεκινήσει από εκεί που σταμάτησε χωρίς να ξαναπληρώσει για κλήσεις που έχουν ήδη γίνει. Παράλληλα κάθε αποτέλεσμα περνάει από έναν έλεγχο εγκυρότητας. Αν εντοπιστούν λιγότερες από 40 εγγραφές, αν κάποιο όνομα δεν αντιστοιχίζεται σε κανονικό νομό ή αν κάποια τιμή βρίσκεται εκτός του εύρους 0,3 έως 4,0 ευρώ ανά λίτρο, το αρχείο σημειώνεται στο `flagged.json` για χειροκίνητο έλεγχο.



Σχήμα 3.6: Συνολικό κόστος της εξαγωγής των ιστορικών δεδομένων από το Gemini API.

Το συνολικό κόστος για την επεξεργασία όλων των ιστορικών δελτίων μέσω του Gemini API ανήλθε σε περίπου €58,34. Δεδομένου του μεγάλου αριθμού αρχείων που έπρεπε να επεξεργαστούν και της δυσκολίας τους θεωρούμε ότι αυτό το κόστος είναι αποδεκτό για μια εφάπαξ διαδικασία η οποία μας έδωσε ένα πλήρες ιστορικό αρχείο τιμών που θα ήταν αδύνατο να εξαχθεί διαφορετικά.

Νέα Δελτία: Εξαγωγή με pdfplumber

Για τα δελτία από το 2023 και έπειτα η εξαγωγή γίνεται με τη βιβλιοθήκη pdfplumber. Επειδή η μορφή είναι πλέον σταθερή δεν υπάρχει λόγος να χρησιμοποιηθεί LLM. Το pdfplumber τρέχει τοπικά, είναι δωρεάν και πολύ πιο γρήγορο από μια κλήση στο API. Η υλοποίηση βρίσκεται στο `daily_parser.py` στη συνάρτηση `parse_pdf`.
Η βασική ιδέα είναι να διαβάζεται το κείμενο κάθε σελίδας σαν απλό string, να εντοπίζονται οι γραμμές που ξεκινούν με όνομα νομού και να εξάγονται οι τιμές που ακολουθούν:

```
with pdfplumber.open(pdf_path) as pdf:
  for page in pdf.pages:
     text = page.extract_text()
     if not text:
         continue
     lines = text.split('\n')

  for line in lines:
    line_norm = normalize_text(line)
    canonical, norm_name = match_prefecture(line_norm)
```

```
    if not canonical:
      continue

      rest_of_line = line_norm.replace(norm_name, '', 1).strip()
      parts = rest_of_line.split()
      price_parts = [
          p for p in parts if re.match(r'^[\d,\.-]+$', p)
      ]

    prices = {}
    for fuel_label,raw_price in zip(ALL_FUELS_ORDER,price_parts):
      prices[fuel_label] = clean_price(raw_price)
```

Η αντιστοίχιση με τους κανονικούς νομούς γίνεται μέσω της `match_prefecture` που χρησιμοποιεί ένα προ-υπολογισμένο λεξικό (`_NORM_PREFECTURES`) από κανονικοποιημένα ονόματα. Όπως και στην περίπτωση του LLM, χρειάζεται normalization για να αντιμετωπιστούν τα προβλήματα τόνων και λατινικών χαρακτήρων. Η συνάρτηση `normalize_text` αναλαμβάνει αυτή τη δουλειά:

```
def normalize_text(text):
    text = text.upper()
    text = ''.join(
        c for c in unicodedata.normalize('NFD', text)
        if unicodedata.category(c) != 'Mn'
    )
    replacements = {
        'A': 'A', 'B': 'B', 'E': 'E', 'H': 'H', 'I': 'I',
        'K': 'K', 'M': 'M', 'N': 'N', 'O': 'O', 'P': 'P',
        'T': 'T', 'X': 'X', 'Y': 'Y', 'Z': 'Z',
    }
    for latin, greek in replacements.items():
        text = text.replace(latin, greek)
    return ' '.join(text.split())
```

Πρώτα γίνεται μετατροπή σε κεφαλαία ώστε να αγνοούμε διαφορές πεζών-κεφαλαίων. Στη συνέχεια αφαιρούνται οι τόνοι μέσω της κανονικής μορφής NFD του Unicode. Τέλος αντικαθίστανται οι λατινικοί χαρακτήρες που είναι οπτικά πανομοιότυποι με ελληνικούς με τους ελληνικούς αντίστοιχους. Τα τρία αυτά βήματα μαζί κάνουν το matching αρκετά ανθεκτικό ακόμα και όταν το PDF έχει μικρά προβλήματα κωδικοποίησης.
Οι τιμές μέσα στο κείμενο είναι σε ευρωπαϊκή μορφή με κόμμα ως δεκαδικό διαχωριστή (π.χ. `1,743`). Η συνάρτηση `clean_price` τις μετατρέπει σε αριθμούς κινητής υποδιαστολής διαχειριζόμενη παράλληλα τις περιπτώσεις κενών τιμών:

```
def clean_price(price_str):
    if not price_str or price_str.strip() in ['-', '', 'NULL']:
        return None
```

```
clean = price_str.replace(',', '.')
clean = re.sub(r'[^\d.]', '', clean)
val = float(clean)
if val == 0.0:
    return None
return val
```

Συμβολισμοί όπως η παύλα (–), η τιμή `0,000` ή ένα κενό κελί ερμηνεύονται όλα ως `null` αφού σημαίνουν απουσία δεδομένων. Με τον συνδυασμό όλων αυτών των βημάτων ο parser του pdfplumber εξάγει αξιόπιστα τα δεδομένα από κάθε νέο δελτίο.
Με τις δύο αυτές προσεγγίσεις καταφέραμε να εξαγάγουμε όλα τα διαθέσιμα ιστορικά δεδομένα από τη μέρα που υπάρχουν online μέχρι σήμερα. Τα παλιά δελτία πέρασαν από το Gemini, τα νέα από το pdfplumber, και όλα τα αποτελέσματα συγκεντρώθηκαν σε αρχεία JSON και στη συνέχεια εισήχθησαν στη βάση. Από εδώ και πέρα η ανάγκη για εξαγωγή πέφτει αποκλειστικά στα νέα δελτία που δημοσιεύονται κάθε μέρα. Αυτή η καθημερινή διαδικασία περιγράφεται στην επόμενη ενότητα.

3.6 Καθημερινή Επεξεργασία και Validation

Η καθημερινή επεξεργασία αφορά μόνο τα νέα δελτία που δημοσιεύονται κάθε μέρα. Επειδή τα νέα δελτία έχουν σταθερή μορφή το ζητούμενο είναι να επεξεργαστούν αυτόματα με μηδενική παρέμβαση και χωρίς κόστος όποτε αυτό είναι εφικτό. Παράλληλα η διαδικασία πρέπει να έχει δίχτυ ασφαλείας αν κάποιο δελτίο αποκλίνει από τη συνηθισμένη μορφή ώστε να μην εισέλθουν λάθος δεδομένα στη βάση. Η σχεδίαση που υιοθετήθηκε ακολουθεί μια διβάθμια λογική. Πρώτα κάθε νέο PDF δοκιμάζεται με τον δωρεάν parser του pdfplumber. Το αποτέλεσμα περνά από αυστηρό validation και αν περάσει θεωρείται έγκυρο. Αν το validation αποτύχει το ίδιο PDF στέλνεται στο Gemini ως fallback. Αυτή η προσέγγιση εκμεταλλεύεται την ταχύτητα και το μηδενικό κόστος του pdfplumber στις 99% των περιπτώσεων διατηρώντας ταυτόχρονα την ευρωστία του LLM για τις περιπτώσεις που πάει κάτι στραβά.

Λειτουργία του validation

Η λογική του validation βρίσκεται στη συνάρτηση `validate_parse` μέσα στο daily_pipeline.py. Παίρνει ως είσοδο τα δεδομένα που επέστρεψε το pdfplumber και ελέγχει πέντε διαφορετικές κατηγορίες προβλημάτων. Η λογική είναι σχεδιασμένη να εντοπίζει σιωπηρές αποτυχίες όπου ο parser τρέχει χωρίς exception αλλά παράγει αλλοιωμένα ή ελλιπή δεδομένα.
Ο πρώτος έλεγχος αφορά τον αριθμό των εγγραφών. Σε κάθε σωστά μορφοποιημένο δελτίο περιμένουμε 51 ή 52 εγγραφές: τους 51 νομούς της Ελλάδας συν την εγγραφή για τον Πανελλήνιο Σταθμισμένο Μέσο Όρο. Αν ο αριθμός των εγγραφών είναι μικρότερος από 50 θεωρούμε ότι ο parser έχασε κάποιες γραμμές. Σε ένα δελτίο όπου το pdfplumber αποτύγχανε να αναγνωρίσει τον πίνακα ολόκληρο, το νούμερο αυτό συνήθως είναι κοντά στο μηδέν.
Ο δεύτερος έλεγχος αφορά τα ονόματα των νομών. Κάθε εγγραφή πρέπει να έχει όνομα νομού που υπάρχει στην κανονική λίστα `CANONICAL_PREFECTURES`. Αν εμφανιστεί κάποιο όνομα που δεν αντιστοιχίζεται σε αυτή τη λίστα είναι σαφής ένδειξη ότι η εξαγωγή πήγε στραβά.

Συνήθως αυτό συμβαίνει όταν το pdfplumber διαβάζει εσφαλμένα έναν χαρακτήρα ή όταν προκύπτει σπασμένη γραμμή.
Ο τρίτος έλεγχος αφορά τις τιμές. Σταθερό λογικό εύρος για τις τιμές καυσίμων στην Ελλάδα είναι μεταξύ 0,3 και 4,0 ευρώ ανά λίτρο. Οποιαδήποτε τιμή πέφτει εκτός αυτού του εύρους είναι ύποπτη. Συνήθως οφείλεται σε λάθος ανάγνωση όπου π.χ. μπερδεύεται η κουκκίδα με το κόμμα και μια τιμή `1,743` γίνεται `1743`.
Ο τέταρτος έλεγχος αφορά την πληρότητα των δεδομένων ανά νομό. Αν μια εγγραφή έχει λιγότερες από 3 αριθμητικές τιμές καυσίμων (δηλαδή 3 ή περισσότερες είναι null) πιθανότατα έγινε λάθος στο διάβασμα.
Ο πέμπτος έλεγχος είναι ο πιο ενδιαφέρων. Συγκρίνει τον απλό μέσο όρο των τιμών των νομών για κάθε καύσιμο με την τιμή που αναφέρει το ίδιο το δελτίο ως Πανελλήνιο Σταθμισμένο Μέσο Όρο. Οι δύο αυτές τιμές δεν είναι ίδιες (η μία είναι σταθμισμένη ανά κατανάλωση, η άλλη απλός μέσος όρος) αλλά πρέπει να είναι σχετικά κοντινές. Αν η απόκλιση ξεπεράσει το 10% είναι ένδειξη ότι κάποιες τιμές έχουν διαβαστεί λάθος και τραβάνε τον μέσο όρο μακριά από την πραγματική κατάσταση.
Παρακάτω φαίνεται ο πυρήνας της συνάρτησης `validate_parse` όπου εκτελούνται οι έλεγχοι:

```
def validate_parse(data):
    warnings = []
    entries = data.get('entries', [])

    if len(entries) < MIN_ENTRIES:
        warnings.append(f"only {len(entries)}
        entries (expected 51-52)")

    unmatched = []
    out_of_range = 0
    low_price_entries = 0

    for entry in entries:
        pref = entry.get('prefecture', '')
        if pref not in CANONICAL_PREFECTURES:
            unmatched.append(pref)

        prices = entry.get('prices', {})
        non_null = [v for v in prices.values()
                    if isinstance(v, (int, float))]
        if len(non_null) < 3:
            low_price_entries += 1

        for fuel, val in prices.items():
            if isinstance(val, (int, float)) and \
               not (PRICE_MIN <= val <= PRICE_MAX):
                out_of_range += 1
```

```
    # ... avg deviation check (see below)
```

Ο έλεγχος απόκλισης μέσου όρου εφαρμόζεται σε κάθε καύσιμο ξεχωριστά. Για κάθε καύσιμο που υπάρχει στον Πανελλήνιο Σταθμισμένο Μέσο Όρο υπολογίζεται ο απλός μέσος όρος των τιμών των νομών και συγκρίνεται με την εθνική τιμή:

```
for fuel, national_val in national.items():
    values = [
        e['prices'][fuel] for e in prefectures_only
        if isinstance(e.get('prices',{}).get(fuel),(int, float))
    ]
    if not values:
        continue
    simple_avg = sum(values) / len(values)
    deviation = abs(simple_avg-national_val) / national_val * 100
    if deviation > AVG_DEVIATION_PCT:
        avg_deviation_fail = True
```

Το αποτέλεσμα του validation είναι μια τιμή `acceptable` που υπολογίζεται από τον συνδυασμό όλων των ελέγχων. Για να θεωρηθεί αποδεκτή η εξαγωγή πρέπει να ισχύουν ταυτόχρονα όλα τα παρακάτω:

```
acceptable = (
    len(entries) >= MIN_ENTRIES
    and len(unmatched) == 0
    and out_of_range <= 5
    and low_price_entries <= 3
    and not avg_deviation_fail
)
```

Οι αριθμοί 5 και 3 στο out_of_range και low_price_entries λειτουργούν ως όρια ανοχής. Επιτρέπουν λίγα μεμονωμένα προβλήματα (π.χ. ένας νομός που πραγματικά δεν έχει δεδομένα για κάποιο καύσιμο) χωρίς να απορρίπτεται ολόκληρο το δελτίο. Αν όμως τα προβλήματα ξεπερνούν αυτά τα όρια θεωρείται ότι ο parser έπεσε έξω και η εξαγωγή απορρίπτεται.

Ενεργοποίηση του LLM Fallback

Όταν το validation αποτύχει το ίδιο PDF στέλνεται στο Gemini μέσω της συνάρτησης `llm_parse`. Αυτή η συνάρτηση καλεί την `extract_data_from_pdf` από το `parceALLpdfs.py` (την ίδια συνάρτηση που χρησιμοποιήθηκε για τα ιστορικά δεδομένα) και επαναχρησιμοποιεί ολόκληρο το system prompt που περιγράψαμε στην προηγούμενη ενότητα. Με αυτή την επιλογή δεν διπλασιάζουμε τον κώδικα και έχουμε τη βεβαιότητα ότι το prompt που λειτούργησε καλά στα ιστορικά δεδομένα θα δουλέψει το ίδιο και στα νέα.

```
def llm_parse(pdf_path):
    from parceALLpdfs import extract_data_from_pdf
    from google import genai

    api_key = os.getenv('GEMINI_API_KEY')
    if not api_key:
        return None, "GEMINI_API_KEY not set"

    client = genai.Client(api_key=api_key)
    data = extract_data_from_pdf(client, pdf_path)

    if data is None:
      return None, "LLM extraction returned None after retries"

    data['date'] = pdf_path.stem  # filename is authoritative
    return data, None
```

Η συνάρτηση `extract_data_from_pdf` έχει ενσωματωμένη λογική επανάληψης (retry) για να αντιμετωπίσει παροδικά σφάλματα δικτύου ή του API. Δοκιμάζει την κλήση μέχρι 3 φορές με exponential backoff. Αν αποτύχει η πρώτη προσπάθεια περιμένει 5 δευτερόλεπτα και ξαναπροσπαθεί, αν αποτύχει η δεύτερη περιμένει 10 δευτερόλεπτα, και η τρίτη 20 δευτερόλεπτα. Αν και οι τρεις αποτύχουν επιστρέφει `None`:

```
for attempt in range(3):
    try:
        response = client.models.generate_content(...)
        data = json.loads(response.text)
        return normalize_entries(data)
    except Exception as e:
        if attempt < 2:
            wait = 5 * (2 ** attempt)
            time.sleep(wait)
        else:
            return None
```

Το exponential backoff είναι σημαντικό γιατί τα περισσότερα σφάλματα API είναι παροδικά (rate limiting, στιγμιαία αδυναμία σύνδεσης) και επιλύονται με μια μικρή αναμονή. Αν συνεχίσει και μετά τις 3 προσπάθειες θεωρούμε ότι υπάρχει σοβαρότερο πρόβλημα και το δελτίο παραμένει ως αποτυχία.
Με αυτή τη λογική η ροή είναι η εξής. Το pdfplumber τρέχει πρώτα. Αν περάσει το validation η εξαγωγή θεωρείται επιτυχής και η μέθοδος καταγράφεται ως `pdfplumber`. Αν αποτύχει το validation καλείται το `llm_parse`. Αν το LLM επιστρέψει έγκυρα δεδομένα η εξαγωγή θεωρείται επιτυχής και η μέθοδος καταγράφεται ως `llm`. Αν αποτύχει και το LLM η επεξεργασία του συγκεκριμένου δελτίου σταματάει και προγραμματίζεται για επανεξέταση. Σημαντικό είναι ότι σε καμία περίπτωση δεν εισέρχονται στη βάση δεδομένα που δεν έχουν περάσει από τους παραπάνω ελέγχους.

3.7 Αποθήκευση Δεδομένων

Μετά την επιτυχή εξαγωγή τα δεδομένα ενός δελτίου περνούν από μια σειρά βημάτων αποθήκευσης. Η σχεδίαση που επιλέχθηκε αποθηκεύει τα δεδομένα σε δύο διαφορετικές μορφές: πρώτα ως αρχείο JSON στο τοπικό filesystem και μετά ως εγγραφές στη σχεσιακή βάση. Αυτή η διπλή αποθήκευση δεν είναι περιττή, εξυπηρετεί συγκεκριμένους σκοπούς που εξηγούνται παρακάτω.

Γιατί JSON ως ενδιάμεσο βήμα

Η πρώτη σκέψη όταν κάποιος βλέπει το pipeline είναι γιατί να μην εισάγονται τα δεδομένα απευθείας στη βάση. Στην πραγματικότητα η αποθήκευση σε JSON πριν την εισαγωγή στη βάση εξυπηρετεί πολλούς λόγους που έγιναν σαφείς κατά την ανάπτυξη του συστήματος.
Πρώτον, το JSON λειτουργεί ως αντίγραφο ασφαλείας. Αν κάποια στιγμή η βάση χαθεί, αλλοιωθεί ή χρειαστεί να μεταφερθεί σε άλλον server, μπορούμε να την ξαναγεμίσουμε από τα αποθηκευμένα JSON χωρίς να χρειαστεί να ξαναεπεξεργαστούμε τα PDF. Αυτό είναι ιδιαίτερα σημαντικό για τα παλιά δελτία που η εξαγωγή τους κόστισε περίπου €58 σε χρήσεις του Gemini API. Αν δεν είχαμε τα JSON θα έπρεπε να πληρώσουμε ξανά για το ίδιο αποτέλεσμα.
Δεύτερον, το JSON χρησιμοποιείται ως marker προόδου του pipeline. Όπως είδαμε στην ενότητα του scraping ο έλεγχος αν ένα δελτίο έχει επεξεργαστεί γίνεται μέσω της ύπαρξης του αντίστοιχου JSON. Αν είχαμε μόνο τη βάση θα έπρεπε να κάνουμε query κάθε φορά για να μάθουμε αν μια ημερομηνία υπάρχει ήδη, κάτι που είναι πιο αργό από έναν απλό έλεγχο ύπαρξης αρχείου στο filesystem.
Τρίτον, ο διαχωρισμός μας επιτρέπει να ξανατρέξουμε μόνο το βήμα εισαγωγής στη βάση χωρίς να ξαναεπεξεργαστούμε τα PDF. Αν για παράδειγμα αλλάξουμε το σχήμα της βάσης ή θελήσουμε να φορτώσουμε τα δεδομένα σε μια δεύτερη βάση μπορούμε να γράψουμε ένα ξεχωριστό script (το `import_to_db.py`) που διαβάζει τα JSON και τα εισάγει.
Τα JSON αρχεία αποθηκεύονται στον φάκελο `json_output` με όνομα την ημερομηνία του δελτίου σε μορφή ISO. Η εγγραφή γίνεται με την τυπική βιβλιοθήκη `json` της Python με ενεργοποιημένο το `ensure_ascii=False` ώστε οι ελληνικοί χαρακτήρες να αποθηκεύονται σωστά:

```
json_path = JSON_DIR / f"{date_str}.json"
with open(json_path, 'w', encoding='utf-8') as f:
    json.dump(data, f, ensure_ascii=False, indent=2)
```

Εισαγωγή στη Βάση

Η εισαγωγή των δεδομένων στη βάση γίνεται μέσω του module `db.py` που περιέχει όλες τις σχετικές helper functions. Η αρχιτεκτονική αυτή έχει το πλεονέκτημα ότι ο κώδικας εισαγωγής βρίσκεται σε ένα μόνο σημείο και χρησιμοποιείται τόσο από το καθημερινό pipeline όσο και από τον αρχικό bulk importer.
Τα στοιχεία σύνδεσης με τη βάση δεν αποθηκεύονται μέσα στον κώδικα. Αυτό είναι σημαντικό για λόγους ασφαλείας αφού ο κώδικας βρίσκεται σε public repository στο GitHub και αν τα credentials ήταν μέσα του θα γίνονταν διαθέσιμα σε όλους. Αντί για αυτό χρησιμοποιείται ένα

αρχείο `.env` στη ρίζα του project που είναι εξαιρεμένο από το git μέσω του `.gitignore`. Από εκεί φορτώνονται κατά την εκκίνηση του script με τη βιβλιοθήκη `python-dotenv`:

```
def get_db_config() -> dict:
    return {
        'host': os.getenv('DB_HOST'),
        'port': int(os.getenv('DB_PORT', '3306')),
        'user': os.getenv('DB_USER'),
        'password': os.getenv('DB_PASSWORD'),
        'database': os.getenv('DB_NAME'),
        'charset': 'utf8mb4',
    }

def connect():
    cfg = get_db_config()
    return pymysql.connect(**cfg)
```

Η σύνδεση γίνεται μέσω της βιβλιοθήκης `pymysql` που είναι ένα καθαρό Python implementation του MySQL πρωτοκόλλου. Επιλέξαμε το `utf8mb4` ως charset για να υποστηρίζονται σωστά οι ελληνικοί χαρακτήρες στα ονόματα νομών και καυσίμων.
Πριν την εισαγωγή των τιμών χρειαζόμαστε τα IDs των νομών και των καυσίμων από τους αντίστοιχους πίνακες αναφοράς. Αυτό γίνεται μία φορά στην αρχή κάθε εκτέλεσης μέσω της `load_id_maps` που φορτώνει όλους τους νομούς και τα καύσιμα σε δύο dictionaries:

```
def load_id_maps(cursor):
    cursor.execute('SELECT id, name FROM prefectures')
    prefs = {normalize_text(name): pid for pid,
    name in cursor.fetchall()}
    cursor.execute('SELECT id, name FROM fuel_types')
    fuels = {name: fid for fid, name in cursor.fetchall()}
    return prefs, fuels
```

Η ίδια συνάρτηση `normalize_text` που είδαμε στην εξαγωγή χρησιμοποιείται και εδώ για την κανονικοποίηση των ονομάτων νομών. Έτσι ένα όνομα που υπάρχει στο JSON μπορεί να αντιστοιχηθεί σωστά με την εγγραφή της βάσης ακόμα και αν έχει μικρές διαφορές γραφής.
Η εισαγωγή των τιμών γίνεται με ένα μόνο SQL statement που έχει τη μορφή `INSERT …ON DUPLICATE KEY UPDATE`. Αυτή η σύνταξη της MySQL/MariaDB συνδυάζει την εισαγωγή και την ενημέρωση σε μία μόνο εντολή και εκμεταλλεύεται τον περιορισμό `UNIQUE` του πίνακα:

```
_INSERT_SQL = """
    INSERT INTO daily_fuel_prices
        (prefecture_id, fuel_type_id, date, price)
    VALUES (%s, %s, %s, %s)
    ON DUPLICATE KEY UPDATE price = VALUES(price)
"""
```

Η λογική είναι απλή: αν δεν υπάρχει εγγραφή για τον συγκεκριμένο συνδυασμό (νομός, καύσιμο, ημερομηνία) γίνεται `INSERT`. Αν υπάρχει ήδη γίνεται `UPDATE` της τιμής. Με αυτόν τον τρόπο η εισαγωγή είναι idempotent. Αν το pipeline τρέξει δύο φορές για την ίδια ημέρα δεν θα δημιουργηθούν διπλές εγγραφές, απλώς θα γραφτεί ξανά η ίδια τιμή.
Όλες οι εγγραφές ενός δελτίου εισάγονται με μια κλήση στην `executemany` αντί για ξεχωριστά `INSERT` ένα προς ένα. Αυτό είναι σημαντικά πιο γρήγορο γιατί στέλνεται ένα μόνο μήνυμα στον MySQL server με όλες τις γραμμές:

```
def insert_entries(cursor, data, pref_ids, fuel_ids):
    date = data.get('date')
    rows = []
    unknown = set()

    for entry in data.get('entries', []):
        pref_name = entry.get('prefecture', '')
        pref_id = find_prefecture_id(pref_name, pref_ids)
        if pref_id is None:
            unknown.add(pref_name)
            continue

        for fuel, price in entry.get('prices', {}).items():
            if price is None:
                continue
            fuel_id = fuel_ids.get(fuel)
            if fuel_id is None:
                continue
            rows.append((pref_id, fuel_id, date, float(price)))

    if rows:
        cursor.executemany(_INSERT_SQL, rows)

    return len(rows), unknown
```

Σημαντικό είναι ότι χρησιμοποιούνται prepared statements με τα `%s` placeholders. Αυτό προστατεύει από SQL injection και είναι η πιο σωστή μέθοδος για κάθε query που περιέχει δεδομένα από εξωτερική πηγή.
Όταν ολοκληρωθεί επιτυχώς η εισαγωγή στη βάση το αρχικό PDF διαγράφεται από τον δίσκο. Αυτό γίνεται για λόγους οικονομίας χώρου αφού όλη η πληροφορία υπάρχει πλέον τόσο στο JSON όσο και στη βάση. Αν για κάποιο λόγο η εισαγωγή στη βάση αποτύχει το PDF παραμένει στον δίσκο και η επανεκτέλεση του pipeline θα ξαναπροσπαθήσει την επόμενη μέρα.

Logging Εκτελέσεων

Παράλληλα με την αποθήκευση των δεδομένων το pipeline κρατάει αναλυτικό αρχείο για κάθε εκτέλεση. Το log βρίσκεται στο `logs/daily_pipeline.json` και είναι μια λίστα από αντι-

κείμενα όπου κάθε αντικείμενο περιγράφει την επεξεργασία ενός δελτίου. Η μορφή JSON επιλέχθηκε αντί για ένα παραδοσιακό text log γιατί επιτρέπει εύκολη αυτόματη ανάλυση. Για παράδειγμα ο εβδομαδιαίος έλεγχος του pipeline διαβάζει το JSON και βγάζει στατιστικά χωρίς να χρειάζεται να γίνει parsing ενός text format.
Κάθε εγγραφή του log περιέχει τα παρακάτω πεδία:

- `date`: η ημερομηνία του δελτίου σε μορφή `EEEE-MM-HH`
- `run_at`: η ημερομηνία και ώρα εκτέλεσης σε ISO 8601 format
- `status`: ένα από τα τρία `success`, `fallback_llm`, `failed`
- `method`: η μέθοδος εξαγωγής (`pdfplumber`, `llm`, `both`, ή `none`)
- `entries_count`: ο αριθμός των εγγραφών που εξήχθησαν
- `warnings`: λίστα με τυχόν προειδοποιήσεις από το validation
- `error`: μήνυμα σφάλματος αν υπάρχει, αλλιώς `null`

Η εγγραφή στο log γίνεται μέσω της `append_log` που αναλαμβάνει τη φόρτωση του υπάρχοντος log, την προσθήκη της νέας εγγραφής και την αποθήκευση πίσω:

```
def append_log(date_str, status, method, entries_count,
               warnings=None, error=None):
    entries = load_log()
    entries.append({
        'date': date_str,
        'run_at': datetime.now().isoformat(timespec='seconds'),
        'status': status,
        'method': method,
        'entries_count': entries_count,
        'warnings': warnings or [],
        'error': error,
    })
    save_log(entries)
```

Παρακάτω φαίνονται τρία παραδείγματα εγγραφών log που αντιστοιχούν σε τρία πιθανά σενάρια εκτέλεσης.
Το πρώτο παράδειγμα δείχνει μια φυσιολογική επιτυχημένη εκτέλεση όπου το pdfplumber κατάφερε να εξαγάγει σωστά τα δεδομένα και χωρίς προειδοποιήσεις:

```
{
  "date": "2026-03-26",
  "run_at": "2026-03-29T19:12:39",
  "status": "success",
  "method": "pdfplumber",
  "entries_count": 52,
```

```
  "warnings": [],
  "error": null
}
```

Το δεύτερο παράδειγμα δείχνει μια εκτέλεση όπου το pdfplumber απέτυχε στο validation και ενεργοποιήθηκε το Gemini fallback. Στις προειδοποιήσεις φαίνεται γιατί το pdfplumber απορρίφθηκε:

```
{
  "date": "2024-08-15",
  "run_at": "2024-08-15T12:46:12",
  "status": "fallback_llm",
  "method": "llm",
  "entries_count": 51,
  "warnings": [
    "only 38 entries (expected 51-52)",
    "3 unmatched prefectures: ['ΝΟΜΟΣ', 'ΝΟΜΟΣ', 'ΑΤΤΙΚΗΣ']",
    "avg deviation Αμόλυβδη 95 οκτ.: avg=1.823
        vs national=1.745 (4.5%)"
  ],
  "error": null
}
```

Το τρίτο παράδειγμα δείχνει μια εντελώς αποτυχημένη εκτέλεση όπου ούτε το pdfplumber ούτε το Gemini κατάφεραν να ολοκληρώσουν την επεξεργασία. Στο πεδίο `error` καταγράφεται και η αιτία της αποτυχίας:

```
{
  "date": "2026-04-02",
  "run_at": "2026-04-02T00:34:58",
  "status": "failed",
  "method": "none",
  "entries_count": 0,
  "warnings": [],
  "error": "Page fetch exception: HTTPSConnectionPool(
    host='www.fuelprices.gr', port=443):
    Read timed out. (read timeout=30)"
}
```

Με αυτή τη μορφή logging μπορούμε εύκολα να απαντήσουμε σε ερωτήσεις όπως πόσες φορές χρειάστηκε fallback τον τελευταίο μήνα, ποιες είναι οι πιο συχνές αιτίες αποτυχίας και ποια δελτία χρειάζονται επανεξέταση. Αυτές οι πληροφορίες είναι χρήσιμες τόσο για την παρακολούθηση της υγείας του συστήματος όσο και για την σύνταξη εβδομαδιαίων αναφορών που περιγράφονται στην επόμενη ενότητα.

3.8 Αυτόματη Ανάκτηση (Cron Job)

Όλα τα παραπάνω βήματα του pipeline δεν θα είχαν πρακτική αξία αν χρειαζόταν κάποιος να τρέχει χειροκίνητα το script κάθε μέρα. Για να γίνει η διαδικασία πραγματικά αυτόματη χρησιμοποιείται το cron του Linux. Παράλληλα έχει υλοποιηθεί ένα σύστημα ειδοποιήσεων μέσω email ώστε ο διαχειριστής να ενημερώνεται για προβλήματα ή να λαμβάνει εβδομαδιαία αναφορά για την κατάσταση του συστήματος.

Ρύθμιση του cron

Στο crontab του server έχει προστεθεί μία γραμμή που εκτελεί το `daily_pipeline.py` κάθε μέρα στις 12:46:

```
46 12 * * * cd /home/pc/fuelprices && \
            ./venv/bin/python scripts/daily_pipeline.py \
            >> logs/cron.log 2>&1
```

Τα πέντε πεδία στην αρχή της γραμμής είναι το λεπτό (46), η ώρα (12), η ημέρα του μήνα (*), ο μήνας (*) και η ημέρα της εβδομάδας (*). Οι αστερίσκοι σημαίνουν “οποιοδήποτε” οπότε η εργασία τρέχει κάθε μέρα. Η αλλαγή του current directory με `cd` είναι απαραίτητη γιατί το script περιμένει να βρίσκεται στη ρίζα του project ώστε να βρει το `.env` και τους σχετικούς φακέλους. Η χρήση του `./venv/bin/python` εξασφαλίζει ότι το script τρέχει με το virtual environment του project και έχει πρόσβαση στις εγκατεστημένες βιβλιοθήκες.
Η ώρα 12:46 επιλέχθηκε με βάση δύο παρατηρήσεις. Πρώτον τα νέα δελτία ανεβαίνουν στον ιστότοπο του Υπουργείου συνήθως μέσα στο πρωί, οπότε το μεσημέρι είναι αρκετά αργά για να έχει ολοκληρωθεί η δημοσίευση. Δεύτερον παρατηρήθηκε ότι ο ιστότοπος του Υπουργείου σε ορισμένες νυχτερινές ώρες είναι περιστασιακά εκτός λειτουργίας (παρουσιάστηκαν αρκετές τέτοιες περιπτώσεις στις πρώτες μέρες λειτουργίας του pipeline). Μια εκτέλεση μέσα στη μέρα όταν ο server λειτουργεί κανονικά είναι σημαντικά πιο αξιόπιστη.
Το ακριβές λεπτό 46 δεν έχει ιδιαίτερη σημασία, απλώς αποφεύγεται η ώρα 12:00 ή 12:30 που είναι “στρογγυλά” σημεία όπου τρέχουν πολλά άλλα cron jobs στο σύστημα και πιθανόν θα δημιουργήσει συμφόρηση.

Email Notifications

Παρόλο που τα logs δίνουν πλήρη εικόνα της λειτουργίας του pipeline, δεν είναι πρακτικό να τα ελέγχει κανείς χειροκίνητα κάθε μέρα. Για αυτό υλοποιήθηκε ένα σύστημα email notifications. Υπάρχουν δύο διαφορετικοί τύποι μηνυμάτων: τα critical alerts που στέλνονται άμεσα όταν συμβεί κάποιο σοβαρό πρόβλημα και η εβδομαδιαία περίληψη που στέλνεται κάθε Κυριακή.
Η αποστολή των emails γίνεται μέσω SMTP με τη βιβλιοθήκη `smtplib` της Python που είναι μέρος της standard library. Ως SMTP server χρησιμοποιείται το Gmail (`smtp.gmail.com` στην πόρτα 587 με STARTTLS). Επειδή το Gmail δεν επιτρέπει πλέον login με τον κανονικό κωδικό για λόγους ασφαλείας, δημιουργήθηκε ένα *app password* από τις ρυθμίσεις του λογαριασμού. Όλα τα στοιχεία σύνδεσης βρίσκονται στο `.env`:

```
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=alerts@example.com
SMTP_PASSWORD=xxxx-xxxx-xxxx-xxxx
ALERT_EMAIL=georgekoutsiamoneys@gmail.com, stoug@ihu.gr
```

Η βασική συνάρτηση αποστολής είναι η `send_email`. Δέχεται το θέμα και το σώμα του μηνύματος, διαβάζει τα στοιχεία σύνδεσης από τα environment variables και κάνει την αποστολή:

```
def send_email(subject, body):
    host = os.getenv('SMTP_HOST')
    port = int(os.getenv('SMTP_PORT', '587'))
    user = os.getenv('SMTP_USER')
    password = os.getenv('SMTP_PASSWORD')
    to_addr = os.getenv('ALERT_EMAIL')

    msg = MIMEText(body, 'plain', 'utf-8')
    msg['Subject'] = subject
    msg['From'] = user
    msg['To'] = to_addr

    with smtplib.SMTP(host, port) as srv:
        srv.starttls()
        srv.login(user, password)
        srv.send_message(msg)
```

Το μήνυμα χτίζεται με `MIMEText` και κωδικοποίηση UTF-8 για να υποστηρίζονται οι ελληνικοί χαρακτήρες. Η σύνδεση γίνεται μέσα σε ένα context manager (`with`) ώστε να κλείνει αυτόματα ακόμα και αν προκύψει σφάλμα.

Critical Alerts

Όταν συμβεί κάποιο σοβαρό πρόβλημα κατά την εκτέλεση του pipeline στέλνεται άμεσα ένα critical alert email. Δεν περιμένουμε την Κυριακή για την εβδομαδιαία περίληψη γιατί ορισμένα προβλήματα χρειάζονται γρήγορη αντίδραση. Τα σενάρια που πυροδοτούν critical alert είναι τα παρακάτω:

- Αποτυχία λήψης της κύριας σελίδας από το `fuelprices.gr` (π.χ. τιμή λήξης χρόνου ή HTTP error)
- Αποτυχία και των δύο parsers (pdfplumber + Gemini fallback) για ένα δελτίο
- Αποτυχία της εισαγωγής στη βάση δεδομένων (π.χ. απώλεια σύνδεσης με τη MariaDB)

Η συνάρτηση `send_critical_alert` συνθέτει ένα κατατοπιστικό μήνυμα με την ημερομηνία του δελτίου, την αιτία της αποτυχίας και τη διαδρομή του log αρχείου για περαιτέρω διερεύνηση:

```
def send_critical_alert(date_str, error):
    send_email(
        f"ALERT: Fuel price pipeline FAILED - {date_str}",
        f"The daily fuel price pipeline failed.\n\n"
        f"Date: {date_str}\n"
        f"Error: {error}\n\n"
        f"Check manually: {TARGET_PAGE}\n"
        f"Log: {LOG_FILE.resolve()}"
    )
```

Ένα πραγματικό critical alert μοιάζει κάπως έτσι:

```
Subject: ALERT: Fuel price pipeline FAILED - 2026-04-02

The daily fuel price pipeline failed.

Date: 2026-04-02
Error: Page fetch exception: HTTPSConnectionPool(host=
'www.fuelprices.gr', port=443): Read timed out. (read timeout=30)

Check manually: https://www.fuelprices.gr/deltia_dn.view
Log: /home/pc/fuelprices/logs/daily_pipeline.json
```

Εβδομαδιαία Περίληψη

Πέρα από τα άμεσα alerts κάθε Κυριακή στέλνεται μια συγκεντρωτική αναφορά για όλη την εβδομάδα που μόλις πέρασε. Σκοπός της είναι να έχει ο διαχειριστής μια γενική εικόνα της υγείας του συστήματος ακόμα και όταν δεν έχει συμβεί κάτι σοβαρό. Ο έλεγχος για την Κυριακή γίνεται με `datetime.now().weekday() == 6` (η Δευτέρα είναι 0 και η Κυριακή είναι 6).
Η εβδομαδιαία περίληψη παράγεται από τη συνάρτηση `send_weekly_summary`. Αρχικά διαβάζει το log αρχείο και κρατάει μόνο τις εκτελέσεις των τελευταίων 7 ημερών:

```
def send_weekly_summary():
    entries = load_log()
    cutoff = (datetime.now() - timedelta(days=7)).isoformat()
    week = [e for e in entries if e.get('run_at', '') >= cutoff]
```

Στη συνέχεια τα δεδομένα ομαδοποιούνται σε τρεις κατηγορίες ανάλογα με το status: επιτυχίες, fallbacks στο LLM, και αποτυχίες. Για κάθε κατηγορία υπολογίζεται ο αριθμός και τα αντίστοιχα δεδομένα. Αν υπάρχουν αποτυχίες ή fallbacks αναγράφονται αναλυτικά μαζί με την αιτία τους. Στο θέμα του email μπαίνει ένα prefix ανάλογα με την κατάσταση: `[OK]` αν δεν υπήρξαν αποτυχίες και `[ALERT]` αν υπήρξαν.
Παρακάτω φαίνεται ένα παράδειγμα εβδομαδιαίας περίληψης όπου όλες οι εκτελέσεις της εβδομάδας ολοκληρώθηκαν χωρίς προβλήματα:

```
Subject: [OK] Weekly Fuel Prices - 7/7 processed

Weekly Fuel Price Pipeline Summary
Period: 2026-05-18 to 2026-05-24

Total runs: 7
  Successful (pdfplumber): 7
  Fallback (LLM):          0
  Failed:                  0

Everything ran smoothly this week.
```

Σε μια άλλη εβδομάδα όπου υπήρξαν προβλήματα η αναφορά είναι πιο αναλυτική και περιλαμβάνει τη συγκεκριμένη αιτία κάθε προβλήματος:

```
Subject: [ALERT] Weekly Fuel Prices - 6/7 processed

Weekly Fuel Price Pipeline Summary
Period: 2026-03-26 to 2026-04-02

Total runs: 7
  Successful (pdfplumber): 5
  Fallback (LLM):          1
  Failed:                  1

LLM Fallbacks:
  2026-03-28: only 38 entries (expected 51-52)

Failures:
  2026-04-02: Page fetch exception: Read timed out.

All warnings:
  2026-03-28: only 38 entries (expected 51-52)
  2026-03-28: 3 unmatched prefectures: ['...']
```

Με αυτή τη ροή ειδοποιήσεων κάθε σοβαρή αποτυχία γίνεται γνωστή μέσα σε λίγα λεπτά ενώ ταυτόχρονα ο διαχειριστής λαμβάνει μια εβδομαδιαία βεβαίωση ότι το σύστημα λειτουργεί σωστά. Αν δεν λάβει αναφορά την Κυριακή (πράγμα που θα σήμαινε ότι ούτε καν το cron job δεν τρέχει) αυτό από μόνο του είναι μια ένδειξη ότι κάτι πάει στραβά και χρειάζεται έλεγχος.

Κεφάλαιο 4

Ανάπτυξη API

4.1 Εισαγωγή

Όπως είδαμε στο πρώτο κεφάλαιο τα δεδομένα τιμών καυσίμων που δημοσιεύει το Υπουργείο Ανάπτυξης βρίσκονται στο πρώτο επίπεδο του σχήματος πέντε αστέρων. Είναι δημόσια αλλά πρακτικά μη αξιοποιήσιμα γιατί δίνονται σε μορφή PDF. Με το pipeline του Κεφαλαίου 3 καταφέραμε να μετατρέψουμε αυτή την πληροφορία σε δομημένα δεδομένα μέσα σε μια σχεσιακή βάση. Όμως η βάση από μόνη της δεν είναι προσβάσιμη από το ευρύ κοινό. Για να ολοκληρωθεί ο κύκλος και τα δεδομένα να γίνουν πραγματικά ανοιχτά χρειάζεται μια διεπαφή που να επιτρέπει σε οποιονδήποτε να τα ανακτήσει εύκολα από το διαδίκτυο. Αυτή τη διεπαφή προσφέρει το Web API που παρουσιάζεται σε αυτό το κεφάλαιο.

Τι είναι ένα Web API

Ο όρος API προέρχεται από το Application Programming Interface και αναφέρεται σε ένα σύνολο κανόνων που επιτρέπει σε δύο προγράμματα να επικοινωνούν μεταξύ τους. Όταν ένα API είναι διαθέσιμο μέσω του διαδικτύου ονομάζεται Web API. Στην ουσία πρόκειται για μια σύμβαση: ο πελάτης στέλνει ένα αίτημα σε μια συγκεκριμένη διεύθυνση, ο server το επεξεργάζεται και επιστρέφει μια απάντηση σε προκαθορισμένη μορφή. Σε αντίθεση με μια κανονική ιστοσελίδα που σχεδιάζεται για να διαβαστεί από άνθρωπο, ένα Web API σχεδιάζεται για να καταναλωθεί από άλλα προγράμματα.
Σήμερα σχεδόν κάθε υπηρεσία στο διαδίκτυο διαθέτει κάποιο Web API. Το Twitter, το Facebook, η Google και χιλιάδες άλλες εταιρείες προσφέρουν APIs που επιτρέπουν σε προγραμματιστές να φτιάχνουν εφαρμογές πάνω στις υπηρεσίες τους. Παράλληλα κυβερνητικοί φορείς και ερευνητικά ιδρύματα δημοσιεύουν τα δεδομένα τους μέσω APIs για να ενισχύσουν τη διαφάνεια και τη δυνατότητα επαναχρησιμοποίησης.

Η αρχιτεκτονική REST

Το επικρατέστερο μοντέλο σχεδίασης Web APIs σήμερα είναι το REST που προτάθηκε από τον Roy Fielding στη διδακτορική του διατριβή το 2000 [17]. Το REST είναι ακρωνύμιο του Representational State Transfer και ορίζει μια σειρά αρχιτεκτονικών αρχών που πρέπει να ακολουθεί ένα σύστημα για να θεωρείται RESTful. Δεν είναι ένα πρότυπο ή ένα framework αλλά

ένα στυλ σχεδίασης πάνω από το πρωτόκολλο HTTP.
Οι βασικές αρχές του REST είναι οι εξής. Η επικοινωνία γίνεται με μοντέλο πελάτη-εξυπηρετητή. Ο server είναι stateless, δηλαδή κάθε αίτημα περιέχει όλη την πληροφορία που χρειάζεται για να εξυπηρετηθεί και ο server δεν διατηρεί κατάσταση μεταξύ διαφορετικών αιτημάτων από τον ίδιο πελάτη. Οι απαντήσεις μπορούν να αποθηκευτούν στην cache για να μειωθεί η επιβάρυνση του server. Η διεπαφή είναι ομοιόμορφη με συγκεκριμένες συμβάσεις για το πώς εκφράζονται οι πόροι και πώς γίνονται οι ενέργειες πάνω τους. Τέλος το σύστημα μπορεί να αποτελείται από πολλαπλά στρώματα (proxies, load balancers, gateways) διαφανώς από τον πελάτη.
Σε ένα REST API κάθε αντικείμενο του συστήματος αντιμετωπίζεται ως πόρος (resource) που έχει μια μοναδική διεύθυνση. Για παράδειγμα στη δική μας περίπτωση οι νομοί είναι ένας πόρος με διεύθυνση `/api/v1/prefectures`, τα καύσιμα είναι ένας άλλος πόρος στη διεύθυνση `/api/v1/fuel-types` και ούτω καθεξής. Η ενέργεια που θα γίνει σε έναν πόρο εκφράζεται μέσω της HTTP μεθόδου του αιτήματος. Στο πρότυπο HTTP [18] ορίζονται μέθοδοι όπως `GET` για ανάκτηση, `POST` για δημιουργία, `PUT` για ενημέρωση και `DELETE` για διαγραφή. Επειδή το δικό μας API είναι read-only χρησιμοποιεί αποκλειστικά τη μέθοδο `GET`.
Η απάντηση κάθε αιτήματος συνοδεύεται από έναν κωδικό HTTP status που υποδεικνύει το αποτέλεσμα της επεξεργασίας. Οι κωδικοί `2xx` σημαίνουν επιτυχία (π.χ. `200 OK`), οι `4xx` σημαίνουν σφάλμα του πελάτη (π.χ. `400 Bad Request` ή `404 Not Found`) και οι `5xx` σφάλμα του server (π.χ. `500 Internal Server Error`). Η σωστή χρήση αυτών των κωδικών είναι βασικό χαρακτηριστικό ενός καλά σχεδιασμένου REST API.

Γιατί φτιάξαμε δικό μας API

Αφού το pipeline του Κεφαλαίου 3 έχει ήδη μετατρέψει τα PDF σε δομημένα δεδομένα στη βάση, γεννάται το ερώτημα γιατί χρειάζεται ένα API πάνω από αυτή. Η απάντηση συνδέεται άμεσα με το κίνητρο της εργασίας. Η βάση δεδομένων τρέχει στον server του τμήματος και δεν είναι προσβάσιμη απευθείας από το εξωτερικό. Αν κάποιος ερευνητής ή δημοσιογράφος θέλει να αξιοποιήσει τα δεδομένα, δεν μπορεί απλώς να συνδεθεί στη βάση. Χρειάζεται έναν τυποποιημένο, ασφαλή τρόπο πρόσβασης που να μην εκθέτει τη βάση σε κίνδυνο και να μην απαιτεί ειδικά εργαλεία από τη μεριά του χρήστη.
Το API λύνει αυτό το πρόβλημα. Με μερικά απλά HTTP αιτήματα οποιοσδήποτε μπορεί να πάρει τα δεδομένα που τον ενδιαφέρουν σε μορφή JSON, μια μορφή που υποστηρίζεται από κάθε γλώσσα προγραμματισμού [19]. Δεν χρειάζεται να ξέρει τη δομή της βάσης ούτε να γράψει SQL ερωτήματα. Επίσης το API μας δίνει τη δυνατότητα να ελέγξουμε την πρόσβαση, να επιβάλουμε rate limiting αν χρειαστεί στο μέλλον, και να αλλάξουμε τη δομή της βάσης χωρίς να σπάσουν οι υπάρχουσες εφαρμογές που χρησιμοποιούν τα δεδομένα.
Με το API τα δεδομένα μας ανεβαίνουν σταθερά στο τρίτο επίπεδο του σχήματος πέντε αστέρων του Berners-Lee αγγίζοντας και τα όρια του τέταρτου. Είναι προσβάσιμα μέσω σταθερών HTTP διευθύνσεων (URIs) σε μηχαναγνώσιμη μορφή ανοιχτού προτύπου και χωρίς κανέναν περιορισμό χρήσης. Το τέταρτο επίπεδο στην αυστηρή του εκδοχή απαιτεί χρήση τεχνολογιών Linked Data (RDF, SPARQL) που δεν υλοποιήθηκαν εδώ, αλλά οι σταθερές URIs που εκθέτει το REST API μας επιτρέπουν στους χρήστες "να δείχνουν στα δεδομένα μας" όπως ζητάει η αρχή του επιπέδου. Παράλληλα ικανοποιούνται και οι τέσσερις αρχές FAIR. Τα δεδομένα μπορούν να βρεθούν (Findable) μέσω της δημόσιας τεκμηρίωσης, είναι προσβάσιμα (Accessible) μέσω

HTTP, είναι διαλειτουργικά (Interoperable) επειδή χρησιμοποιούν JSON, και είναι επαναχρησιμοποιήσιμα (Reusable) χωρίς όρους.

Σχεδιαστικές αρχές του δικού μας API

Κατά τη σχεδίαση του API ακολουθήθηκαν συγκεκριμένες αρχές που εξυπηρετούν τόσο τον τελικό χρήστη όσο και την μελλοντική συντήρηση. Οι πιο σημαντικές είναι οι παρακάτω:

- **Versioning**. Όλα τα endpoints είναι κάτω από το prefix `/api/v1/`. Αυτό μας επιτρέπει στο μέλλον να εισάγουμε μια νέα έκδοση (`/api/v2/`) με διαφορετική δομή χωρίς να σπάσει η συμβατότητα με τους υπάρχοντες χρήστες.
- **JSON ως μοναδική μορφή απάντησης**. Όλα τα endpoints επιστρέφουν JSON με header `Content-Type: application/json`. Δεν υποστηρίζεται XML ή άλλη μορφή. Αυτή η απλότητα κρατάει το API εύκολο στη χρήση και στη συντήρηση.
- **Response envelope**. Κάθε απάντηση έχει σταθερή δομή της μορφής `{data, meta}`. Το πεδίο `data` περιέχει τα δεδομένα που ζητήθηκαν ενώ το `meta` περιέχει πληροφορίες όπως ο συνολικός αριθμός εγγραφών ή το χρονικό εύρος. Σε περίπτωση σφάλματος υπάρχει αντί για `data` ένα πεδίο `error` με κωδικοποιημένη περιγραφή του προβλήματος.
- **Public access χωρίς authentication**. Δεν απαιτείται API key ή login. Αυτή η επιλογή ευθυγραμμίζεται με τη φιλοσοφία των ανοιχτών δεδομένων και κάνει τη χρήση του API τετριμμένη. Αν στο μέλλον προκύψει ανάγκη για αυθεντικοποίηση μπορεί να προστεθεί χωρίς να αλλάξει η υπόλοιπη υλοποίηση.
- **CORS ενεργοποιημένο για όλες τις προελεύσεις**. Το API επιτρέπει αιτήματα από οποιοδήποτε domain μέσω του header `Access-Control-Allow-Origin: *`. Έτσι μπορεί να χρησιμοποιηθεί απευθείας από JavaScript σε οποιαδήποτε ιστοσελίδα τρίτου χωρίς να μπλοκάρει ο browser το αίτημα.
- **Pretty URLs**. Τα endpoints έχουν φιλικές διευθύνσεις χωρίς query strings που να φαίνονται τεχνικά (π.χ. /api/v1/prefectures αντί για /api/index.php?route=prefectures). Αυτό κάνει το API πιο ευκολανάγνωστο και πιο επαγγελματικό.

Stack υλοποίησης

Το API υλοποιήθηκε σε PHP 8 και τρέχει πάνω από τον Apache HTTP Server στον server του τμήματος. Η σύνδεση με τη MariaDB γίνεται μέσω της επέκτασης PHP Data Objects (PDO) που είναι ο επίσημος τρόπος της PHP για ασφαλή ερωτήματα με prepared statements. Δεν χρησιμοποιήθηκε κάποιο PHP framework όπως το Laravel ή το Symfony. Η κλίμακα του project δεν δικαιολογεί την πολυπλοκότητα ενός framework και η vanilla PHP κάνει τον κώδικα πιο εύκολο να διαβαστεί και να συντηρηθεί. Οι λεπτομέρειες της υλοποίησης (front controller, routing, helpers, ρυθμίσεις Apache) αναλύονται στο Κεφάλαιο 5 μαζί με την υπόλοιπη εφαρμογή ιστού. Στις επόμενες ενότητες παρουσιάζεται κάθε endpoint ξεχωριστά μαζί με τις παραμέτρους του και παραδείγματα αιτημάτων και απαντήσεων.

4.2 GET /api/v1/

Το πρώτο endpoint του API είναι αυτό που αντιστοιχεί στη ρίζα της διεπαφής. Σκοπός του είναι να λειτουργεί ως σημείο εκκίνησης για όποιον επισκέπτεται το API για πρώτη φορά. Όταν κάποιος πληκτρολογεί απλώς `/api/v1/` χωρίς να ξέρει τι ακριβώς προσφέρει η υπηρεσία πρέπει να παίρνει μια περιγραφή του τι μπορεί να κάνει. Αυτή η ιδέα ονομάζεται API discovery και είναι μια βασική συνιστώμενη πρακτική σε REST APIs.
Το endpoint δεν δέχεται καμία παράμετρο. Ένα απλό `GET` αίτημα αρκεί για να επιστρέψει το API έναν χάρτη με όλα όσα προσφέρει. Πέρα από τη λίστα των endpoints, η απάντηση περιλαμβάνει και μερικά γενικά στοιχεία για το ίδιο το API: το όνομα της υπηρεσίας, την τρέχουσα έκδοση και μια σύντομη περιγραφή. Επιπλέον περιλαμβάνεται ένα πεδίο `data_coverage` με στατιστικά για τη βάση: η παλαιότερη και η πιο πρόσφατη ημερομηνία για την οποία υπάρχουν δεδομένα, καθώς και ο συνολικός αριθμός εγγραφών. Αυτή η πληροφορία βοηθάει τον χρήστη να καταλάβει αμέσως τι όγκο και τι χρονικό εύρος δεδομένων μπορεί να αναμένει.

Παράδειγμα αιτήματος και απάντησης

Μια απλή κλήση από τη γραμμή εντολών με το εργαλείο `curl` επιστρέφει ολόκληρο τον χάρτη του API:

```
$ curl https://nireas.iee.ihu.gr/fuel/api/v1/
```

Το API απαντά με JSON της παρακάτω μορφής:

```
{
  "data": {
    "name": "Greek Fuel Prices API",
    "version": "v1",
    "description": "Greek daily fuel prices per prefecture, ...",
    "endpoints": {
      "root": {
        "method": "GET",
        "path": "/v1/",
        "description": "This page"
      },
      "prefectures": {
        "method": "GET",
        "path": "/v1/prefectures",
        "description": "List all prefectures ..."
      },
      "fuel_types": {
        "method": "GET",
        "path": "/v1/fuel-types",
        "description": "List all tracked fuel types"
      },
      "prices": {
```

```
        "method": "GET",
        "path": "/v1/prices",
        "description": "Price time series ...",
        "query_params": {
          "prefecture_id": "integer, required",
          "fuel_type_id":  "integer, required",
          "from":          "YYYY-MM-DD, required",
          "to":            "YYYY-MM-DD, required"
        }
      },
      "prices_latest": {
        "method": "GET",
        "path": "/v1/prices/latest",
        "description": "Most recent bulletin: all prices ..."
      }
    },
    "data_coverage": {
      "earliest": "2017-01-02",
      "latest":   "2026-05-23",
      "total_rows": 458291,
      "unit": "EUR/L"
    }
  },
  "meta": {
    "version": "v1"
  }
}
```

Με αυτή την απάντηση ένας προγραμματιστής μπορεί να πάρει μια πλήρη εικόνα της λειτουργίας του API χωρίς να ανατρέξει σε ξεχωριστή τεκμηρίωση. Βλέπει αμέσως ποια endpoints υπάρχουν, πώς καλούνται, ποιες παραμέτρους δέχονται και τι όγκο δεδομένων κρύβεται από πίσω. Στις επόμενες ενότητες θα εξετάσουμε ένα προς ένα τα υπόλοιπα endpoints με την ίδια λογική.

4.3 GET /api/v1/prefectures

Το συγκεκριμένο endpoint επιστρέφει τη λίστα με όλους τους νομούς που υπάρχουν στη βάση δεδομένων. Είναι ένα από τα ποιο απλά endpoints αλλά εξυπηρετεί έναν κρίσιμο σκοπό. Όπως θα δούμε στα επόμενα endpoints για να ζητήσει κανείς τιμές για έναν νομό χρειάζεται το ID του και όχι το όνομά του. Αυτό το endpoint δίνει στους χρήστες τη δυνατότητα να ανακαλύψουν τα διαθέσιμα IDs μαζί με τα αντίστοιχα ονόματα ώστε να καταρτίσουν τα ερωτήματά τους.
Η λίστα περιλαμβάνει τους 51 νομούς της Ελλάδας καθώς και μια ειδική εγγραφή με το όνομα “ΠΑΝΕΛΛΗΝΙΟΣ ΣΤΑΘΜΙΣΜΕΝΟΣ Μ.Ο.” που αντιστοιχεί στον εθνικό σταθμισμένο μέσο όρο και θεωρείται από τη βάση ως ένας ακόμα ”νομός”. Με αυτόν τον τρόπο ο μέσος όρος μπορεί να ζητηθεί με τον ίδιο τρόπο που ζητείται ένας οποιοσδήποτε νομός.

Παράδειγμα απάντησης

Μια κλήση στο endpoint επιστρέφει την παρακάτω απάντηση (παρουσιάζεται περικομμένη για συντομία):

```
{
  "data": [
    { "id": 1, "name": "ΝΟΜΟΣ ΑΤΤΙΚΗΣ" },
    { "id": 2, "name": "ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ" },
    { "id": 3, "name": "ΝΟΜΟΣ ΑΡΓΟΛΙΔΟΣ" },
    ...
    { "id": 51, "name": "ΝΟΜΟΣ ΧΙΟΥ" },
    { "id": 52, "name": "ΠΑΝΕΛΛΗΝΙΟΣ ΣΤΑΘΜΙΣΜΕΝΟΣ Μ.Ο." }
  ],
  "meta": { "count": 52 }
}
```

Το πεδίο `count` στο `meta` δίνει στον χρήστη τον συνολικό αριθμό των εγγραφών χωρίς να χρειάζεται να τις μετρήσει χειροκίνητα.

4.4 GET /api/v1/fuel-types

Παρόμοιο σε λογική με το προηγούμενο, αυτό το endpoint επιστρέφει τη λίστα των τύπων καυσίμων που παρακολουθεί το σύστημα. Όπως και στους νομούς, ο χρήστης χρειάζεται το ID ενός καυσίμου για να το χρησιμοποιήσει σε ερωτήματα τιμών. Η λίστα περιέχει πέντε εγγραφές που αντιστοιχούν στα καύσιμα που δημοσιεύει το Υπουργείο: Αμόλυβδη 95 οκτανίων, Αμόλυβδη 100 οκτανίων, Diesel Κίνησης, Υγραέριο Κίνησης (Autogas) και Diesel Θέρμανσης Κατ᾽ οίκον. Σημειώνεται ότι το Diesel Θέρμανσης Κατ᾽ οίκον είναι εποχικό προϊόν και πωλείται μόνο τους χειμερινούς μήνες. Σε δελτία ανοιξιάτικων και καλοκαιρινών μηνών οι τιμές για αυτό το καύσιμο δεν υπάρχουν αλλά η ίδια η εγγραφή στον πίνακα `fuel_types` παραμένει σταθερή.

Παράδειγμα απάντησης

Η απάντηση είναι μια λίστα από πέντε αντικείμενα:

```
{
  "data": [
    { "id": 1, "name": "Αμόλυβδη 95 οκτ." },
    { "id": 2, "name": "Αμόλυβδη 100 οκτ." },
    { "id": 3, "name": "Diesel Κίνησης" },
    { "id": 4, "name": "Υγραέριο κίνησης (Autogas)" },
    { "id": 5, "name": "Diesel Θέρμανσης Κατ΄ οίκον" }
  ],
  "meta": { "count": 5 }
}
```

4.5 GET /api/v1/prices

Το endpoint `prices` είναι το βασικό εργαλείο ανάκτησης ιστορικών δεδομένων. Επιστρέφει την ιστορική σειρά τιμών για έναν συγκεκριμένο συνδυασμό νομού και καυσίμου μέσα σε ένα συγκεκριμένο χρονικό διάστημα. Είναι το endpoint που χρησιμοποιεί κυρίως η εφαρμογή ιστού για να φτιάχνει τα διαγράμματα της εξέλιξης των τιμών.

Παράμετροι

Το endpoint αυτό λειτουργεί με δύο τρόπους. Στη βασική του χρήση δέχεται τέσσερις παραμέτρους που περνούν στο URL ως query string:

- `prefecture_id`: ακέραιος, το ID του νομού για τον οποίο ζητούνται οι τιμές
- `fuel_type_id`: ακέραιος, το ID του καυσίμου
- `from`: ημερομηνία σε μορφή `EEEE-MM-HH`, η αρχή του διαστήματος
- `to`: ημερομηνία σε μορφή `EEEE-MM-HH`, το τέλος του διαστήματος

Οι τέσσερις παράμετροι λειτουργούν ως σύνολο. Αν δοθεί έστω μία από αυτές τότε απαιτούνται και οι τέσσερις. Αν λείπει κάποια το API επιστρέφει σφάλμα `400 Bad Request` με σαφές μήνυμα για το τι έλειπε. Το ίδιο γίνεται αν οι τιμές δεν έχουν την αναμενόμενη μορφή (π.χ. μια ημερομηνία σε λάθος format). Επιπλέον γίνεται έλεγχος ότι το `from` είναι πριν ή ίσο με το `to`. Τέλος αν το `prefecture_id` ή το `fuel_type_id` δεν αντιστοιχεί σε υπαρκτή εγγραφή της βάσης, επιστρέφεται σφάλμα `404 Not Found` αντί για μια σιωπηρή κενή απάντηση που θα μπέρδευε τον χρήστη.
Ο δεύτερος τρόπος είναι να μην δοθεί καμία παράμετρος. Σε αυτή την περίπτωση το endpoint επιστρέφει ολόκληρο το dataset, δηλαδή κάθε τιμή για κάθε νομό και κάθε καύσιμο σε όλες τις ημερομηνίες. Επειδή εδώ δεν υπάρχει συγκεκριμένος νομός ή καύσιμο, κάθε γραμμή της απάντησης περιέχει το ID και το όνομα τόσο του νομού όσο και του καυσίμου ώστε να είναι αυτόνομα αναγνωρίσιμη. Πρέπει να σημειωθεί ότι αυτή η κλήση επιστρέφει εκατοντάδες χιλιάδες εγγραφές και παράγει αρκετά μεγάλη απάντηση, οπότε χρησιμοποιείται κυρίως για μαζική λήψη του συνόλου των δεδομένων και όχι σε καθημερινά αιτήματα.
Τα αποτελέσματα και στους δύο τρόπους επιστρέφονται ταξινομημένα κατά ημερομηνία από το παλαιότερο προς το νεότερο ώστε να μπορούν να γίνουν plot απευθείας σε ένα διάγραμμα χωρίς επιπλέον ταξινόμηση από τη μεριά του πελάτη.

Παράδειγμα αιτήματος και απάντησης

Για να πάρει κανείς την εξέλιξη της τιμής της Αμόλυβδης 95 στον Νομό Θεσσαλονίκης τον Ιανουάριο του 2025, η κλήση είναι:

```
$ curl "https://nireas.iee.ihu.gr/fuel/api/v1/prices\
?prefecture_id=19&fuel_type_id=1\
&from=2025-01-01&to=2025-01-31"
```

Η απάντηση είναι μια λίστα από αντικείμενα `{date, price}`, συνοδευόμενα από μεταδεδομένα που επαναλαμβάνουν τις παραμέτρους του ερωτήματος:

```
{
  "data": [
    { "date": "2025-01-02", "price": 1.743 },
    { "date": "2025-01-03", "price": 1.745 },
    { "date": "2025-01-04", "price": 1.751 },
    ...
    { "date": "2025-01-31", "price": 1.768 }
  ],
  "meta": {
    "count": 30,
    "unit":  "EUR/L",
    "prefecture": { "id": 19, "name": "ΝΟΜΟΣ ΘΕΣΣΑΛΟΝΙΚΗΣ" },
    "fuel_type":  { "id": 1,  "name": "Αμόλυβδη 95 οκτ." },
    "from": "2025-01-01",
    "to":   "2025-01-31"
  }
}
```

Με αυτή τη μορφή απάντησης ο πελάτης έχει όλα όσα χρειάζεται για να φτιάξει το διάγραμμα: τη λίστα σημείων προς γραφική απεικόνιση και τα μεταδεδομένα για τους τίτλους.
Στη δεύτερη μορφή χρήσης, χωρίς καμία παράμετρο, η κλήση είναι απλώς:

```
$ curl https://nireas.iee.ihu.gr/fuel/api/v1/prices
```

Η απάντηση είναι μια λίστα από εγγραφές όπου η καθεμία περιέχει τον νομό, το καύσιμο, την ημερομηνία και την τιμή:

```
{
  "data": [
    {
      "prefecture": { "id": 1, "name": "ΝΟΜΟΣ ΑΤΤΙΚΗΣ" },
      "fuel_type":  { "id": 1, "name": "Αμόλυβδη 95 οκτ." },
      "date": "2017-01-02",
      "price": 1.412
    },
    ...
  ],
  "meta": { "count": 458291, "unit": "EUR/L" }
}
```

4.6 GET /api/v1/prices/latest

Αυτό το endpoint επιστρέφει όλες τις τιμές του πιο πρόσφατου διαθέσιμου δελτίου. Είναι σχεδιασμένο για χρήσεις όπου ο πελάτης θέλει μια στιγμιαία εικόνα της κατάστασης χωρίς να γνω-

ρίζει από πριν ποια είναι η πιο πρόσφατη ημερομηνία. Στην εφαρμογή ιστού χρησιμοποιείται για να γεμίσουν οι κάρτες στην αρχική σελίδα που δείχνουν τις τρέχουσες τιμές των βασικών καυσίμων.
Σε αντίθεση με τα άλλα endpoints δεν δέχεται καμία παράμετρο. Η ίδια η βάση καθορίζει ποια είναι η πιο πρόσφατη ημερομηνία.
Στο εσωτερικό η απάντηση χτίζεται σε δύο βήματα. Πρώτα εντοπίζεται η πιο πρόσφατη ημερομηνία και έπειτα ανακτώνται όλες οι εγγραφές αυτής της ημερομηνίας μαζί με τα ονόματα νομών και καυσίμων. Τα αποτελέσματα ομαδοποιούνται σε δομή `prefecture` → `{fuel: price}` ώστε ο πελάτης να μπορεί εύκολα να εντοπίσει όλες τις τιμές ενός νομού χωρίς να ψάχνει σε επίπεδη λίστα.

Παράδειγμα απάντησης

```
{
  "data": {
    "date": "2026-05-23",
    "entries": [
      {
        "prefecture": { "id": 1, "name": "ΝΟΜΟΣ ΑΤΤΙΚΗΣ" },
        "prices": {
          "Αμόλυβδη 95 οκτ.":          1.785,
          "Αμόλυβδη 100 οκτ.":         2.034,
          "Diesel Κίνησης":            1.612,
          "Υγραέριο κίνησης (Autogas)": 1.045
        }
      },
      {
        "prefecture": { "id": 2, "name": "ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ
                                                ΑΚΑΡΝΑΝΙΑΣ" },
        "prices": {
          "Αμόλυβδη 95 οκτ.":  1.791,
          "Αμόλυβδη 100 οκτ.": 2.038,
          "Diesel Κίνησης":    1.618
        }
      },
      ...
    ]
  },
  "meta": { "count": 52, "unit": "EUR/L" }
}
```

Παρατηρούμε ότι σε ορισμένες εγγραφές κάποια καύσιμα μπορεί να λείπουν. Αυτό συμβαίνει όταν στο συγκεκριμένο δελτίο δεν υπήρχε τιμή για το καύσιμο σε αυτόν τον νομό (π.χ. όταν το Diesel Θέρμανσης λείπει τους καλοκαιρινούς μήνες). Η αναπαράσταση `{fuel: price}` κάνει αυτή τη συμπεριφορά φυσική: αν δεν υπάρχει η τιμή απλώς λείπει το αντίστοιχο κλειδί.

4.7 Διαχείριση Σφαλμάτων

Όταν κάτι πάει στραβά σε ένα αίτημα το API επιστρέφει μια απάντηση που ακολουθεί σταθερή δομή. Αυτό είναι σημαντικό γιατί επιτρέπει στους πελάτες να γράψουν μία γενική λογική χειρισμού σφαλμάτων αντί να ελέγχουν κάθε φορά διαφορετική μορφή. Η δομή της απάντησης σφάλματος είναι η εξής:

```
{
  "error": {
    "code":    "<machine-readable code>",
    "message": "<human-readable message>",
    "details": { ... }
  },
  "meta": { "status": <http-status-code> }
}
```

Το πεδίο `code` είναι ένας σταθερός κωδικός που το πρόγραμμα-πελάτης μπορεί να ελέγξει εύκολα. Το `message` είναι μια περιγραφή στα αγγλικά κατάλληλη για εμφάνιση σε αναπτυξιακή κονσόλα. Το προαιρετικό `details` περιέχει επιπλέον πληροφορίες για το σφάλμα όταν χρειάζεται.
Παρακάτω συνοψίζονται οι κωδικοί που χρησιμοποιεί το API.

Status	Code	Πότε εμφανίζεται
400	`missing_param`	Λείπει υποχρεωτική παράμετρος
400	`invalid_param`	Παράμετρος με λάθος τύπο ή μορφή
400	`invalid_range`	Το `from` είναι μετά το `to`
404	`not_found`	Endpoint, νομός ή καύσιμο δεν υπάρχει
404	`no_data`	Η βάση δεν περιέχει ακόμα δεδομένα
405	`method_not_allowed`	Χρήση HTTP μεθόδου εκτός από `GET`
500	`internal_error`	Απρόσμενο σφάλμα στον server

Πίνακας 4.1: Κωδικοί σφαλμάτων του API.

Ένα παράδειγμα: αν κάποιος καλέσει το `/api/v1/prices` χωρίς να δώσει την παράμετρο `prefecture_id` το API απαντά με κωδικό HTTP 400 και σώμα:

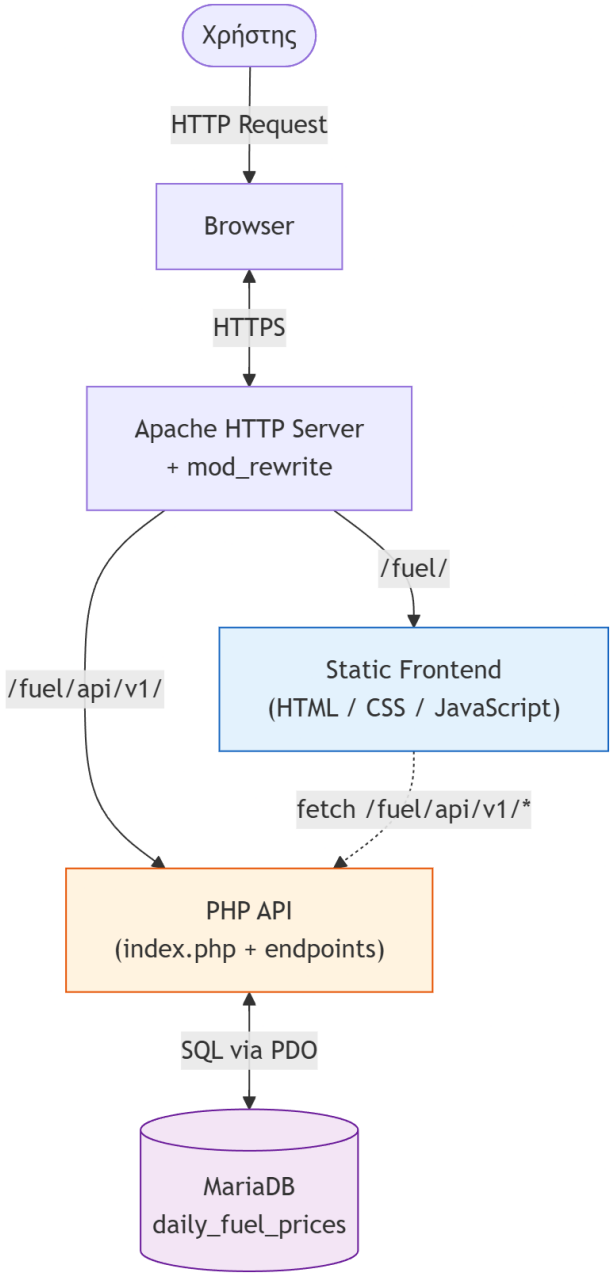
```
{
  "error": {
    "code":    "missing_param",
    "message": "Missing required parameter: prefecture_id"
  },
  "meta": { "status": 400 }
}
```

Κεφάλαιο 5

Ανάπτυξη και Παρουσίαση της Εφαρμογής Ιστού

5.1 Αρχιτεκτονική

Η εφαρμογή ιστού ακολουθεί τη γνώριμη αρχιτεκτονική πελάτη-εξυπηρετητή με σαφή διαχωρισμό μεταξύ frontend και backend. Ο τελικός χρήστης αλληλεπιδρά με την εφαρμογή μέσω του φυλλομετρητή, ο οποίος ζητάει στατικά αρχεία (HTML, CSS, JavaScript) και στη συνέχεια εκτελεί ασύγχρονα αιτήματα στο API για να ανακτήσει δεδομένα. Όλη η οπτικοποίηση και η διαδραστικότητα γίνεται μέσα στον φυλλομετρητή ενώ ο server περιορίζεται στο να εξυπηρετεί αρχεία και να απαντάει σε ερωτήματα δεδομένων.



Σχήμα 5.1: Αρχιτεκτονική της εφαρμογής ιστού.

Στο επάνω μέρος του διαγράμματος βρίσκεται ο χρήστης που αλληλεπιδρά μέσω του φυλλομετρητή του. Όλα τα αιτήματα φτάνουν αρχικά στον Apache HTTP Server ο οποίος αναλαμβάνει τη δρομολόγηση. Με βάση τη διεύθυνση που ζητήθηκε, ο Apache αποφασίζει αν θα σερβίρει στατικό αρχείο από το frontend ή θα προωθήσει το αίτημα στο PHP API. Διευθύνσεις της μορφής `/fuel/` εξυπηρετούνται από τα στατικά αρχεία ενώ διευθύνσεις της μορφής `/fuel/api/v1/` προωθούνται μέσω του front controller του API.
Το frontend τρέχει εξ ολοκλήρου στον φυλλομετρητή και δεν επικοινωνεί με τη βάση. Όταν χρειαστεί δεδομένα κάνει `fetch` κλήσεις στο API ακριβώς όπως θα έκανε οποιοσδήποτε τρί-

τος. Αυτή η σχεδίαση είναι σκόπιμη γιατί κρατάει το API ως μοναδικό σημείο πρόσβασης στα δεδομένα και διασφαλίζει ότι το API δοκιμάζεται από την ίδια την εφαρμογή που ξέρει καλύτερα από τον καθένα τι χρειάζεται. Το API με τη σειρά του είναι το μόνο στοιχείο που επικοινωνεί με τη MariaDB μέσω του PDO με παραμετροποιημένα ερωτήματα.
Για να γίνει πιο συγκεκριμένη η ροή ας δούμε τι συμβαίνει όταν ο χρήστης επιλέγει έναν νομό και ένα καύσιμο από τα dropdowns της εφαρμογής. Πρώτα γίνεται κλικ στο dropdown και επιλέγεται ο συνδυασμός. Η JavaScript στο frontend συνθέτει το URL του αιτήματος και καλεί το fetch('/fuel/api/v1/prices?prefecture_id=19&fuel_type_id=1&from=...&to=...'). Ο Apache λαμβάνει το αίτημα, ταιριάζει το prefix `/fuel/api/` και το προωθεί στο PHP API. Το `index.php` αναγνωρίζει τη διαδρομή `v1/prices`, εκτελεί το αντίστοιχο endpoint, ζητάει τα δεδομένα από τη MariaDB και επιστρέφει JSON. Ο Apache μεταφέρει την απάντηση πίσω στον φυλλομετρητή. Η JavaScript παραλαμβάνει τα δεδομένα, τα δίνει στο Chart.js και ενημερώνει το διάγραμμα. Όλη αυτή η ροή ολοκληρώνεται συνήθως σε λιγότερο από ένα δευτερόλεπτο.
Ένα σημαντικό σχεδιαστικό χαρακτηριστικό είναι ότι δεν υπάρχει build step στο frontend. Δεν χρησιμοποιείται κάποιο bundler όπως το webpack ή το Vite και τα αρχεία ανεβαίνουν στον server έτσι όπως γράφονται. Αυτή η επιλογή βασίστηκε στη σχετικά μικρή κλίμακα της εφαρμογής και κρατάει το deployment όσο πιο απλό γίνεται. Η αλλαγή ενός αρχείου JavaScript ή CSS αρκεί να ανέβει στον server και είναι αμέσως διαθέσιμη χωρίς compilation.

5.2 Υλοποίηση του Backend

Το backend της εφαρμογής είναι ο κώδικας που τρέχει στον server και εκθέτει τα δεδομένα της βάσης μέσω του API που περιγράφηκε στο Κεφάλαιο 4. Στην ενότητα αυτή παρουσιάζονται οι λεπτομέρειες υλοποίησης: η δομή του project, η αρχιτεκτονική του front controller, τα helper functions, η σύνδεση με τη βάση και οι ρυθμίσεις του Apache για ασφάλεια και routing.

Δομή του project

Όλος ο PHP κώδικας του backend βρίσκεται στον φάκελο `api/` του project. Η δομή είναι η εξής:

```
api/
├── .htaccess           # Apache config (routing + protection)
├── index.php           # Front controller
├── config.php          # .env loading + PDO connection
├── helpers.php         # Response and validation helpers
└── endpoints/
    ├── root.php        # GET /api/v1/
    ├── prefectures.php # GET /api/v1/prefectures
    ├── fuel_types.php  # GET /api/v1/fuel-types
    ├── prices.php      # GET /api/v1/prices
    └── prices_latest.php  # GET /api/v1/prices/latest
```

Η οργάνωση αυτή ακολουθεί το πρότυπο Front Controller που είδαμε στην εισαγωγή του προηγούμενου κεφαλαίου. Ένα μόνο σημείο εισόδου (`index.php`) λαμβάνει όλα τα αιτήματα και

τα δρομολογεί στο κατάλληλο endpoint αρχείο. Η κοινή λογική (CORS, error handling, σύνδεση με τη βάση, validation παραμέτρων) ζει σε ξεχωριστά αρχεία ώστε τα endpoints να μένουν μικρά και εστιασμένα.

Front Controller: index.php

Το `index.php` έχει τέσσερις βασικές ευθύνες: να ρυθμίζει τα CORS headers, να χειρίζεται το OPTIONS preflight του browser, να εγκαθιστά έναν καθολικό exception handler και τέλος να δρομολογεί το αίτημα στο σωστό endpoint.
Στην αρχή του αρχείου ορίζονται τα CORS headers που επιτρέπουν αιτήματα από οποιοδήποτε domain:

```
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET, OPTIONS');
header('Access-Control-Allow-Headers: Content-Type');

if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    http_response_code(204);
    exit;
}

if ($_SERVER['REQUEST_METHOD'] !== 'GET') {
    error_response('method_not_allowed',
                   'Only GET requests are supported', 405);
}
```

Το OPTIONS preflight είναι ένας έλεγχος που στέλνει ο browser πριν από συγκεκριμένα cross-origin αιτήματα. Απαντάμε με 204 No Content χωρίς σώμα όπως απαιτεί το πρότυπο CORS. Παράλληλα απορρίπτονται όλες οι άλλες HTTP μέθοδοι πέρα από `GET` αφού το API είναι read-only.
Στη συνέχεια εγκαθίσταται ένας global exception handler που πιάνει οποιοδήποτε exception δεν χειρίστηκε αλλού:

```
set_exception_handler(function (Throwable $e) {
    error_log($e->getMessage() . "\n" . $e->getTraceAsString());
    $details = [];
    if (getenv('DEBUG') === '1') {
        $details = [
            'exception' => get_class($e),
            'message' => $e->getMessage(),
            'file'   => basename($e->getFile()) . ':' . $e->getLine(),
        ];
    }
    error_response('internal_error',
                   'An unexpected error occurred', 500, $details);
});
```

Όλες οι λεπτομέρειες του σφάλματος καταγράφονται στο log του Apache για ανάλυση από τον διαχειριστή. Στον τελικό χρήστη όμως επιστρέφεται ένα γενικό μήνυμα ώστε να μην διαρρεύσει εσωτερική πληροφορία (paths αρχείων, ονόματα κλάσεων κ.λπ.). Σε περιβάλλον ανάπτυξης μπορεί να ενεργοποιηθεί `DEBUG=1` στο `.env` και τότε επιστρέφονται και αυτές οι πληροφορίες για εύκολο troubleshooting.
Το τελευταίο και πιο σημαντικό κομμάτι είναι η δρομολόγηση. Η διαδρομή φτάνει στον front controller ως παράμετρος `_route` χάρη στο rewrite που κάνει ο Apache. Αφού γίνει normalize και αφαιρεθεί το prefix έκδοσης (`v1/`) γίνεται lookup στον route table:

```
$raw = $_GET['_route'] ?? '';
$route = trim($raw, '/');
// strip "v1/" prefix...

$routes = [
    ''              => 'root.php',
    'prefectures'   => 'prefectures.php',
    'fuel-types'    => 'fuel_types.php',
    'prices'        => 'prices.php',
    'prices/latest' => 'prices_latest.php',
];

if (!array_key_exists($route, $routes)) {
    error_response('not_found',
                   "Endpoint not found: /v1/$route", 404);
}

require __DIR__ . '/endpoints/' . $routes[$route];
```

Ο route table είναι το μόνο σημείο όπου ορίζονται όλα τα endpoints. Η προσθήκη ενός νέου endpoint είναι μια απλή γραμμή στον πίνακα και ένα νέο αρχείο στον φάκελο `endpoints/`. Αν η διαδρομή δεν αντιστοιχεί σε κανένα γνωστό endpoint επιστρέφεται `404` με σαφές μήνυμα και λίστα των διαθέσιμων routes.

Configuration: config.php

Το `config.php` κάνει δύο πράγματα. Πρώτον φορτώνει τις μεταβλητές περιβάλλοντος από το `.env`. Δεύτερον παρέχει μια singleton σύνδεση με τη βάση δεδομένων μέσω PDO.
Η φόρτωση του `.env` γίνεται με δικό μας mini parser αντί για κάποια εξωτερική βιβλιοθήκη. Η PHP δεν έχει built-in υποστήριξη για `.env` αρχεία οπότε γράφτηκε μια απλή συνάρτηση `load_env` που διαβάζει γραμμή προς γραμμή και ορίζει κάθε ζευγάρι ως environment variable. Παραλείπονται σχόλια, κενές γραμμές και αφαιρούνται προαιρετικά εισαγωγικά γύρω από τις τιμές.
Σημαντικό σημείο: το `.env` αναζητείται πρώτα στον γονικό φάκελο του project και μετά στον φάκελο του API. Σε production τοποθετείται έξω από τη ρίζα του webroot ώστε ο Apache να μην μπορεί φυσικά να το σερβίρει ακόμα και αν χαλάσουν οι κανόνες του `.htaccess`.

```
foreach ([__DIR__ . '/../../.env',
__DIR__ . '/../.env'] as $candidate) {
    if (is_readable($candidate)) {
        load_env($candidate);
        break;
    }
}
```

Η σύνδεση με τη βάση γίνεται μέσω της `get_db()` που υλοποιεί το pattern του singleton. Η πρώτη κλήση δημιουργεί τη σύνδεση, οι επόμενες επιστρέφουν την ίδια ώστε να μην ανοίγουμε πολλαπλές συνδέσεις μέσα στην ίδια εκτέλεση:

```
function get_db(): PDO
{
    static $pdo = null;
    if ($pdo !== null) {
        return $pdo;
    }

    $host = getenv('DB_HOST') ?: 'localhost';
    $port = getenv('DB_PORT') ?: '3306';
    $name = getenv('DB_NAME') ?: '';
    $user = getenv('DB_USER') ?: '';
    $pass = getenv('DB_PASSWORD') ?: '';

    $dsn = "mysql:host=$host;port=$port;
    dbname=$name;charset=utf8mb4";
    $pdo = new PDO($dsn, $user, $pass, [
        PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        PDO::ATTR_EMULATE_PREPARES   => false,
    ]);
    return $pdo;
}
```

Τρεις ρυθμίσεις του PDO αξίζει να τονιστούν. Το `ERRMODE_EXCEPTION` κάνει τη βιβλιοθήκη να πετάει εξαιρέσεις σε σφάλματα της βάσης αντί για σιωπηρή αποτυχία. Το `FETCH_ASSOC` κάνει τα αποτελέσματα να επιστρέφονται ως associative arrays που είναι η φυσική μορφή για JSON output. Τέλος το `EMULATE_PREPARES => false` εξασφαλίζει ότι χρησιμοποιούνται πραγματικά prepared statements στη βάση και όχι emulated από την PHP, κάτι που είναι πιο ασφαλές και πιο γρήγορο.

Helpers: helpers.php

Στο `helpers.php` βρίσκονται οι κοινές συναρτήσεις που χρησιμοποιούν τα endpoints. Είναι ουσιαστικά η εργαλειοθήκη που εξασφαλίζει ομοιόμορφη συμπεριφορά σε όλο το API.

Η πιο σημαντική είναι η `json_response` που τυποποιεί την επιτυχημένη απάντηση. Όλα τα endpoints περνούν από αυτή τη συνάρτηση οπότε το envelope `{data, meta}` είναι εγγυημένα σταθερό:

```
function json_response(mixed $data, array $meta = [],
                       int $status = 200): never
{
    http_response_code($status);
    header('Content-Type: application/json; charset=utf-8');
    echo json_encode(
        ['data' => $data, 'meta' => $meta],
        JSON_UNESCAPED_UNICODE
            | JSON_UNESCAPED_SLASHES
            | JSON_PRETTY_PRINT
    );
    exit;
}
```

Το flag `JSON_UNESCAPED_UNICODE` είναι σημαντικό γιατί διατηρεί τους ελληνικούς χαρακτήρες ως κανονικά γράμματα αντί να τους κωδικοποιεί ως `\u` sequences που είναι δυσανάγνωστα. Η αντίστοιχη `error_response` έχει παρόμοια δομή αλλά παράγει envelope με `{error, meta}`.
Άλλες δύο συναρτήσεις που είδαμε ήδη να χρησιμοποιούνται στο endpoint `prices` είναι οι `require_int` και `require_date`. Διαβάζουν μια παράμετρο από το query string, την επικυρώνουν, και αν δεν είναι έγκυρη απαντάνε αυτόματα με `400` χωρίς ο κώδικας του endpoint να χρειάζεται να ασχοληθεί:

```
function require_int(string $name): int
{
    $raw = $_GET[$name] ?? null;
    if ($raw === null || $raw === '') {
        error_response('missing_param',
                       "Missing required parameter: $name");
    }
    if (!ctype_digit((string) $raw) || (int) $raw <= 0) {
        error_response('invalid_param',
                "Parameter '$name' must be a positive integer");
    }
    return (int) $raw;
}
```

Με αυτή τη σχεδίαση κάθε endpoint γράφεται απλά $id = require_int('prefecture_id'); χωρίς να επαναλαμβάνει τη λογική επικύρωσης. Παρόμοια λογική ακολουθούν οι `fetch_prefecture` και `fetch_fuel_type` που επιστρέφουν εγγραφές της βάσης από το ID τους και απαντούν αυτόματα με `404` αν δεν υπάρχουν.

Apache Security

Για να λειτουργήσουν τα όλα τα παραπάνω χρειάζονται ρυθμίσεις στον Apache. Η πιο σημαντική είναι το `mod_rewrite` που υλοποιεί τα Pretty URLs. Στον φάκελο του API υπάρχει το παρακάτω `.htaccess`:

```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php?_route=$1 [QSA,L]
```

Οι δύο `RewriteCond` εξασφαλίζουν ότι αν το request αντιστοιχεί σε υπαρκτό αρχείο ή φάκελο σερβίρεται κανονικά. Διαφορετικά ανακατευθύνεται στο `index.php` με την αρχική διαδρομή ως παράμετρο `_route`. Το flag `[QSA]` (”Query String Append”) διατηρεί τυχόν υπάρχουσες παραμέτρους του URL.
Παράλληλα μπλοκάρεται η άμεση πρόσβαση σε ευαίσθητα αρχεία ρύθμισης ώστε να εκτελείται μόνο το `index.php`:

```
<Files "config.php"> Require all denied </Files>
<Files "helpers.php"> Require all denied </Files>
<FilesMatch "^\.env"> Require all denied </FilesMatch>
```

Στη ρίζα του project υπάρχει επιπλέον `.htaccess` που προστατεύει ολόκληρο το εσωτερικό του project (`scripts`, `logs`, `pdfs`, `json_output`, `venv`, dotfiles κ.λπ.) ώστε η μόνη δημόσια επιφάνεια να είναι το API και το frontend.

Ρύθμιση AllowOverride

Κατά την ανάπτυξη παρουσιάστηκε ένα πρόβλημα που χρειάστηκε χρόνο για να εντοπιστεί. Οι κανόνες των `.htaccess` αρχείων δεν είχαν καμία απολύτως επίδραση. Το API έδινε `404` για όλα τα Pretty URLs και τα ευαίσθητα αρχεία ήταν προσβάσιμα κανονικά.
Η αιτία ήταν μια προεπιλογή του Apache. Από τις σύγχρονες εκδόσεις του Apache, η οδηγία `AllowOverride` είναι εξ ορισμού ρυθμισμένη σε `None` στο εικονικό host. Αυτό σημαίνει ότι τα `.htaccess` αρχεία αγνοούνται εντελώς για λόγους επίδοσης. Για να λειτουργήσουν χρειάζεται ρητή ενεργοποίηση στο αρχείο διαμόρφωσης του virtual host:

```
<Directory /var/www/html>
    AllowOverride All
    Require all granted
</Directory>
```

Μόλις προστέθηκε αυτή η ρύθμιση και επανεκκινήθηκε ο Apache, όλοι οι κανόνες των `.htaccess` τέθηκαν σε ισχύ και το API άρχισε να λειτουργεί κανονικά. Αυτό το θέμα δεν είναι προφανές γιατί τα `.htaccess` σε άλλους servers (π.χ. shared hosting) συνήθως δουλεύουν εξ αρχής και δεν περιμένει κανείς ότι σε έναν dedicated server χρειάζεται επιπλέον ενεργοποίηση.

5.3 Υλοποίηση του Frontend

Το frontend της εφαρμογής είναι ο κώδικας που τρέχει στον φυλλομετρητή του χρήστη και αναλαμβάνει την παρουσίαση των δεδομένων με διαδραστικό τρόπο. Όπως αναφέρθηκε στην αρχιτεκτονική, η υλοποίηση γίνεται με vanilla HTML, CSS και JavaScript χωρίς κανένα framework και χωρίς build step. Η μόνη εξωτερική εξάρτηση είναι η βιβλιοθήκη Chart.js που φορτώνεται από CDN. Στην ενότητα αυτή παρουσιάζονται τα βασικά κομμάτια της υλοποίησης.

Δομή του project

Το frontend αποτελείται από τα παρακάτω αρχεία:

```
fuelprices/
├── index.html          # Αρχική σελίδα (κάρτες + διάγραμμα)
├── docs.html           # Σελίδα τεκμηρίωσης του API
├── about.html          # Σελίδα περιγραφής του project
├── favicon.svg         # Εικονίδιο σελίδας
├── og-image.png        # Εικόνα για social media preview
└── frontend/
    ├── app.js          # Όλη η λογική JavaScript
    └── style.css       # Όλο το styling
```

Η αρχική σελίδα (`index.html`) είναι το κύριο σημείο της εφαρμογής όπου ο χρήστης βλέπει τις τιμές και αλληλεπιδρά με το διάγραμμα. Οι σελίδες τεκμηρίωσης και πληροφοριών (`docs.html`, `about.html`) είναι κυρίως στατικό περιεχόμενο και δεν αναλύονται εδώ.

Δομή της αρχικής σελίδας

Η `index.html` χρησιμοποιεί σημασιολογικά HTML5 στοιχεία (`<header>`, `<main>`, `<section>`, `<footer>`) για να οργανώσει το περιεχόμενο σε λογικές περιοχές. Η συνολική δομή είναι η εξής:

```
<header class="site-header">
    <a class="brand" href="/fuel"> Fuel Prices Greece </a>
    <nav> About | API Docs </nav>
</header>

<main>
    <section class="hero">
        <div class="latest-section">
            <!-- Κάρτες με τρέχουσες πανελλήνιες τιμές -->
        </div>
        <div class="controls">
            <!-- Multi-select dropdowns για καύσιμα και νομούς-->
            <!-- Chips για επιλογή χρονικού διαστήματος -->
```

```
        </div>
    </section>
    <section class="chart-card">
        <canvas id="price-chart"></canvas>
    </section>
</main>

<footer class="site-footer"> ... </footer>
```

Στο head της σελίδας ορίζονται επίσης τα Open Graph και Twitter Card meta tags ώστε όταν ο σύνδεσμος της εφαρμογής διαμοιραστεί σε μέσα κοινωνικής δικτύωσης να εμφανίζεται με τίτλο, περιγραφή και εικόνα προεπισκόπησης. Φορτώνεται επίσης η γραμματοσειρά Commissioner από τις Google Fonts και στο τέλος του `<body>` τα Chart.js και το `app.js`.

Επικοινωνία με το API

Όλες οι κλήσεις στο API περνούν από μία και μόνη συνάρτηση `apiGet` που χτίζει το URL και διαβάζει την απάντηση. Αν το API επιστρέψει σφάλμα η συνάρτηση πετάει exception ώστε ο καλούντας να το χειριστεί με `try/catch`:

```
async function apiGet(path, params = {}) {
    const qs = new URLSearchParams(params).toString();
    const url = `api/v1/${path}${qs ? '?' + qs : ''}`;
    const res = await fetch(url, {
        headers: { 'Accept': 'application/json' }
    });
    const body = await res.json();
    if (!res.ok || body.error) {
     throw new Error(body?.error?.message || `HTTP ${res.status}`);
    }
    return body;
}
```

Με αυτόν τον τρόπο όλος ο κώδικας του frontend ζητάει δεδομένα με γραμμές της μορφής `const r = await apiGet('prices', {...});` χωρίς να ασχολείται με το πρωτόκολλο.

Custom Multi-Select Dropdown

Τα native `<select multiple>` του browser είναι λειτουργικά αλλά δυσκολοχείριστα και άσχημα οπτικά. Για αυτό υλοποιήθηκε ένα custom multi-select component ως κλάση JavaScript. Δέχεται μια λίστα από items και ένα DOM element και χτίζει ένα dropdown με checkboxes. Όταν αλλάζει η επιλογή ειδοποιεί τους εγγεγραμμένους listeners:

```
class MultiSelect {
    constructor(root, items, { defaultIds = []
```

```
    , placeholder, countLabel }) {
        this.root = root;
        this.trigger = root.querySelector('.multi-select-trigger');
        this.label = root.querySelector('.trigger-label');
        this.panel = root.querySelector('.multi-select-panel');
        this.items = items;
        this.selected = new Set(defaultIds);
        this.changeListeners = [];

        this.renderOptions();
        this.refreshLabel();
        this.bindEvents();
    }

    toggle(id, on, row) {
        if (on) this.selected.add(id);
        else this.selected.delete(id);
        this.refreshLabel();
        this.changeListeners.forEach(fn => fn());
    }

    refreshLabel() {
        const n = this.selected.size;
        if (n === 0) this.label.textContent = this.placeholder;
        else if (n === 1) {
            const item = this.items.find(
            i => i.id === [...this.selected][0]);
            this.label.textContent = item.name;
        } else this.label.textContent = `${n} ${this.countLabel}`;
    }
    // ... bindEvents, setOpen, onChange, getSelectedItems
}
```

Στο component προστέθηκαν επίσης λεπτομέρειες accessibility (ARIA attributes), αυτόματο κλείσιμο όταν ο χρήστης κάνει click έξω από το dropdown ή πατάει το πλήκτρο Escape, και ένα label που αλλάζει δυναμικά ώστε να δείχνει είτε το όνομα της μοναδικής επιλογής είτε τον αριθμό των επιλεγμένων.

Setup του Chart.js

Το διάγραμμα χτίζεται μία φορά κατά την εκκίνηση της εφαρμογής και στη συνέχεια ενημερώνεται κάθε φορά που αλλάζει η επιλογή του χρήστη. Δύο σημαντικές προσαρμογές έγιναν για την ελληνική έκδοση. Πρώτον ο άξονας x ρυθμίστηκε ως `time scale` ώστε ο Chart.js να ερμηνεύει τις τιμές ως ημερομηνίες και να επιλέγει αυτόματα τη σωστή κλίμακα. Δεύτερον τα labels του άξονα εμφανίζουν τους ελληνικούς μήνες αντί για τα προεπιλεγμένα αγγλικά:

```
const GREEK_MONTHS = ['Ιαν', 'Φεβ', 'Μαρ', 'Απρ', 'Μάι', 'Ιουν',
                      'Ιουλ', 'Αυγ', 'Σεπ', 'Οκτ', 'Νοε', 'Δεκ'];

scales: {
    x: {
        type: 'time',
        time: { unit: 'month' },
        ticks: {
            callback: (value) =>
                GREEK_MONTHS[new Date(value).getMonth()],
        },
    },
    y: {
        ticks: {
            callback: (v) => v.toFixed(3) + '€',
        },
    },
}
```

Παρόμοια λογική εφαρμόζεται στο tooltip όπου εμφανίζεται η πλήρης ελληνική ημερομηνία (`15 Ιαν 2025`) μέσω της `formatGreekDate`. Οι τιμές εμφανίζονται πάντα με τρία δεκαδικά ψηφία και τη μονάδα `€/L` ώστε να ταιριάζουν με τη μορφή του Υπουργείου.

Πολλαπλές σειρές δεδομένων

Ένα από τα πιο χρήσιμα χαρακτηριστικά της εφαρμογής είναι η δυνατότητα σύγκρισης πολλαπλών συνδυασμών νομών και καυσίμων στο ίδιο διάγραμμα. Όταν ο χρήστης επιλέξει 2 νομούς και 3 καύσιμα η εφαρμογή πρέπει να δείξει 6 γραμμές, μία για κάθε συνδυασμό. Επειδή κάθε συνδυασμός χρειάζεται ξεχωριστή κλήση στο API, οι κλήσεις γίνονται παράλληλα με `Promise.all`:

```
const combos = [];
for (const p of prefectures) {
    for (const f of fuels) combos.push({ prefecture: p, fuel: f });
}

const responses =
await Promise.all(combos.map(({ prefecture, fuel }) =>
    apiGet('prices', {
        prefecture_id: prefecture.id,
        fuel_type_id: fuel.id,
        from, to,
    }).then(r => ({ prefecture, fuel, points: r.data }))
));
```

Έτσι το συνολικό latency είναι όσο και της πιο αργής κλήσης, αντί για το άθροισμα όλων. Σε ένα τοπικό δίκτυο η διαφορά είναι ορατή αλλά όχι κρίσιμη. Σε αργές συνδέσεις όμως μειώνει σημαντικά τον χρόνο φόρτωσης.
Ένα λεπτό σημείο που χρειάστηκε προσοχή είναι το race condition. Αν ο χρήστης αλλάξει γρήγορα τις επιλογές του, μπορεί μια παλιά κλήση να επιστρέψει αφού έχει ξεκινήσει η νέα, και να γράψει λάθος δεδομένα στο διάγραμμα. Για να αποφευχθεί αυτό κάθε refresh παίρνει ένα μοναδικό token και πριν γράψει στο διάγραμμα ελέγχει αν εξακολουθεί να είναι το πιο πρόσφατο:

```
let fetchToken = 0;

async function refreshChart() {
  // ...
  const myToken = ++fetchToken;
  const responses = await Promise.all(...);

  if (myToken !== fetchToken) return;//newer request superseded us
  // ... draw chart
}
```

Κάρτες με τις τρέχουσες τιμές

Στην αρχική σελίδα εμφανίζονται τέσσερις κάρτες με τις πιο πρόσφατες πανελλήνιες τιμές για τα κύρια καύσιμα. Αυτές γεμίζουν με μία κλήση στο `prices/latest` και φιλτράρονται ώστε να δείχνουν τις τιμές του Πανελλήνιου Σταθμισμένου Μέσου Όρου. Η σταθερά `HEADLINE_FUEL_IDS` ορίζει ποια καύσιμα εμφανίζονται και με ποια σειρά. Σκόπιμα έχει εξαιρεθεί το Diesel Θέρμανσης (ID 5) γιατί είναι εποχικό και θα ήταν παραπλανητικό να εμφανίζεται μια καλοκαιρινή τιμή:

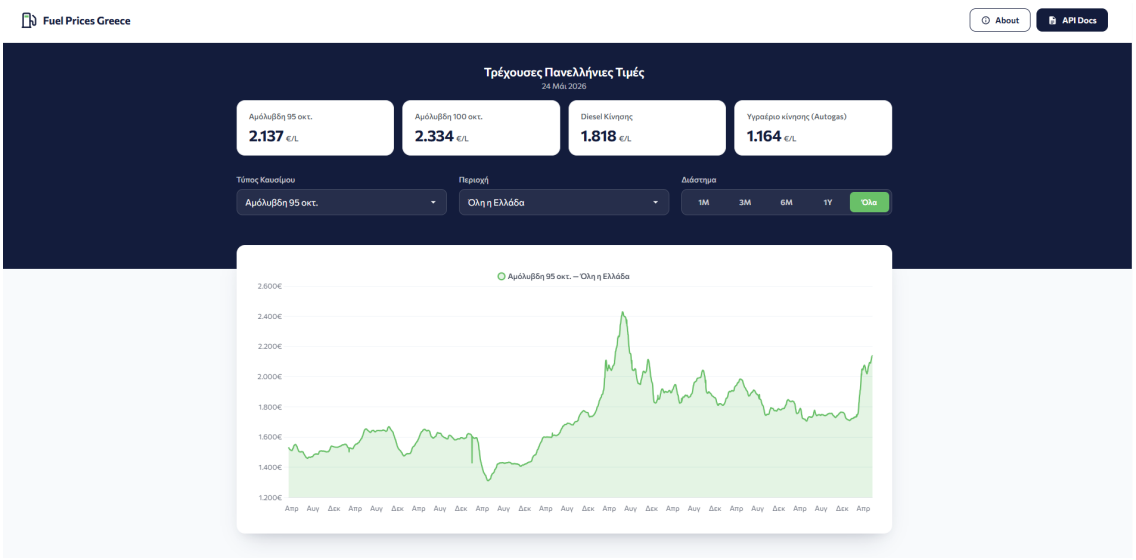
```
const HEADLINE_FUEL_IDS = [1, 2, 3, 4];

async function loadLatestPrices(fuels) {
    const res = await apiGet('prices/latest');
    const national = res.data.entries.find(
        e => e.prefecture.name === DEFAULT_PREFECTURE
    );

    for (const fuelId of HEADLINE_FUEL_IDS) {
      const fuel = fuels.find(f => f.id === fuelId);
      const price = national.prices[fuel.name];
      if (price == null) continue;
      els.latestGrid.appendChild
      (buildLatestCard(fuel.name, price));
    }
}
```

Responsive CSS

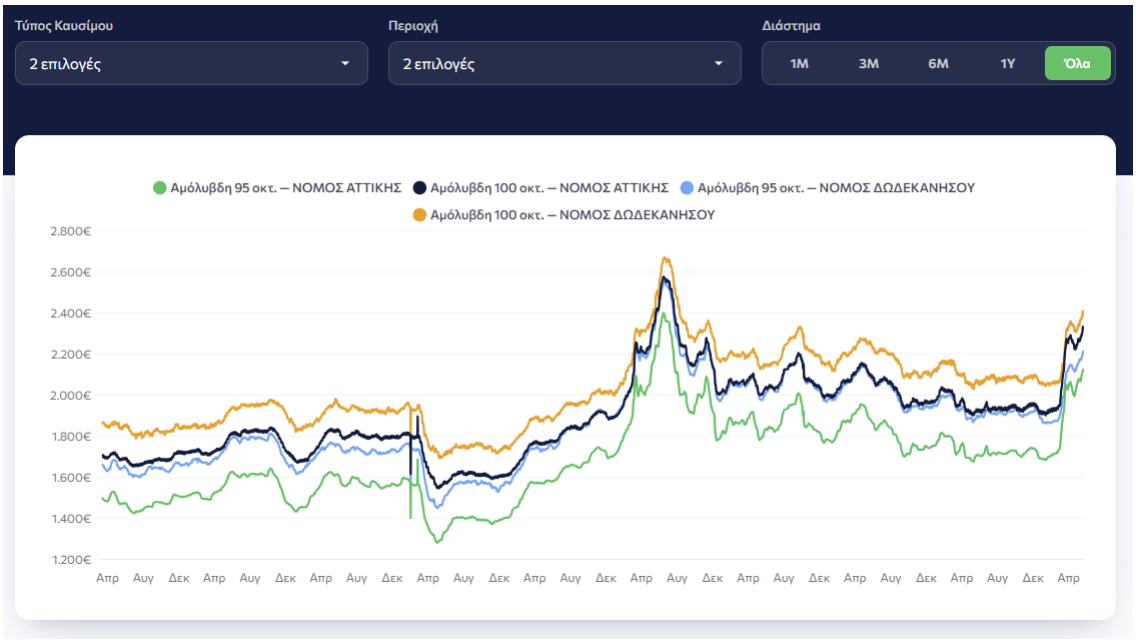
Το styling είναι γραμμένο σε καθαρό CSS χωρίς preprocessor. Χρησιμοποιεί CSS Grid και Flexbox για τη διάταξη και CSS custom properties (variables) για χρώματα και σταθερές διαστάσεις ώστε να μπορούν να αλλαχθούν εύκολα.
Για την υποστήριξη κινητών συσκευών έχουν οριστεί media queries σε δύο βασικά breakpoints (768px και 640px). Σε μικρότερες οθόνες το layout αλλάζει από οριζόντιο σε κάθετο, οι κάρτες γίνονται μικρότερες και τα dropdowns επιτρέπουν περισσότερο χώρο για το dropdown panel. Οι κανόνες είναι γραμμένοι με χρήση `max-width` queries οπότε η βασική σχεδίαση στοχεύει στις μεγάλες οθόνες και προσαρμόζεται προς τα κάτω για τις μικρότερες.

5.4 Παρουσίαση της Εφαρμογής

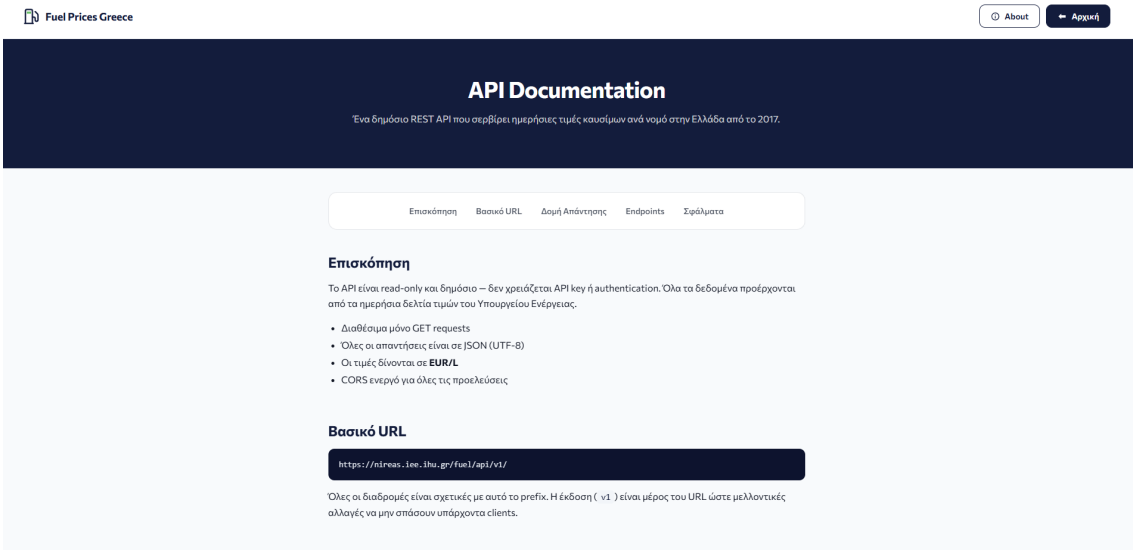
Η εφαρμογή είναι δημόσια διαθέσιμη στη διεύθυνση https://nireas.iee.ihu.gr/fuel/. Παρακάτω παρουσιάζονται στιγμιότυπα οθόνης από τις βασικές σελίδες όπως φαίνονται στον τελικό χρήστη.



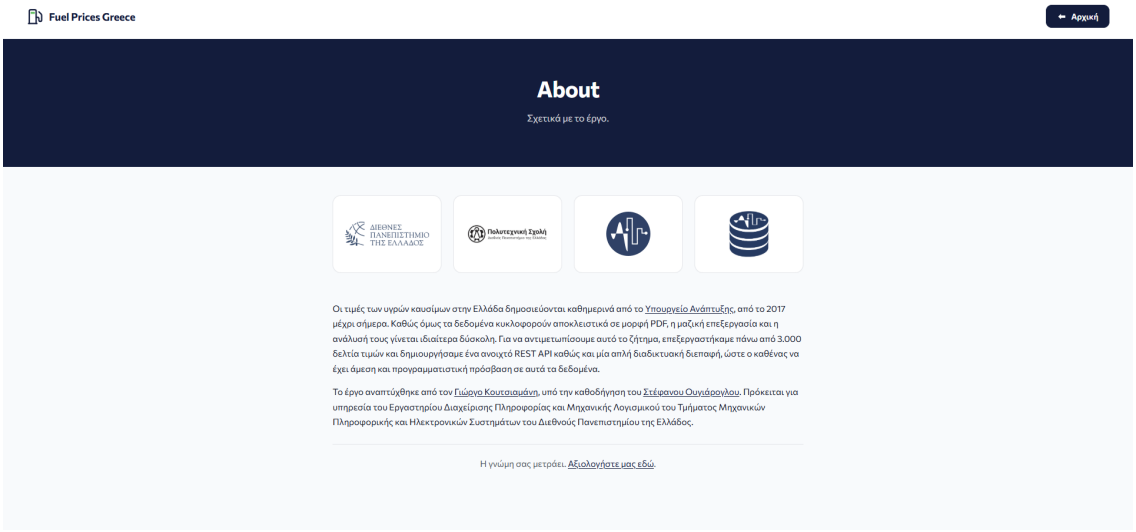
Σχήμα 5.2: Αρχική σελίδα: κάρτες με τις πρόσφατες πανελλήνιες τιμές και διαδραστικό διάγραμμα.



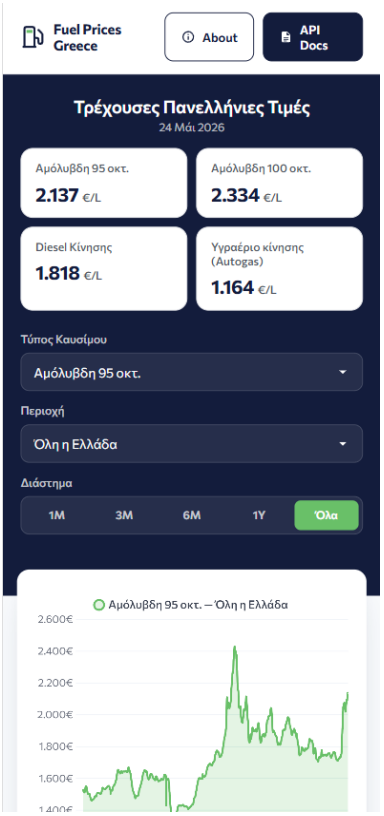
Σχήμα 5.3: Σύγκριση πολλαπλών καυσίμων και νομών στο ίδιο διάγραμμα.



Σχήμα 5.4: Σελίδα τεκμηρίωσης του API.



Σχήμα 5.5: Σελίδα περιγραφής του project.



Σχήμα 5.6: Προβολή της αρχικής σελίδας σε κινητή συσκευή.

Κεφάλαιο 6

Εμπειρία Χρήσης

6.1 SUS — System Usability Scale Questionnaire

Για να αξιολογηθεί η ευχρηστία της εφαρμογής επιλέχθηκε το ερωτηματολόγιο System Usability Scale (SUS). Το SUS δημιουργήθηκε από τον John Brooke το 1986 στην Digital Equipment Corporation και δημοσιεύτηκε επίσημα το 1996 [20]. Από τότε έχει καθιερωθεί ως ένα από τα πιο διαδεδομένα εργαλεία για γρήγορη και αξιόπιστη εκτίμηση της ευχρηστίας ενός συστήματος. Παρά την απλότητά του πολλές μελέτες έχουν δείξει ότι παράγει αξιόπιστα αποτελέσματα ακόμα και με μικρό αριθμό συμμετεχόντων.
Το ερωτηματολόγιο αποτελείται από 10 σταθερές ερωτήσεις τις οποίες ο συμμετέχων καλείται να βαθμολογήσει σε κλίμακα Likert πέντε σημείων, από “Διαφωνώ απόλυτα” (1) μέχρι “Συμφωνώ απόλυτα” (5). Ένα από τα πιο ενδιαφέροντα σχεδιαστικά χαρακτηριστικά του είναι ότι οι ερωτήσεις εναλλάσσονται μεταξύ θετικής και αρνητικής διατύπωσης. Οι μονές (1, 3, 5, 7, 9) είναι διατυπωμένες θετικά ενώ οι ζυγές (2, 4, 6, 8, 10) αρνητικά. Αυτή η εναλλαγή αναγκάζει τον συμμετέχοντα να διαβάζει προσεκτικά κάθε ερώτηση πριν απαντήσει και αποτρέπει αυτόματες απαντήσεις (acquiescence bias) όπου κάποιος θα διάλεγε μηχανικά μια σταθερή τιμή χωρίς να το σκεφτεί.

Οι ερωτήσεις

Στην παρούσα μελέτη οι 10 ερωτήσεις του SUS διανεμήθηκαν στα αγγλικά. Η μόνη προσαρμογή σε σχέση με την πρωτότυπη διατύπωση είναι ότι η λέξη “system” αντικαταστάθηκε με τη λέξη “website” ώστε να ταιριάζει με το αντικείμενο της αξιολόγησης. Πρόκειται για συνηθισμένη και αποδεκτή πρακτική που πρότεινε ο ίδιος ο Brooke. Ο πίνακας 6.1 παρουσιάζει τις ερωτήσεις όπως μοιράστηκαν στους συμμετέχοντες.

#	Question
1	I think that I would like to use this website frequently.
2	I found the website unnecessarily complex.
3	I thought the website was easy to use.
4	I think that I would need the support of a technical person to be able to use this website.
5	I found the various functions in this website were well integrated.
6	I thought there was too much inconsistency in this website.
7	I would imagine that most people would learn to use this website very quickly.
8	I found the website very cumbersome to use.
9	I felt very confident using the website.
10	I needed to learn a lot of things before I could get going with this website.

Πίνακας 6.1: Οι 10 ερωτήσεις του ερωτηματολογίου SUS.

Διανομή του ερωτηματολογίου

Το ερωτηματολόγιο διανεμήθηκε ηλεκτρονικά μέσω της υπηρεσίας Google Forms. Η επιλογή αυτή έγινε γιατί η υπηρεσία είναι δωρεάν, εύκολη στη χρήση και διαθέτει αυτόματη συγκέντρωση των απαντήσεων σε υπολογιστικό φύλλο που μπορεί να εξαχθεί για περαιτέρω ανάλυση. Παράλληλα ο σύνδεσμος του ερωτηματολογίου μπορεί να μοιραστεί εύκολα μέσω email ή κοινωνικών δικτύων.
Πριν συμπληρώσουν το ερωτηματολόγιο, οι συμμετέχοντες είχαν τη δυνατότητα να επισκεφθούν την εφαρμογή στη διεύθυνση https://nireas.iee.ihu.gr/fuel/ και να τη δοκιμάσουν για όσο χρόνο επιθυμούσαν χωρίς συγκεκριμένες οδηγίες ή σενάριο χρήσης. Αυτή η ελεύθερη εξερεύνηση είναι κοντά στο πώς θα προσέγγιζε την εφαρμογή ένας πραγματικός χρήστης για πρώτη φορά και επιτρέπει στο SUS να μετρήσει το πραγματικό συνολικό αίσθημα της εμπειρίας χρήσης.
Η ανάλυση των απαντήσεων, ο υπολογισμός του τελικού SUS score και η ερμηνεία των αποτελεσμάτων παρουσιάζονται στην επόμενη ενότητα.

6.2 Υπολογισμός SUS Score

Συγκεντρώθηκαν συνολικά 14 απαντήσεις στο ερωτηματολόγιο. Σε αυτή την ενότητα παρουσιάζεται ο τρόπος υπολογισμού του τελικού SUS score, τα αποτελέσματα και η ερμηνεία τους.

Μεθοδολογία υπολογισμού

Το SUS score δεν είναι ο απλός μέσος όρος των απαντήσεων. Ο υπολογισμός γίνεται με συγκεκριμένο τρόπο που λαμβάνει υπόψη την εναλλαγή θετικών και αρνητικών ερωτήσεων. Για κάθε συμμετέχοντα η συνεισφορά κάθε ερώτησης υπολογίζεται ως εξής:

- Για τις θετικές ερωτήσεις (1, 3, 5, 7, 9) η συνεισφορά είναι η απάντηση μείον 1.

- Για τις αρνητικές ερωτήσεις (2, 4, 6, 8, 10) η συνεισφορά είναι το 5 μείον την απάντηση.

Με αυτόν τον τρόπο κάθε ερώτηση συνεισφέρει μια τιμή από 0 έως 4 και το άθροισμα των δέκα ερωτήσεων κυμαίνεται από 0 έως 40. Το άθροισμα αυτό πολλαπλασιάζεται επί 2,5 ώστε το τελικό score να βρίσκεται στην κλίμακα 0 έως 100. Πρέπει να τονιστεί ότι το SUS score δεν είναι ποσοστό. Ένα score 90 δεν σημαίνει 90%, είναι απλώς μια τιμή στην κλίμακα του SUS.

Αποτελέσματα

Ο πίνακας 6.2 δείχνει το τελικό SUS score για καθέναν από τους 14 συμμετέχοντες.

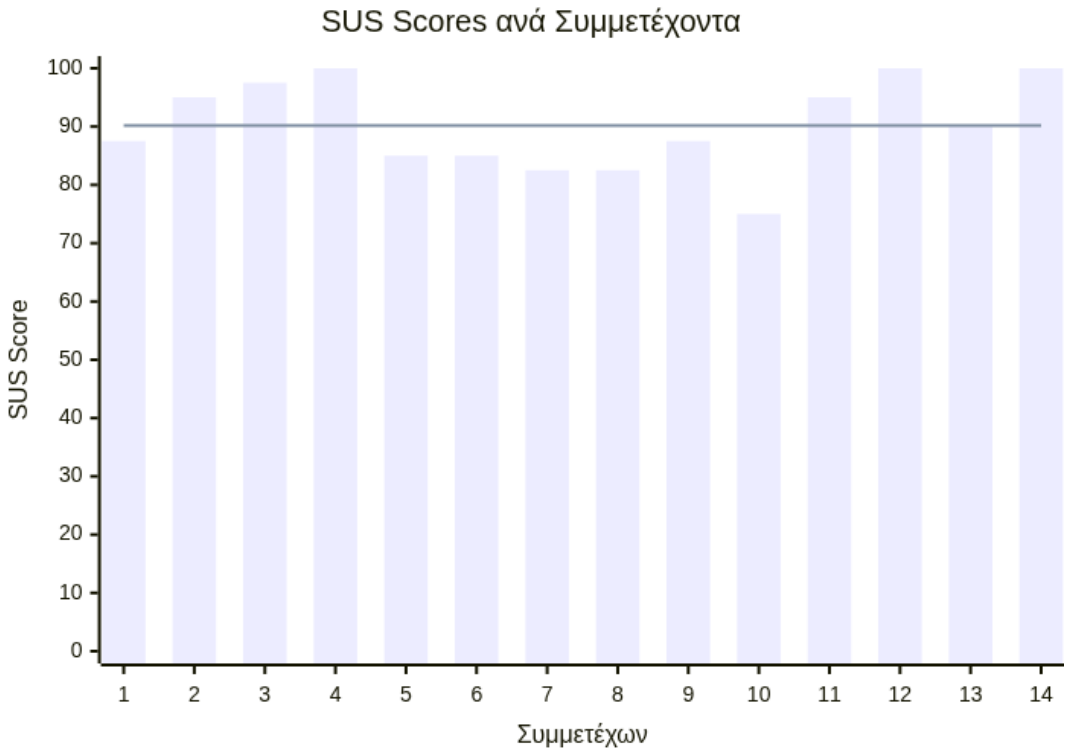
Συμμετέχων	**SUS Score**	**Συμμετέχων**	**SUS Score**
1	87,5	8	82,5
2	95,0	9	87,5
3	97,5	10	75,0
4	100,0	11	95,0
5	85,0	12	100,0
6	85,0	13	90,0
7	82,5	14	100,0

Πίνακας 6.2: Τα SUS scores των 14 συμμετεχόντων.

Από τα παραπάνω δεδομένα προκύπτουν τα εξής συγκεντρωτικά στατιστικά:

- **Μέσος όρος**: 90,18
- **Διάμεσος**: 88,75
- **Τυπική απόκλιση**: 7,82
- **Ελάχιστο**: 75,0
- **Μέγιστο**: 100,0

Η σχετικά μικρή τυπική απόκλιση δείχνει ότι οι απαντήσεις ήταν αρκετά συγκεντρωμένες. Δεν υπήρξε κάποιος συμμετέχων με ιδιαίτερα χαμηλή βαθμολογία αφού ακόμα και το ελάχιστο score (75) θεωρείται καλό.



Σχήμα 6.1: Κατανομή των SUS scores των συμμετεχόντων.

Ερμηνεία των αποτελεσμάτων

Για να αποκτήσει νόημα ένα SUS score χρειάζεται ένα σημείο αναφοράς. Σύμφωνα με μελέτες που έχουν συγκεντρώσει χιλιάδες αξιολογήσεις, ο μέσος όρος του SUS σε όλα τα συστήματα είναι περίπου 68 [21]. Ένα σύστημα με score 68 θεωρείται μέσο, ένα score κάτω από 50 θεωρείται προβληματικό και ένα score πάνω από 80 θεωρείται εξαιρετικό.
Ο μέσος όρος της εφαρμογής μας είναι 90,18, αρκετά υψηλότερος από τον μέσο όρο των 68. Με βάση την κλίμακα επιθέτων που πρότειναν οι Bangor και συνεργάτες [22], ένα score αυτού του επιπέδου αντιστοιχεί στον χαρακτηρισμό “Excellent”. Στην καμπύλη βαθμολόγησης των Sauro και Lewis ένα score πάνω από 84 αντιστοιχεί στον υψηλότερο βαθμό “A+” και τοποθετεί την εφαρμογή στο κορυφαίο ποσοστό των συστημάτων που έχουν αξιολογηθεί με το SUS.
Συνολικά τα αποτελέσματα του ερωτηματολογίου δείχνουν ότι οι χρήστες βρήκαν την εφαρμογή εύκολη και ευχάριστη στη χρήση. Δεδομένου ότι η ευχρηστία ήταν ένας από τους βασικούς στόχους της εφαρμογής ιστού, το υψηλό SUS score επιβεβαιώνει ότι οι σχεδιαστικές επιλογές που έγιναν ανταποκρίνονται στις ανάγκες των τελικών χρηστών.

Κεφάλαιο 7

Συμπεράσματα και Μελλοντικές Επεκτάσεις

7.1 Συμπεράσματα

Στην εργασία αυτή ασχοληθήκαμε με το πρόβλημα της προσβασιμότητας των δημόσιων δεδομένων τιμών καυσίμων στην Ελλάδα. Όπως είδαμε στο Κεφάλαιο 1, το Υπουργείο Ανάπτυξης δημοσιεύει καθημερινά τα δελτία τιμών αλλά σε μορφή PDF που δεν επιτρέπει την αυτόματη επεξεργασία. Στόχος της εργασίας ήταν να μετατραπεί αυτό το δύσχρηστο σύνολο δεδομένων σε πραγματικά ανοιχτά δεδομένα και να γίνει διαθέσιμο σε όλους μέσω σύγχρονων εργαλείων. Για να επιτευχθεί αυτό αναπτύχθηκε ένα ολοκληρωμένο σύστημα τριών στοιχείων. Πρώτο ένα αυτοματοποιημένο pipeline σε Python που κατεβάζει τα δελτία από τον ιστότοπο του Υπουργείου, εξάγει τις τιμές και τις αποθηκεύει σε σχεσιακή βάση δεδομένων. Για τα παλιά PDF χρησιμοποιήθηκε το Gemini API ως μηχανή εξαγωγής, ενώ για τα νέα δελτία η εξαγωγή γίνεται με τη βιβλιοθήκη pdfplumber. Δεύτερο ένα ανοιχτό REST API σε PHP που εκθέτει τα δεδομένα της βάσης μέσω καλά σχεδιασμένων endpoints σε μορφή JSON. Τρίτο μια εφαρμογή ιστού σε vanilla HTML, CSS και JavaScript με Chart.js που οπτικοποιεί τα δεδομένα διαδραστικά. Όλο το σύστημα τρέχει στον server του τμήματος και ενημερώνεται καθημερινά αυτόματα μέσω cron job.

Σε ό,τι αφορά τους στόχους της εργασίας, αυτοί επιτεύχθηκαν σε σημαντικό βαθμό. Τα δεδομένα του `fuelprices.gr` βρίσκονταν στο πρώτο επίπεδο (1★) του σχήματος Berners-Lee. Μετά την εργασία αυτή τα ίδια δεδομένα ανεβαίνουν σταθερά στο τρίτο επίπεδο (3★) με χαρακτηριστικά που τα προσεγγίζουν στο τέταρτο. Το JSON είναι ανοιχτή μηχαναγνώσιμη μορφή και το REST API εκθέτει τα δεδομένα μέσω σταθερών HTTP διευθύνσεων (URIs) που επιτρέπουν σε άλλα συστήματα να δείχνουν σε αυτά. Δεν εφαρμόστηκαν τεχνολογίες Linked Data όπως RDF και SPARQL που είναι ο αυστηρός ορισμός του τέταρτου επιπέδου, αλλά η πρακτική προσβασιμότητα των δεδομένων είναι εφάμιλλη. Παράλληλα ικανοποιούνται και οι τέσσερις αρχές FAIR, οπότε από τη σκοπιά της σύγχρονης πρακτικής τα δεδομένα είναι πλέον πραγματικά ανοιχτά και αξιοποιήσιμα.

Πέρα από τα παραπάνω η εργασία ήταν μια ευκαιρία να αντιμετωπίσουμε αρκετά πραγματικά προβλήματα ανάπτυξης που σπάνια εμφανίζονται σε ένα ακαδημαϊκό μάθημα. Η πιο σημαντική εμπειρία αφορά την εξαγωγή δεδομένων από PDF. Αρχικά υπήρχε η εντύπωση ότι ένας απλός parser θα έφτανε αλλά γρήγορα φάνηκε ότι τα παλαιότερα δελτία είχαν τόσες πα-

ραλλαγές μορφής (σπασμένες στήλες, encoding mismatches, σκαναρισμένες σελίδες, εποχικές στήλες) που κανένας παραδοσιακός parser δεν θα λειτουργούσε καθολικά. Η χρήση ενός LLM ως parser ήταν λύση που λίγα χρόνια πριν δεν θα ήταν εφικτή και έδωσε αξιόπιστα αποτελέσματα με αποδεκτό κόστος. Ένα δεύτερο μάθημα ήταν η σημασία του defensive design σε ένα pipeline που τρέχει χωρίς επίβλεψη. Οι έλεγχοι εγκυρότητας, το exponential backoff στις κλήσεις του API, η idempotent εισαγωγή στη βάση και τα emails ειδοποίησης δεν είναι περιττές ευκολίες, είναι αυτά που κάνουν τη διαφορά μεταξύ ενός pipeline που δουλεύει στις δοκιμές και ενός που επιβιώνει στην πράξη.
Τέλος, ο διαχωρισμός σε τρία ξεχωριστά στοιχεία (pipeline, API, frontend) με σαφή όρια μεταξύ τους αποδείχθηκε σημαντικός για τη συντηρησιμότητα. Κάθε στοιχείο μπορεί να εξελιχθεί ή να αντικατασταθεί ανεξάρτητα χωρίς να επηρεαστούν τα υπόλοιπα. Αν για παράδειγμα κάποια στιγμή θελήσουμε να προσθέσουμε mobile εφαρμογή, μπορούμε να την χτίσουμε πάνω στο ίδιο API χωρίς να αλλάξουμε τίποτα στο pipeline ή στη βάση. Αυτή η ευελιξία ανοίγει τον δρόμο για τις μελλοντικές επεκτάσεις που συζητώνται στην επόμενη ενότητα.

7.2 Μελλοντικές Επεκτάσεις

Παρόλο που το σύστημα που αναπτύχθηκε στην παρούσα εργασία καλύπτει τους αρχικούς στόχους, υπάρχουν αρκετές κατευθύνσεις για περαιτέρω εξέλιξη. Στην ενότητα αυτή παρουσιάζονται οι πιο ενδιαφέρουσες προτάσεις για μελλοντική δουλειά. Πρόκειται για επεκτάσεις σημαντικού βεληνεκούς που η καθεμία θα μπορούσε να αποτελέσει αυτόνομη εργασία και υπερβαίνει το πλαίσιο μιας πτυχιακής.

Πρόβλεψη τιμών με χρήση Machine Learning

Μια από τις πιο φιλόδοξες επεκτάσεις θα ήταν η ανάπτυξη ενός μοντέλου πρόβλεψης για την εξέλιξη των τιμών των καυσίμων. Έχοντας πλέον στη διάθεσή μας πολυετή ιστορικά δεδομένα μέσω της βάσης, μπορεί να εκπαιδευτεί ένα μοντέλο time series forecasting (π.χ. ARIMA, Prophet ή LSTM) που να προβλέπει τις τιμές των επόμενων ημερών ή εβδομάδων. Για να είναι πραγματικά χρήσιμη μια τέτοια πρόβλεψη θα έπρεπε να ενσωματώνει και εξωτερικούς παράγοντες όπως η διεθνής τιμή του αργού πετρελαίου, η ισοτιμία ευρώ/δολαρίου και γεωπολιτικά γεγονότα. Το έργο αυτό απαιτεί εξειδικευμένη γνώση σε ML, ξεχωριστή υποδομή για την εκπαίδευση και την περιοδική επανεκπαίδευση των μοντέλων, και ένα παράλληλο pipeline για τα εξωτερικά δεδομένα.

Επέκταση σε άλλα ανοιχτά δεδομένα

Η αρχιτεκτονική του pipeline μπορεί να εφαρμοστεί σε παρόμοιες κατηγορίες δημόσιων δεδομένων που υποφέρουν από τα ίδια προβλήματα μη μηχαναγνώσιμης δημοσίευσης. Παραδείγματα είναι οι τιμές του φυσικού αερίου, του ρεύματος, βασικών τροφίμων ή ακόμα και δεδομένα από άλλους φορείς όπως η ΕΛΣΤΑΤ. Κάθε νέα κατηγορία δεδομένων όμως απαιτεί δικό της scraper, δικό της parser με τη δική του λογική validation, δικό της σχήμα βάσης δεδομένων και αντίστοιχα νέα endpoints στο API. Πρακτικά κάθε επέκταση είναι ένα νέο μικρό project, χωρίς να μπορεί να γίνει γενική αυτοματοποίηση πάνω από διαφορετικές πηγές.

Διεθνής σύγκριση τιμών

Μια ιδιαίτερα ενδιαφέρουσα προοπτική είναι η σύγκριση των τιμών της Ελλάδας με αυτές άλλων ευρωπαϊκών χωρών. Η Eurostat και το Weekly Oil Bulletin της Ευρωπαϊκής Επιτροπής δημοσιεύουν αντίστοιχα δεδομένα για όλα τα κράτη μέλη. Η ενσωμάτωσή τους θα έδινε στους χρήστες τη δυνατότητα να δουν πώς συγκρίνεται η ελληνική αγορά καυσίμων με τις γειτονικές. Η υλοποίηση όμως δεν είναι τετριμμένη γιατί κάθε χώρα δημοσιεύει σε διαφορετική μορφή, με διαφορετική συχνότητα ενημέρωσης, διαφορετική κατηγοριοποίηση καυσίμων και διαφορετικό καθεστώς φορολόγησης. Χρειάζεται ξεχωριστή μελέτη για το πώς θα κανονικοποιηθούν αυτές οι διαφορές ώστε οι συγκρίσεις να είναι ουσιαστικές και όχι παραπλανητικές.

Μετάβαση σε πραγματικό Linked Data

Όπως αναφέρθηκε στα συμπεράσματα, το API μας πιάνει τα όρια του τέταρτου επιπέδου του σχήματος Berners-Lee χωρίς όμως να εφαρμόζει τις αυστηρές τεχνολογίες Linked Data. Μια ολοκληρωμένη μετάβαση στο πέμπτο επίπεδο (5★) θα απαιτούσε σημαντική επένδυση. Πρώτον τα δεδομένα θα έπρεπε να αναπαρίστανται σε RDF χρησιμοποιώντας καθιερωμένα λεξιλόγια όπως το schema.org και η Dublin Core για τα μεταδεδομένα. Δεύτερον θα έπρεπε να στηθεί ένας triplestore (π.χ. Apache Jena, Virtuoso) και να εκτεθεί ένα SPARQL endpoint παράλληλα με το υπάρχον REST API. Τέλος και το πιο σημαντικό, τα δεδομένα θα πρέπει να συνδεθούν με άλλα open datasets όπως το GeoNames για τους νομούς ή το DBpedia για τα είδη καυσίμων. Αυτή η εργασία απαιτεί εξειδίκευση στις τεχνολογίες του Semantic Web και ουσιαστική σχεδιαστική δουλειά για το ontology mapping.

Caching και rate limiting

Με τη μορφή που έχει σήμερα το API εξυπηρετεί άνετα τον προβλεπόμενο φόρτο. Αν όμως κάποια στιγμή κερδίσει σημαντική δημοτικότητα ή χρησιμοποιηθεί από εφαρμογές τρίτων με υψηλή κίνηση, θα ήταν χρήσιμο να προστεθεί ένα επίπεδο caching (π.χ. Redis) για τις πιο συχνές απαντήσεις και ένα σύστημα rate limiting για να αποφεύγεται η κατάχρηση. Αυτές οι βελτιώσεις δεν είναι κρίσιμες στην παρούσα κλίμακα αλλά θα αποτελούσαν φυσικό επόμενο βήμα στην ωρίμανση του συστήματος.

Βιβλιογραφία

[1] Ελληνική Στατιστική Αρχή, "Εξωτερικό Εμπόριο, Δεκέμβριος 2025." https://www.statistics.gr/el/statistics/-/publication/SFC02/2025-M12, 2026. Ανακτήθηκε: 2026-05-23.

[2] Eurostat, "Energy in Europe: imports dependency." https://ec.europa.eu/eurostat/web/products-eurostat-news/w/wdn-20260318-1, 2026. Ανακτήθηκε: 2026-05-23.

[3] Open Knowledge Foundation, "The open definition." https://opendefinition.org/od/2.1/en/, 2015. Ανακτήθηκε: 2026-05-23.

[4] T. Berners-Lee, "Linked data — design issues." https://www.w3.org/DesignIssues/LinkedData.html, 2006. Ανακτήθηκε: 2026-05-23.

[5] M. D. Wilkinson *et al.*, "The FAIR guiding principles for scientific data management and stewardship." Scientific Data, vol. 3, article 160018, 2016. https://doi.org/10.1038/sdata.2016.18.

[6] European Commission, "Open data maturity report 2024." https://data.europa.eu/en/publications/open-data-maturity, 2024. Ανακτήθηκε: 2026-05-23.

[7] H. P. Langtangen, *A Primer on Scientific Programming with Python*. Texts in Computational Science and Engineering, Berlin, Heidelberg: Springer, 5 ed., 2016.

[8] W. J. Gilmore, *Beginning PHP and MySQL: From Novice to Professional*. Berkeley, CA: Apress, 4 ed., 2010.

[9] D. Bartholomew, *MariaDB and MySQL Common Table Expressions and Window Functions Revealed*. Berkeley, CA: Apress, 2017.

[10] D. R. Brooks, *Guide to HTML, JavaScript and PHP: For Scientists and Engineers*. London: Springer, 2011.

[11] L. Richardson, "Beautiful Soup documentation." https://www.crummy.com/software/BeautifulSoup/bs4/doc/, 2024. Ανακτήθηκε: 2026-05-23.

[12] J. Singer-Vine, "pdfplumber: Plumb a PDF for detailed information about each char, rectangle, line, et cetera." https://github.com/jsvine/pdfplumber, 2024. Ανακτήθηκε: 2026-05-23.

[13] Gemini Team, Google, “Gemini: A family of highly capable multimodal models.” arXiv:2312.11805, 2023. https://arxiv.org/abs/2312.11805.

[14] Chart.js Contributors, “Chart.js documentation.” https://www.chartjs.org/docs/latest/, 2024. Ανακτήθηκε: 2026-05-23.

[15] P. Wainwright, *Pro Apache*. Berkeley, CA: Apress, 3 ed., 2004.

[16] S. van Vugt, *Beginning the Linux Command Line*. Berkeley, CA: Apress, 2 ed., 2015.

[17] R. T. Fielding, “Architectural styles and the design of network-based software architectures.” Ph.D. dissertation, University of California, Irvine, 2000. https://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm.

[18] R. T. Fielding, M. Nottingham, and J. Reschke, “HTTP semantics.” RFC 9110, Internet Engineering Task Force (IETF), 2022. https://datatracker.ietf.org/doc/html/rfc9110.

[19] T. Bray, “The javascript object notation (JSON) data interchange format.” RFC 8259, Internet Engineering Task Force (IETF), 2017. https://datatracker.ietf.org/doc/html/rfc8259.

[20] J. Brooke, “SUS: A ‘quick and dirty’ usability scale.” In P. W. Jordan, B. Thomas, B. A. Weerdmeester, & I. L. McClelland (Eds.), Usability Evaluation in Industry, pp. 189–194. Taylor & Francis, London, 1996.

[21] J. Sauro, “A practical guide to the system usability scale: Background, benchmarks & best practices.” Measuring Usability LLC, 2011. https://measuringu.com/sus/.

[22] A. Bangor, P. Kortum, and J. Miller, “Determining what individual SUS scores mean: Adding an adjective rating scale.” Journal of Usability Studies, vol. 4, no. 3, pp. 114–123, 2009.