ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΥΠΟΛΟΓΙΣΤΩΝ

Σύστημα παρακολούθησης και ελέγχου λιανικών τιμών από ετερογενείς πηγές δεδομένων μεγάλης κλίμακας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΕΟΔΩΡΟΣ - ΦΡΙΞΟΣ ΠΑΠΑΡΡΗΓΟΠΟΥΛΟΣ

Επίβλεψη: Παναγιώτης Τσανάκας Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2023

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΥΠΟΛΟΓΙΣΤΩΝ

Σύστημα παρακολούθησης και ελέγχου λιανικών τιμών από ετερογενείς πηγές δεδομένων μεγάλης κλίμακας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΕΟΔΩΡΟΣ - ΦΡΙΞΟΣ ΠΑΠΑΡΡΗΓΟΠΟΥΛΟΣ

Επίβλεψη: Παναγιώτης Τσανάκας Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή στις 14 Μαρτίου 2023

……	……	……...
Παναγιώτης Τσανάκας	Σωτήριος Ξύδης	Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2023

(Υπογραφή)
.....

Θεόδωρος Φρίξος Παπαρρηγόπουλος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

Περίληψη

Στην παρούσα διατριβή προτείνω ένα σύστημα για την παρακολούθηση και τον έλεγχο των τιμών λιανικής από ετερογενείς πηγές δεδομένων μεγάλης κλίμακας. Το σύστημα στοχεύει στην παροχή πολύτιμων πληροφοριών τόσο στους λιανοπωλητές όσο και στους πελάτες, παρακολουθώντας και συγκρίνοντας τις τιμές των προϊόντων σε διαφορετικά σούπερ μάρκετ και παρέχοντας συστάσεις στους πελάτες.

Οι λιανικές τιμές των προϊόντων αλλάζουν συχνά και απρόβλεπτα λόγω διαφόρων παραγόντων, όπως η προσφορά και η ζήτηση, οι προσφορές και ο ανταγωνισμός. Τόσο οι καταναλωτές όσο και οι έμποροι λιανικής μπορούν να επωφεληθούν από την ακριβή και έγκαιρη παρακολούθηση των τιμών λιανικής για να λαμβάνουν τεκμηριωμένες αποφάσεις σχετικά με τις στρατηγικές αγοράς και τιμολόγησης. Επιπλέον, οι συστάσεις προϊόντων μπορούν να βελτιώσουν την εμπειρία του χρήστη και να αυξήσουν τις πωλήσεις.

Το Bestcart είναι μια πολυεπίπεδη εφαρμογή που παρακολουθεί τις τιμές λιανικής και παρέχει συστάσεις προϊόντων. Η εφαρμογή περιλαμβάνει ένα web crawler που συλλέγει δεδομένα από πολλαπλά σούπερ μάρκετ χρησιμοποιώντας τη βιβλιοθήκη Selenium, ένα API βασισμένο στη βιβλιοθήκη της Python Flask που αποθηκεύει τα επεξεργασμένα δεδομένα σε μια βάση δεδομένων SQLite και μια frontend υπηρεσία που οπτικοποιεί τα ερωτήματα (queries) και παρέχει ένα σύστημα συστάσεων για συστάσεις προϊόντων.

Το σύστημα συστάσεων χρησιμοποιεί το TF-IDF για να δημιουργήσει μια αριθμητική αναπαράσταση των περιγραφών των προϊόντων (vectors) και στη συνέχεια χρησιμοποιεί το cosine distance για την εύρεση παρόμοιων προϊόντων. Στην παρούσα διατριβή παρουσιάζεται ο σχεδιασμός, η υλοποίηση και η αξιολόγηση του Bestcart. Η αξιολόγηση αποδεικνύει ότι το Bestcart παρέχει ακριβή και έγκαιρη παρακολούθηση των τιμών λιανικής πώλησης και αποτελεσματικές συστάσεις προϊόντων, καθιστώντας το ένα χρήσιμο εργαλείο για τους καταναλωτές και τους λιανοπωλητές.

Λέξεις – Κλειδιά

Web Crawlers, web scrapers, παρακολούθηση τιμών, ενημέρωση τιμών, κατηγοριοποιήση προϊόντων, συσχετισμός προϊόντων

Abstract

In this thesis, I propose a system for tracking and controlling retail prices from large-scale heterogeneous data sources. The system aims to provide valuable insights to retailers and customers alike by tracking and comparing prices of products across different supermarkets and providing recommendations to customers.

Retail prices of products change frequently and unpredictably due to various factors such as supply and demand, promotions, and competition. Consumers and retailers alike can benefit from accurate and timely retail price tracking to make informed decisions about purchasing and pricing strategies. Additionally, product recommendations can enhance the user experience and increase sales.

Bestcart is a multilayer application that tracks retail prices and provides product recommendations. The application comprises a web crawler that collects data from multiple supermarkets using the Selenium library, a Flask-based API that stores the processed data in an SQLite database, and a frontend service that visualizes queries and provides a recommender system for product recommendations.

The recommender system uses TF-IDF to create a numerical representation of product descriptions and then employs cosine distance to find similar products. This thesis presents the design, implementation, and evaluation of Bestcart. The evaluation demonstrates that Bestcart provides accurate and timely retail price tracking and effective product recommendations, making it a useful tool for consumers and retailers.

Words - Keywords

Web Crawlers, web scrapers, price tracking, price updating, product categorization, product recommendation

Ευχαριστίες

Ευχαριστώ ιδιαίτερα τον καθηγητή κ. Παναγιώτη Τσανάκα για την ανάθεση και επίβλεψη αυτού του θέματος διπλωματικής εργασίας. Ευχαριστώ ιδιαίτερα και την οικογένεια μου και τους φίλους μου για την στήριξη, υλική και ηθική, κατά την διάρκεια των σπουδών μου.

Table of Contents

Κεφάλαιο 1: Web Crawlers: ένα σύστημα για την παρακολούθηση και τον έλεγχο των τιμών λιανικής από ετερογενείς πηγές δεδομένων μεγάλης κλίμακας

1.1 Εισαγωγή

Οι ανιχνευτές ιστού, γνωστοί και ως αράχνες ή bots, είναι αυτοματοποιημένα προγράμματα που διασχίζουν τον Παγκόσμιο Ιστό με μεθοδικό, αυτοματοποιημένο τρόπο. Χρησιμοποιούνται για την ανακάλυψη και την οργάνωση πληροφοριών στο διαδίκτυο, καθιστώντας τις εύκολα προσβάσιμες στους χρήστες μέσω των μηχανών αναζήτησης. Οι ανιχνευτές ιστού λειτουργούν ακολουθώντας συνδέσμους από τη μία ιστοσελίδα στην άλλη και στη συνέχεια αναζητούν τις πληροφορίες σε αυτές τις σελίδες ώστε να χρησιμοποιηθούν από τις μηχανές αναζήτησης. Η διαδικασία αυτή είναι γνωστή ως web crawling. Οι web crawlers αποτελούν βασικό συστατικό του σύγχρονου διαδικτύου, συμβάλλοντας στη γρήγορη και εύκολη πρόσβαση των χρηστών στις επιθυμητές πληροφορίες.

Η άρνηση πρόσβασης των σούπερ μάρκετ σε καθημερινά δεδομένα αποτελεί σημαντική πρόκληση. Μια λύση είναι η χρήση τεχνικών web crawling, καθώς είναι ιδιαίτερα ικανές στη γρήγορη συλλογή και επεξεργασία μεγάλων ποσοτήτων δεδομένων. Ωστόσο, θα πρέπει να σημειωθεί ότι, ενώ τα προγράμματα περιήγησης είναι αποτελεσματικά στην εκτέλεση JavaScript, στην εμφάνιση εικόνων και στη μορφοποίηση πληροφοριών με φιλικό προς τον χρήστη τρόπο, η αξιοποίηση των εσωτερικών διεπαφών προγραμματισμού εφαρμογών (API) που παρέχονται από τους ιστότοπους θα μπορούσε επίσης να διευκολύνει την απόκτηση δεδομένων. Παρ' όλα αυτά, είναι σημαντικό να ληφθεί υπόψη ότι οι διάφοροι ιστότοποι συχνά εφαρμόζουν μέτρα για τον περιορισμό της πρόσβασης σε αυτές τις εσωτερικές διεπαφές API. Ως αποτέλεσμα, το web scraping αναδεικνύεται ως η μόνη βιώσιμη λύση σε αυτό το πρόβλημα.

1.2 Διαθέσιμες Διεπαφές

Στον τομέα του web scraping και της εξαγωγής δεδομένων, υπάρχουν διάφορες βιβλιοθήκες που μπορούν να υλοποιηθούν με τη χρήση της γλώσσας προγραμματισμού Python. Ορισμένες από τις πιο συχνά χρησιμοποιούμενες βιβλιοθήκες περιλαμβάνουν το Scrapy [3], το Beautiful Soup [4], το Selenium [5] και το Requests-HTML [6] [2].

Το Scrapy, ένα ανοιχτού κώδικα και συνεργατικό πλαίσιο web scraping, επιτρέπει την αποτελεσματική εξαγωγή δεδομένων από ιστότοπους, καθώς και τη χρήση APIs. Το Beautiful Soup, μια βιβλιοθήκη Python για την ανάλυση εγγράφων HTML και XML, επιτρέπει τη δημιουργία δέντρων ανάλυσης για την εξαγωγή πληροφοριών από αυτούς τους τύπους εγγράφων. Το Selenium, ένα εργαλείο αυτοματοποίησης του προγράμματος περιήγησης, επιτρέπει την αυτοματοποίηση δραστηριοτήτων του προγράμματος περιήγησης, όπως το πάτημα κουμπιών, η συμπλήρωση φορμών και η πλοήγηση σε σελίδες. Τέλος, το Requests-HTML, μια βιβλιοθήκη ανάλυσης HTML, επιτρέπει την εκτέλεση εργασιών απόξεσης ιστού με τη χρήση της βιβλιοθήκης Requests της Python [7].

Το Scrapy είναι ένα ανοιχτού κώδικα και συνεργατικό πλαίσιο web scraping που επιτρέπει την αποτελεσματική εξαγωγή δεδομένων από ιστότοπους, καθώς και τη χρήση APIs. Διαθέτει

ενσωματωμένη υποστήριξη για τον χειρισμό κοινών εργασιών απόξεσης ιστού, όπως η σύνδεση, ο χειρισμός των cookies και η παρακολούθηση ανακατευθύνσεων. Το Scrapy επιτρέπει επίσης την εύκολη εξαγωγή δεδομένων σε πολλαπλές μορφές, όπως CSV, JSON ή XML. Η βιβλιοθήκη συντηρείται ενεργά και διαθέτει μια μεγάλη κοινότητα που μπορεί να παρέχει υποστήριξη και καθοδήγηση. Ωστόσο, μπορεί να έχει μια πιο απότομη καμπύλη εκμάθησης για όσους είναι αρχάριοι στο web scraping και μπορεί να μην είναι κατάλληλη για μικρές εργασίες [8].

Το Beautiful Soup είναι μια βιβλιοθήκη Python για την ανάλυση εγγράφων HTML και XML, η οποία επιτρέπει την εξαγωγή πληροφοριών από αυτούς τους τύπους εγγράφων μέσω δέντρων ανάλυσης. Είναι απλή στη χρήση και εύκολη στην εκμάθηση και διαθέτει μεγάλη κοινότητα που μπορεί να παρέχει υποστήριξη και καθοδήγηση. Ωστόσο, το Beautiful Soup αναλύει μόνο HTML και XML, δεν αλληλεπιδρά με ιστοσελίδες όπως το Selenium και μπορεί να μην είναι η καλύτερη επιλογή για την απόξεση μεγάλων και πολύπλοκων ιστοσελίδων [7].

Το Selenium είναι ένα εργαλείο αυτοματοποίησης του προγράμματος περιήγησης που επιτρέπει την αυτοματοποίηση δραστηριοτήτων του προγράμματος περιήγησης, όπως το πάτημα κουμπιών, η συμπλήρωση φορμών και η πλοήγηση σε σελίδες. Αυτό το καθιστά ιδανικό για την αλληλεπίδραση με δυναμικές ιστοσελίδες. Ωστόσο, το Selenium μπορεί να απαιτεί περισσότερες τεχνικές γνώσεις και ρυθμίσεις σε σύγκριση με άλλες βιβλιοθήκες και μπορεί να είναι λιγότερο αποτελεσματικό για web scraping μεγάλης κλίμακας [7].

Το Requests-HTML είναι μια βιβλιοθήκη ανάλυσης HTML που επιτρέπει την εκτέλεση εργασιών απόξεσης ιστού με τη χρήση της βιβλιοθήκης Requests της Python. Είναι απλή στη χρήση, αλλά έχει περιορισμένες δυνατότητες σε σύγκριση με άλλες βιβλιοθήκες, χρησιμοποιείται κυρίως για μικρές εργασίες web scraping και μπορεί να μην είναι κατάλληλη για πιο σύνθετες εργασίες απόξεσης που απαιτούν αυτοματοποίηση του προγράμματος περιήγησης ή χειρισμό των cookies και των ανακατευθύνσεων.

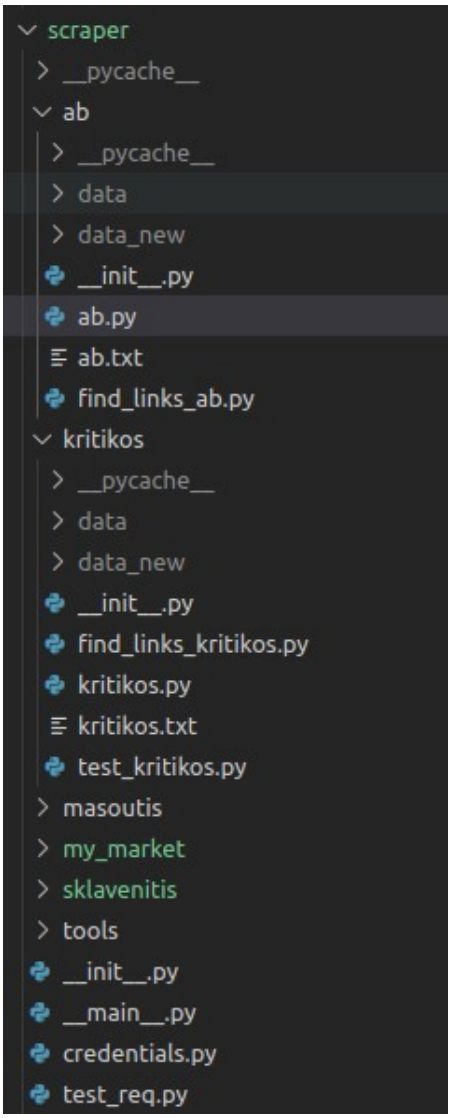
Για τους σκοπούς της διατριβής μου, αποφάσισα να χρησιμοποιήσω το Selenium ως το κύριο εργαλείο για το web scraping. Οι ιστότοποι των σούπερ μάρκετ που σκοπεύω να σαρώσω απαιτούν φόρτωση σελίδων με scrolling και κάνουν χρήση JavaScript, καθώς και απόδοση από την πλευρά του διακομιστή. Αυτά τα χαρακτηριστικά μπορούν να καταστήσουν το web scraping πιο δύσκολη και οι παραδοσιακές μέθοδοι του web scraping μπορεί να μην είναι επαρκείς. Το Selenium, ως εργαλείο αυτοματοποίησης του προγράμματος περιήγησης, επιτρέπει την αυτοματοποίηση δραστηριοτήτων του προγράμματος περιήγησης, όπως η φόρτωση σελίδων με scrolling, η αλληλεπίδραση με JavaScript και ο χειρισμός δυναμικών ιστοσελίδων. Αυτό το καθιστά ιδανική επιλογή για το web scraping που διαθέτουν αυτά τα χαρακτηριστικά, καθώς μιμείται τις ενέργειες ενός χρήστη και μπορεί να παρακάμψει τυχόν περιορισμούς που μπορεί να επιβάλλονται από τις παραδοσιακές μεθόδους web scraping. Επιπλέον, το Selenium διαθέτει μεγάλη κοινότητα και πληθώρα διαθέσιμων πόρων, οι οποίοι μπορούν να είναι εξαιρετικά χρήσιμοι για την αντιμετώπιση τυχόν προβλημάτων που μπορεί να προκύψουν κατά τη διαδικασία απόξεσης.

1.3 Γενικοί Ανιχνευτές Ιστού (Web Crawlers)

Η ανάγκη για γενικούς ανιχνευτές ιστού προκύπτει από την απαίτηση για συλλογή δεδομένων από πολλαπλές πηγές με ελάχιστη προσπάθεια. Στην περίπτωσή μου, καθώς εργάζομαι σε μια διπλωματική εργασία που περιλαμβάνει την αποκομή πληροφοριών προϊόντων από σούπερ μάρκετ, χρειάζομαι έναν web crawler που μπορεί εύκολα να ρυθμιστεί και να προσαρμοστεί ώστε να λειτουργεί εύκολα και με διαφορετικά σούπερ μάρκετ. Καθώς κάθε σούπερ μάρκετ έχει τη δική του δομή ιστοτόπου και τη δική

του μορφή δεδομένων, η δημιουργία ενός νέου προγράμματος ανίχνευσης για κάθε σούπερ μάρκετ θα ήταν χρονοβόρα και απαιτητική σε πόρους. Ένας γενικός ανιχνευτής ιστού μπορεί να προσαρμοστεί ώστε να λειτουργεί με διαφορετικούς ιστότοπους αλλάζοντας τις διαμορφώσεις, τους κανόνες και τα μοτίβα ώστε να ταιριάζουν με τον ιστότοπο-στόχο. Αυτή η προσέγγιση μου επιτρέπει να προσθέτω εύκολα νέες υπεραγορές στη διαδικασία απόξεσης χωρίς να χρειάζεται να δημιουργήσω νέους ανιχνευτές από την αρχή. Επιπλέον, η χρήση ενός γενικού web crawler μπορεί να με βοηθήσει να διασφαλίσω τη συνοχή και την ακρίβεια των δεδομένων που συλλέγονται από τις διάφορες υπεραγορές.

Προκειμένου να επιτύχω μια συνεπή και οργανωμένη προσέγγιση για την απόκτηση δεδομένων από πολλαπλά σούπερ μάρκετ, έχω εφαρμόσει μια συγκεκριμένη μορφή για κάθε σούπερ μάρκετ. Συγκεκριμένα, δημιούργησα έναν φάκελο "scraper", εντός του οποίου δημιούργησα έναν ξεχωριστό φάκελο για κάθε σούπερ μάρκετ. Μέσα σε κάθε έναν από αυτούς τους φακέλους, έχω συμπεριλάβει ένα αρχείο απόξεσης που αφορά ειδικά το συγκεκριμένο σούπερ μάρκετ, ένα αρχείο που περιέχει τους συνδέσμους προς scraping, ένα αρχείο ".txt" για την αποθήκευση σχετικών πληροφοριών και έναν φάκελο για την αποθήκευση δεδομένων του παρελθόντος. Αυτή η μορφή επιτρέπει την εύκολη οργάνωση και πρόσβαση στα δεδομένα που σαρώνονται από κάθε σούπερ μάρκετ και εξασφαλίζει ότι η διαδικασία προσθήκης νέων σούπερ μάρκετ είναι απλοποιημένη και αποτελεσματική.



Εικόνα 1.1 Generic web crawlers format

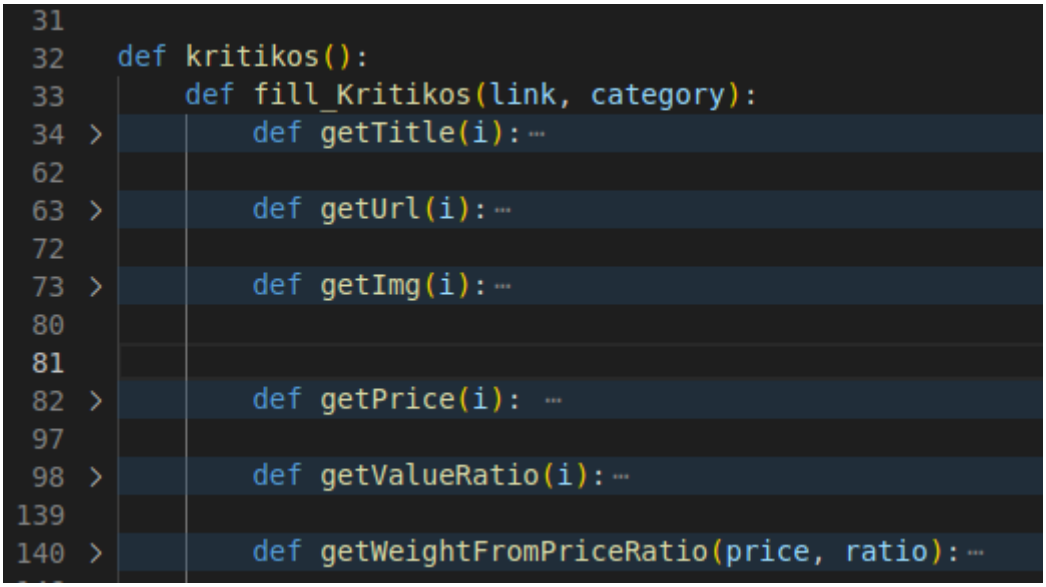
Έχω υλοποιήσει μια γενική συνάρτηση, με το όνομα "get_categories_{supermarket}", μέσα στο αρχείο "find_links", η οποία είναι ειδικά σχεδιασμένη για να χρησιμοποιεί το Selenium με σκοπό την πλοήγηση στη σελίδα του ηλεκτρονικού καταστήματος ενός συγκεκριμένου σούπερ μάρκετ και την απόξεση των συνδέσμων και των κατηγοριών αυτού του σούπερ μάρκετ. Η λειτουργία αυτή επιτρέπει την αποτελεσματική ανάκτηση των σχετικών δεδομένων και διατηρεί επίσης μια λίστα που περιέχει όλα τα δεδομένα που έχουν αποκομηθεί, εξαλείφοντας έτσι την ανάγκη επανειλημμένης ανάκτησης των δεδομένων αυτών. Η προσέγγιση αυτή διασφαλίζει ότι η διαδικασία αποκομής των συνδέσμων και των κατηγοριών ενός σούπερ μάρκετ είναι τόσο αποτελεσματική όσο και ακριβής.

```
14  class_conventions = {
15      'title': "ProductListItem_title__e6MEz",
16      'product': "ProductListItem_productItem__cKUyG",
17      "additional_info": "ProductListItem_titleDesc__JzvBv",
18
19      'price': 'ProductListItem_finalPrice__sEMjs',
20      "beginPrice": "ProductListItem_beginPrice__vK_Dk",
21
22      "weight": "ProductListItem_titleDesc__JzvBv",
23      'ratio': 'ProductListItem_description__DRAGa',
24      'img': "ProductListItem_productImage__HbseK",
25      'url': "ProductListItem_productLink__BZo3P",
26  }
```

Εικόνα 1.2 Ενα βασικό “class_conventions” dictionary των HTML & CSS attributes

Προκειμένου να μεγιστοποιήσω την παραγωγικότητα και την αποτελεσματικότητα κατά τη δημιουργία web crawlers για κάθε σούπερ μάρκετ, έχω αναπτύξει μια βασική δομή που πρέπει να ακολουθήσετε. Αυτή η δομή περιλαμβάνει τη δημιουργία ενός νέου αντικειμένου "class_convention" για κάθε νέο scraper, το οποίο περιλαμβάνει όλα τα απαραίτητα χαρακτηριστικά HTML και CSS για την απόκτηση των επιθυμητών δεδομένων. Συγκεκριμένα, η δομή αυτή περιλαμβάνει το όνομα του προϊόντος, την τιμή, την αρχική τιμή εάν το προϊόν είναι σε έκπτωση, το βάρος (εάν ισχύει), την αναλογία τιμής-αξίας, τη διεύθυνση URL της εικόνας (καθώς κάθε σούπερ μάρκετ φορτώνει τις εικόνες του μέσω ενός API) και τη διεύθυνση URL του προϊόντος. Αυτή η προσέγγιση επιτρέπει μια συνεπή και αποτελεσματική μέθοδο απόκτησης δεδομένων από πολλά σούπερ μάρκετ, ενώ παράλληλα διευκολύνει την προσθήκη νέων σούπερ μάρκετ στη διαδικασία αποκομής.

Επιπλέον, κάθε πρόγραμμα ανίχνευσης ιστού χρησιμοποιεί μια συγκεκριμένη διεπαφή προγραμματισμού εφαρμογών (API) η οποία, μέσω της χρήσης του webdriver, εκτελεί εντολές JavaScript προκειμένου να ανακτήσει τα δεδομένα. Η μέθοδος αξιοποίησης αυτών των λειτουργιών περιλαμβάνει αρχικά τον προσδιορισμό του αριθμού των προϊόντων που υπάρχουν σε μια ιστοσελίδα και στη συνέχεια την επαναληπτική εξέταση κάθε προϊόντος για την εξαγωγή των χαρακτηριστικών του. Αυτή η προσέγγιση επιτρέπει μια αποτελεσματική και ολοκληρωμένη μέθοδο ανάκτησης δεδομένων, ενώ παράλληλα διασφαλίζει ότι η διαδικασία είναι αυτοματοποιημένη και βελτιστοποιημένη.



Εικόνα 1.3 Basic web crawler's API

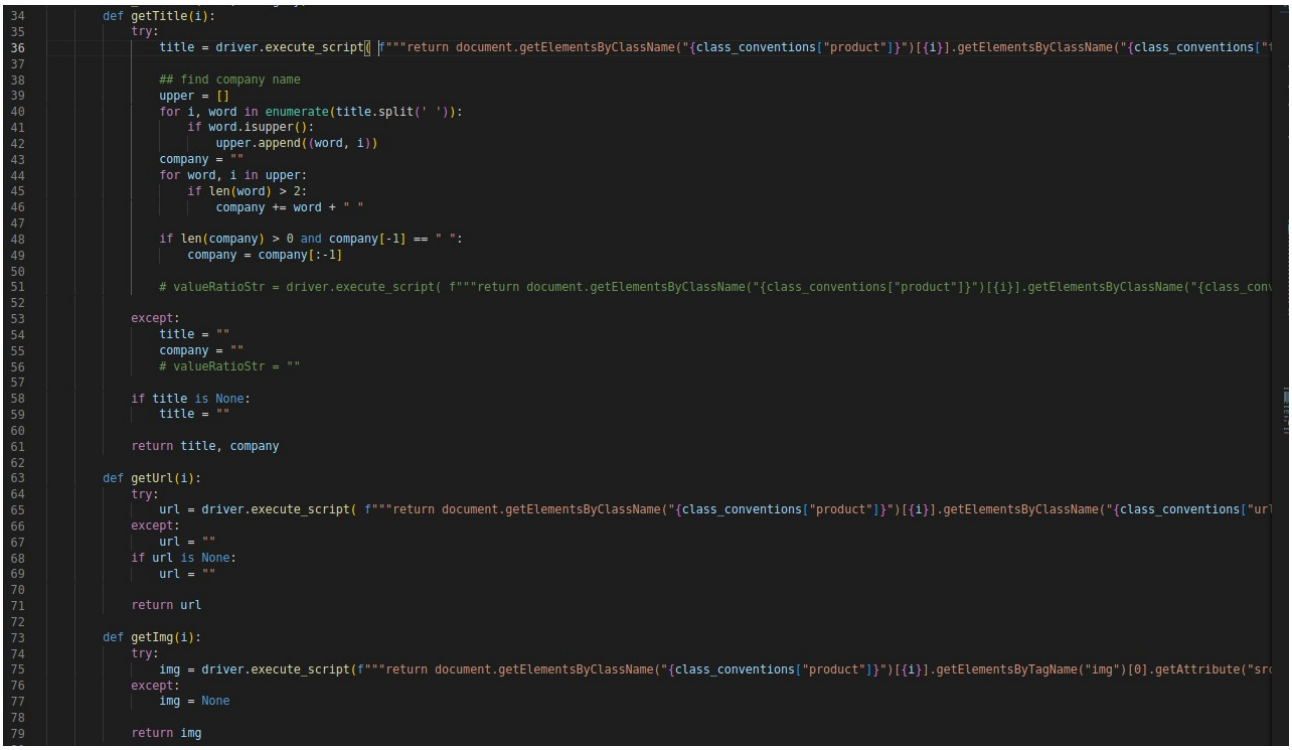
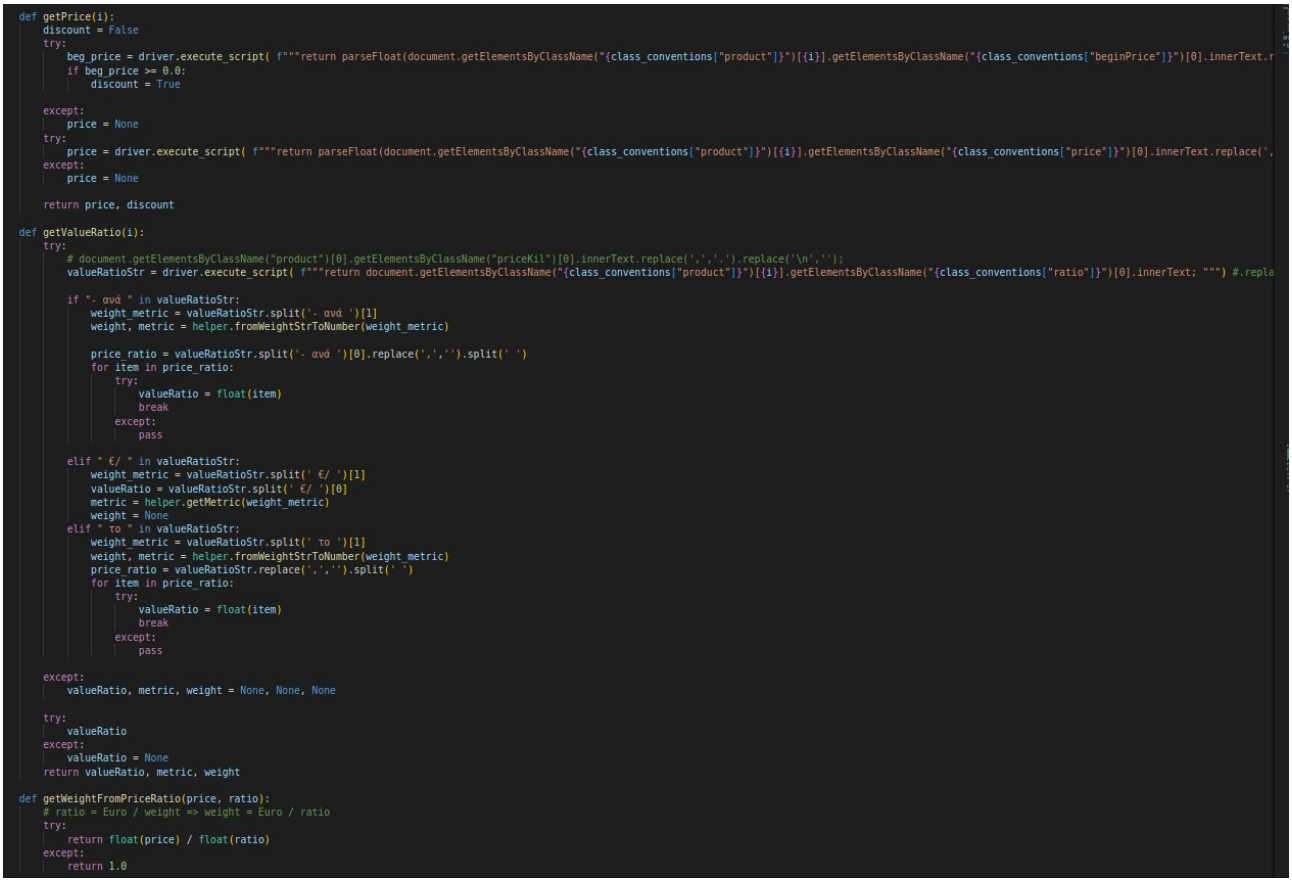


Image 1.3.1 Implementation

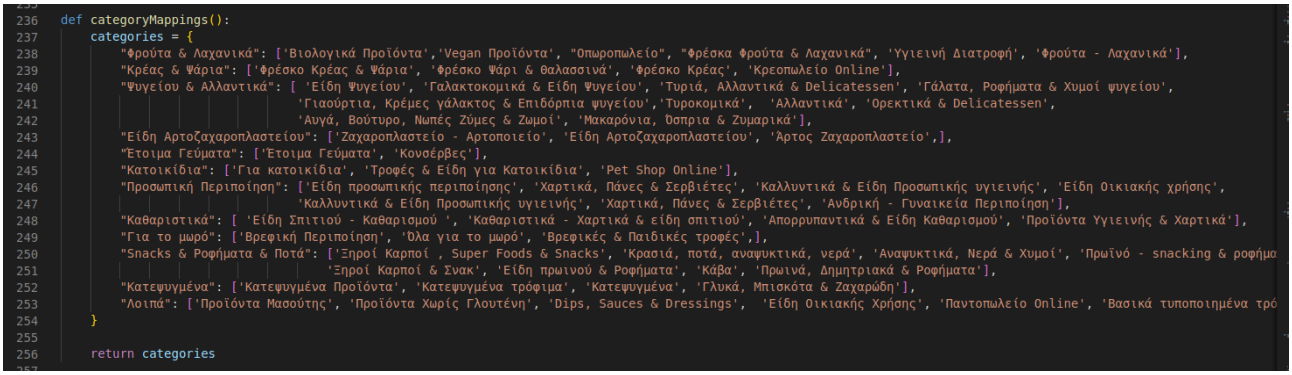


```
def getPrice(i):
    discount = False
    try:
        beg_price = driver.execute_script( f"""return parseFloat(document.getElementsByClassName("{class_conventions["product"]}")[{i}].getElementsByClassName("{class_conventions["beginPrice"]}")[0].innerText.r
        if beg_price >= 0.0:
            discount = True

    except:
        price = None
    try:
        price = driver.execute_script( f"""return parseFloat(document.getElementsByClassName("{class_conventions["product"]}")[{i}].getElementsByClassName("{class_conventions["price"]}")[0].innerText.replace(',
    except:
        price = None

    return price, discount

def getValueRatio(i):
    try:
        # document.getElementsByClassName("product")[0].getElementsByClassName("priceKil")[0].innerText.replace(',',',').replace('\n','');
        valueRatioStr = driver.execute_script( f"""return document.getElementsByClassName("{class_conventions["product"]}")[{i}].getElementsByClassName("{class_conventions["ratio"]}")[0].innerText; """) #.repla

        if "- ανά " in valueRatioStr:
            weight_metric = valueRatioStr.split('- ανά ')[1]
            weight, metric = helper.fromWeightStrToNumber(weight_metric)

            price_ratio = valueRatioStr.split('- ανά ')[0].replace(',','').split(' ')
            for item in price_ratio:
                try:
                    valueRatio = float(item)
                    break
                except:
                    pass

        elif " €/ " in valueRatioStr:
            weight_metric = valueRatioStr.split(' €/ ')[1]
            valueRatio = valueRatioStr.split(' €/ ')[0]
            metric = helper.getMetric(weight_metric)
            weight = None
        elif " το " in valueRatioStr:
            weight_metric = valueRatioStr.split(' το ')[1]
            weight, metric = helper.fromWeightStrToNumber(weight_metric)
            price_ratio = valueRatioStr.replace(',','').split(' ')
            for item in price_ratio:
                try:
                    valueRatio = float(item)
                    break
                except:
                    pass

    except:
        valueRatio, metric, weight = None, None, None

    try:
        valueRatio
    except:
        valueRatio = None
    return valueRatio, metric, weight

def getWeightFromPriceRatio(price, ratio):
    # ratio = Euro / weight => weight = Euro / ratio
    try:
        return float(price) / float(ratio)
    except:
        return 1.0
```

Image 1.3.2 Implementation

Μια άλλη πρόκληση που προκύπτει κατά την ανάκτηση δεδομένων από πολλαπλά σούπερ μάρκετ είναι η ποικιλία των κατηγοριών που δηλώνει κάθε σούπερ μάρκετ. Για να το αντιμετωπίσω αυτό, δημιούργησα ένα σύστημα αντιστοίχισης της κατηγορίας κάθε σούπερ μάρκετ σε μια γενική κατηγορία εντός της εφαρμογής μου. Η προσέγγιση αυτή επιτρέπει τη συνεπή κατηγοριοποίηση των προϊόντων και διευκολύνει την ανάλυση των δεδομένων που έχουν ανακτηθεί με απόξεση (crawling), αποτελώντας έτσι μία αποτελεσματική και ολοκληρωμένη μέθοδο ανάκτησης δεδομένων, διασφαλίζοντας επιπλέον τον αυτοματοποιημένο χαρακτήρα της.

Οι αντιστοιχίσεις κατηγοριών που έχουν υλοποιηθεί είναι οι εξής:



```
def categoryMappings():
    categories = {
        "Φρούτα & Λαχανικά": ['Βιολογικά Προϊόντα','Vegan Προϊόντα', "Οπωροπωλείο", "Φρέσκα Φρούτα & Λαχανικά", 'Υγιεινή Διατροφή', 'Φρούτα - Λαχανικά'],
        "Κρέας & Ψάρια": ['Φρέσκο Κρέας & Ψάρια', 'Φρέσκο Ψάρι & Θαλασσινά', 'Φρέσκο Κρέας', 'Κρεοπωλείο Online'],
        "Ψυγείου & Αλλαντικά": [ 'Είδη Ψυγείου', 'Γαλακτοκομικά & Είδη Ψυγείου', 'Τυριά, Αλλαντικά & Delicatessen', 'Γάλατα, Ροφήματα & Χυμοί ψυγείου',
                                 'Γιαούρτια, Κρέμες γάλακτος & Επιδόρπια ψυγείου','Τυροκομικά',  'Αλλαντικά', 'Ορεκτικά & Delicatessen',
                                 'Αυγά, Βούτυρο, Νωπές Ζύμες & Ζωμοί', 'Μακαρόνια, Όσπρια & Ζυμαρικά'],
        "Είδη Αρτοζαχαροπλαστείου": ['Ζαχαροπλαστείο - Αρτοποιείο', 'Είδη Αρτοζαχαροπλαστείου', 'Άρτος Ζαχαροπλαστείο',],
        "Έτοιμα Γεύματα": ['Έτοιμα Γεύματα', 'Κονσέρβες'],
        "Κατοικίδια": ['Για κατοικίδια', 'Τροφές & Είδη για Κατοικίδια', 'Pet Shop Online'],
        "Προσωπική Περιποίηση": ['Είδη προσωπικής περιποίησης', 'Χαρτικά, Πάνες & Σερβιέτες', 'Καλλυντικά & Είδη Προσωπικής υγιεινής', 'Είδη Οικιακής χρήσης',
                                 'Καλλυντικά & Είδη Προσωπικής υγιεινής', 'Χαρτικά, Πάνες & Σερβιέτες', 'Ανδρική - Γυναικεία Περιποίηση'],
        "Καθαριστικά": [ 'Είδη Σπιτιού - Καθαρισμού ', 'Καθαριστικά - Χαρτικά & είδη σπιτιού', 'Απορρυπαντικά & Είδη Καθαρισμού', 'Προϊόντα Υγιεινής & Χαρτικά'],
        "Για το μωρό": ['Βρεφική Περιποίηση', 'Όλα για το μωρό', 'Βρεφικές & Παιδικές τροφές',],
        "Snacks & Ροφήματα & Ποτά": ['Ξηροί Καρποί , Super Foods & Snacks', 'Κρασιά, ποτά, αναψυκτικά, νερά', 'Αναψυκτικά, Νερά & Χυμοί', 'Πρωϊνό - snacking & ροφήμα
                                     'Ξηροί Καρποί & Σνακ', 'Είδη πρωινού & Ροφήματα', 'Κάβα', 'Πρωινά, Δημητριακά & Ροφήματα'],
        "Κατεψυγμένα": ['Κατεψυγμένα Προϊόντα', 'Κατεψυγμένα τρόφιμα', 'Κατεψυγμένα', 'Γλυκά, Μπισκότα & Ζαχαρώδη'],
        "Λοιπά": ['Προϊόντα Μασούτης', 'Προϊόντα Χωρίς Γλουτένη', 'Dips, Sauces & Dressings',  'Είδη Οικιακής Χρήσης', 'Παντοπωλείο Online', 'Βασικά τυποποιημένα τρό
    }

    return categories
```

Image 1.4 Category mappings

Προκειμένου να αποθηκεύσω και να οργανώσω αποτελεσματικά τα δεδομένα των προϊόντων, ανέπτυξα μια σειρά από κλάσεις για τη διαχείριση των αντικειμένων με δομημένο τρόπο. Συγκεκριμένα, δημιούργησα μια κλάση "Product" που μου επιτρέπει να χειρίζομαι το αντικείμενο, αντί

να βασίζομαι σε μοναδικές μεταβλητές για κάθε προϊόν. Αυτό επιτρέπει τον εύκολο χειρισμό των δεδομένων και διευκολύνει τη διαδικασία ενημέρωσης της βάσης δεδομένων. Για κάθε προϊόν, δημιουργώ ένα αντικείμενο "Product", το αποθηκεύω σε μια λίστα και ταυτόχρονα το γράφω σε ένα αρχείο. Αυτό το αρχείο χρησιμοποιείται στη συνέχεια για την ενημέρωση της βάσης δεδομένων, εξασφαλίζοντας έτσι ότι η διαδικασία είναι αυτοματοποιημένη και αποτελεσματική. Αυτή η προσέγγιση επιτρέπει μια συνεπή και οργανωμένη μέθοδο αποθήκευσης δεδομένων προϊόντων, ενώ παράλληλα διευκολύνει την ανάλυση και την ανάκτηση των δεδομένων.

Ένα άλλο ζήτημα που προκύπτει είναι ότι κάθε σούπερ μάρκετ φορτώνει τα προϊόντα του με δύο διαφορετικούς τρόπους. Ορισμένοι επιλέγουν να χρησιμοποιούν σελίδες ως τρόπο φόρτωσης μιας παρτίδας προϊόντων. Άλλοι επιλέγουν να φορτώνουν περισσότερα προϊόντα καθώς ο χρήστης κάνει scrolling. Για την πρώτη περίπτωση, προκειμένου να το διευκολύνω αυτό, έχω σχεδιάσει μια σειρά από εντολές Javascript για να βρω το συνολικό αριθμό των σελίδων, ώστε να μπορώ να μπω σε κάθε μία από αυτές. Για τη δεύτερη περίπτωση, ορισμένα σούπερ μάρκετ προσθέτουν τον συνολικό αριθμό των προϊόντων στο πάνω μέρος της σελίδας. Αν αυτό δεν υπάρχει, κάνουμε κύλιση μέχρι να μην φορτωθούν άλλα δεδομένα, αλλιώς, το πρόγραμμα περιήγησης στο διαδίκτυο κυλάει και φορτώνει μέχρι να βρει αυτόν τον αριθμό προϊόντων. Ωστόσο, λόγω κάποιων σφαλμάτων στις ιστοσελίδες τους, ο αριθμός αυτός μπορεί να είναι ψευδής. Για το λόγο αυτό, σε κάθε περίπτωση έχω εφαρμόσει ένα κανόνα ασφαλείας, σύμφωνα με τον οποίο αν ο webdriver κάνει κύλιση 5 φορές πάνω κάτω και δεν εμφανιστούν νέα προϊόντα, έχει φτάσει στο τέλος της σελίδας. Και εκτυπώνει ένα μήνυμα για να με ειδοποιήσει ότι μπορεί να έχουν χαθεί κάποια προϊόντα.

Προκειμένου να ενσωματώσω αποτελεσματικά όλα τα προαναφερθέντα στοιχεία και να διασφαλίσω την αποτελεσματική και ακριβή απόξεση δεδομένων, έχω αναπτύξει μια βασική δομή για τον web crawler. Η δομή ξεκινά με τον προσδιορισμό του μηχανισμού φόρτωσης της σελίδας, αν πρόκειται για μηχανισμό φόρτωσης με κύλιση ή όχι. Εάν η σελίδα χρησιμοποιεί το μηχανισμό φόρτωσης κύλισης, το πρώτο βήμα είναι η κύλιση μέχρι το τέλος της σελίδας, ενώ σε διαφορετική περίπτωση, η διαδικασία προχωρά στο επόμενο βήμα. Στη συνέχεια υλοποιείται μια διαδικασία επανάληψης μέσω του συνολικού αριθμού των προϊόντων, όπου αποκτώνται τα βασικά δεδομένα για κάθε προϊόν. Για κάθε προϊόν δημιουργείται ένα αντικείμενο "Product" το οποίο εγγράφεται σε ένα αρχείο με τη δυνατότητα να χρησιμοποιηθεί για μεταγενέστερη ανάλυση. Αυτή η προσέγγιση επιτρέπει μια αυτοματοποιημένη και βελτιστοποιημένη διαδικασία απόξεσης δεδομένων, ενώ παράλληλα διασφαλίζει ότι τα δεδομένα είναι οργανωμένα και προσβάσιμα για περαιτέρω ανάλυση.

1.4 Μεγιστοποίηση της Απόδοσης των Web Crawlers

Η καθυστέρηση του δικτύου είναι ένα πρωταρχικό εμπόδιο που μπορεί να εμποδίσει την απόδοση οποιουδήποτε έργου web scraping. Τόσο η μετάδοση ενός αιτήματος στον διακομιστή ιστού όσο και η επακόλουθη λήψη μιας απάντησης οδηγούν σε καθυστέρηση. Αν και υπάρχουν περιορισμοί στο τι μπορεί να γίνει για την επίλυση αυτού του προβλήματος, όπως η ύπαρξη καλύτερης σύνδεσης ή κάρτας δικτύου, υπάρχουν διάφοροι τρόποι για την ενίσχυση της απόδοσης.

Μια τέτοια μέθοδος είναι η χρήση headless προγραμμάτων περιήγησης, το οποίο υποστηρίζεται από το Selenium. Σε ορισμένες περιπτώσεις, ένα headless πρόγραμμα περιήγησης μπορεί να μην φέρνει τα επιθυμητά αποτελέσματα, καθώς ορισμένοι ιστότοποι χρησιμοποιούν ελέγχους από ανιχνευτές ιστού και δεν θα φορτώσουν τη σελίδα εάν δεν εμφανίζεται κάποια οθόνη. Ωστόσο, όταν είναι δυνατή η χρήση ενός προγράμματος περιήγησης χωρίς κεφαλή, μπορεί να υπάρξει εξοικονόμηση μέχρι του μισού χρόνου.

Επιπλέον, η χρήση πολλαπλών νημάτων (threads) μπορεί επίσης να βελτιώσει την απόδοση, επιτρέποντας σε πολλούς ανιχνευτές ιστού να εκτελούνται ανεξάρτητα [9]. Αυτή η μέθοδος είναι αποτελεσματική μόνο εάν η CPU είναι σε θέση να χειριστεί τα πολλαπλά νήματα Selenium που εκτελούνται ταυτόχρονα. Στον υπολογιστή μου, επιτεύχθηκε σημαντική μείωση του χρόνου μέσω της εφαρμογής πολυνηματικότητας. Χωρίς πολυνηματικότητα, ο χρόνος που απαιτούνταν για την απόξεση όλων των δεδομένων από όλους τους δικτυακούς τόπους ήταν περίπου έξι ώρες, ενώ με μια αμιγώς πολυνηματική ρύθμιση, ο χρόνος μειώθηκε σε μιάμιση με δύο ώρες. Ακόμη μεγαλύτερη μείωση του απαιτούμενου χρόνου μπορεί να επιτευχθεί με το διαχωρισμό των συνδέσμων ενός σούπερ μαρκετ σε δύο ίσες λίστες έτσι ώστε κάθε λίστα να τρέχει σε διαφορετικό thread, με αποτέλεσμα τη μείωση του χρόνου κατά το ήμισυ. Στην περίπτωσή μου, αυτό μείωσε τον χρόνο από μιάμιση ώρα σε 30-45 λεπτά, ανάλογα με τη σύνδεση.

Τέλος, είναι σημαντικό να σημειωθεί ότι κατά τη χρήση πολλαπλών νημάτων, πολλά νήματα μπορεί να επιχειρήσουν να γράψουν στο ίδιο αρχείο ταυτόχρονα. Για να διασφαλίσω την ασφάλεια και την ακεραιότητα του συστήματος, εφάρμοσα spinlocks. Ένα spinlock είναι ένας μηχανισμός συγχρονισμού που χρησιμοποιείται για να κλειδώνει ένα αρχείο όταν ένα νήμα επιχειρεί να γράψει σε αυτό και στη συνέχεια το ξεκλειδώνει όταν ολοκληρωθεί η διαδικασία εγγραφής. Αυτή η προσέγγιση διασφαλίζει ότι μόνο ένα νήμα είναι σε θέση να έχει πρόσβαση στο αρχείο κάθε φορά, αποφεύγοντας πιθανές συγκρούσεις ή σφάλματα που μπορεί να προκύψουν λόγω ταυτόχρονης πρόσβασης.

1.5 Μελλοντικές Βελτιώσεις

Μια προσέγγιση για τη βελτίωση της απόδοσης των web scrapers είναι η χρήση απομακρυσμένων διακομιστών για την απόξεση. Με την αξιοποίηση των πόρων των απομακρυσμένων διακομιστών, οι web scrapers μπορούν να αυξήσουν την ταχύτητα απόξεσης, να χειριστούν περισσότερα αιτήματα ταυτόχρονα και να μειώσουν τον κίνδυνο να μπλοκαριστούν από τους web servers [1]. Αυτή η προσέγγιση περιλαμβάνει την εκτέλεση των web scrapers σε διακομιστές που δεν βρίσκονται στο ίδιο δίκτυο με το μηχάνημα που εκτελεί το scraping script. Αυτό μπορεί να επιτευχθεί είτε με τη μίσθωση ενός εικονικού ιδιωτικού διακομιστή (VPS) είτε με τη χρήση μιας υπηρεσίας που βασίζεται στο νέφος, όπως η Amazon Web Services (AWS) ή η Google Cloud Platform (GCP).

Επιπλέον, η χρήση απομακρυσμένων διακομιστών επιτρέπει τη χρήση πολλαπλών διευθύνσεων IP, μειώνοντας έτσι τις πιθανότητες εντοπισμού και αποκλεισμού από τους διαχειριστές ιστότοπων, επιτρέποντας επιπλέον τη χρήση πολλαπλών νημάτων, αυξάνοντας έτσι την ταχύτητα απόξεσης. Ωστόσο, είναι σημαντικό να σημειωθεί ότι η απόξεση σε απομακρυσμένους διακομιστές συνοδεύεται επίσης από ορισμένα μειονεκτήματα, όπως το αυξημένο κόστος και η ανάγκη για αξιόπιστη σύνδεση στο διαδίκτυο.

Κεφάλαιο 2: Back-end Υπηρεσία

2.1 Επιλογή Βάσης Δεδομένων και Δομή

Η επιλογή του κατάλληλου μοντέλου βάσης δεδομένων μπορεί να αποτελέσει σημαντική πρόκληση, καθώς υπάρχει πληθώρα διαθέσιμων επιλογών. Οι επιλογές αυτές περιλαμβάνουν βάσεις δεδομένων βασισμένες σε SQL, βάσεις δεδομένων NoSQL και βάσεις δεδομένων γράφων, καθεμία από τις οποίες έχει σχεδιαστεί για την αντιμετώπιση συγκεκριμένων τύπων προκλήσεων. Στο πλαίσιο του παρόντος έργου, ο φόρτος δεδομένων είναι σχετικά χαμηλός, με συνολικά 70.000 διακριτά προϊόντα και τα σχετικά δεδομένα τους. Επιπλέον, η ανάγκη για σχέσεις στα δεδομένα, όπως αυτές που απαιτούνται για τις προτάσεις, δεν είναι ιδιαίτερα απαιτητική. Λαμβάνοντας υπόψη αυτές τις εκτιμήσεις, μια απλή βάση δεδομένων SQLite θεωρείται επαρκής για τις ανάγκες του παρόντος έργου.

Επιπλέον, μια υπηρεσία ιστού είναι απαραίτητη για τον χειρισμό των ερωτημάτων SQL και την παροχή ενός μέσου αλληλεπίδρασης με τη βάση δεδομένων. Δεδομένου του αναμενόμενου χαμηλού όγκου αιτημάτων HTTP, μια απλή εφαρμογή Flask θεωρείται επαρκής για το σκοπό αυτό, καθώς προσφέρει μια ελαφριά και αποτελεσματική λύση για το χειρισμό αυτών των τύπων αιτημάτων.

Οι εφαρμογές Flask είναι εξοπλισμένες με διάφορα εργαλεία που μπορούν να αξιοποιηθούν για τη βελτίωση της λειτουργικότητάς τους. Ένα τέτοιο εργαλείο είναι η SQLAlchemy, μια βιβλιοθήκη Αντικειμενοσχεσιακής Αντιστοίχισης (ORM) που επιτρέπει την αλληλεπίδραση με βάσεις δεδομένων με πιο αποτελεσματικό και βολικό τρόπο. Η SQLAlchemy είναι μια κοινή επιλογή για τους προγραμματιστές όταν πρόκειται να εργαστούν με βάσεις δεδομένων σε εφαρμογές Flask.

Η αξιοποίηση της SQLAlchemy επιτρέπει τη δημιουργία μιας σχεσιακής αντιστοίχισης μεταξύ κλάσεων Python και κώδικα SQL. Αυτό επιτρέπει τον χειρισμό της βάσης δεδομένων μέσω της χρήσης κλάσεων Python, αντί της συγγραφής ακατέργαστου κώδικα SQL, με αποτέλεσμα μια πιο αποτελεσματική και βολική διαδικασία ανάπτυξης.

Τέλος, είναι ένας ασφαλής τρόπος για την προστασία των δεδομένων από επιθέσεις όπως η SQL Injection. Η SQLAlchemy αποτρέπει την έγχυση SQL χρησιμοποιώντας μια τεχνική που ονομάζεται "δέσμευση παραμέτρων". Αυτή η τεχνική περιλαμβάνει το διαχωρισμό των δεδομένων από το ερώτημα SQL και τη διαβίβαση των δεδομένων ως ξεχωριστή παράμετρο. Με αυτόν τον τρόπο, κάθε δεδομένο που περνάει στο ερώτημα αντιμετωπίζεται ως δεσμευμένη μεταβλητή και όχι ως μέρος του ίδιου του ερωτήματος SQL. Αυτό διασφαλίζει ότι οποιαδήποτε δεδομένα που περνούν στο ερώτημα, ανεξάρτητα από το περιεχόμενό τους, αντιμετωπίζονται ως τιμή και όχι ως μέρος του ερωτήματος SQL, εξουδετερώνοντας αποτελεσματικά τυχόν απόπειρες SQL Injection. Επιπλέον, η SQLAlchemy παρέχει έναν τρόπο για τη χρήση "sql expressions" αντί της απλής συνένωσης συμβολοσειρών, η οποία βοηθά στην αποτροπή του SQL Injection.

Για παράδειγμα, όταν χρησιμοποιείται η απλή συνένωση συμβολοσειρών, μια είσοδος χρήστη προστίθεται στη συμβολοσειρά ερωτήματος, ωστόσο, όταν χρησιμοποιείται η SQLAlchemy, η είσοδος χρήστη περνάει ως ξεχωριστή παράμετρος στο ερώτημα και δεν προστίθεται στη συμβολοσειρά ερωτήματος. Έτσι, ακόμη και αν η είσοδος χρήστη περιέχει κάποιο κακόβουλο κώδικα SQL, δεν θα εκτελεστεί επειδή δεν αποτελεί μέρος της συμβολοσειράς ερωτήματος.

2.1.1 Πίνακας Προϊόντων

```python
class Product(Base):
    __tablename__ = 'product'

    id = db.Column(db.Integer, primary_key=True, unique=True)

    supermarket = db.Column(db.String)
    supermarket_english = db.Column(db.String)

    product = db.Column(db.String, nullable=False)
    product_english = db.Column(db.String)

    company = db.Column(db.String)
    company_english = db.Column(db.String)

    category = db.Column(db.String)
    category_english = db.Column(db.String)

    metric = db.Column(db.String)

    _prices = db.Column(db.String)
    _datesUpdated = db.Column(db.String)

    value_ratio = db.Column(db.Float)
    weight = db.Column(db.Float)

    img = db.Column(db.String)
    url = db.Column(db.String)

    discount = db.Column(db.Boolean)

    exist = db.Column(db.Boolean)
```

Image 2.1.1.1 Product class: Basic attributes

Στην εφαρμογή αυτή, έχουν χρησιμοποιηθεί ορισμένες μεταβλητές με αυτονόητες περιγραφές. Μια αξιοσημείωτη πρόκληση που αντιμετωπίστηκε κατά τη διαδικασία ανάπτυξης ήταν η αποθήκευση των ιστορικών δεδομένων τιμολόγησης για κάθε προϊόν. Για την αντιμετώπιση αυτής της πρόκλησης, εφαρμόστηκε η χρήση των μεταβλητών «_prices» και «_datesUpdated». Αυτές οι μεταβλητές χρησιμεύουν για τη σύνδεση και αποθήκευση των ιστορικών πληροφοριών τιμολόγησης και των αντίστοιχων ημερομηνιών για κάθε προϊόν.

Ένα από τα βασικά χαρακτηριστικά του SQLAlchemy είναι η ικανότητά του να ορίζει properties, hybrid properties, and setters σε κάθε κλάση. Τα properties είναι χαρακτηριστικά σε επίπεδο κλάσης που χρησιμοποιούνται για τον καθορισμό των στηλών ενός πίνακα και των τύπων δεδομένων τους. Τα hybrid properties, από την άλλη, είναι ένας συνδυασμός Python expressions και ιδιοτήτων που βασίζονται σε στήλη. Χρησιμοποιούνται για τον καθορισμό υπολογιστικών πεδίων που δεν αποθηκεύονται στη βάση δεδομένων αλλά υπολογίζονται αμέσως. Τέλος, οι setters χρησιμοποιούνται για τον ορισμό προσαρμοσμένων μεθόδων ρυθμιστή για ιδιότητες, οι οποίες μπορούν να

χρησιμοποιηθούν για την εκτέλεση πρόσθετων ενεργειών όταν ορίζεται ένα property, όπως η επικύρωση ή ο μετασχηματισμός δεδομένων.

```
41
42      @property
43      def prices(self):
44          return [float(x) for x in self._prices.split(';') if x != '']
45
46      @hybrid_property
47      def price(self):
48          try:
49              return float(self._prices.split(';')[-1])
50          except:
51              return -1.0
52
53      @prices.setter
54      def prices(self, price):
55          if self._prices is None:
56              self._prices = ';%s' % price
57          else:
58              self._prices += ';%s' % price
59
```

Image 2.1.1.2 How a property, a hybrid property and a setter is used for the prices

Όταν ολοκληρωθεί η φάση του scraping εκτελώ

```
857     insertFromInputFile(main_dir + "/scraper/ab/ab.txt", "AB")
858     insertFromInputFile(main_dir + "/scraper/sklavenitis/sklavenitis.txt", "Sklavenitis")
859     insertFromInputFile(main_dir + "/scraper/kritikos/kritikos.txt", "Kritikos")
860     insertFromInputFile(main_dir + "/scraper/my_market/my-market.txt", "My Market")
861     insertFromInputFile(main_dir + "/scraper/masoutis/masoutis.txt", "Masoutis")
862
```

Image 2.1.1.3 Update functions

Η συνάρτηση έχει οριστεί για την ανάγνωση δεδομένων από αρχεία, χωρίζοντάς τα στη συνέχεια σε μια λίστα λεξικών και επικαλούμενη τη συνάρτηση "insertProducts(products)".

Για να ολοκληρώσει αυτήν την εργασία, το API χρησιμοποιεί πρώτα το path για να προσδιορίσει την τρέχουσα ημερομηνία και το αντίστοιχο σούπερ μάρκετ. Αυτό επιτρέπει στη λειτουργία να ανακτά τα κατάλληλα δεδομένα και να διασφαλίζει ότι οι πληροφορίες που εμφανίζονται στον χρήστη είναι ενημερωμένες και σχετικές.

```
303 def insertFromInputFile(file, name=""):
304     from datetime import datetime
305
306     current_date = datetime.now()
307     file_date = current_date.strftime("%Y/%m/%d")
308
309     print(file_date)
310
311     if "ab" in file:
312         supermarket_name = "Αβ"
313         supermarket_name_english = "Ab"
314     elif "sklavenitis" in file:
315         supermarket_name = "Σκλαβενίτης"
316         supermarket_name_english = "Sklavenitis"
317     elif "my_market" in file:
318         supermarket_name = "My Market"
319         supermarket_name_english = "My Market"
320     elif "kritikos" in file:
321         supermarket_name = "Κρητικός"
322         supermarket_name_english = "Kritikos"
323     elif "masoutis" in file:
324         supermarket_name = "Μασούτης"
325         supermarket_name_english = "Masoutis"
326     else:
327         raise "Error"
```

Image 2.1.1.4 How to identify each supermarket

Στη συνέχεια προχωρά στην ανάγνωση του αρχείου και στη δημιουργία μιας λίστας αντικειμένων

```
329     with open(file, "r") as f:
330         products = []
331 
332         for line in f.readlines():
333             # print(line)
334             items = line.replace('\n', '').split(', ')
335             j = len(items) - 7
336             p = ""
337 
338             for k in range(1, j):
339                 p += items[k]
340                 if k != j - 1:
341                     p += " "
342 
343             if items[j+6] == "True":
344                 disc = True
345             else:
346                 disc = False
347 
348             obj = {
349                 "supermarket": supermarket_name,
350                 "supermarket_english": supermarket_name_english,
351                 "category": items[0],
352                 "product": p,
353                 "company": items[j+1],
354                 "url": items[j+4],
355                 "img": items[j+5],
356                 "discount": disc,
357                 "date": file_date,
358             }
359 
360             try:
361                 it = items[j+3].split('€/')
362                 obj["value_ratio"] = float(it[0])
363                 obj["metric"] = it[1]
364             except:
365                 continue
366 
367             try:
368                 obj["price"] = float(items[2])
369             except:
370                 continue
371 
372             try:
373                 obj["weight"] = float(items[j+2])
374             except:
375                 if obj["price"] >= 0.0 and obj["value_ratio"] > 0.0:
376                     obj["weight"] = obj["price"] / obj["value_ratio"]
377                 else:
378                     obj["weight"] = 1.0
379 
380             if obj["metric"] == "":
381                 obj["metric"] = "τεμάχιο"
```

Image 2.1.1.5 Spliting the data file into a list of objects

Η συνάρτηση εκτελεί μια επανάληψη μέσω της λίστας των προϊόντων που λαμβάνονται από το αρχείο και στη συνέχεια αναζητά την ύπαρξη κάθε προϊόντος ξεχωριστά. Σε περίπτωση που το προϊόν υπάρχει ήδη, η συνάρτηση ενημερώνει τις τιμές του προϊόντος και συνεχίζει την επανάληψη.

```
107             prod_exist = session.query(Product).filter(Product.url == product["url"]).first()
108
109             if prod_exist is not None:
110                 prod_exist.product = help.removeAccent(product["product"])
111                 prod_exist.category = help.removeAccent(product["category"])
112                 prod_exist.prices = product["price"]
113                 prod_exist.dates = product["date"]
114                 prod_exist.value_ratio = product["value_ratio"]
115                 prod_exist.weight = product["weight"]
116                 prod_exist.discount = product["discount"]
117                 prod_exist.img = product["img"]
118                 prod_exist.url = product["url"]
119                 prod_exist.exist = True
120                 prod_exist.metric = product["metric"]
121
122                 continue
```

Image 2.1.1.6 If product exists update it’s attributes

Διαφορετικά, εάν χρειάζεται να προστεθεί ένα νέο προϊόν, εκτελείται ο ακόλουθος κώδικας:

```
import googletrans
translator = googletrans.Translator()
```

```
125             translated_product = translator.translate(str(product["product"]), src='el', dest='en').text
126
127             if len(product["company"]) == 0:
128                 translated_company = ""
129             else:
130                 translated_company = translator.translate(str(product["company"]), src='el', dest='en').text
131
132             prod = Product(
133                 supermarket=product["supermarket"],
134                 supermarket_english=product["supermarket_english"],
135
136                 product=help.getProperWord(help.removeAccent(product["product"])),
137                 product_english = translated_product,
138
139                 company=help.getProperWord(help.removeAccent(product["company"])),
140                 company_english=translated_company,
141
142                 category=help.getProperWord(help.removeAccent(product["category"])),
143                 category_english = categories[help.getProperWord(help.removeAccent(product["category"]))],
144
145                 metric=product["metric"],
146                 img=product["img"],
147                 url=product["url"],
148                 discount=product["discount"],
149             )
150             prod.exist = True
151
152             if isinstance(product["price"], float):
153                 prod.prices = product["price"]
154             else:
155                 continue
156
157             prod._datesUpdated = ";" + product["date"]
158
159             if isinstance(product["value_ratio"], float):
160                 prod.value_ratio = product["value_ratio"]
161
162             if isinstance(product["weight"], float):
163                 prod.weight = product["weight"]
164
165             session.add(prod)
```

Image 2.1.7 Create new instance of product

2.1.2 Deals Table

Στην αρχική έκδοση της εφαρμογής, χρησιμοποιήθηκε ένα μόνο αίτημα HTTP για τον υπολογισμό του drop value ενός προϊόντος με βάση τη μέση τιμή του, επιστρέφοντας στη συνέχεια τυχόν σχετικές προσφορές. Ωστόσο, λόγω των περιορισμένων 2 GB μνήμης RAM στο VPS που φιλοξενούσε την εφαρμογή εκείνη τη στιγμή, αυτός ο υπολογισμός διαπιστώθηκε ότι ήταν σχετικά αργός, χρειάζονταν περίπου 4-6 δευτερόλεπτα για να ολοκληρωθεί. Επιπλέον, αυτή η προσέγγιση απαιτούσε τον επανυπολογισμό των dropValues κάθε φορά που ένας χρήστης ζητούσε να δει τις προσφορές. Προκειμένου να αντιμετωπιστούν αυτά τα ζητήματα, αποφασίστηκε να εφαρμοστεί ένας νέος πίνακας που διατηρούσε τις σχέσεις μεταξύ των αναγνωριστικών ευκαιριών προϊόντων σε φθίνουσα σειρά, επιτρέποντας την παρουσίαση των πιο ευνοϊκών προσφορών στην κορυφή της λίστας. Με την περιοδική εκκαθάριση αυτού του πίνακα και τη σύνταξη νέων προσφορών, ήταν δυνατή η αποτελεσματική ενημέρωση των τιμών των προϊόντων χωρίς την ανάγκη επανυπολογισμών.

```python
class Deal(Base):
    __tablename__ = 'deal'

    primary_id = db.Column(db.Integer, primary_key=True, unique=True)
    id = db.Column(db.Integer, db.ForeignKey('product.id'), nullable=False)  # this is 1-1 with parent class

    # relationships
    parent = relationship("Product", uselist=True) # this is 1-1 with parent class
```

Image 2.1.2.1 Deal class

Μετά την απόξεση νέων δεδομένων και την αποθήκευσή τους στη βάση δεδομένων, επικαλούμαι αυτό το API που υπολογίζει το dropPrice, το οποίο είναι μια ιδιότητα κλάσης του Product. Μόνο προϊόντα με εκπτώσεις διατηρούνται στη βάση δεδομένων. Αυτή η διαδικασία επιτρέπει την αποτελεσματική διαχείριση των δεδομένων των προϊόντων και διασφαλίζει ότι μόνο σχετικά προϊόντα και προϊόντα με έκπτωση διατίθενται στους χρήστες.

```python
def findBestDeals():
    session = Session()
    try:
        session.query(Deal).delete()
        session.commit()
    except Exception as e:
        print(e)

    ret = [prod for prod in session.query(Product).filter(Product.exist == True).all() if prod.dropPrice < -0.1]

    ret.sort(key=lambda x: x.dropPrice, reverse=False)

    from datetime import date
    today = date.today().strftime("%d/%m/%Y")

    for p in ret:
        deal = Deal()
        deal.parent.append(p)
        session.add(deal)

    session.commit()


    session.close()
```

Image 2.1.2.2 Find best Deals function

2.1.3 Users Table

Όπως συμβαίνει με κάθε εφαρμογή, προκύπτει η ανάγκη αποθήκευσης δεδομένων χρήστη. Για να διευκολυνθεί αυτό, οι χρήστες έχουν την επιλογή να συνδεθούν με τον λογαριασμό τους Google και να παρακολουθούν το καλάθι αγορών τους. Συγκεκριμένα, εφαρμόστηκε μια ιδιότητα καλαθιού που περιέχει μια λίστα με αναγνωριστικά προϊόντων και την αντίστοιχη ποσότητα τους σε μία συμβολοσειρά. Αυτό επιτρέπει την εύκολη διαχείριση και ανάκτηση των δεδομένων του καλαθιού του χρήστη.

```python
class User(Base):
    __tablename__ = 'user'

    id = db.Column(db.Integer, primary_key=True, unique=True)
    user_id = db.Column(db.Integer, unique=True)
    provider = db.Column(db.String)
    email = db.Column(db.String)
    name = db.Column(db.String)
    firstName = db.Column(db.String)
    lastName = db.Column(db.String)
    authToken = db.Column(db.String)
    idToken = db.Column(db.String, unique=True)
    authorizationCode = db.Column(db.String)
```

Image 2.1.3.1 User Class

Το API για την ενημέρωση του καλαθιού αγορών ενός χρήστη έχει δύο τρόπους λειτουργίας: "overwrite" και "update". Στη λειτουργία "overwrite", ο χρήστης αντικαθιστά πλήρως το υπάρχον καλάθι με ένα νέο, όπως καθορίζεται στην παράμετρο "product_quantities". Πριν από αυτήν την ενημέρωση, πραγματοποιείται έλεγχος για να διασφαλιστεί ότι υπάρχουν τα προϊόντα που προστίθενται στο καλάθι. Στη συνέχεια, το API επιστρέφει τα αποτελέσματα αυτής της ενημέρωσης. Στη λειτουργία "update", ο χρήστης μπορεί να προσθέσει ή να αφαιρέσει προϊόντα από το καλάθι του, όπως καθορίζεται στην παράμετρο "product_quantities". Πραγματοποιείται εκ νέου έλεγχος για να διασφαλιστεί ότι υπάρχουν τα προϊόντα που προστίθενται ή αφαιρούνται από το καλάθι και στη συνέχεια, το API επιστρέφει τα αποτελέσματα αυτής της ενημέρωσης.

```python
def updateUserCart(ids, user_id, user_idToken, setting, product_quantities=None):
    session = Session()
    presaved_cart = np.array([])

    if user_id is not None and user_idToken is not None:
        user = session.query(User).filter(User.user_id == user_id).first()
        stored = user.cart
        quantities = product_quantities
        if setting == "overwrite":
            try:
                s = ""
                for i, id in enumerate(ids):
                    if len(id) > 0:
                        s += id + "&" + str(product_quantities[i]) + ", "

                user.cart = s
                session.add(user)
                session.commit()
            except:
                pass
        else:
            if len(stored) > 0:
                data = stored.split(', ')[:-1]
                ids = []
                quantities = []
                for d in data:
                    spl = d.split('&')
                    ids.append(spl[0])
                    quantities.append(float(spl[1]))

            else:
                ids = []
                quantities = []



        products = []
        for id in ids:
            s = session.query(Product).filter(Product.id == id).first()
            if s is not None:
                products.append(s)

        session.close()
        return products, quantities

    products = []
    for id in ids:
        s = session.query(Product).filter(Product.id == id).first()
        if s is not None:
            products.append(s)

    session.close()
    return products, product_quantities
```

Image 2.1.3.2 How to update the users cart

2.1.4 User’s Messages Table

Επιπλέον, έχει υλοποιηθεί ένας πίνακας μηνυμάτων που ουσιαστικά αποθηκεύει τα μηνύματα από τους χρήστες.

```python
from .base import Base

class Message(Base):
    __tablename__ = 'message'

    id = db.Column(db.Integer, primary_key=True, unique=True, nullable=False, autoincrement=True)

    viewed = db.Column(db.Boolean)
    name = db.Column(db.String)
    email = db.Column(db.String)
    date = db.Column(db.Date)
    message = db.Column(db.String)
    answer = db.Column(db.String)

    def serialize(self):
        return {
            "id": self.id,
            "viewed": self.viewed,
            "name": self.name,
            "email": self.email,
            "date": self.date,
            "message": self.message,
            "answer": self.answer,
        }
```

Image 2.1.4.1 Message Table

2.1.5 Suggestions Tables

Η δομή του πίνακα προτάσεων είναι πιο περίπλοκη σε σύγκριση με τους προηγούμενους. Για να δημιουργήσετε μια σχέση many to many χρησιμοποιώντας το SQLAlchemy, είναι απαραίτητο να δημιουργήσετε έναν προσωρινό πίνακα για την αποθήκευση των μοναδικών σχέσεων. Επιπλέον, πρέπει επίσης να δημιουργηθεί μια σχέση γονέων-παιδιών, με τα παιδιά να αποθηκεύονται σε μορφή λίστας. Αυτή η προσέγγιση απαιτείται για την αποτελεσματική εφαρμογή της επιθυμητής σχέσης στον πίνακα προτάσεων. Να τονιστεί πως σε αυτή την λίστα μπαίνουν μόνο Deals.

```python
suggestions_list_deals = db.Table('suggestions_list_deal', Base.metadata,
    db.Column('id', db.Integer, primary_key=True),
    db.Column('child1_id', db.Integer, db.ForeignKey('deal.id')),
    db.Column('child2_id', db.Integer, db.ForeignKey('suggestion.primary_id'))
)
```

Image 2.1.5.1 Temporary Table

```python
class Suggestion(Base):
    __tablename__ = 'suggestion'

    primary_id = db.Column(db.Integer, primary_key=True, unique=True)
    id = db.Column(db.Integer, db.ForeignKey('product.id'))  # this is 1-1 with parent class

    parent = relationship("Product", uselist=True) # this is 1-1 with parent class
    associations = relationship('Deal', secondary=suggestions_list_deals) # , backref=db.backref('child', lazy='dynamic'))
```

Image 2.1.5.2 Suggestion Table

Όπως περιγράφεται στο Κεφάλαιο 3, η εκτέλεση του αλγορίθμου K-means δημιουργεί ένα αρχείο “cluster label file”. Αυτό το αρχείο είναι δομημένο έτσι ώστε η σειρά i να αντιστοιχεί στο προϊόν με ένα id = i-1. Κάθε σειρά του αρχείου περιέχει μια αριθμητική τιμή που αντιστοιχεί στο σύμπλεγμα στο οποίο ανήκει το αντίστοιχο προϊόν, όπως καθορίζεται από τον αλγόριθμο.

```
558 def fillSuggestions():
559     session = Session()
560 
561     try:
562         session.query(Suggestion).delete()
563         session.commit()
564     except Exception as e:
565         print(e)
566     try:
567         session.query(suggestions_list_deals).delete()
568         session.commit()
569     except Exception as e:
570         print(e)
571 
572     for p in session.query(Product).all():
573         sug = Suggestion()
574         sug.parent.append(p)
575         session.add(sug)
576 
577     session.commit()
578 
579     print("Done with deletion")
580     import numpy as np
581 
582     suggestions = session.query(Suggestion).all()
583 
584     K = 1800
585     with open("clust_labels_K1800.txt", "r") as f:
586         cluster_labels = np.array([int(line.replace('\n', '')) for line in f.readlines() ])
587 
588         for i in range(K):
589             # print(i)
590             new_line = list([int(l+1) for l in np.where(cluster_labels == i)[0]])
591 
592             results = session.query(Deal).filter(Deal.id.in_(new_line))
593 
594             ids = [r.id for r in results.all()]
595             print(i, len(ids))
596             for idx, id in enumerate(ids):
597                 for j in range(idx):
598                     suggestions[id - 1].associations.append(results[j])
599                 for j in range(idx + 1, len(ids), 1):
600                     suggestions[id - 1].associations.append(results[j])
601 
602     session.commit()
603         # for line in f.readlines():
604         #     print(line.replace('\n'))
605     session.close()
```

Image 2.1.5.3 Fill Suggestions implementation

2.2 Scraped Data: Data Immunity

Το Web scraping, ως μέθοδος συλλογής δεδομένων, μπορεί μερικές φορές να παράγει αναξιόπιστα ή ανακριβή αποτελέσματα. Αυτό οφείλεται συχνά στην έλλειψη κατάλληλο formatting δεδομένων στους ιστότοπους που εξετάζονται. Για να μετριαστεί αυτό το ζήτημα, έχω εφαρμόσει έναν αλγόριθμο για την επικύρωση και τη διόρθωση βασικών δεδομένων όπως η τιμή, το price value ratio και το βάρος.

Συγκεκριμένα, ο αλγόριθμος συγκρίνει την αναλογία αξίας προϊόντος πολλαπλασιασμένη με το βάρος προς την τιμή και διορθώνει τυχόν αποκλίσεις που μπορεί να είναι αποτέλεσμα σφαλμάτων στον ιστότοπο προέλευσης. Αυτή η προσέγγιση μου επιτρέπει να βελτιώσω την ακρίβεια και την αξιοπιστία των δεδομένων που συλλέγονται.

```
def fixPrices():
    session = Session()
    f = open("temp.txt", "w")
    for prod in session.query(Product).all():
        price = prod.price
        weight = prod.weight
        value_ratio = prod.value_ratio

        if abs(float(round(value_ratio * weight, 4)) - price) > 0.5  :
            if value_ratio == 0.0 or value_ratio == -1.0:
                prod.value_ratio = float(round(price / weight, 2))
            elif weight == 0.0 or weight == -1.0:
                prod.weight = float(round(price / value_ratio, 4))
            elif price == -1.0:
                prod.exist = False
            else:
                f.write(str(prod.id) + " " + str(price) + " " + str(weight) + " " + str(value_ratio) + " " + str( value_ratio * weight) + " " + str(prod.product) + "\n")

    session.commit()
    session.close()
```

Image 2.2.1 Fix Prices

2.3 VPS Setup

Στο πλαίσιο της παρούσας διπλωματικής εργασίας, κρίνεται απαραίτητος ένας εικονικός ιδιωτικός διακομιστής (Virtual Private Server - VPS) για την ασταμάτητη λειτουργία της εφαρμογής. Ωστόσο, πρέπει να ξεπεραστούν ορισμένα εμπόδια προκειμένου να διασφαλιστεί η ασφάλεια του VPS και η ομαλή διανομή των κλήσεων δικτύου, καθώς και η διαρκής λειτουργία της εφαρμογής flask. Στο πλαίσιο της παρούσας διπλωματικής εργασίας, κρίνεται απαραίτητος ένας εικονικός ιδιωτικός διακομιστής (Virtual Private Server - VPS) για την συνεχή λειτουργία της εφαρμογής.

2.3.1 NGINX

Το NGINX είναι ένα λογισμικό διακομιστή ιστού που χρησιμοποιείται συχνά ως reverse proxy, load balancer, και HTTP cache. Είναι γνωστό για την υψηλή απόδοση και τη χαμηλή κατανάλωση πόρων, καθιστώντας το μια δημοφιλή επιλογή για ιστοσελίδες υψηλής επισκεψιμότητας και εφαρμογές web. Μπορεί επίσης να χρησιμοποιηθεί για την εξυπηρέτηση στατικού περιεχομένου, ως διακομιστή μεσολάβησης για άλλους διακομιστές Ιστού και ως load balancer για τη διανομή των εισερχόμενων αιτημάτων μεταξύ πολλών διακομιστών. Παράλληλα, μπορεί να χρησιμοποιηθεί για τον τερματισμό συνδέσεων SSL/TLS και να χρησιμεύσει ως διακομιστής προέλευσης για το πρωτόκολλο HTTP/2 [12]. Η διαμόρφωση του διακομιστή NGINX σε αυτό το σενάριο έχει σχεδιαστεί για να ανακατευθύνει όλα τα εισερχόμενα αιτήματα HTTP στα αντίστοιχα ισοδύναμα HTTPS. Αυτό επιτυγχάνεται με τη χρήση των κατάλληλων οδηγιών ανακατεύθυνσης μέσα στο αρχείο διαμόρφωσης NGINX. Επιπλέον, η διαμόρφωση διαχειρίζεται τη συμπερίληψη του "www." subdomain, διασφαλίζοντας ότι όλα τα σχετικά αιτήματα δικτύου κατευθύνονται σωστά στον προβλεπόμενο διακομιστή. Στην περίπτωση αιτημάτων HTTPS, η διαμόρφωση χρησιμοποιεί το βοηθητικό πρόγραμμα Certbot για τη δημιουργία πιστοποιητικών SSL και ανακατευθύνει αυτά τα αιτήματα στον τοπικό διακομιστή που εκτελείται στη θύρα 8080 για περαιτέρω επεξεργασία. Συνολικά, η διαμόρφωση NGINX έχει σχεδιαστεί για να παρέχει βέλτιστη ασφάλεια και απόδοση, διασφαλίζοντας παράλληλα ότι όλα τα αιτήματα δικτύου αντιμετωπίζονται και κατευθύνονται σωστά.

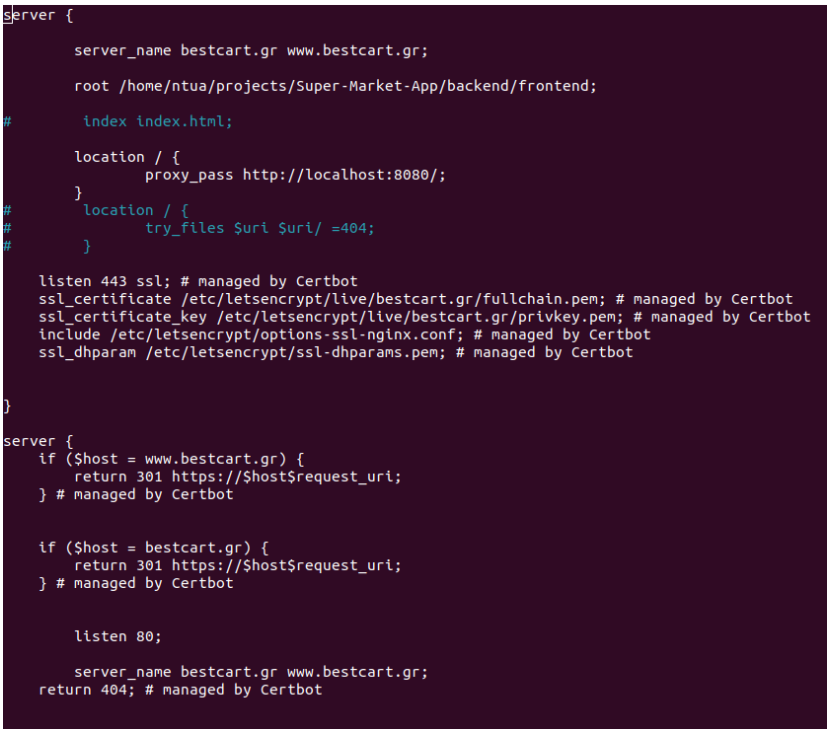


Image 2.3.1.1 NGINX Configurations

2.3.2 Docker με Gunicorn

Το Docker είναι μια πλατφόρμα που χρησιμοποιεί κοντέινερ για τη συσκευασία και τη διανομή εφαρμογών σε μια μορφή που μπορεί να εκτελείται με συνέπεια σε διαφορετικά περιβάλλοντα. Επιτρέπει την εύκολη δημιουργία, δοκιμή και ανάπτυξη εφαρμογών και επιτρέπει στους προγραμματιστές να επικεντρωθούν στη σύνταξη κώδικα χωρίς να ανησυχούν για το σύστημα στο οποίο θα εκτελείται. Βοηθά επίσης τις ομάδες λειτουργιών πληροφορικής να μετακινούν γρήγορα εφαρμογές μεταξύ περιβαλλόντων και να απλοποιούν την κλιμάκωση και τη διαχείριση εφαρμογών χρησιμοποιώντας εργαλεία ενορχήστρωσης. Επιπλέον, παρέχει μια μεγάλη συλλογή προκατασκευασμένων εικόνων που μπορούν να χρησιμοποιηθούν για γρήγορη εκκίνηση μιας εφαρμογής ή υπηρεσίας, εξοικονομώντας χρόνο και προσπάθεια.

Το Gunicorn (συντομογραφία του Greenlet-based Worker for UNIX) είναι ένας διακομιστής HTTP διασύνδεσης πύλης διακομιστή Web (WSGI) της Python. Αξιοποιήτε για την εξυπηρέτηση εφαρμογών ιστού με χρήση Python και χρησιμοποιείται συχνά ως διακομιστής web έτοιμος για παραγωγή για εφαρμογές που έχουν δημιουργηθεί χρησιμοποιώντας βιβλιοθήκες όπως το Flask. Το Gunicorn έχει σχεδιαστεί για να είναι ελαφρύ, γρήγορο και αποτελεσματικό και μπορεί να χειριστεί πολλαπλές ταυτόχρονες συνδέσεις χρησιμοποιώντας ένα pre-fork μοντέλο εργασίας, επιτρέποντάς του να χειρίζεται μεγάλο αριθμό εισερχόμενων αιτημάτων χωρίς σημαντική αύξηση στη χρήση πόρων.
Είναι επίσης γνωστό για την ικανότητά του να χειρίζεται αργούς πελάτες χωρίς να μπλοκάρει τις διαδικασίες εργασίας, μπορεί επίσης να δαιμονοποιηθεί και μπορεί επίσης να εκτελεστεί πίσω από έναν αντίστροφο διακομιστή μεσολάβησης όπως το nginx ή το Apache.

Το Gunicorn χρησιμοποιείται συνήθως σε συνδυασμό με έναν reverse proxy και έναν διακομιστή web front-end όπως το nginx ή ο Apache, ο οποίος μπορεί να χειριστεί εργασίες όπως ο τερματισμός SSL, η εξισορρόπηση φορτίου και η προσωρινή αποθήκευση.

Για να εκτελέσετε ένα Dockerfile, πρώτα, βεβαιωθείτε ότι έχετε εγκαταστήσει το Docker στο σύστημά σας. Αφού ενεργοποιήσετε και εκτελείτε το Docker, μεταβείτε στον κατάλογο όπου βρίσκεται το Dockerfile σας χρησιμοποιώντας τη γραμμή εντολών. Στη συνέχεια, μπορείτε να χρησιμοποιήσετε την εντολή "docker build" για να δημιουργήσετε μια εικόνα από το δικό σας Dockerfile. Η βασική σύνταξη για την εντολή build είναι "docker build -t [image_name] [path_to_dockerfile]". Για παράδειγμα, εάν το Dockerfile σας βρίσκεται στον τοπικό κατάλογο "myapp" και θέλετε να ονομάσετε την εικόνα "myapp-image", θα χρησιμοποιήσετε την εντολή "docker build -t myapp-image myapp/".

Μόλις δημιουργηθεί η εικόνα, μπορείτε να χρησιμοποιήσετε την εντολή "docker run" για να ξεκινήσετε ένα νέο κοντέινερ χρησιμοποιώντας την εικόνα. Η βασική σύνταξη για την εντολή εκτέλεσης είναι "Docker run [image_name] [command_to_run]". Για παράδειγμα, για να ξεκινήσετε ένα κοντέινερ χρησιμοποιώντας την εικόνα "myapp-image" και να εκτελέσετε την εντολή "python app.py", θα χρησιμοποιούσατε την εντολή "docker run myapp-image python app.py".

Μπορείτε επίσης να χρησιμοποιήσετε την επιλογή -p στην εντολή εκτέλεσης για να αντιστοιχίσετε τις θύρες του κοντέινερ στον κεντρικό υπολογιστή. Αυτή η επιλογή είναι χρήσιμη εάν θέλετε να αποκτήσετε πρόσβαση στην εφαρμογή που εκτελείται στο κοντέινερ από τον κεντρικό υπολογιστή. Για παράδειγμα, για να αντιστοιχίσετε τη θύρα 8080 του κοντέινερ στη θύρα 80 του κεντρικού υπολογιστή, θα χρησιμοποιούσατε την εντολή "docker run -p 80:8080 myapp-image python app.py"

Επιπλέον, μπορείτε επίσης να χρησιμοποιήσετε την επιλογή -d για να εκτελέσετε το κοντέινερ σε λειτουργία αποσύνδεσης, η οποία επιτρέπει στο κοντέινερ να τρέχει στο παρασκήνιο, σας επιτρέπει επίσης να προβάλετε αρχεία καταγραφής και να σταματήσετε το κοντέινερ.

Είναι σημαντικό να έχετε κατά νου ότι όταν ένα κοντέινερ σταματάει όλα τα δεδομένα χάνονται, εάν θέλετε να διατηρηθούν τα δεδομένα, θα πρέπει να χρησιμοποιήσετε τόμους ή προσαρτήσεις σύνδεσης με την επιλογή -v. [13]

Παρακάτω, συμπεριέλαβα το Dockerfile μου και την εντολή εκτέλεσης του κοντέινερ κατασκευής.

```
1  # syntax=docker/dockerfile:1
2
3  FROM python:3.8
4
5  WORKDIR /app
6  COPY . .
7
8  RUN pip install -r requirements.txt
9
10 ENTRYPOINT [ "gunicorn" ]
11 CMD ["--bind", "0.0.0.0:8080", "--workers=2", "--access-logfile", "-", "app:app"]
12
13 # Run as
14 # docker run -p 8080:8080 --name test --rm -v /mnt/AEBA35E3BA35A8AB/Super-Market-App/backend/databases2:/app/databases/ -d supermarket
15 # docker run -p 8080:8080 -v /mnt/AEBA35E3BA35A8AB/Super-Market-App/backend/:/app --rm -d --name supermarket_image supermarket
```

Image 2.3.2.1 Dockerfile

Κεφάλαιο 3: Σύστημα Συστάσεων

3.1 Η Ανάγκη για Έναν Καλό Αλγόριθμο Συστάσεων

Η ανάγκη για έναν καλό αλγόριθμο συστάσεων είναι υψίστης σημασίας στο σημερινό ταχέως εξελισσόμενο περιβάλλον λιανικής πώλησης. Με την πληθώρα επιλογών που έχουν στη διάθεσή τους οι καταναλωτές, είναι συχνά δύσκολο για αυτούς να λάβουν τεκμηριωμένες αποφάσεις σχετικά με το ποια προϊόντα θα αγοράσουν. Ένας καλός αλγόριθμος συστάσεων αντιμετωπίζει αυτό το ζήτημα προτείνοντας προϊόντα στους καταναλωτές με βάση τις προτιμήσεις και την προηγούμενη συμπεριφορά τους. Αυτό όχι μόνο βοηθάει τους καταναλωτές να θυμηθούν προϊόντα που μπορεί να έχουν ξεχάσει, αλλά και τους βοηθάει να ανακαλύψουν νέα προϊόντα που μπορεί να είναι πιο κατάλληλα γι' αυτούς. Επιπλέον, ένας καλός αλγόριθμος συστάσεων μπορεί να βοηθήσει στη χαρτογράφηση πανομοιότυπων προϊόντων σε διαφορετικά σούπερ μάρκετ, παρέχοντας στους καταναλωτές μια πιο ολοκληρωμένη εικόνα των προϊόντων και των τιμών που είναι διαθέσιμα σε αυτούς. Αυτή η χαρτογράφηση μπορεί επίσης να αποτελέσει μεγάλο πλεονέκτημα για τους λιανοπωλητές, καθώς τους επιτρέπει να έχουν μια πιο ακριβή αντίληψη των προϊόντων που είναι διαθέσιμα στην αγορά, η οποία με τη σειρά της μπορεί να τους βοηθήσει να λάβουν καλύτερες αποφάσεις σχετικά με τα προϊόντα που διαθέτουν και τις τιμές τους. Συνολικά, ένας καλός αλγόριθμος συστάσεων μπορεί να βελτιώσει σημαντικά την εμπειρία αγορών τόσο για τους καταναλωτές όσο και για τους λιανοπωλητές.

3.2 TF-IDF

3.2.1 Βασική Θεωρία

Ξεκινάμε με την απλή δυαδική σακούλα λέξεων του μοντέλου όπου κάθε έγγραφο αναπαρίσταται ως ένα διάνυσμα σταθερού μεγέθους από 0 και 1, όπου αν μια λέξη εμφανίζεται σε ένα έγγραφο παίρνει 1 και αν δεν εμφανίζεται τότε παίρνει 0. Ως παράδειγμα, θεωρήστε τα παρακάτω τέσσερα έγγραφα:

D1: the movie was a very *indulging* cinematic experience.
D2: standard of this movie is above its contemporaries.
D3: director brought out the best of the pair.
D4: moviegoers won’t mind seeing the pair again. [14]

Η δυαδική σακούλα των λέξεων που αναπαριστάται από τα 4 κείμενα, είναι η εξής:

Table Binary Bag of Words Model [14]

Docs/ Words	the	movie	of	pair	was	a	wont	mind
D1	1	1	0	0	1	1	0	0
D2	0	1	1	0	0	0	0	0
D3	0	0	1	1	0	0	0	0
D4	0	0	0	1	0	0	1	1

Σε αυτό το μοντέλο, αναπαριστούμε μόνο την ύπαρξη κάθε λέξης, αλλά δεν λαμβάνουμε υπόψη τη σημασία μιας λέξης. Για παράδειγμα, η λέξη "indulging" είναι μια σημαντική λέξη.

Το TF-IDF, συντομογραφία του term frequency-inverse document frequency, είναι ένα στατιστικό μέτρο που χρησιμοποιείται για την κατανόηση της σημασίας μιας λέξης σε ένα δεδομένο έγγραφο ή σώμα κειμένων. Είναι μια ευρέως χρησιμοποιούμενη τεχνική στην επεξεργασία φυσικής γλώσσας και στην ανάκτηση πληροφοριών και είναι ιδιαίτερα χρήσιμη για την ταξινόμηση κειμένων και τις μηχανές αναζήτησης.

Το μέτρο TF-IDF αποτελείται από δύο μέρη: τη συχνότητα όρων (TF) και την αντίστροφη συχνότητα εγγράφων (IDF). Η συχνότητα όρων είναι ένα μέτρο του αριθμού των φορών που εμφανίζεται μια λέξη σε ένα έγγραφο και χρησιμοποιείται για να μετρήσει τη συνάφεια μιας λέξης με ένα συγκεκριμένο έγγραφο. Υπάρχουν διάφοροι τρόποι για τον ορισμό της συχνότητας όρων, όπως η ακατέργαστη καταμέτρηση, η συχνότητα όρων προσαρμοσμένη στο μήκος του εγγράφου, η λογαριθμικά κλιμακούμενη συχνότητα και η boolean συχνότητα.

Από την άλλη πλευρά, η αντίστροφη συχνότητα εγγράφων είναι ένα μέτρο του πόσο κοινή ή ασυνήθιστη είναι μια λέξη σε ένα σώμα εγγράφων. Χρησιμοποιείται για τη μέτρηση της σημασίας μιας λέξης στο σύνολο του σώματος κειμένων. Η αντίστροφη συχνότητα εγγράφων υπολογίζεται με τη λήψη του λογαρίθμου του λόγου του συνολικού αριθμού των εγγράφων στο σώμα κειμένων προς τον αριθμό των εγγράφων στα οποία εμφανίζεται η λέξη [14].

$$idf\,(t, D) = log\,(\ \frac{N}{count\ (d \in D : t \in d)}\ )$$

Image 3.2.1.1 [14]

Scikit-Learn

- $\text{IDF}(t) = \log\frac{1+n}{1+\text{df}(t)} + 1$

Standard notation

- $\text{IDF}(t) = \log\frac{n}{\text{df}(t)}$

Image 3.2.1.2 [14]

3.2.2 Προεπεξεργασία

Είναι ζωτικής σημασίας η προεπεξεργασία της περιγραφής του προϊόντος πριν από τη χρήση του αλγορίθμου tf-idf, καθώς είναι ιδιαίτερα ευαίσθητος στις νέες λέξεις. Μια μέθοδος προεπεξεργασίας είναι η χρήση μιας βιβλιοθήκης python με την ονομασία "greek_stemmer", η οποία στελεχώνει τις ελληνικές λέξεις. Ο σκοπός του stemming είναι να μειώσει τις πολλαπλές παραλλαγές μιας λέξης, όπως "Μπανάνες" και "Μπανάνα", σε μια ενιαία, κοινή μορφή, στην προκειμένη περίπτωση "ΜΠΑΝΑΝ". Επιπλέον, κατά την εξέταση του συνόλου δεδομένων, διαπιστώθηκε ότι υπήρχαν πολυάριθμες περιπτώσεις λαθών, όπως ο συνδυασμός πολλαπλών λέξεων σε μία λέξη, όπως φαίνεται στο παράδειγμα "ΠΟΡΤΟΚΑΛΙ/ΜΗΛΟ/ΒΕΡΙΚΟΚΟ. " Ως αποτέλεσμα, ήταν απαραίτητες περαιτέρω τροποποιήσεις πριν προχωρήσουμε στη χρήση οποιουδήποτε αλγορίθμου. Μετά την εκτέλεση αυτού του αλγόριθμου, οι μοναδικές λέξεις και οι τιμές tf-idf ήταν καλύτερες.

```python
def applyWeightChanges(s):
    '''
        gets a string in greek finds and fixes weird
    '''
    upper = removeAccent(s.upper())
    upper = upper.replace(',', '.')

    if upper.endswith(" GR"):
        upper = upper[:-3] + "GR"

    '''
        Starting with kg, gr etc
    '''
    # Search of "{number}G" if it exist and after the G is a space character or the string ends
    # add R to make it GR
    # # remove redundant end spaces
    upper = re.sub(r'(\d+G)(\s|$)', r'\1R ', upper)
    while upper[-1] == " ":
        upper = upper[:-1]

    # Search of "{number}ΓΡ." , relpace it with GR to make it GR
    # # remove redundant end spaces
    upper = re.sub(r'(\d+)ΓΡ\.', r'\1GR ', upper)
    while upper[-1] == " ":
        upper = upper[:-1]

    upper = upper.replace(' KG', 'KG').replace(' KG.', 'KG').replace('KG.', 'KG').replace(' ΚΙΛΩΝ', 'KG')

    upper = upper.replace(' ML', 'ML').replace(' ML.', 'ML').replace('ML.', 'ML')
    upper = upper.replace(' LT', 'LT').replace(' LT.', 'LT').replace('LT.', 'LT')

    upper = upper.replace(' ΤΕΜΑΧΙΟ', 'ΤΕΜΑΧ').replace('ΤΕΜΑΧΙΟ', 'ΤΕΜΑΧ').replace(' ΤΕΜΑΧΙΑ', 'ΤΕΜΑΧ').replace('ΤΕΜΑΧΙΑ', 'ΤΕΜΑΧ')
    upper = upper.replace('ΚΑΡΑΜΕΛΑ-ΜΠΑΝΑΝΑ', 'ΚΟΤΟΠΟΥΛΟ ΛΑΧΑΝΙΚΑ').replace('ΚΟΥΝΕΛΙ&ΛΑΧΑΝΙΚΑ', 'ΚΟΥΝΕΛΙ ΛΑΧΑΝΙΚΑ')
    upper = upper.replace('ΜΗΛΟ-ΑΧΛΑΔΙ-ΣΤΑΦΥΛΙ-ΜΠΑΝΑΝΑ', 'ΜΗΛΟ ΑΧΛΑΔΙ ΣΤΑΦΥΛΙ ΜΠΑΝΑΝΑ')
    upper = upper.replace('ΜΗΛΟ-ΜΠΑΝΑΝΑ-ΒΑΤΟΜΟΥΡΟ-ΔΗΜΗΤΡΙΑΚΑ', 'ΜΗΛΟ ΜΠΑΝΑΝΑ ΒΑΤΟΜΟΥΡΟ ΔΗΜΗΤΡΙΑΚΑ')
    upper = upper.replace('ΚΑΡΑΜΕΛΑ-ΦΙΣΤΙΚΙ-ΜΠΑΝΑΝΑ', 'ΚΑΡΑΜΕΛΑ ΦΙΣΤΙΚΙ ΜΠΑΝΑΝΑ')
    upper = upper.replace('ΜΠΑΝΑΝΑ-ΒΑΝΙΛΙΑ', 'ΜΠΑΝΑΝΑ ΒΑΝΙΛΙΑ')
    upper = upper.replace('ΜΠΑΝΑΝΑ-ΣΟΚΟΛΑΤΑ', 'ΜΠΑΝΑΝΑ ΣΟΚΟΛΑΤΑ')
    upper = upper.replace('ΣΟΚΟΛΑΤΑ&ΜΠΑΝΑΝΑ', 'ΣΟΚΟΛΑΤΑ ΜΠΑΝΑΝΑ')
    upper = upper.replace('ΜΠΑΝΑΝΑ&ΣΟΚΟΛΑΤΑ', 'ΜΠΑΝΑΝΑ ΣΟΚΟΛΑΤΑ')
    upper = upper.replace('ΜΠΑΝΑΝΑ-ΜΗΛΟ-ΒΡΩΜΗ', 'ΜΠΑΝΑΝΑ ΜΗΛΟ ΒΡΩΜΗ')
    upper = upper.replace('ΜΗΛΟ-ΒΕΡΙΚΟΚΟ-ΜΠΑΝΑΝΑ', 'ΜΗΛΟ ΒΕΡΙΚΟΚΟ ΜΠΑΝΑΝΑ')
    upper = upper.replace('ΜΠΑΝΑΝΑ+ΣΟΚΟΜΠΙΛ', 'ΜΠΑΝΑΝΑ ΣΟΚΟΜΠΙΛ')
    upper = upper.replace('ΦΡΑΟΥΛΑ-ΜΠΑΝΑΝΑ', 'ΦΡΑΟΥΛΑ ΜΠΑΝΑΝΑ')
    upper = upper.replace('ΜΗΛΟ/ΣΤΑΦΥΛΙ/ΦΡΑΟΥΛΑ', 'ΜΗΛΟ ΣΤΑΦΥΛΙ ΦΡΑΟΥΛΑ')
    upper = upper.replace('ΜΗΛΟ/ΑΧΛΑΔΙ/ΣΤΑΦΥΛΙ/ΜΠΑΝΑΝΑ', 'ΜΗΛΟ ΑΧΛΑΔΙ ΣΤΑΦΥΛΙ ΜΠΑΝΑΝΑ')
    upper = upper.replace('ΜΗΛΟ/ΠΟΡΤΟΚΑΛΙ/ΡΟΔΑΚΙΝΟ/ΒΕΡΙΚΟΚΟ', 'ΜΗΛΟ ΠΟΡΤΟΚΑΛΙ ΡΟΔΑΚΙΝΟ ΒΕΡΙΚΟΚΟ')
    upper = upper.replace('ΠΟΡΤΟΚΑΛΙ/ΜΗΛΟ/ΒΕΡΙΚΟΚΟ', 'ΠΟΡΤΟΚΑΛΙ ΜΗΛΟ ΒΕΡΙΚΟΚΟ')

    return upper

def stemLine(stemmer, line):
    return " ".join([stemmer.stem(removeAccent(l.upper())) for l in line.split(' ')])
```

Image 3.2.3.1 Line Preprocessing

```
12889   μιγ
12890   μιγμ
12891   μικ
12892   μικρ
12893   μικρα
12894   μικροβιοκτον
12895   μικρογκοφρ
12896   μικροι
12897   μικροιν
12898   μικροκαρπ
12899   μικροκυμ
12900   μικροσπερμ
12901   μικροσωμ
12902   μικτ
12903   μικτες
12904   μικτη
12905   μικτο
12906   μικτσικελ
12907   μιλαν
12908   μιλανεζ
12909   μιλφει
12910   μιλφειγ
12911   μιμ
12912   μιμοζ
12913   μιν
12914   μινερβ
12915   μινερβιν
12916   μινεστρον
12917   μινθ
12918   μινι
12919   μινιί
12920   μινιατουρ
12921   μιξ
12922   μιξερ
12923   μιξερακ
12924   μιρ
12925   μιραντ
12926   μις
12927   μισκοτ
12928   μισκοτιν
12929   μισο
12930   μιστ
12931   μιστικανζ
12932   μιτατ
12933   μιτατοτυρ
12934   μιτσικελ
12935   μιχ
12936   μιχαηλιδ
12937   μιχαλ
12938   μιχαλακ
```

Image 3.2.3.2 Example of the unique words

3.3 Αλγόριθμοι Νέων Προτάσεων

Το ζήτημα της σύστασης νέων προϊόντων στους χρήστες από ένα τεράστιο φάσμα επιλογών είναι πολύπλοκο. Ωστόσο, είναι δυνατή η ανάπτυξη συστημάτων συστάσεων με τη χρήση των διαθέσιμων δεδομένων.

Στην παρούσα διατριβή, προτείνονται δύο συστήματα συστάσεων με διαφορετικές μεθόδους: K-means, και Cosine similarity.

3.3.1 K-Means

Μια πιθανή λύση είναι η χρήση του αλγορίθμου K-Means, μιας μεθόδου χωρίς επίβλεψη που απαιτεί μόνο τον καθορισμό του αριθμού των ομάδων, ο οποίος συμβολίζεται ως K, που πρέπει να δημιουργηθούν. Αυτή η προσέγγιση αποτελεί μια βασική επιλογή για την πραγματοποίηση εξατομικευμένων συστάσεων προς τους χρήστες.

Η μέθοδος k-means είναι ένας δημοφιλής τρόπος ομαδοποίησης σημείων δεδομένων σε ομάδες προσπαθώντας να ελαχιστοποιήσει την απόσταση μεταξύ σημείων στην ίδια ομάδα. Είναι απλή και γρήγορη, αλλά δεν παρέχει εγγυήσεις για την ακρίβεια. Με την προσθήκη ενός τυχαίου στοιχείου στην αρχική τοποθέτηση των συστάδων, η μέθοδος μπορεί να γίνει πιο αποτελεσματική και ακριβής. Επιπλέον, πέρα από την επιλογή K δεν μπορούμε να επηρεάσουμε το πλήθος των ομαδοποιήσεων, έτσι είνια δυνατόν μια κατηγορία να περιέχει πολλά προϊόντα, ενώ άλλες να έχουν λιγότερα. Οι αρχικές δοκιμές έδειξαν σημαντικές βελτιώσεις τόσο στην ταχύτητα όσο και στην ακρίβεια [17].

For the `k-means` problem, we are given an integer k and a set of n data points $\mathcal{X} \subset \mathbb{R}^d$. We wish to choose k centers $\mathcal{C}$ so as to minimize the potential function,

$$\phi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} \|x - c\|^2.$$

Image 3.3.1.1 Definition of k-means [17]

Στην παρούσα διατριβή, πραγματοποιείται αξιολόγηση του αλγορίθμου k-means μέσω της χρήσης δύο μεθόδων: του συντελεστή σιλουέτας και των οπτικών αναπαραστάσεων που μπορούμε να φτιάξουμε.

Ο συντελεστής Silhouette Coefficient, γνωστός και ως silhouette score, είναι μια μετρική που χρησιμοποιείται για τη μέτρηση της αποτελεσματικότητας μιας τεχνικής ομαδοποίησης, με εύρος τιμών από -1 έως 1. Η τιμή 1 υποδηλώνει ότι οι συστάδες είναι καλά διαχωρισμένες, η τιμή 0 υποδηλώνει ότι οι συστάδες είναι αδιάφορες και η τιμή -1 υποδηλώνει ότι οι συστάδες έχουν αποδοθεί εσφαλμένα [19].

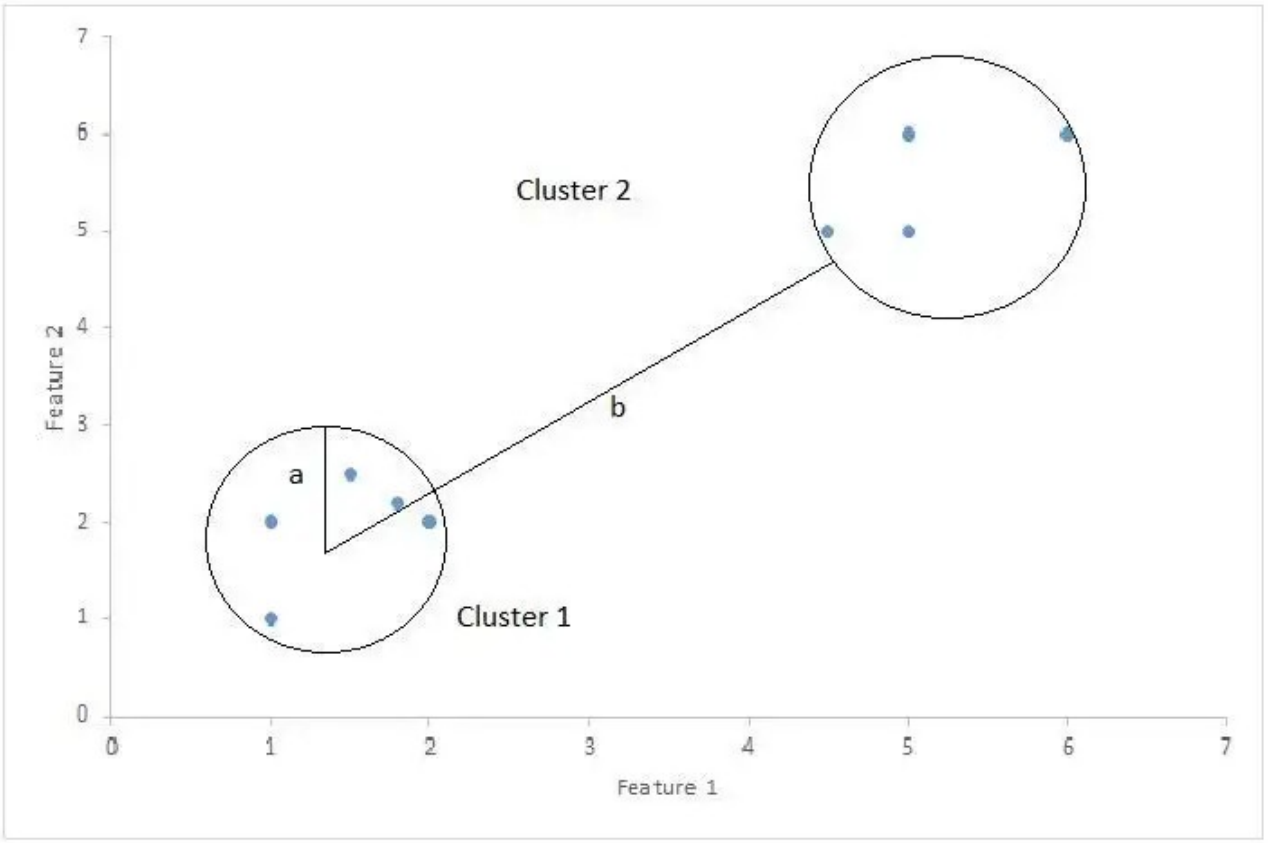


Image 3.3.1.2 Silhouette Score Representation

Silhouette Score = $\frac{b(i)-a(i)}{max(a(i),b(i))}$, όπου

a(i): μέση απόσταση ενός σημείο i με όλα τα υπόλοιπα σημεία στην ομάδα
b(i): μέση απόσταση μεταξύ του i σε όλες τις άλλες ομάδες που δεν ανήκει

Table 1. Silhouette Score for various K values

K Value	Silhouette Score
500	0.064123910053892
1000	0.083192436793098
3000	0.114632552705606
5000	0.143830595692552
10000	0.201307073686309
15000	0.237640750096125
20000	0.257919619180028

Κατά την εκτέλεση του αλγορίθμου k-means στο σύνολο δεδομένων για διάφορες τιμές του K, διαπιστώθηκε ότι καθώς αυξάνεται η τιμή του K, αυξάνεται και το σκορ σιλουέτας. Ωστόσο, καθώς αυξάνεται το K, μειώνεται ο αριθμός των προϊόντων σε κάθε κατηγορία. Σε περιπτώσεις όπου αναζητούνται παρόμοια προϊόντα, αυτό έχει συχνά ως αποτέλεσμα την εύρεση μηδενικών προϊόντων. Ως εκ τούτου, μια τιμή K μεταξύ 5000 και 10000 θεωρείται βέλτιστη για αυτή την περίπτωση χρήσης και τελικά επιλέγεται η τιμή K=5000. Τέλος, να σημειωθεί πως τα σκορ που προκείπτουν κλείνουν προς το αδιάφορο (0) [18].

Image 3.3.1.3 Implementation of the k-means.

[18]:
```
import numpy as np

def getLabels(path="clust_labels_K13000_6k_columns.txt"):
    with open(path, "r") as f:
        cluster_labels = [int(line.replace('\n', '')) for line in f.readlines() ]

    cluster_labels = np.array(cluster_labels)
    return cluster_labels


file_data = np.array(product_data)

def writeNewCategories(file_data, cluster_labels, K, columns="6k"):
    with open("new_categories_K" + str(K) + "_" + columns + ".txt", "w") as f:
        for i in range(K):
            new_line = [(l, file_data[l]) for l in np.where(cluster_labels == i)[0]]
            for l, prod in new_line:
                f.write(str(l) + ": " + prod + " |&|\t")
            f.write("\n")

# print(cluster_labels)
```

+ Code + Markdown

```
import cuml

from sklearn.cluster import KMeans

range_n_clusters = [500] # 18000, 21000, 25000, 27500] # , 12000, 15000, 20000, 25000, 30000]
import numpy as np

X = X.todense()# X = np.array(X)
silhouette_scores = []
for n_clusters in range_n_clusters:
    clusterer = cuml.KMeans(n_clusters=n_clusters)
    cluster_labels = clusterer.fit_predict(X)

    with open("/kaggle/working/cluster_labels_K" + str(n_clusters) + ".txt", "w") as f:
        for c in cluster_labels:
            f.write(str(c) +"\n")

    s = silhouette_score(X, cluster_labels)
    silhouette_scores.append(s)
    print(n_clusters, s)


print(silhouette_scores)
```

Image 3.3.1.4 Write new categories in a file

Αφού ολοκληρωθεί η εκτέλεση του kmeans, μπορώ να εκτελέσω τις δύο συναρτήσεις για να λάβω μια οπτική αναπαράσταση του τρόπου με τον οποίο σχηματίζονται οι κατηγορίες μου.

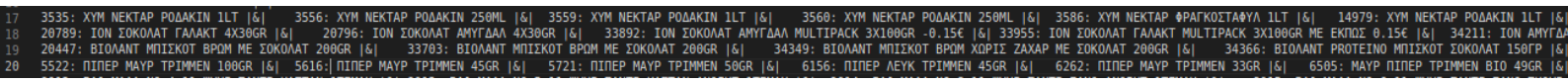


Image 3.3.1.5 Example of categories formed with K = 3000

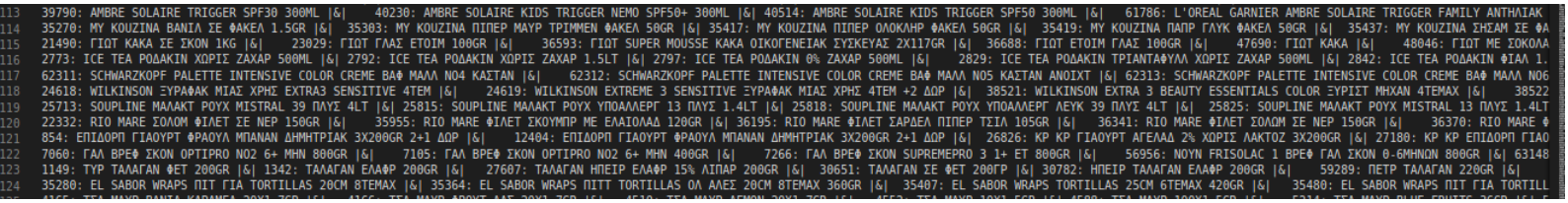


Image 3.3.1.6 Example of categories formed with K = 5000

3.3.2 Cosine Distance

Η λύση για τη σύσταση προϊόντων μέσω της απόδοσης ομοιότητας με χρήση του cosine distance περιελάμβανε αρκετά βήματα για την οριστικοποίηση του. Αρχικά, η ομοιότητα Jaccard θεωρήθηκε ως η πρωταρχική επιλογή για τον προσδιορισμό της ομοιότητας μεταξύ των προϊόντων. Ωστόσο, διαπιστώθηκε ότι η ομοιότητα Jaccard δεν αντικατοπτρίζει με ακρίβεια την ομοιότητα μεταξύ των προϊόντων, οδηγώντας έτσι στην εφαρμογή της απόστασης συνημίτονου ως εναλλακτικής μεθόδου. Η απόσταση συνημίτονου είναι ένα μέτρο ομοιότητας μεταξύ δύο μη μηδενικών διανυσμάτων και υπολογίζεται ως το τετραγωνικό γινόμενο των διανυσμάτων διαιρούμενο με το γινόμενο των μεγεθών τους. Ένα πλεονέκτημα της χρήσης της απόστασης συνημίτονου έναντι άλλων μεθόδων, όπως η k-means, είναι η δυνατότητα ρύθμισης του μέγιστου αριθμού συνιστώμενων προϊόντων. Με την επιλογή ενός χαμηλού κατωφλίου, ο συνολικός αριθμός των αποτελεσμάτων μπορεί να περιοριστεί και το κατώφλι μπορεί στη συνέχεια να ρυθμιστεί ανάλογα με βάση τον επιθυμητό αριθμό συνιστώμενων προϊόντων [20]. Συγκεκριμένα υπολογίζεται ως: $S_c(A,B) = \cos(\theta) = \frac{\boldsymbol{A}\cdot\boldsymbol{B}}{\|\boldsymbol{A}\|\cdot\|\boldsymbol{B}\|}$

Table 2. Example of similar products through cosine similarity

Product' Id	Product Name	Similar Product	Similar's id	Cosine Similarity
6479	ΜΟΥΣΤΑΡΔ DIJONNAISE 200GR	ΜΟΥΣΤΑΡΔ DIJONNAISE 240GR	5672	0.83
		ΜΟΥΣΤΑΡΔ 200GR	6019	0.63
		MAILLE ΜΟΥΣΤΑΡΔ DIJONNAISE 200GR	36829	0.8
6480	ΕΛΑΙΟΛΑΔ ΕΞΑΙΡΕΤ ΠΑΡΘΕΝ ΠΑΡΑΔΟΣΙΑΚ 1LT 9.22	ΔΩΡ ΕΞΑΙΡΕΤ ΠΑΡΘΕΝ ΕΛΑΙΟΛΑΔ 1LT	37053	0.7
		ΕΛΑΙΟΛΑΔ ΕΞΑΙΡΕΤ ΠΑΡΘΕΝ 1LT	6064	0.74
16479	ΚΑΡΑΜΟΛΕΓΚ ΨΩΜ ΣΕ ΦΕΤ 500GR	ΖΚΑΡΑΜΟΛΕΓΚ ΨΩΜ Ζ ΣΕ ΦΕΤ 500GR	56748	1
		ΚΑΡΑΜΟΛΕΓΚ ΟΚΤΑΣΠΟΡ ΨΩΜ	16577	0.78

			ΣΕ ΦΕΤ 500GR	
16480	ΚΑΤΣΕΛ ΤΟΣΤ 720GR	ΨΩΜΨΩΜ ΣΤΑΡΕΝΣΤΑΡΕΝ 720GR	ΤΟΣΤ1952	0.88

Είναι σημαντικό να σημειωθεί ότι ο λόγος που οι εν λόγω περιγραφές δεν περιέχουν πλήρεις λέξεις οφείλεται στο γεγονός ότι πρόκειται για λέξεις με βάση.

Συγκεκριμένα, για να λάβω αυτά τα αποτελέσματα, υλοποίησα τον παρακάτω κώδικα.

```python
from numba import jit, cuda

# @jit(target_backend='cuda')
def similarity_check_for_list(tfIdfX, product_data, id):
    # print("Starting", product_data[id])

    len_words = tfIdfX.getrow(id).data.shape[0]
    minimum_similarity = (len_words - 1)/(2 * len_words + 1) + 0.1

    # print(minimum_similarity)

    # minimum_similarity = 0.6

    ids = []

    a = list(zip(list(X.getrow(id).data), list(X.getrow(id).indices)))

    for i in range(product_data):
        if i == id:
            continue

        # and_ = len(set(a_indices) & set(b_indices))
        # union_ = len(set(a_indices) | set(b_indices))

        b = list(zip(list(X.getrow(i).data), list(X.getrow(i).indices)))

        and_ = list(set(X.getrow(id).indices) & set(X.getrow(i).indices))
        temp1 = {}

        product = 0.0
        a_l2, b_l2 = 0.0, 0.0

        for q in a:
            a_l2 += q[0] ** 2

            if q[1] in and_:
                temp1[q[1]] = q[0]
        for q in b:
            b_l2 += q[0] ** 2

            if q[1] in and_:
                product += temp1[q[1]] * q[0]

        div = (a_l2 * b_l2) ** 0.5

        cosine_similarity = product / div

        if cosine_similarity > minimum_similarity:
            # print(i, cosine_similarity, product_data[id], "&", product_data[i])
            ids.append((i, cosine_similarity))

    return ids
```

Image 3.3.2.1 Cosine Similarity check

Όσον αφορά το χρόνο, χρειάζονται περίπου 7-8 δευτερόλεπτα ανά προϊόν για να βρεθούν όλες οι ομοιότητες σε έναν laptop με 12 GB RAM και επεξεργαστή Intel i7. Επιπλέον, κατά τη δοκιμή σε ένα δεύτερο laptop με 8GB RAM και επεξεργαστή Intel i5, η διαδικασία απαιτούσε 12-13 δευτερόλεπτα για να ολοκληρωθεί. Αξίζει να σημειωθεί ότι προσπάθησα επίσης να χρησιμοποιήσω το numba, ένα εργαλείο που στοχεύει στη βελτίωση των επιδόσεων με τη χρήση της GPU, προκειμένου να βελτιστοποιήσω τη διαδικασία. Ωστόσο, αυτό δεν είχε το επιθυμητό αποτέλεσμα. Στο laptop με μνήμη RAM 12 GB, η διαδικασία απαιτούσε 0.25 δευτερόλεπτα ανά προϊόν, ενώ στο laptop με μνήμη 8 GB απαιτούσε 0.35 δευτερόλεπτα για να ολοκληρωθεί. Αυτή η έλλειψη βελτίωσης και μείωση της απόδοσης οφείλεται πιθανότατα στο γεγονός ότι η numba δεν είναι κατάλληλη για το χειρισμό αραιών πινάκων (X).

3.4 Σώσιμο Στην Βάση Δεδομένων

Πριν κλείσουμε το κεφάλαιο σχετικά με τους recommenders, είναι σημαντικό να συζητήσουμε την ενσωμάτωση των συστάσεων στη βάση δεδομένων. Για να το επιτύχω αυτό, υλοποίησα μια συνάρτηση εντός του φακέλου backend που είναι ειδικά σχεδιασμένη για το σκοπό αυτό.

```python
def fillSuggestions():
    session = Session()

    try:
        session.query(Suggestion).delete()
        session.commit()
    except Exception as e:
        print(e)
    try:
        session.query(suggestions_list_deals).delete()
        session.commit()
    except Exception as e:
        print(e)

    for p in session.query(Product).all():
        sug = Suggestion()
        sug.parent.append(p)
        session.add(sug)

    session.commit()

    print("Done with deletion")
    import numpy as np

    suggestions = session.query(Suggestion).all()

    K = 5000
    with open("clust_labels_K5000.txt", "r") as f:
        cluster_labels = np.array([int(line.replace('\n', '')) for line in f.readlines() ])

        for i in range(K):
            # print(i)
            new_line = list([int(l+1) for l in np.where(cluster_labels == i)[0]])

            results = session.query(Deal).filter(Deal.id.in_(new_line))

            ids = [r.id for r in results.all()]
            print(i, len(ids))
            for idx, id in enumerate(ids):
                for j in range(idx):
                    suggestions[id - 1].associations.append(results[
                for j in range(idx + 1, len(ids), 1):
                    suggestions[id - 1].associations.append(results[j])

    session.commit()
        # for line in f.readlines():
        #     print(line.replace('\n'))
    session.close()
```

(variable) j: int

Image 3.4.1 Fill Suggestions implementation with K-Means results

```
with open("similarity_output.txt", "r") as file:
    data = file.read()

# Convert the string data into a Python object
data = data.split('\n')

corellations = []

for index, line in enumerate(data):
    l = line.replace('(', '').replace(')', '').replace('[', '').replace(']', '')
    l = l.split(', ')
    i = 0
    corellations_temp = []
    while 2 * i + 1 < len(l):
        corellations_temp.append((int(l[2 * i]), float(l[2 * i + 1])))
        i += 1

    corellations_temp.sort(key=lambda a: a[1])

    corellations.append(corellations_temp)
```

```
suggestions = session.query(Suggestion).all()

for i, cor in enumerate(corellations):
    ids = [c[0] for c in cor]
    results = session.query(Deal).filter(Deal.id.in_(ids)).all(

    if i % 100 == 0:
        print(len(ids), len(results))
        print(i, len(results))

    for res in results:
        suggestions[i].associations.append(res)

session.commit()
session.close()
```

Image 3.4.2 Cosine similarity update

Η διαδικασία προσθήκης των προτάσεων στη βάση δεδομένων περιλαμβάνει διάφορα στάδια. Πρώτον, είναι απαραίτητο να διαγραφούν όλες οι προηγούμενες προτάσεις, καθώς τα deals έχουν αλλάξει. Στη συνέχεια, για κάθε cluster, εντοπίζονται όλα τα προϊόντα που αντιστοιχούν στο εν λόγω cluster και προστίθενται σε μια λίστα. Για κάθε προϊόν στην λίστα, δημιουργείται ξεχωριστή λίστα, με τα προϊόντα χωρίς το ίδιο το προϊόν. Τέλος, οι αλλαγές σώζονται στη βάση δεδομένων.

```
session = Session()

try:
    session.query(Suggestion).delete()
    session.commit()
except Exception as e:
    print(e)
try:
    session.query(suggestions_list_deals).delete()
    session.commit()
except Exception as e:
    print(e)

for p in session.query(Product).all():
    sug = Suggestion()
    sug.parent.append(p)
    session.add(sug)

session.commit()

print("Done with deletion")
```

Image 3.4.3 Delete previous data

Κεφάλαιο 4: Αυτοματοποιήστε την Όλη Διαδικασία

4.1 Γιατί Αυτοματοποίηση;

Κατά την ανάπτυξη αυτής της εφαρμογής, έγινε φανερό ότι η χειροκίνητη εκτέλεση δεν είναι ο βέλτιστος τρόπος λειτουργίας. Ως αποτέλεσμα, η δημιουργία αυτοματοποιημένων scripts για τη διαχείριση όλων των διαδικασιών έγινε αναγκαία.

4.2 Αυτοματοποίηση

4.2.1 Αυτοματοποίηση Διαδικασίας Web Scraping

Η εκκίνηση των web crawlers είναι απλή και περιλαμβάνει μία μόνο εντολή στο τερματικό (termina). Συγκεκριμένα από τον root φάκελο του έργου, μπορούμε να εκτελέσουμε το “python3 scraper/”. Εσωτερικά, στον κατάλογο scraper, υπάρχει ένα αρχείο __main__.py που διαβάζει ορισμένες εισόδους συστήματος και δημιουργεί τα βασικά νήματα.

```
import sys
threads = []

if len(sys.argv) <= 1:
    threads.append(threading.Thread(target=AB, args=(1,)))
    threads.append(threading.Thread(target=AB, args=(2,)))

    threads.append(threading.Thread(target=sklavenitis, args=(1,)))
    threads.append(threading.Thread(target=sklavenitis, args=(2,)))

    threads.append(threading.Thread(target=kritikos))
    threads.append(threading.Thread(target=MyMarket))
    threads.append(threading.Thread(target=masoutis))
else:
    for i in range(1,len(sys.argv)):
        if sys.argv[i] == 'AB':
            threads.append(threading.Thread(target=AB, args=(1,)))
            threads.append(threading.Thread(target=AB, args=(2,)))
        elif sys.argv[i] == 'Sklavenitis':
            threads.append(threading.Thread(target=sklavenitis, args=(1,)))
            threads.append(threading.Thread(target=sklavenitis, args=(2,)))
        elif sys.argv[i] == 'Kritikos':
            threads.append(threading.Thread(target=kritikos))
        elif sys.argv[i] == 'MyMarket':
            threads.append(threading.Thread(target=MyMarket))
        elif sys.argv[i] == 'Masoutis':
            threads.append(threading.Thread(target=masoutis))

for thread in threads:
    thread.start()

for thread in threads:
    thread.join()

print("Done")
```

Image 4.2.1.1 Automate web scraping

4.2.2 Αυτοματοποίηση Back-end

Μια άλλη ακολουθία που απαιτεί αυτοματοποίηση είναι η ενημέρωση του back-end και της βάσης δεδομένων. Μόλις ο web crawler ολοκληρώσει την εργασία του, το αρχείο backend/backend.py μπορεί να εκτελεστεί. Αυτό το αρχείο περιλαμβάνει πολλά βήματα. Το πρώτο βήμα είναι να διαγράψει τις πληροφορίες του προηγούμενου προϊόντος, κάτι που επιτυγχάνεται ορίζοντας τη μεταβλητή “exist”, η οποία υποδεικνύει εάν ένα προϊόν υπάρχει αυτήν τη στιγμή στο κατάστημα, σε false.

```
861     session = Session()
862
863     for prod in session.query(Product).all():
864         prod.discount = False
865         prod.exist = False
866         session.add(prod)
867
868     session.commit()
869     session.close()
```

Image 4.2.2.1 Clearing Products

Στη συνέχεια, τα αρχεία που αποκτήθηκαν από τον web crawler διαβάζονται και αποθηκεύονται στη βάση δεδομένων.

Image 4.2.2.2 Data update

```
871     import os
872
873     os.chdir("..")
874     main_dir = os.getcwd() # /home/.../Super-Market-app
875
876     insertFromInputFile(main_dir + "/scraper/ab/ab.txt", "AB")
877     insertFromInputFile(main_dir + "/scraper/sklavenitis/sklavenitis.txt", "Sklavenitis")
878     insertFromInputFile(main_dir + "/scraper/kritikos/kritikos.txt", "Kritikos")
879     insertFromInputFile(main_dir + "/scraper/my_market/my-market.txt", "My Market")
880     insertFromInputFile(main_dir + "/scraper/masoutis/masoutis.txt", "Masoutis")
881
882     os.chdir("backend")
```

Στη συνέχεια, εφαρμόζονται τυχόν απαραίτητες διορθώσεις στα δεδομένα, όπως η ενημέρωση των αγγλικών κατηγοριών και διόρθωση των τιμών. Το script προσδιορίζει τις καλύτερες προσφορές και συμπληρώνει τις προτάσεις.

```
884     updateEnglishCategories()
885
886     fixPrices()
887
888     findBestDeals()
889
890     fillSuggestions()
```

Image 4.2.2.3 Fixes

4.2.3 Διατήρηση Δεδομένων Χρήστη

Είναι σημαντικό να σημειωθεί ότι πριν από την ενημέρωση των προϊόντων, πρέπει να διατηρηθούν τα δεδομένα χρήστη και συγκεκριμένα το καλάθι. Για να επιτευχθεί αυτό, το script συνδέεται στο VPS χρησιμοποιώντας τους κωδικούς SSH, χρησιμοποιώντας τη βιβλιοθήκη pexpect Python για να παρέχει μια άμεση διεπαφή SSH API. Επιπλέον, επειδή χρησιμοποιώ το git ως σύστημα ενημέρωσης, έχω συμπεριλάβει μια διαδικασία συμπίεσης για τη μείωση του μεγέθους του αρχείου της βάσης δεδομένων, καθώς η εναλλακτική θα ήταν να χρησιμοποιήσω το git-lfs, ένα σύστημα επί πληρωμή.

```python
from pexpect import pxssh
import credentials

try:
    conn = pxssh.pxssh()
    hostname = credentials.hostname
    username = credentials.username
    password = credentials.password

    conn.login(hostname, username, password)

    conn.sendline("cd projects/Super-Market-App/backend/databases")
    conn.prompt()
    print(conn.before)

    conn.sendline("git pull")
    conn.prompt()
    print(conn.before)

    conn.sendline("gzip data.sqlite -k -f")
    conn.prompt()
    print(conn.before)

    conn.sendline("git status")
    conn.prompt()
    print(conn.before)

    conn.sendline("git add .")
    conn.prompt()
    print(conn.before)

    conn.sendline("git commit -m 'update users'")
    conn.prompt()
    print(conn.before)

    conn.sendline("git push")
    conn.prompt()
    print(conn.before)

    conn.logout()

except pxssh.ExceptionPxssh as ex:
    print("Could not login")
    print(str(ex))
```

Image 4.2.3.1 User update script

4.2.4 Ενημέρωση του Server

Μια άλλη πτυχή που απαιτεί αυτοματοποίηση είναι η ενημέρωση του διακομιστή. Συγκεκριμένα, έχω αναπτύξει ένα script που συμπιέζει τη βάση δεδομένων σε ένα αρχείο “.gz”, το κάνει commit στο git, συνδέεται στον διακομιστή και τραβάει τα δεδομένα. Το script χρησιμοποιεί την επιλογή volume του Docker container, το οποίο κάνει αυτόματο override το προϋπάρχον αρχείο βάσης δεδομένων μόλις εκτελεστεί το "git pull".

```python
from pexpect import pxssh
from time import sleep
import credentials

conn = pxssh.pxssh()
hostname = credentials.hostname
username = credentials.username
password = credentials.password
connected = False

while not connected:
    try:
        conn.login(hostname, username, password)
        connected = True

    except pxssh.ExceptionPxssh as ex:
        print("Could not login")
        print(str(ex))
        sleep(5)

try:
    conn.sendline("cd projects/Super-Market-App/backend/databases/")
    conn.prompt()
    print(conn.before)

    conn.sendline("git status")
    conn.prompt()
    print(conn.before)

    conn.sendline("git pull")
    conn.prompt()
    print(conn.before)

    conn.sendline("gunzip data.sqlite.gz -f -k")
    conn.prompt()
    print(conn.before)

    # conn.sendline("sudo docker stop supermarket_image")
    # conn.prompt()
    # conn.sendline(password)

    # conn.sendline("sudo docker run -p 8080:8080 -v /home/ntua/projects/Super-Market-App/backend/:/app --rm -d --name supermarket_image supermarket")
    # conn.prompt()

    conn.logout()

except pxssh.ExceptionPxssh as ex:
    print("Could not login")
    print(str(ex))
```

Image 4.2.4.1 Server Update

4.2.5 Άθροισμα των Συστημάτων

Είναι επίσης σημαντικό να υπάρχει ένας τρόπος αυτόματης αποθήκευσης των δεδομένων μετά από κάθε scraping και να διασφαλίζεται ότι δεν θα χαθούν νέα αρχεία δεδομένων. Για να επιτευχθεί αυτό, υπάρχει ένας φάκελος data_new σε κάθε κατάλογο σούπερ μάρκετ. Το script μετακινεί το αρχείο {supermarket}/{supermaket}.txt σε αυτόν τον φάκελο με την αντίστοιχη ημερομηνία και εκτελεί επίσης τις εντολές συμπίεσης. Με την εισαγωγή του script ssh_update_users, ενημερώνει αυτόματα τους χρήστες. Στη συνέχεια, το σενάριο τραβά τις αλλαγές, τις αποσυμπιέζει και εκτελεί το backend script, και επιπλέον σχηματίζει τα νέα αρχεία στους φακέλους δεδομένων.

```python
import subprocess
import time
import os

import ssh_update_users

result = subprocess.run(['git', 'pull'], stdout=subprocess.PIPE)
print(result.stdout.decode('utf-8'))

main_dir = os.getcwd() # /home/.../Super-Market-app


os.chdir(main_dir + "/backend/databases/")

result = subprocess.run(['gunzip', 'data.sqlite.gz', '-f', '-k'], stdout=subprocess.PIPE)
print(result.stdout.decode('utf-8'))

os.chdir(main_dir + "/backend/")

# run backend.py
result = subprocess.run(['python3', 'backend.py'], stdout=subprocess.PIPE)
print(result.stdout.decode('utf-8'))

print("Waiting 10 seconds before continuing...")
time.sleep(10)

# Get list of files on scraper/ab/data and scraper/sklavenitis

os.chdir(main_dir) # back in the main dir

from datetime import datetime
current_date = datetime.now()
formatted_date = current_date.strftime("%Y_%m_%d")

def copy_new_file(main_dir, old_file, path, date):
    new_file = main_dir + path + date + ".txt"
    result = subprocess.run(['touch', new_file], stdout=subprocess.PIPE)
    print(result.stdout.decode('utf-8'))

    result = subprocess.run(['cp', main_dir + old_file, new_file], stdout=subprocess.PIPE)
    print(result.stdout.decode('utf-8'))

copy_new_file(main_dir, "/scraper/ab/ab.txt", "/scraper/ab/data_new/", formatted_date)
copy_new_file(main_dir, "/scraper/sklavenitis/sklavenitis.txt", "/scraper/sklavenitis/data_new/", formatted_date)
copy_new_file(main_dir, "/scraper/kritikos/kritikos.txt", "/scraper/kritikos/data_new/", formatted_date)
copy_new_file(main_dir, "/scraper/my_market/my-market.txt", "/scraper/my_market/data_new/", formatted_date)
copy_new_file(main_dir, "/scraper/masoutis/masoutis.txt", "/scraper/masoutis/data_new/", formatted_date)


print("ab.txt and sklavenitis.txt, kritikos.txt, my_market.txt, masoutis.txt were copied to the new files")
```

Image 4.3.5.1 Initial __main__.py script in the server-update folder

Το script διαγράφει επίσης τα αρχικά αρχεία για κάθε σούπερ μάρκετ, συμπιέζει τη βάση δεδομένων και κάνει commit τα νέα δεδομένα στο git, ενώ τέλος, εκτελεί το σενάριο ssh_server που ενημερώνει τον διακομιστή με τα νέα δεδομένα.

```
52
53  # Clear ab.txt and sklavenitis.txt
54
55  os.chdir(main_dir) # back in the main dir
56
57  result = subprocess.run(['rm', main_dir + "/scraper/ab/ab.txt"], stdout=subprocess.PIPE)
58  print(result.stdout.decode('utf-8'))
59  result = subprocess.run(['touch', main_dir + "/scraper/ab/ab.txt"], stdout=subprocess.PIPE)
60  print(result.stdout.decode('utf-8'))
61
62  result = subprocess.run(['rm', main_dir + "/scraper/sklavenitis/sklavenitis.txt"], stdout=subprocess.PIPE)
63  print(result.stdout.decode('utf-8'))
64  result = subprocess.run(['touch', main_dir + "/scraper/sklavenitis/sklavenitis.txt"], stdout=subprocess.PIPE)
65  print(result.stdout.decode('utf-8'))
66
67  result = subprocess.run(['rm', main_dir + "/scraper/kritikos/kritikos.txt"], stdout=subprocess.PIPE)
68  print(result.stdout.decode('utf-8'))
69  result = subprocess.run(['touch', main_dir + "/scraper/kritikos/kritikos.txt"], stdout=subprocess.PIPE)
70  print(result.stdout.decode('utf-8'))
71
72  result = subprocess.run(['rm', main_dir + "/scraper/my_market/my-market.txt"], stdout=subprocess.PIPE)
73  print(result.stdout.decode('utf-8'))
74  result = subprocess.run(['touch', main_dir + "/scraper/my_market/my-market.txt"], stdout=subprocess.PIPE)
75  print(result.stdout.decode('utf-8'))
76
77  result = subprocess.run(['rm', main_dir + "/scraper/masoutis/masoutis.txt"], stdout=subprocess.PIPE)
78  print(result.stdout.decode('utf-8'))
79  result = subprocess.run(['touch', main_dir + "/scraper/masoutis/masoutis.txt"], stdout=subprocess.PIPE)
80  print(result.stdout.decode('utf-8'))
81
82  print("Files were cleared")
83
84  # submit changes to server
85  os.chdir(main_dir + "/backend/databases/")
86  result = subprocess.run(['gzip', 'data.sqlite', '-k', '-f'], stdout=subprocess.PIPE)
87  print(result.stdout.decode('utf-8'))
88
89  os.chdir(main_dir) # back in the main dir
90
91  result = subprocess.run(['git', 'add', '.'], stdout=subprocess.PIPE)
92  print(result.stdout.decode('utf-8'))
93
94  result = subprocess.run(['git', 'commit', '-m' '"update new values"'], stdout=subprocess.PIPE)
95  print(result.stdout.decode('utf-8'))
96
97
98  result = subprocess.run(['git', 'push'], stdout=subprocess.PIPE)
99  print(result.stdout.decode('utf-8'))
100
101
102 # possibly add docker update
103 # docker build --tag supermarket .
104 # docker stop supermarket_image
105 # docker run -p 8080:8080 -v /home/thodoris/projects/Super-Market-App/backend/:/app --rm -d --name supermarket_image supermarket
106
107 import ssh_server
```

Image 4.3.5.2 The rest

Για να εκτελέσετε όλη την ακολουθία, απαιτείται η εκτέλεση το "python3 scraper/ ; python3 server-update/", μια ακολουθία δύο εντολών.

Κεφάλαιο 5: Frontend, ένας τρόπος οπτικοποίησης των δεδομένων μου

Μια πρόσφατη μελέτη δείχνει ότι το 60% της συνολικής κίνησης στο Διαδίκτυο αποτελείται από κίνηση ιστού, υποδεικνύοντας ότι οι ιστότοποι έχουν γίνει το κύριο μέσο οργάνωσης και μετάδοσης δεδομένων. Η διαδικτυακή επικοινωνία είναι ένας κρίσιμος ενδιάμεσος κρίκος για τη μετάδοση δεδομένων Διαδικτύου. (Cantelon, 2015).

Στη διατριβή μου, υποστηρίζω ότι μια διεπαφή χρήστη (UI) είναι ζωτικής σημασίας για την επιτυχία οποιουδήποτε συστήματος που βασίζεται στο web. Η διεπαφή χρήστη του frontend χρησιμεύει ως η γέφυρα μεταξύ του χρήστη και του συστήματος υποστήριξης, επιτρέποντας στον χρήστη να αλληλεπιδρά και να πλοηγείται στις λειτουργίες και τα δεδομένα του συστήματος. Ένα καλά σχεδιασμένο περιβάλλον χρήστη διασφαλίζει ότι η εμπειρία χρήστη είναι διαισθητική και απρόσκοπτη, διευκολύνοντας τους χρήστες να έχουν πρόσβαση και να χρησιμοποιούν τις δυνατότητες του συστήματος. Επιπλέον, ένα περιβάλλον διεπαφής χρήστη μπορεί επίσης να διαδραματίσει βασικό ρόλο στη συνολική αισθητική και την επωνυμία ενός συστήματος που βασίζεται στο web, καθιστώντας το πιο ελκυστικό οπτικά και επαγγελματικό. [16].

5.1 Frameworks, libraries και Τεχνολογίες

Μετά το έτος 2000, καθώς η τεχνολογία του προγράμματος περιήγησης ιστού προχώρησε γρήγορα, η JavaScript αναδείχθηκε ως ηγετική γλώσσα στην ανάπτυξη Ιστού. Επιπλέον, οι τεχνολογίες web front-end γνώρισαν σημαντικές προόδους [16].

Μετά την εξάπλωση της JavaScript, μια πληθώρα frameworks και βιβλιοθηκών JavaScript έχουν αναπτυχθεί για να βοηθήσουν στη δόμηση του βασικού κώδικα. Αυτές περιλαμβάνουν δημοφιλείς επιλογές όπως Angular, React js, Vue js, Vanilla js και Node js. Μετά από προσεκτική εξέταση, επέλεξα να χρησιμοποιήσω την Angular για την ανάπτυξη του έργου μου. Ενώ μπορεί να απαιτεί αρχικά μια απότομη καμπύλη εκμάθησης, η Angular προσφέρει μια πληθώρα ενσωματωμένων εργαλείων που επιτρέπουν στον προγραμματιστή να εφαρμόσει ένα ευρύ φάσμα λειτουργιών. Επιπλέον, η δομή που βασίζεται σε συνιστώσες του Angular επιτρέπει τη δημιουργία, τον συνδυασμό και τον χειρισμό μεμονωμένων στοιχείων για την κατασκευή ενός συνεκτικού συνόλου [16].

5.2 Available Features

Ξεκινώντας, παρέχω μια γενική επισκόπηση της εφαρμογής ιστού μου.

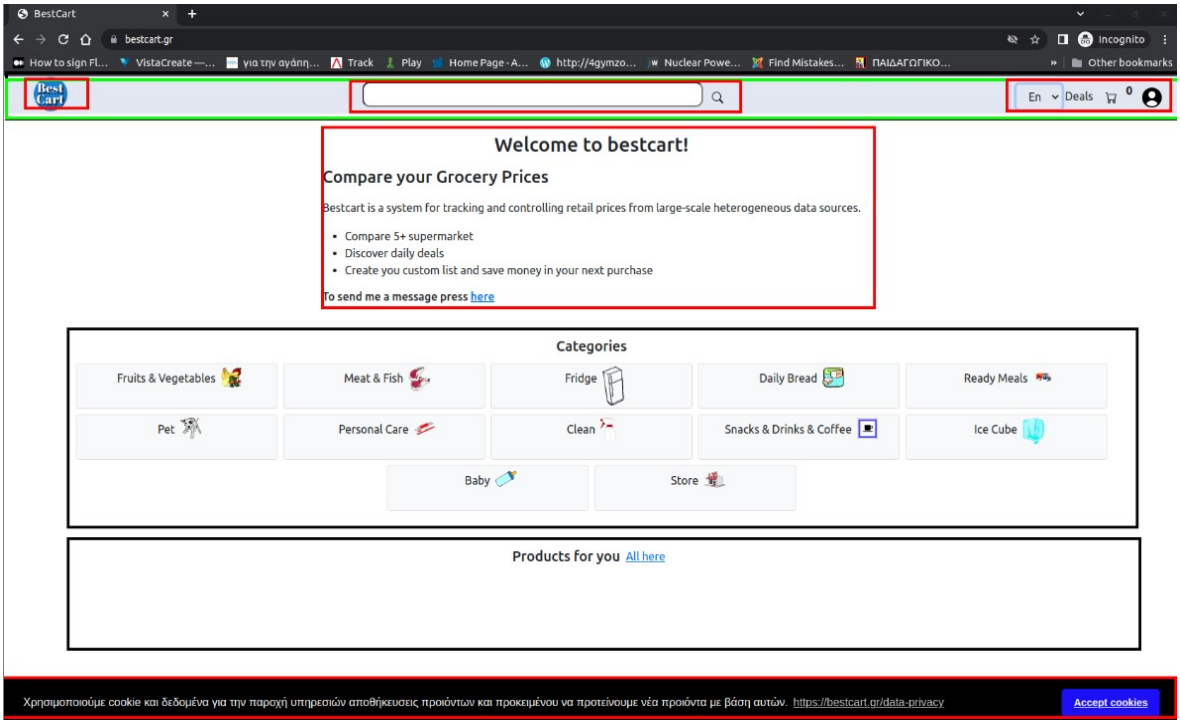


Image 5.2.1 Home page with all the components

Εκτός από τον τυπικό κώδικα HTML, αξίζει να επισημάνω την προσέγγιση που έχω ακολουθήσει σχετικά με το routing. Έχω επιλέξει να υλοποιήσω έναν συγκεκριμένο αριθμό σημείων δρομολόγησης, καθένα από τα οποία φορτώνει ένα ξεχωριστό στοιχείο.

```
const routes: Routes = [
  {path: '', pathMatch: 'full', component: HomeComponent},
  {path: 'cart', component: CheckoutPageComponent},
  {path: 'search', component: ItemsListComponent},
  {path: 'hot-deals', component: HotDealsComponent},
  {path: 'login', component: LoginComponent},
  {path: 'write-message', component: WriteMessageComponent},
  {path: 'root', component: RootComponent},
  {path: 'personalized-deals', component: PersonalizedDealsComponent},
  {path: 'data-privacy', component: DataPrivacyComponent},
];
```

Image 5.2.2 Routes

Πάνω από κάθε στοιχείο βρίσκεται το Header component, το οποίο περιλαμβάνει το λογότυπό μου στα αριστερά, μια γραμμή αναζήτησης στο κέντρο και διάφορα κουμπιά όπως το κουμπί γλώσσας, το κουμπί προσφορών, το προσωπικό καλάθι και το κουμπί σύνδεσης στα δεξιά. Το Header component είναι υπεύθυνο για τη διαχείριση των αλλαγών δρομολόγησης που ξεκινούν από την κεφαλίδα. Συγκεκριμένα, η συνάρτηση onNavigate() καλείται σε κάθε κλικ κουμπιού.

```
35  onNavigate(input: any) {
36    if (input === 'main') {
37      this.router.navigate(['']);
38      this.search = '';
39    } else if (input === 'hot-deals') {
40      this.router.navigate(['hot-deals']);
41    }else if (input === 'cart') {
42      this.router.navigate(['cart']);
43      this.search = '';
44    } else if (input === 'search' && this.search.length > 0) {
45      this.router.navigate(['search'],
46        {
47          queryParams: {
48            'search': this.search,
49            'page': 1
50          },
51        }
52      );
53    } else if (input === 'login') {
54      // this.router.navigate(['login']);
55      if (this.loggingService.userValue == null) {
56        this.loggingService.googleLogin();
57      }
58    } else if (input === 'profile') {
59      this.router.navigate(['login']);
```

Image 5.2.3 Above code ov onNavigate()

```
19    constructor(
20      private router: Router,
21      public cartService: CartService,
22      public loggingService: LoggingService,
23      private dataService: DataService
24    ) { }
```

Image 5.2.4 Header Components Services

Ανάλογα με το κουμπί που πατάτε, το αντίστοιχο σημείο δρομολόγησης αλλάζει. Επιπλέον, για να βελτιστοποιήσω τη χρήση του χώρου στις οθόνες κινητών, έχω ενσωματώσει κινούμενα σχέδια CSS για να διασφαλίσω ότι η γραμμή αναζήτησης εμφανίζεται μόνο όταν ο χρήστης κάνει κλικ στο κουμπί αναζήτησης σε κινητές συσκευές.

```
60      } else if (input === 'search-full') {
61        let small_header = document.querySelectorAll<HTMLElement>('.small-header')[0].className;
62        let inside_header = document.querySelectorAll<HTMLElement>('.inside-header')[0].className;
63
64        let btns: string[] = [];
65        let b = document.getElementsByTagName("button");
66        for (let i = 0; i < b.length; ++i) {
67          btns.push(b[i].className);
68          b[i].className += " btn-disabled disabled"
69        }
70
71        // document.querySelectorAll<HTMLElement>('.remove-header-button')[0].className += " btn-disa
72
73
74        document.querySelectorAll<HTMLElement>('.small-header')[0].style.display = 'inline';
75
76        document.querySelectorAll<HTMLElement>('.inside-header')[0].className += " disappear";
77        document.querySelectorAll<HTMLElement>('.small-header')[0].className += " appear";
78
79        setTimeout( () => {
80          for (let i = 0; i < btns.length; ++i) {
81            b[i].className += btns[i];
82          }
83
84          document.querySelectorAll<HTMLElement>('.inside-header')[0].style.display = 'none';
85
86          document.querySelectorAll<HTMLElement>('.small-header')[0].className = small_header;
87          document.querySelectorAll<HTMLElement>('.inside-header')[0].className = inside_header;
88        }, 2000);
```

Image 5.2.5 Open search on mobile

Για να το πετύχω αυτό, έχω εφαρμόσει μια κλάση small-header CSS, μια κλάση inside-header και μια κλάση appear και disappear που μπορώ να αλλάζω κάθε φορά που ο χρήστης κάνει κλικ στο κουμπί. Ένα σημαντικό στοιχείο είναι ότι για να διασφαλιστεί η σωστή λειτουργία της κινούμενης εικόνας, προστέθηκε ένα χρονικό όριο (καθυστέρηση JavaScript) για να αφαιρεθεί η εμφάνιση της υπόλοιπης κεφαλίδας. Ομοίως, αυτή η προσέγγιση χρησιμοποιήθηκε για την εφαρμογή του κλεισίματος της γραμμής αναζήτησης.

```
90      } else if (input === 'close-search') {
91        let small_header = document.querySelectorAll<HTMLElement>('.small-header')[0].className;
92        let inside_header = document.querySelectorAll<HTMLElement>('.inside-header')[0].className;
93
94        let btns: string[] = [];
95        let b =
96        for (le
97          btns.
98          b[i].className += " btn-disabled disabled"
99        }
100
101       document.querySelectorAll<HTMLElement>('.inside-header')[0].style.display = 'inline';
102
103       document.querySelectorAll<HTMLElement>('.small-header')[0].className += " disappear";
104       document.querySelectorAll<HTMLElement>('.inside-header')[0].className += " appear";
105
106       setTimeout( () => {
107         for (let i = 0; i < btns.length; ++i) {
108           b[i].className += btns[i];
109         }
110
111         document.querySelectorAll<HTMLElement>('.small-header')[0].style.display = 'none';
112
113         document.querySelectorAll<HTMLElement>('.small-header')[0].className = small_header;
114         document.querySelectorAll<HTMLElement>('.inside-header')[0].className = inside_header;
115       }, 2000);
116     }
117
118   }
```

(property) Element.className: string

Returns the value of element's class content attribute. Can be set to change it.

Image 5.2.6 Search bar closure

Πριν εμβαθύνω στα άλλα στοιχεία, είναι επιτακτική ανάγκη να εξηγήσω τις προσαρμοσμένες υπηρεσίες μου όπως εμφανίζονται στον class constructor. Μια τέτοια υπηρεσία είναι η LoggingService, η οποία είναι υπεύθυνη για τη διαχείριση των δεδομένων χρήστη, το χειρισμό του API σύνδεσης της Google και τη διατήρηση ενός subject που επιτρέπει σε κάθε στοιχείο που επιθυμεί να παρακολουθεί τις αλλαγές στη μεταβλητή χρήστη να εγγραφεί σε αυτό. Με αυτόν τον τρόπο, οποιοδήποτε στοιχείο μπορεί να ειδοποιηθεί όταν τροποποιηθεί η τιμή χρήστη.

```
10      private _user = new BehaviorSubject<SocialUser | null>(null);
11      public userValue !: SocialUser | null;
12      public readonly user: Observable<SocialUser | null> = this._user.asObservable();
```

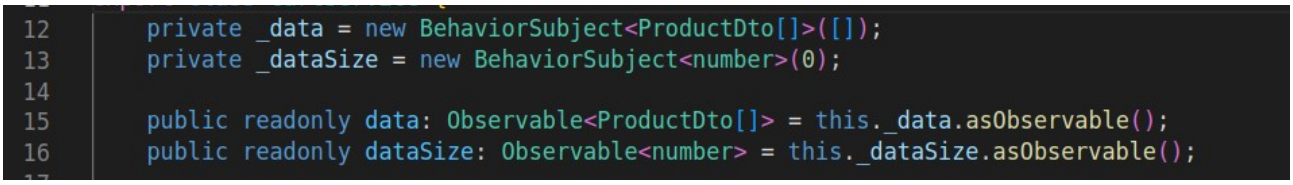
Image 5.2.7 User data

Επιπλέον, για να εγγραφώ στη βάση δεδομένων μου, υπάρχει μια συνάρτηση loginServer που καλείται μόλις συνδεθεί ένας χρήστης.

```
50    loginServer(userValue: SocialUser) {
51
52        let params = new HttpParams();
53        // console.log("login server: ", userValue.id)
54        if (userValue) {
55            params = params
56                .set('user_id', userValue.id)
57                .set('user_firstName', userValue.firstName)
58                .set('user_lastName', userValue.lastName)
59                .set('user_email', userValue.email)
60                .set('user_name', userValue.name)
61                .set('user_authToken', userValue.authToken)
62                .set('user_idToken', userValue.idToken)
63                .set('user_authorizationCode', userValue.authorizationCode)
64                .set('user_provider', userValue.provider);
65        }
66        // console.log(params);
67
68        return this.httpClient.get<void>(environment.path + 'login', { params });
69    }
```

Image 5.2.8 login Server

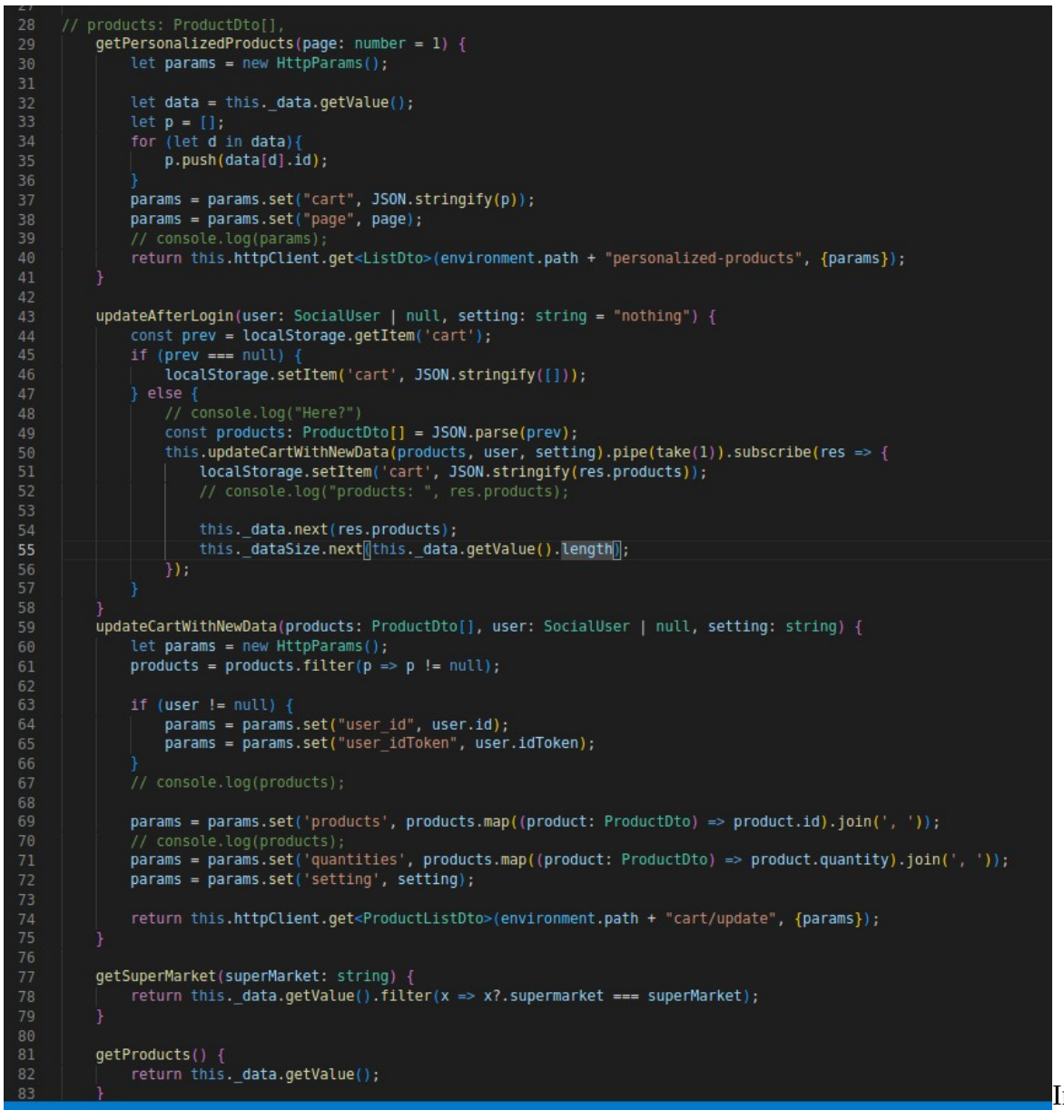
Μια άλλη τέτοια υπηρεσία είναι η CartService, η οποία είναι υπεύθυνη για τη διαχείριση των δεδομένων του καλαθιού. Συγκεκριμένα, έχει ένα θέμα (ως χρήστης) στο οποίο όλα τα αντικείμενα μεταφοράς δεδομένων προϊόντος (Data Transfer Object - DTO) φτάνουν από το backend. Αυτή τη φορά, χρειαζόμαστε δύο μεταβλητές, μία για τα δεδομένα και μία για το μέγεθος.



```
12      private _data = new BehaviorSubject<ProductDto[]>([]);
13      private _dataSize = new BehaviorSubject<number>(0);
14
15      public readonly data: Observable<ProductDto[]> = this._data.asObservable();
16      public readonly dataSize: Observable<number> = this._dataSize.asObservable();
```

Image 5.2.9 data and datasize Subjects

Η LoggingService περιλαμβάνει μια ποικιλία λειτουργιών, όπως την ανάκτηση εξατομικευμένων προϊόντων για την αρχική σελίδα, την ενημέρωση του προσωπικού καλαθιού στη βάση δεδομένων και την αναδιοργάνωση του καλαθιού.



```
28  // products: ProductDto[],
29      getPersonalizedProducts(page: number = 1) {
30          let params = new HttpParams();
31
32          let data = this._data.getValue();
33          let p = [];
34          for (let d in data){
35              p.push(data[d].id);
36          }
37          params = params.set("cart", JSON.stringify(p));
38          params = params.set("page", page);
39          // console.log(params);
40          return this.httpClient.get<ListDto>(environment.path + "personalized-products", {params});
41      }
42
43      updateAfterLogin(user: SocialUser | null, setting: string = "nothing") {
44          const prev = localStorage.getItem('cart');
45          if (prev === null) {
46              localStorage.setItem('cart', JSON.stringify([]));
47          } else {
48              // console.log("Here?")
49              const products: ProductDto[] = JSON.parse(prev);
50              this.updateCartWithNewData(products, user, setting).pipe(take(1)).subscribe(res => {
51                  localStorage.setItem('cart', JSON.stringify(res.products));
52                  // console.log("products: ", res.products);
53
54                  this._data.next(res.products);
55                  this._dataSize.next(this._data.getValue().length);
56              });
57          }
58      }
59      updateCartWithNewData(products: ProductDto[], user: SocialUser | null, setting: string) {
60          let params = new HttpParams();
61          products = products.filter(p => p != null);
62
63          if (user != null) {
64              params = params.set("user_id", user.id);
65              params = params.set("user_idToken", user.idToken);
66          }
67          // console.log(products);
68
69          params = params.set('products', products.map((product: ProductDto) => product.id).join(', '));
70          // console.log(products);
71          params = params.set('quantities', products.map((product: ProductDto) => product.quantity).join(', '));
72          params = params.set('setting', setting);
73
74          return this.httpClient.get<ProductListDto>(environment.path + "cart/update", {params});
75      }
76
77      getSuperMarket(superMarket: string) {
78          return this._data.getValue().filter(x => x?.supermarket === superMarket);
79      }
80
81      getProducts() {
82          return this._data.getValue();
83      }
```

Image 5.2.10 Basic functions

```
    updateProduct(product: ProductDto) {
        let list = this._data.getValue().filter(x => x.product != product.product);

        let v = this._data.getValue();
        for (let i = 0; i < v.length; i++) {
            if (v[i].product == product.product) {
                list.splice(i, 0, product);

                this._data.next(v);
                this._dataSize.next(v.length);

                this.updateLocalStorage("overwrite");

                return ;
            }
        }
    }
    updateProducts(products: ProductDto[]) {
        this._data.next(products);
        this._dataSize.next(products.length);

        this.updateLocalStorage("overwrite");
    }

    deleteProduct(product: ProductDto) {
        const list = this._data.getValue().filter(item => item != product);

        this._data.next(list);
        this._dataSize.next(list.length);
        this.updateLocalStorage("overwrite");
    }

    moveProduct(supermarket: string, startIndex: number, endIndex: number) {
        const list = this._data.getValue();

        // Find the 2 item indexes
        let index1 = -1, index2 = -1;
        let j = 0;
        for (let i = 0; i < list.length; i++) {
            if (j === startIndex) {
                index1 = i;
            }
            if (j === endIndex) {
                index2 = i;
            }

            if (list[i].supermarket == supermarket) {
                j++;
            }
        }
        let p_rem = list[index1];
        let list2 = this._data.getValue().filter(item => item != p_rem);

        list2.splice(index2, 0, p_rem);

        this._data.next(list2);
```

Image 5.2.11 Basic functions

```
export interface ProductDto {
    id: number;

    product : string;
    product_english: string;

    price: number;
    metric: string;

    supermarket: string;
    supermarket_english: string;

    quantity: number;

    company ?: string;
    company_english ?: string;

    category ?: string;
    category_english : string;

    img ?: string;
    url ?: string;

    ratio ?: number;
    weight ?: number;

    discount ?: boolean;
    discount_value : number;
    lastUpdated?: string;

    dropPrice: number;
}
```

Image 5.2.12 The ProductDto object

Μια άλλη τέτοια υπηρεσία είναι η DataService, η οποία είναι υπεύθυνη για τη διαχείριση των οπτικών δεδομένων. Συγκεκριμένα, χειρίζεται την αναζήτηση, τις προσφορές, τον ιστορικό των τιμών σε ένα μόνο προϊόν.

```
41      constructor(private httpClient: HttpClient){}
42
43      search(search: string, page: number) {
44          let params = new HttpParams();
45          params = params.set('search', search);
46          params = params.set('page', page);
47          // params = params.set('filters', filters);
48
49          return this.httpClient.get<ListDto>(environment.path + 'search', { params }); // pass DTO HERE
50      }
51      getHotDeals(page: number) {
52          let params = new HttpParams();
53          params = params.set('page', page);
54
55          return this.httpClient.get<ListDto>(environment.path + 'bestdeals', {params});
56      }
57      getPrices(id: number) {
58          return this.httpClient.get<{prices: number[], dates: string[]}>(environment.path + 'prices/' + id);
59      }
60
61  }
```

Image 5.2.13 DataService API

Επιπλέον, σχεδίασα μια λειτουργική μονάδα λίστας που χειρίζεται την ενημέρωση διεπαφής χρήστη μιας λίστας προϊόντων. Σημειώση ότι σε κάθε αναζήτηση φορτώνει μόνο 25 προϊόντα για καλύτερη απόδοση.

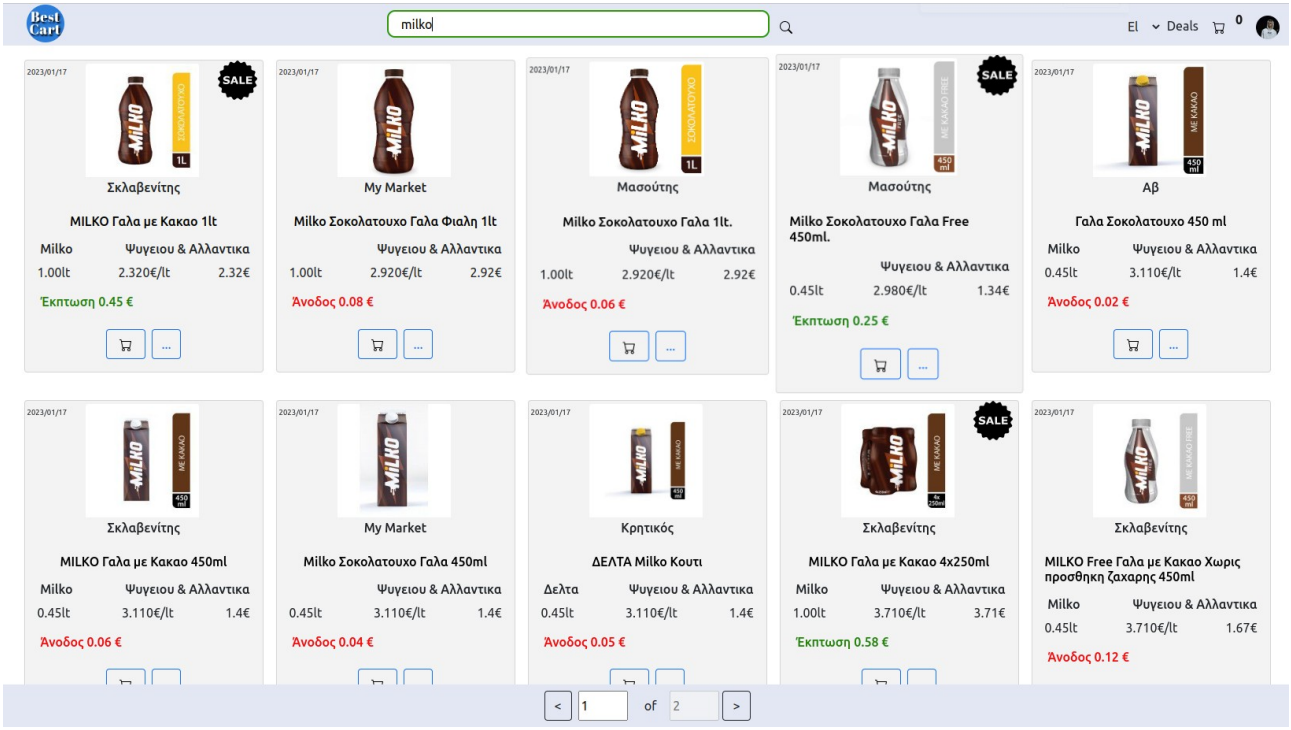


Image 5.2.14 Custom search.

Στη συνέχεια, η προσθήκη ενός προϊόντος στο καλάθι εμφανίζεται ως

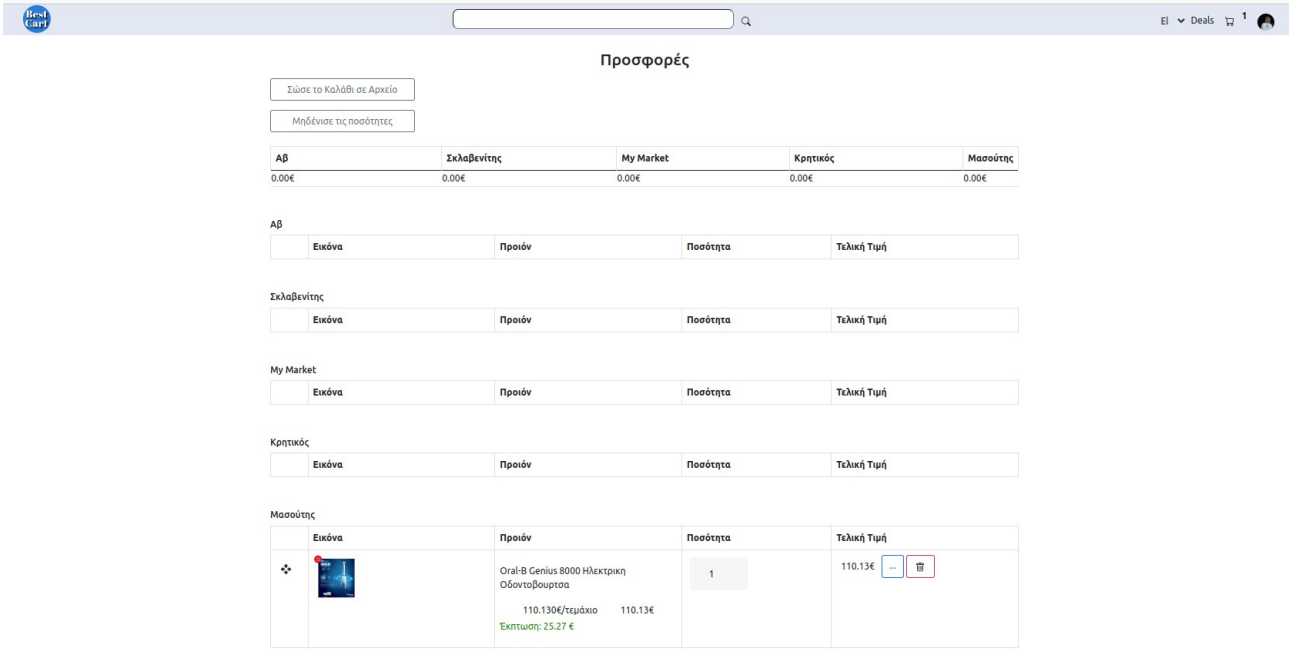


Image 5.2.15 Cart

Αυτό φορτώνει προτάσεις στην αρχική σελίδα.

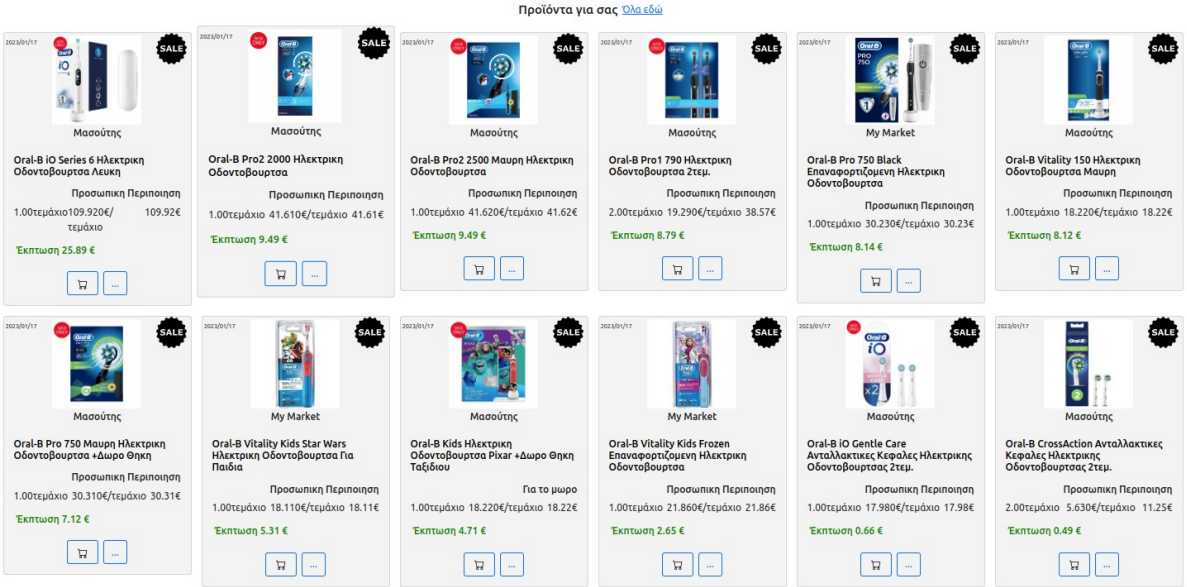


Image 5.2.16 Suggestions

Τέλος, έχουμε τις καυτές προσφορές

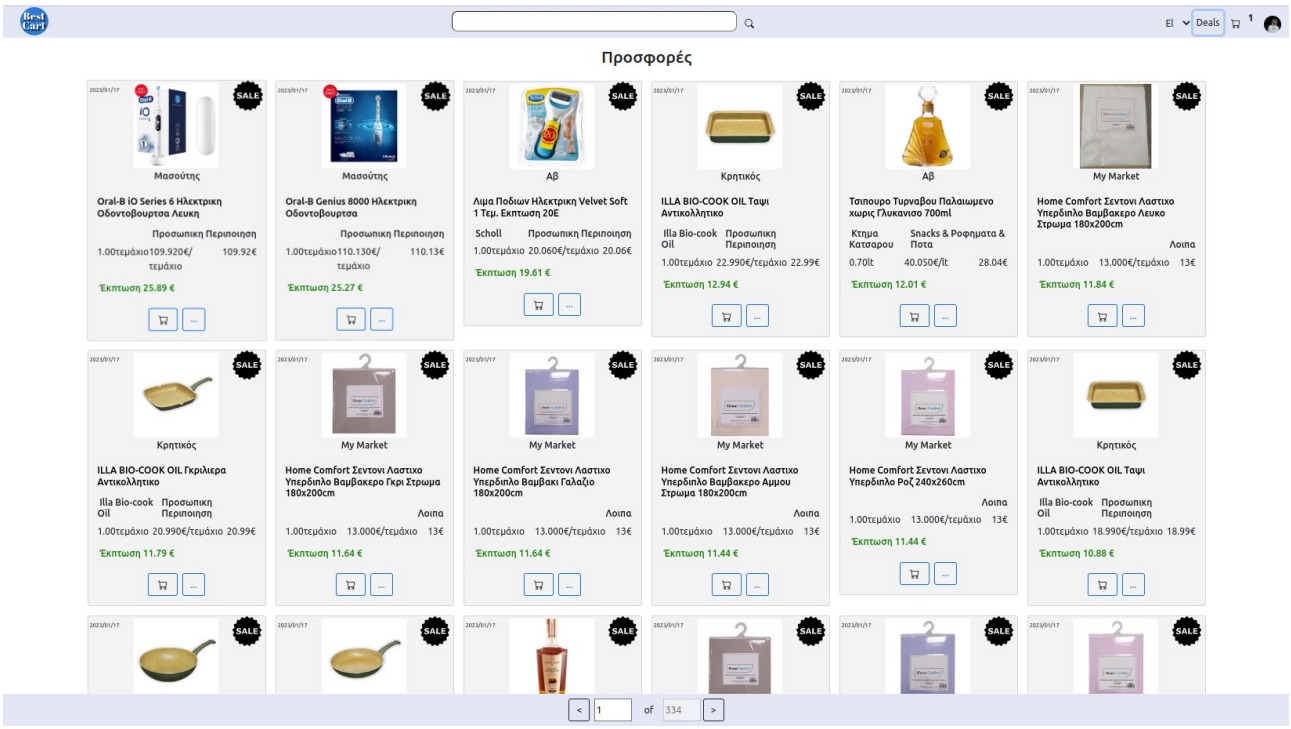


Image 5.2.17 Hot deals

5.3 Αποτελέσματα και Απόδοση

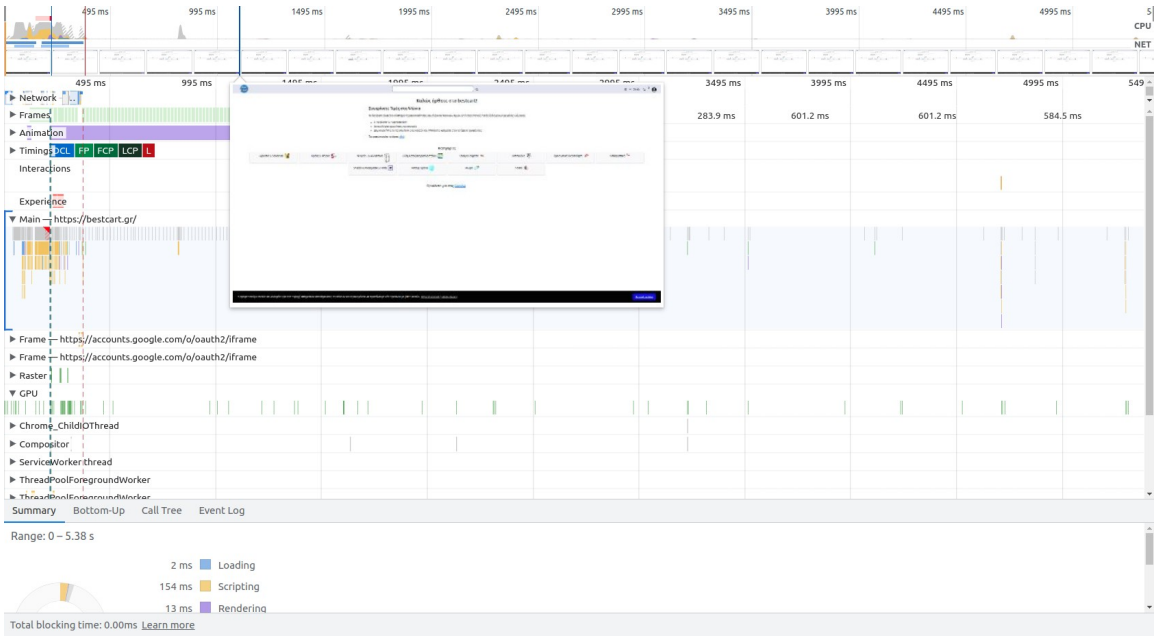


Image 5.3.1 Application load on a new browser

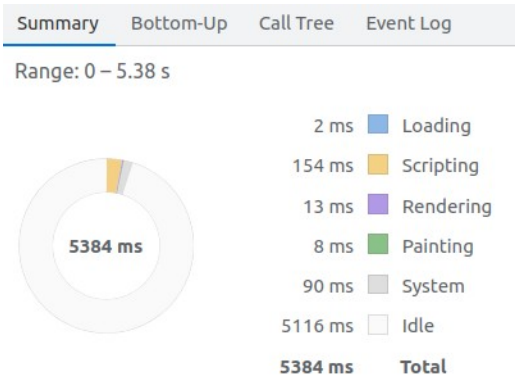


Image 5.3.2 Summary

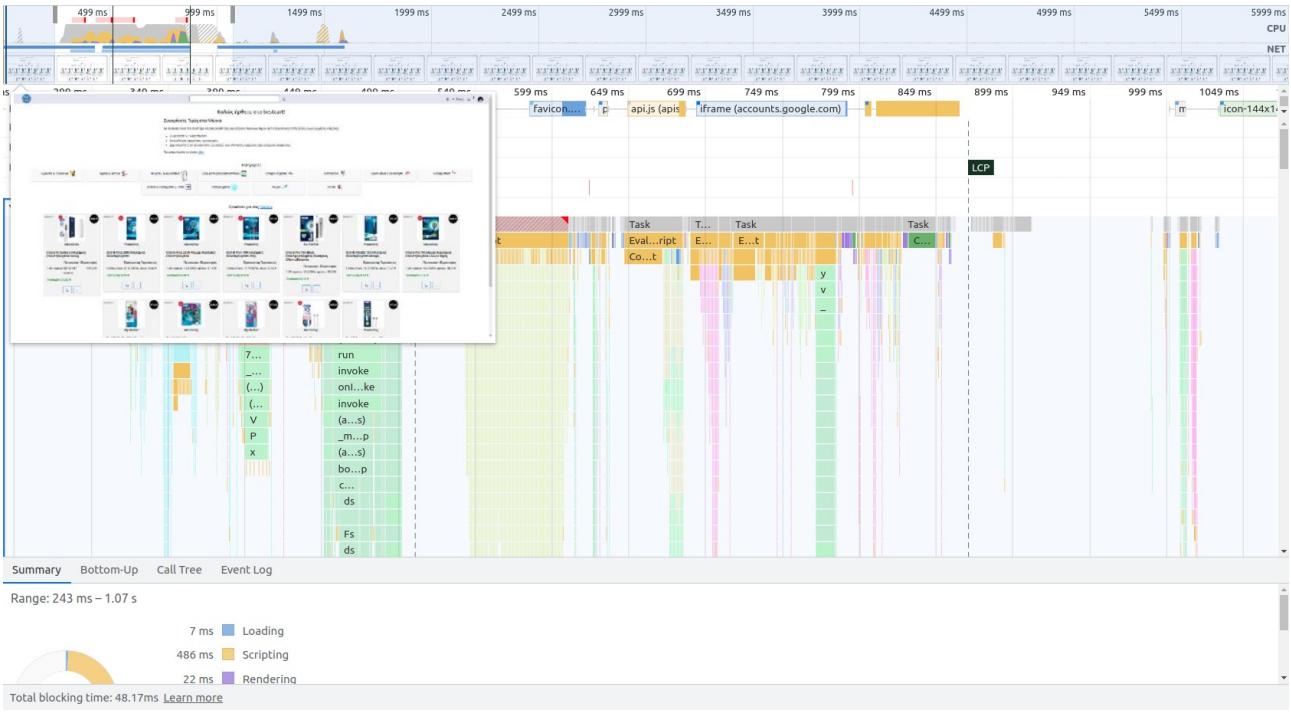


Image 5.3.3 Application load on a previously logged in device

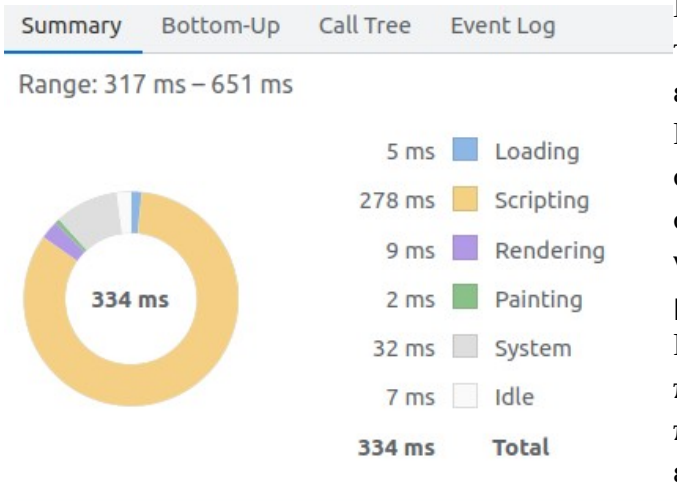


Image 5.3.4 Summary

Μπορούμε να δούμε την αισθητή διαφορά στην ταχύτητα. Αυτό είναι αναμενόμενο. Η εφαρμογή είναι προοδευτική (Progressive Web App, PWA). Τα PWA είναι εφαρμογές ιστού που χρησιμοποιούν σύγχρονες τεχνολογίες ιστού, όπως οι εργαζόμενοι σε υπηρεσίες και τα μανιφέστα εφαρμογών ιστού για να παρέχουν μια εμπειρία χρήστη παρόμοια με αυτή μιας εγγενούς εφαρμογής σε φορητή συσκευή. Τα PWA μπορούν να προσπελαστούν μέσω ενός προγράμματος περιήγησης ιστού όπως οι παραδοσιακοί ιστότοποι, αλλά μπορούν επίσης να εγκατασταθούν στη συσκευή ενός χρήστη όπως μια εγγενής εφαρμογή και μπορούν να λειτουργήσουν εκτός σύνδεσης ή με δίκτυο χαμηλής ποιότητας. Τα PWA έχουν σχεδιαστεί για να είναι γρήγορα, αξιόπιστα και ελκυστικά και μπορούν να χρησιμοποιηθούν σε πολλές πλατφόρμες όπως το Android και το iOS.

Τα PWA μπορούν επίσης να προστεθούν στην αρχική οθόνη της συσκευής ενός χρήστη και μπορούν να λαμβάνουν push notifications. Μπορούν επίσης να γίνουν indexed από τις μηχανές αναζήτησης, διευκολύνοντας τους χρήστες να τα βρουν. Για όλες αυτές τις λειτουργίες, πρέπει να φορτώσει περισσότερα αρχεία από ό,τι ήταν αρχικά η ιστοσελίδα. Ως εκ τούτου, η αρχική καθυστέρηση είναι εύλογη.

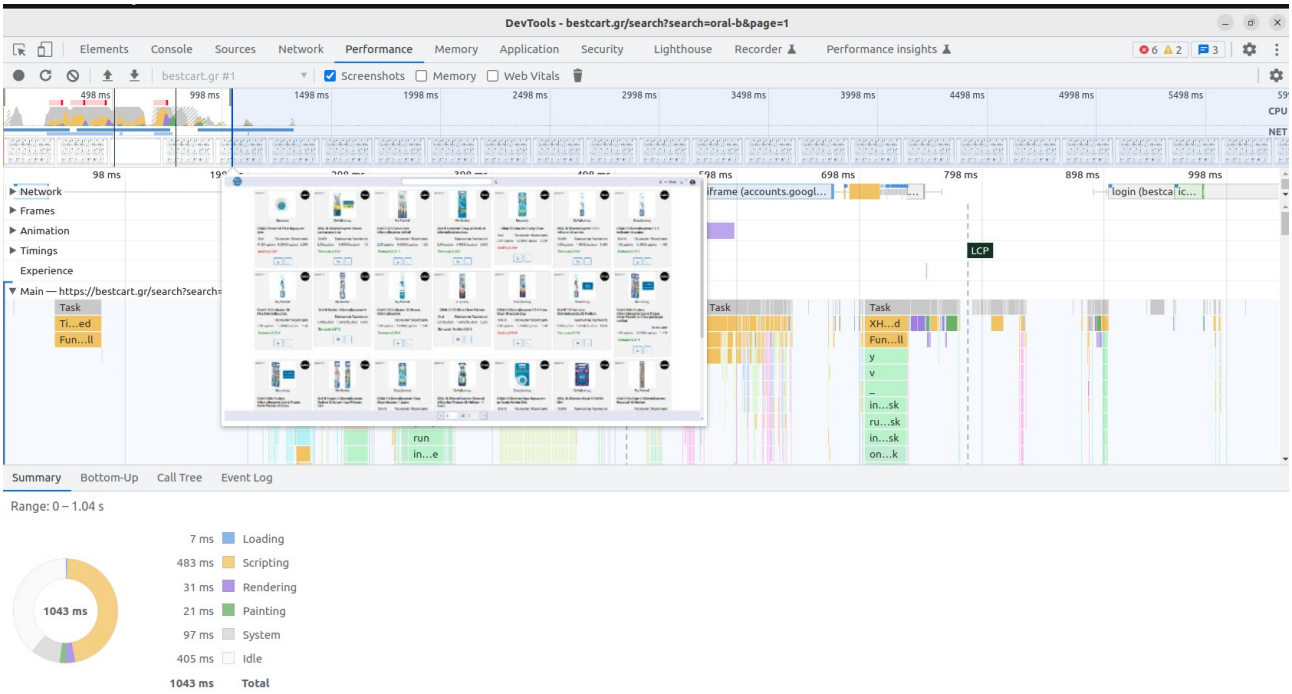


Image 5.3.5 “Oral-B” search results

Όπως βλέπουμε, αυτός είναι ο χρόνος απόκρισης API που προστίθεται στις καθυστερήσεις του UI.

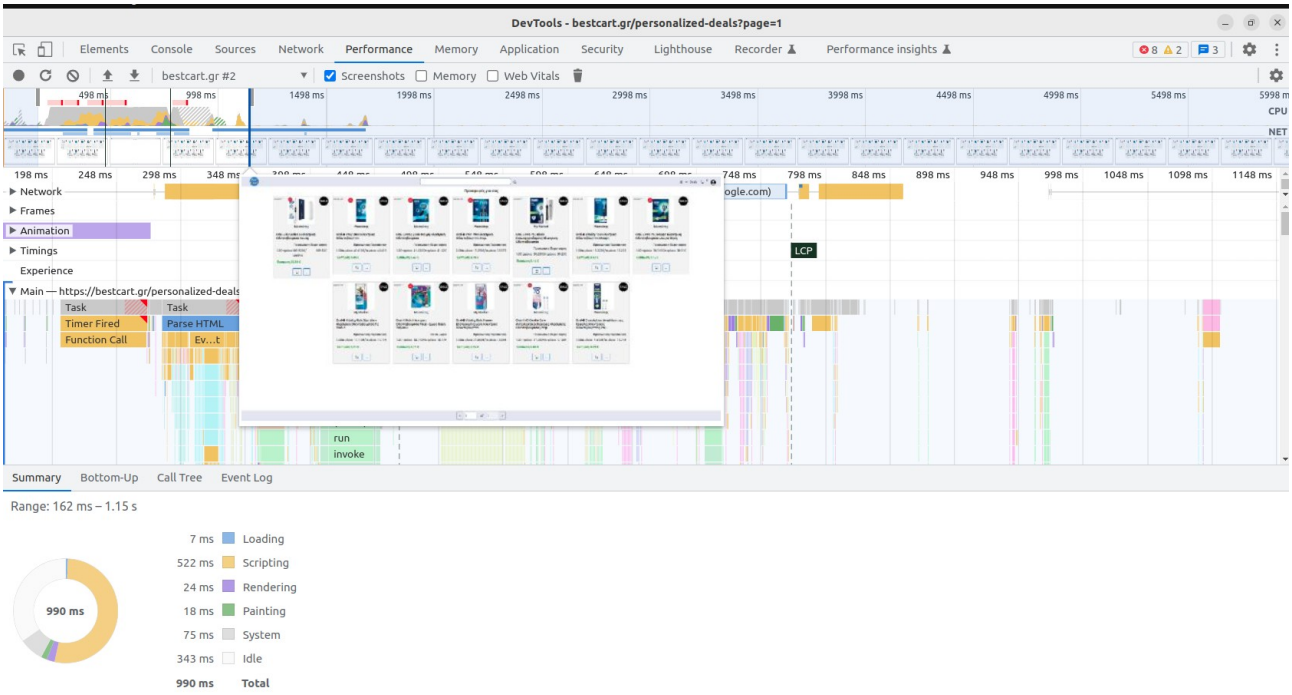


Image 5.3.6 Personalized deals result

Συμπερασματικά, τα αποτελέσματα της δοκιμής της διεπαφής χρήστη υποδεικνύουν ότι κατά μέσο όρο, η σελίδα φορτώνεται πλήρως μέσα σε 1 δευτερόλεπτο. Αξίζει να σημειωθεί ότι αυτά τα αποτελέσματα λήφθηκαν με χρήση VPS με 8 GB RAM, κάτι που πιθανότατα συνέβαλε στον αποτελεσματικό χρόνο φόρτωσης. Συνολικά, η σχεδίαση και η υλοποίηση της διεπαφής χρήστη ήταν επιτυχείς στην παροχή γρήγορης και ανταποκρινόμενης εμπειρίας στους χρήστες. Είναι σημαντικό να συνεχιστεί η παρακολούθηση και η βελτιστοποίηση της απόδοσης της διεπαφής χρήστη για να διασφαλιστεί ότι η εφαρμογή Ιστού παραμένει αποτελεσματική και φιλική προς το χρήστη.

Chapter 1: Web Crawlers: a system for tracking and controlling retail prices from large-scale heterogeneous data sources

1.1 Introduction

Web crawlers, also known as spiders or bots, are automated programs that traverse the World Wide Web in a methodical, automated manner. They are used to discover, index, and organize information on the internet, making it easily accessible to users through search engines. Web crawlers work by following links from one webpage to another, and then indexing the information on those pages for search engines to use. This process is known as web crawling. Web crawlers are an essential component of the modern internet, helping to ensure that users can quickly and easily find the information they need.

The denial of access to daily data by supermarkets poses a significant challenge. One solution is to utilize web crawling techniques, as web scraping is particularly adept at swiftly gathering and processing large amounts of data. However, it should be noted that while browsers are efficient in executing JavaScript, displaying images, and formatting information in a user-friendly manner, utilizing internal Application Programming Interfaces (APIs) provided by the websites could also facilitate the acquisition of data. Nevertheless, it is important to take into consideration that various websites often implement measures to restrict access to these internal APIs. As a result, web scraping emerges as the sole viable solution to this problem.

1.2 Available Interfaces

In the field of web scraping and data extraction, there are several techniques that can be implemented using the Python programming language. Some of the most commonly used libraries and frameworks include Scrapy [3], Beautiful Soup [4], Selenium [5], and Requests-HTML [6] [2]

Scrapy, an open-source and collaborative web crawling framework, allows for the efficient extraction of data from websites, as well as the use of APIs. Beautiful Soup, a Python library for parsing HTML and XML documents, enables the creation of parse trees for the extraction of information from these types of documents. Selenium, a browser automation tool, allows for the automation of browser activities such as clicking buttons, filling out forms and navigating pages. Finally, Requests-HTML, an HTML parsing library, allows for web scraping tasks to be performed using the Python requests library [7].

Scrapy is an open-source and collaborative web scraping framework that allows for the efficient extraction of data from websites, as well as the use of APIs. It has built-in support for handling common web scraping tasks such as logging in, handling cookies, and following redirects. Scrapy also allows for the easy export of data in multiple formats, such as CSV, JSON, or XML. The library is actively maintained, and it has a large community that can provide support and guidance. However, it may have a steeper learning curve for those who are new to web scraping and it may not be suitable for small scraping tasks [8].

Beautiful Soup is a Python library for parsing HTML and XML documents, which enables the extraction of information from these types of documents through parse trees. It is simple to use and easy to learn, and it has a large community that can provide support and guidance. However, Beautiful Soup only parses HTML and XML, it does not interact with web pages like Selenium and it may not be the best option for scraping large and complex web pages [7].

Selenium is a browser automation tool that allows for the automation of browser activities such as clicking buttons, filling out forms and navigating pages. This makes it ideal for interacting with dynamic web pages. However, Selenium may require more technical knowledge and setup compared to other libraries and it may be less efficient for large scale scraping [7].

Requests-HTML is an HTML parsing library that allows for web scraping tasks to be performed using the Python requests library. It's simple to use but it has a limited capability compared to other libraries, it's mainly used for small web scraping tasks and it may not be suitable for more complex scraping tasks that require browser automation or handling of cookies and redirects.

For the purpose of my thesis, I have decided to use Selenium as the primary tool for web scraping. The supermarket websites that I aim to scrape require loading pages with scrolling and make use of JavaScript, as well as server-side rendering. These features can make scraping more challenging and traditional web scraping methods may not be sufficient. Selenium, as a browser automation tool, allows for the automation of browser activities such as loading pages with scrolling, interacting with JavaScript, and handling dynamic web pages. This makes it an ideal choice for scraping websites that have these features, as it mimics the actions of a user and can bypass any limitations that may be imposed by traditional web scraping methods. Additionally, Selenium has a large community and a wealth of resources available, which can be extremely helpful for troubleshooting any issues that may arise during the scraping process.

1.3 Generic Web Crawlers

The need for generic web crawlers arises from the requirement to scrape data from multiple sources with minimal effort. In my case, as I am working on a thesis that involves scraping product information from supermarkets, I need a web crawler that can be easily configured and adapted to work with different supermarkets. Each supermarket has its own website structure and data format, and having to set up a new crawler for each individual supermarket would be time-consuming and resource-intensive. A generic web crawler can be customized to work with different websites by changing the configurations, rules and patterns to match the target website. This approach allows me to easily add new supermarkets to my scraping process without the need to build new crawlers from scratch. Additionally, using a generic web crawler can help me to ensure consistency and accuracy of the data being scraped from the different supermarkets.

In order to achieve a consistent and organized approach to scraping data from multiple supermarkets, I have implemented a specific format for each supermarket. Specifically, I have created a 'scrape' folder, within which I have created a separate folder for each supermarket. Within each of these folders, I have included a scrape file that is specific to that supermarket, a file containing the links to be scraped, a ".txt" file for storing relevant information, and a folder for storing past data. This format allows for easy organization and access to the data being scraped from each supermarket, and ensures that the process of adding new supermarkets is streamlined and efficient.

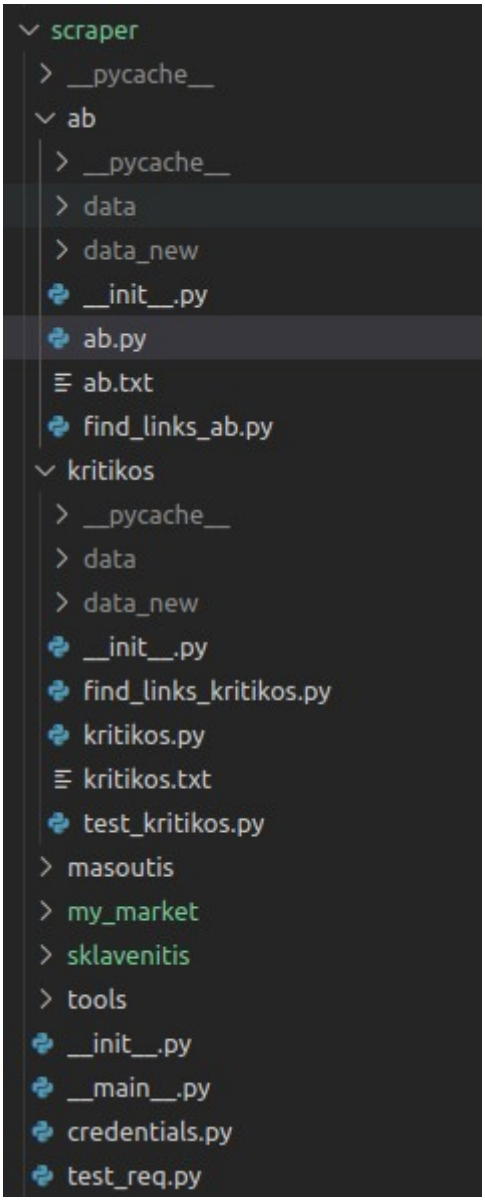


Image 1.1 Generic web crawlers

I have implemented a generic function, named "get_categories_{supermarket}", within the "find_links" file, that is specifically designed to utilize Selenium for the purpose of navigating to the e-shop page of a particular supermarket and scraping the links and categories of that supermarket. This function enables the efficient retrieval of relevant data, and also maintains a list containing all of the scraped data, thus eliminating the need to repeatedly retrieve such data. This approach ensures that the process of scraping the links and categories of a supermarket is both efficient and accurate.

```
class_conventions = {
    'title': "ProductListItem_title__e6MEz",
    'product': "ProductListItem_productItem__cKUyG",
    "additional_info": "ProductListItem_titleDesc__JzvBv",

    'price': 'ProductListItem_finalPrice__sEMjs',
    "beginPrice": "ProductListItem_beginPrice__vK_Dk",

    "weight": "ProductListItem_titleDesc__JzvBv",
    'ratio': 'ProductListItem_description__DRAGa',
    'img': "ProductListItem_productImage__HbseK",
    'url': "ProductListItem_productLink__BZo3P",
}
```

Image 1.2 A basic “class_conventions” dictionary of HTML & CSS attributes

In order to maximize productivity and efficiency when creating web crawlers for each supermarket, I have developed a basic structure to follow. This structure involves creating a new "class_convention" object for each new scraper, which includes all of the necessary HTML and CSS attributes for acquiring the desired data. In particular, this structure includes the product name, the price, the original price if the product is on discount, the weight (if applicable), the price-value ratio, the image URL (as each supermarket loads their images through an API), and the product URL. This approach allows for a consistent and efficient method of acquiring data from multiple supermarkets, while also facilitating the addition of new supermarkets to the scraping process.

Additionally, each web crawler utilizes a specific Application Programming Interface (API) that, through the use of the webdriver, executes JavaScript commands in order to retrieve the data. The method of utilizing these functions involves first determining the number of products present on a webpage, and subsequently iterating through each product to extract their attributes. This approach allows for an efficient and comprehensive method of data retrieval, while also ensuring that the process is automated and streamlined.

```
def kritikos():
    def fill_Kritikos(link, category):
        def getTitle(i):…

        def getUrl(i):…

        def getImg(i):…


        def getPrice(i): …

        def getValueRatio(i):…

        def getWeightFromPriceRatio(price, ratio):…
```

Image 1.3 Basic web crawler’s API

```
        def getTitle(i):
            try:
                title = driver.execute_script( f"""return document.getElementsByClassName("{class_conventions["product"]}")[{i}].getElementsByClassName("{class_conventions["

                ## find company name
                upper = []
                for i, word in enumerate(title.split(' ')):
                    if word.isupper():
                        upper.append((word, i))
                company = ""
                for word, i in upper:
                    if len(word) > 2:
                        company += word + " "

                if len(company) > 0 and company[-1] == " ":
                    company = company[:-1]

                # valueRatioStr = driver.execute_script( f"""return document.getElementsByClassName("{class_conventions["product"]}")[{i}].getElementsByClassName("{class_con

            except:
                title = ""
                company = ""
                # valueRatioStr = ""

            if title is None:
                title = ""

            return title, company

        def getUrl(i):
            try:
                url = driver.execute_script( f"""return document.getElementsByClassName("{class_conventions["product"]}")[{i}].getElementsByClassName("{class_conventions["ur
            except:
                url = ""
            if url is None:
                url = ""

            return url

        def getImg(i):
            try:
                img = driver.execute_script(f"""return document.getElementsByClassName("{class_conventions["product"]}")[{i}].getElementsByTagName("img")[0].getAttribute("sr
            except:
                img = None

            return img
```

Image 1.3.1 Implementation

```
def getPrice(i):
    discount = False
    try:
        beg_price = driver.execute_script( f"""return parseFloat(document.getElementsByClassName("{class_conventions["product"]}")[{i}].getElementsByClassName("{class_conventions["beginPrice"]}")[0].innerText.r
        if beg_price >= 0.0:
            discount = True

    except:
        price = None
    try:
        price = driver.execute_script( f"""return parseFloat(document.getElementsByClassName("{class_conventions["product"]}")[{i}].getElementsByClassName("{class_conventions["price"]}")[0].innerText.replace(',
    except:
        price = None

    return price, discount

def getValueRatio(i):
    try:
        # document.getElementsByClassName("product")[0].getElementsByClassName("priceKil")[0].innerText.replace(',','.').replace('\n','');
        valueRatioStr = driver.execute_script( f"""return document.getElementsByClassName("{class_conventions["product"]}")[{i}].getElementsByClassName("{class_conventions["ratio"]}")[0].innerText; """) #.repla

        if "- αντί " in valueRatioStr:
            weight_metric = valueRatioStr.split('- αντί ')[1]
            weight, metric = helper.fromWeightStrToNumber(weight_metric)

            price_ratio = valueRatioStr.split('- αντί ')[0].replace(',','').split(' ')
            for item in price_ratio:
                try:
                    valueRatio = float(item)
                    break
                except:
                    pass

        elif " €/ " in valueRatioStr:
            weight_metric = valueRatioStr.split(' €/ ')[1]
            valueRatio = valueRatioStr.split(' €/ ')[0]
            metric = helper.getMetric(weight_metric)
            weight = None
        elif " το " in valueRatioStr:
            weight_metric = valueRatioStr.split(' το ')[1]
            weight, metric = helper.fromWeightStrToNumber(weight_metric)
            price_ratio = valueRatioStr.replace(',','').split(' ')
            for item in price_ratio:
                try:
                    valueRatio = float(item)
                    break
                except:
                    pass

    except:
        valueRatio, metric, weight = None, None, None

    try:
        valueRatio
    except:
        valueRatio = None
    return valueRatio, metric, weight

def getWeightFromPriceRatio(price, ratio):
    # ratio = Euro / weight => weight = Euro / ratio
    try:
        return float(price) / float(ratio)
    except:
        return 1.0
```

Image 1.3.2 Implementation

Another challenge that arises when scraping data from multiple supermarkets is the diversity of categories declared by each supermarket. To address this, I have created a mapping system that maps each supermarket's category to a generic category within my application. This approach allows for consistency in categorization and facilitates the analysis of the scraped data.
The category mappings that have been implemented are as follows:

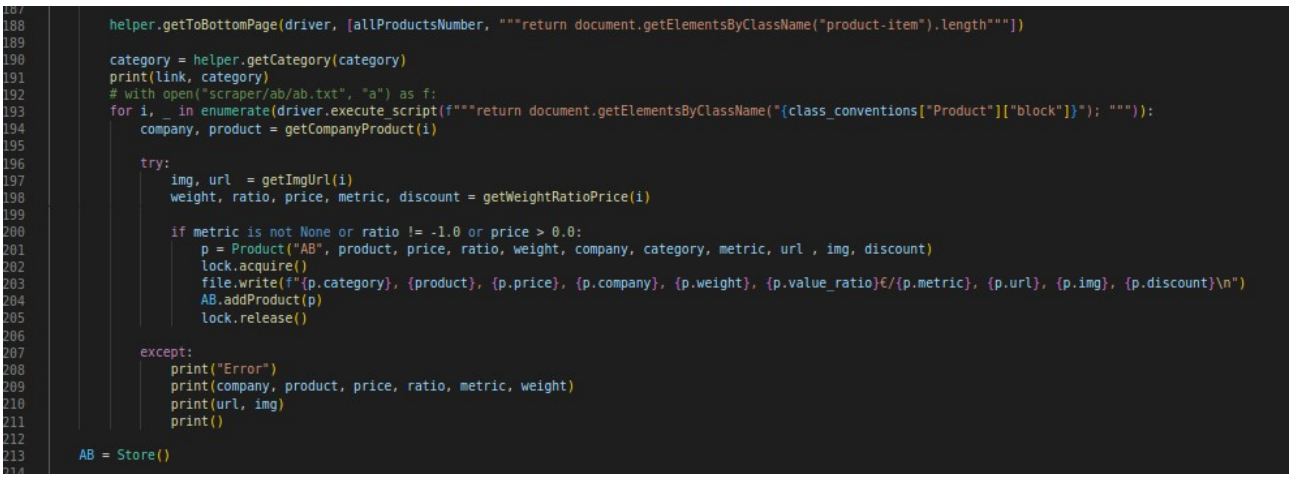
```
def categoryMappings():
    categories = {
        "Φρούτα & Λαχανικά": ['Βιολογικά Προϊόντα','Vegan Προϊόντα', "Οπωροπωλείο", "Φρέσκα Φρούτα & Λαχανικά", 'Υγιεινή Διατροφή', 'Φρούτα - Λαχανικά'],
        "Κρέας & Ψάρια": ['Φρέσκο Κρέας & Ψάρια', 'Φρέσκο Ψάρι & Θαλασσινά', 'Φρέσκο Κρέας', 'Κρεοπωλείο Online'],
        "Ψυγείου & Αλλαντικά": [ 'Είδη Ψυγείου', 'Γαλακτοκομικά & Είδη Ψυγείου', 'Τυριά, Αλλαντικά & Delicatessen', 'Γάλατα, Ροφήματα & Χυμοί ψυγείου',
                                 'Γιαούρτια, Κρέμες γάλακτος & Επιδόρπια ψυγείου','Τυροκομικά',  'Αλλαντικά', 'Ορεκτικά & Delicatessen',
                                 'Αυγά, Βούτυρο, Νωπές Ζύμες & Ζωμοί', 'Μακαρόνια, Όσπρια & Ζυμαρικά'],
        "Είδη Αρτοζαχαροπλαστείου": ['Ζαχαροπλαστείο - Αρτοποιείο', 'Είδη Αρτοζαχαροπλαστείου', 'Άρτος Ζαχαροπλαστείο',],
        "Έτοιμα Γεύματα": ['Έτοιμα Γεύματα', 'Κονσέρβες'],
        "Κατοικίδια": ['Για κατοικίδια', 'Τροφές & Είδη για Κατοικίδια', 'Pet Shop Online'],
        "Προσωπική Περιποίηση": ['Είδη προσωπικής περιποίησης', 'Χαρτικά, Πάνες & Σερβιέτες', 'Καλλυντικά & Είδη Προσωπικής υγιεινής', 'Είδη Οικιακής χρήσης',
                                 'Καλλυντικά & Είδη Προσωπικής υγιεινής', 'Χαρτικά, Πάνες & Σερβιέτες', 'Ανδρική - Γυναικεία Περιποίηση'],
        "Καθαριστικά": [ 'Είδη Σπιτιού - Καθαρισμού ', 'Καθαριστικά - Χαρτικά & είδη σπιτιού', 'Απορρυπαντικά & Είδη Καθαρισμού', 'Προϊόντα Υγιεινής & Χαρτικά'],
        "Για το μωρό": ['Βρεφική Περιποίηση', 'Όλα για το μωρό', 'Βρεφικές & Παιδικές τροφές',],
        "Snacks & Ροφήματα & Ποτά": ['Ξηροί Καρποί , Super Foods & Snacks', 'Κρασιά, ποτά, αναψυκτικά, νερά', 'Αναψυκτικά, Νερά & Χυμοί', 'Πρωϊνό - snacking & ροφήμα
                                     'Ξηροί Καρποί & Σνακ', 'Είδη πρωινού & Ροφήματα', 'Κάβα', 'Πρωινά, Δημητριακά & Ροφήματα'],
        "Κατεψυγμένα": ['Κατεψυγμένα Προϊόντα', 'Κατεψυγμένα τρόφιμα', 'Κατεψυγμένα', 'Γλυκά, Μπισκότα & Ζαχαρώδη'],
        "Λοιπά": ['Προϊόντα Μασούτης', 'Προϊόντα Χωρίς Γλουτένη', 'Dips, Sauces & Dressings',  'Είδη Οικιακής Χρήσης', 'Παντοπωλείο Online', 'Βασικά τυποποιημένα τρό
    }

    return categories
```

Image 1.4 Category mappings

In order to efficiently store and organize the product data, I have developed a series of classes to manage the objects in a structured manner. Specifically, I have created a "Product" class that allows me to manipulate the object, rather than relying on unique variables for each product. This allows for easy manipulation of the data and facilitates the process of updating the database. For each product, I create a "Product" object, store it in a list, and simultaneously write it to a file. This file is then used to update the database, thus ensuring that the process is automated and efficient. This approach allows for a consistent and organized method of storing product data, while also facilitating the analysis and retrieval of the data.

Another issue that arrives is that each supermarket loads their products in two different ways. Some choose to use pages as the way to load a batch of products. Other’s choose to load more products as the user scrolls. For the first case, in order to facilitate that, I have designed a series of Javascript commands to find the total number of pages so that I can get into each one. For the second case, some supermarkets add the total number of products in the top of the page. If this doesn’t exist, the webdriver scrolls until no more data is loaded. If this exist, the web crawler scrolls and loads until it finds this number of products. However, due to some faults on their websites, that number might be false. For this reason, I have implemented a security rule that if the webdriver scrolls 5 times and no new products appear, it has reached the end of the page. And it debugs a message to notify me that there might have been lost some products.

In order to effectively integrate all of the previously mentioned components and to ensure the efficient and accurate scraping of data, I have developed a basic structure for the web crawler. The structure begins by determining the loading mechanism of the page, whether it is the scroll loading mechanism or not. If the page uses the scroll loading mechanism, the first step is to scroll through the end of the page. If not, the process proceeds to the next step. An iteration process is then implemented through the total number of products, where the basic data for each product is acquired. A "Product" object is created for each product and the object is written to a file, which can be used for later analysis. This approach allows for an automated and streamlined process of data scraping, while also ensuring that the data is organized and accessible for further analysis.



```
        helper.getToBottomPage(driver, [allProductsNumber, """return document.getElementsByClassName("product-item").length"""])

        category = helper.getCategory(category)
        print(link, category)
        # with open("scraper/ab/ab.txt", "a") as f:
        for i, _ in enumerate(driver.execute_script(f"""return document.getElementsByClassName("{class_conventions["Product"]["block"]}"); """)):
            company, product = getCompanyProduct(i)

            try:
                img, url  = getImgUrl(i)
                weight, ratio, price, metric, discount = getWeightRatioPrice(i)

                if metric is not None or ratio != -1.0 or price > 0.0:
                    p = Product("AB", product, price, ratio, weight, company, category, metric, url , img, discount)
                    lock.acquire()
                    file.write(f"{p.category}, {product}, {p.price}, {p.company}, {p.weight}, {p.value_ratio}€/{p.metric}, {p.url}, {p.img}, {p.discount}\n")
                    AB.addProduct(p)
                    lock.release()

            except:
                print("Error")
                print(company, product, price, ratio, metric, weight)
                print(url, img)
                print()

    AB = Store()
```

Image 1.5 Basic web scraper structure

1.4 Maximizing Web Crawlers Performance

The network delay is a primary bottleneck that can impede the performance of any web scraping project. The transmission of a request to the web server and the subsequent receipt of a response both result in a delay. While there are limitations to what can be done to resolve this issue, such as having a better connection or network card, there are several ways to boost performance.

One such method is the use of headless browsers, as supported by Selenium. In certain situations, a headless browser may not be viable as some websites employ web crawler checks and will not load the page if it does not appear as a browser. However, when a headless browser can be utilized, it can result in a speedup of approximately half the time.

Additionally, the use of multithreading can also improve performance, allowing for many web crawlers to run independently [9]. This method is effective only if the CPU is able to handle the multiple Selenium threads running simultaneously. In my personal experience, a speedup of approximately two times was achieved through the implementation of multithreading. Without any multi-threading, the time required to scrape all data from all websites was approximately six hours. With a pure multithreading setup, the time was reduced to one and a half hours. However, an even greater speedup can be achieved by splitting the links into two equal lists, resulting in a reduction of time by half. In my case, this reduced the time from one and a half hours to 30-45 minutes, depending on the connection.

Lastly, it is important to note that when utilizing multithreading, multiple threads may attempt to write to the same file simultaneously. To ensure the safety and integrity of the system, I implemented spinlocks. A spinlock is a synchronization mechanism that is used to lock a file when a thread attempts to write to it, and subsequently unlock it when the writing process is completed. This approach ensures that only one thread is able to access the file at a time, avoiding any potential conflicts or errors that may occur due to simultaneous access.

1.5 Future Improvements

One approach to improving the performance of web scrapers is to utilize remote servers for scraping. By leveraging the resources of remote servers, web scrapers can increase their scraping speed, handle more requests simultaneously, and reduce the risk of being blocked by web servers [1]. This approach involves running web scrapers on servers that are not located on the same network as the machine running the scraping script. This can be achieved by either renting a virtual private server (VPS) or using a cloud-based service such as Amazon Web Services (AWS) or Google Cloud Platform (GCP).

Additionally, using remote servers also allows for the use of multiple IP addresses, thus reducing the chances of being detected and blocked by website administrators. Furthermore, using remote servers also allows for the use of multiple threads, thus increasing the scraping speed. However, it's important to note that scraping on remote servers also comes with some drawbacks, such as increased costs and the need for a reliable internet connection.

Chapter 2: Back-end Service

2.1 Database Selection and the Structure

The selection of an appropriate database model can present a significant challenge, as there are a multitude of options available. These options include SQL-based databases, NoSQL databases, and graph databases, each of which are designed to address specific types of challenges. In the context of this project, the data load is relatively low, with a total of 70,000 distinct products and their associated data. Furthermore, the need for relationships in the data, such as those required for suggestions, is not particularly demanding. Given these considerations, a simple SQLite database is deemed sufficient for the needs of this project [10].

Additionally, a web service is necessary to handle the SQL queries, and provide a means for interacting with the database. Given the anticipated low volume of HTTP requests, a simple Flask application is deemed sufficient for this purpose, as it offers a lightweight and efficient solution for handling these types of requests.

Flask applications come equipped with various tools that can be utilized to enhance their functionality. One such tool is SQLAlchemy, an Object-Relational Mapping (ORM) library that allows for the interaction with databases in a more efficient and convenient manner. SQLAlchemy is a common choice for developers when it comes to working with databases in Flask applications. The utilization of SQLAlchemy allows for the establishment of a relational mapping between Python classes and SQL code. This enables the manipulation of the database through the use of Python classes, rather than writing raw SQL code, resulting in a more efficient and convenient development process [11].

Lastly, it is a secure way to protect the data from attacks as SQL Injection. SQLAlchemy prevents SQL injection by using a technique called "parameter binding". This technique involves separating the data from the SQL query and passing the data as a separate parameter. In this way, any data passed to the query is treated as a bound variable rather than as part of the SQL query itself. This ensures that any data passed to the query, regardless of its content, is treated as a value rather than as part of the SQL query, effectively neutralizing any SQL injection attempts. Additionally, SQLAlchemy also provides a way to use "sql expressions" instead of plain string concatenation, which also helps in preventing SQL injection. For example, when using plain string concatenation, a user input is added to the query string, however, when using SQLAlchemy, the user input is passed as a separate parameter to the query and it is not added to the query string. So even if the user input contain some malicious SQL code, it will not be executed because it is not part of the query string [11].

2.1.1 Products Table

```python
class Product(Base):
    __tablename__ = 'product'

    id = db.Column(db.Integer, primary_key=True, unique=True)

    supermarket = db.Column(db.String)
    supermarket_english = db.Column(db.String)

    product = db.Column(db.String, nullable=False)
    product_english = db.Column(db.String)

    company = db.Column(db.String)
    company_english = db.Column(db.String)

    category = db.Column(db.String)
    category_english = db.Column(db.String)

    metric = db.Column(db.String)

    _prices = db.Column(db.String)
    _datesUpdated = db.Column(db.String)

    value_ratio = db.Column(db.Float)
    weight = db.Column(db.Float)

    img = db.Column(db.String)
    url = db.Column(db.String)

    discount = db.Column(db.Boolean)

    exist = db.Column(db.Boolean)
```

Image 2.1.1.1 Product class: Basic attributes

In the implementation of the present system, certain self-defined variables have been utilized. One notable challenge encountered during the development process was the storage of the historical pricing data for each product. To address this challenge, the utilization of the variables "_prices" and "_datesUpdated" was implemented. These variables serve to concatenate and store the historical pricing information and respective dates for each product.

One of the key features of SQLAlchemy is its ability to define properties, hybrid properties, and setters within each class. Properties are class-level attributes that are used to define the columns of a table and their data types. Hybrid properties, on the other hand, are a combination of Python expressions and column-based properties. They are used to define computed fields that are not stored in the database but are calculated on the fly. Lastly, setters are used to define custom setter methods for properties, which can be used to perform additional actions when a property is set, such as validation or data transformation.

```python
    @property
    def prices(self):
        return [float(x) for x in self._prices.split(';') if x != '']

    @hybrid_property
    def price(self):
        try:
            return float(self._prices.split(';')[-1])
        except:
            return -1.0

    @prices.setter
    def prices(self, price):
        if self._prices is None:
            self._prices = ';%s' % price
        else:
            self._prices += ';%s' % price
```

Image 2.1.1.2 How a property, a hybrid property and a setter is used for the prices

When the scraping phase is complete I execute

```python
insertFromInputFile(main_dir + "/scraper/ab/ab.txt", "AB")
insertFromInputFile(main_dir + "/scraper/sklavenitis/sklavenitis.txt", "Sklavenitis")
insertFromInputFile(main_dir + "/scraper/kritikos/kritikos.txt", "Kritikos")
insertFromInputFile(main_dir + "/scraper/my_market/my-market.txt", "My Market")
insertFromInputFile(main_dir + "/scraper/masoutis/masoutis.txt", "Masoutis")
```

Image 2.1.1.3 Update functions

The function is designated to read data from files, subsequently splitting it into a list of dictionaries and invoking the "insertProducts(products)" function. To accomplish this task, the API first uses the path to determine the current date and the corresponding supermarket. This enables the function to retrieve the appropriate data and ensure that the information displayed to the user is up-to-date and relevant.

```python
def insertFromInputFile(file, name=""):
    from datetime import datetime

    current_date = datetime.now()
    file_date = current_date.strftime("%Y/%m/%d")

    print(file_date)

    if "ab" in file:
        supermarket_name = "Αβ"
        supermarket_name_english = "Ab"
    elif "sklavenitis" in file:
        supermarket_name = "Σκλαβενίτης"
        supermarket_name_english = "Sklavenitis"
    elif "my_market" in file:
        supermarket_name = "My Market"
        supermarket_name_english = "My Market"
    elif "kritikos" in file:
        supermarket_name = "Κρητικός"
        supermarket_name_english = "Kritikos"
    elif "masoutis" in file:
        supermarket_name = "Μασούτης"
        supermarket_name_english = "Masoutis"
    else:
        raise "Error"
```

Image 2.1.1.4 How to identify each supermarket

It then proceeds to read the file and create a list of objects

```
329     with open(file, "r") as f:
330         products = []
331 
332         for line in f.readlines():
333             # print(line)
334             items = line.replace('\n', '').split(', ')
335             j = len(items) - 7
336             p = ""
337 
338             for k in range(1, j):
339                 p += items[k]
340                 if k != j - 1:
341                     p += " "
342 
343             if items[j+6] == "True":
344                 disc = True
345             else:
346                 disc = False
347 
348             obj = {
349                 "supermarket": supermarket_name,
350                 "supermarket_english": supermarket_name_english,
351                 "category": items[0],
352                 "product": p,
353                 "company": items[j+1],
354                 "url": items[j+4],
355                 "img": items[j+5],
356                 "discount": disc,
357                 "date": file_date,
358             }
359 
360             try:
361                 it = items[j+3].split('€/')
362                 obj["value_ratio"] = float(it[0])
363                 obj["metric"] = it[1]
364             except:
365                 continue
366 
367             try:
368                 obj["price"] = float(items[2])
369             except:
370                 continue
371 
372             try:
373                 obj["weight"] = float(items[j+2])
374             except:
375                 if obj["price"] >= 0.0 and obj["value_ratio"] > 0.0:
376                     obj["weight"] = obj["price"] / obj["value_ratio"]
377                 else:
378                     obj["weight"] = 1.0
379 
380             if obj["metric"] == "":
381                 obj["metric"] = "τεμάχιο"
```

Image 2.1.1.5 Spliting the data file into a list of objects

The function performs an iteration through the list of products obtained from the file, it then searches for the existence of each product individually. In case the product already exists, the function updates the prices of the product and continues the iteration.

```
107             prod_exist = session.query(Product).filter(Product.url == product["url"]).first()
108
109             if prod_exist is not None:
110                 prod_exist.product = help.removeAccent(product["product"])
111                 prod_exist.category = help.removeAccent(product["category"])
112                 prod_exist.prices = product["price"]
113                 prod_exist.dates = product["date"]
114                 prod_exist.value_ratio = product["value_ratio"]
115                 prod_exist.weight = product["weight"]
116                 prod_exist.discount = product["discount"]
117                 prod_exist.img = product["img"]
118                 prod_exist.url = product["url"]
119                 prod_exist.exist = True
120                 prod_exist.metric = product["metric"]
121
122                 continue
```

Image 2.1.1.6 If product exists update it's attributes

Else, if a new product needs to be added, the following code is executed:

```
import googletrans
translator = googletrans.Translator()
```

```
125             translated_product = translator.translate(str(product["product"]), src='el', dest='en').text
126
127             if len(product["company"]) == 0:
128                 translated_company = ""
129             else:
130                 translated_company = translator.translate(str(product["company"]), src='el', dest='en').text
131
132             prod = Product(
133                 supermarket=product["supermarket"],
134                 supermarket_english=product["supermarket_english"],
135
136                 product=help.getProperWord(help.removeAccent(product["product"])),
137                 product_english = translated_product,
138
139                 company=help.getProperWord(help.removeAccent(product["company"])),
140                 company_english=translated_company,
141
142                 category=help.getProperWord(help.removeAccent(product["category"])),
143                 category_english = categories[help.getProperWord(help.removeAccent(product["category"]))],
144
145                 metric=product["metric"],
146                 img=product["img"],
147                 url=product["url"],
148                 discount=product["discount"],
149             )
150             prod.exist = True
151
152             if isinstance(product["price"], float):
153                 prod.prices = product["price"]
154             else:
155                 continue
156
157             prod._datesUpdated = ";" + product["date"]
158
159             if isinstance(product["value_ratio"], float):
160                 prod.value_ratio = product["value_ratio"]
161
162             if isinstance(product["weight"], float):
163                 prod.weight = product["weight"]
164
165             session.add(prod)
```

Image 2.1.7 Create new instance of product

2.1.2 Deals Table

In the initial iteration of the application, a single HTTP request was utilized to calculate the drop value of a given product based on its average price, subsequently returning any relevant deals. However, due to the limited 2GB of RAM on the VPS hosting the app at the time, this calculation was found to be relatively slow, taking approximately 4-6 seconds to complete. Additionally, this approach required recalculating the dropValues every time a user requested to view the deals. In order to address these issues, it was decided to implement a new table that maintained the relationships between deal-product IDs in descending order, allowing for the most favorable deals to be presented at the top of the list. By periodically clearing this table and writing new deals, it was possible to efficiently update product prices without the need for recalculations.

```
15  class Deal(Base):
16      __tablename__ = 'deal'
17
18      primary_id = db.Column(db.Integer, primary_key=True, unique=True)
19      id = db.Column(db.Integer, db.ForeignKey('product.id'), nullable=False)  # this is 1-1 with parent class
20
21      # relationships
22      parent = relationship("Product", uselist=True) # this is 1-1 with parent class
23
```

Image 2.1.2.1 Deal class

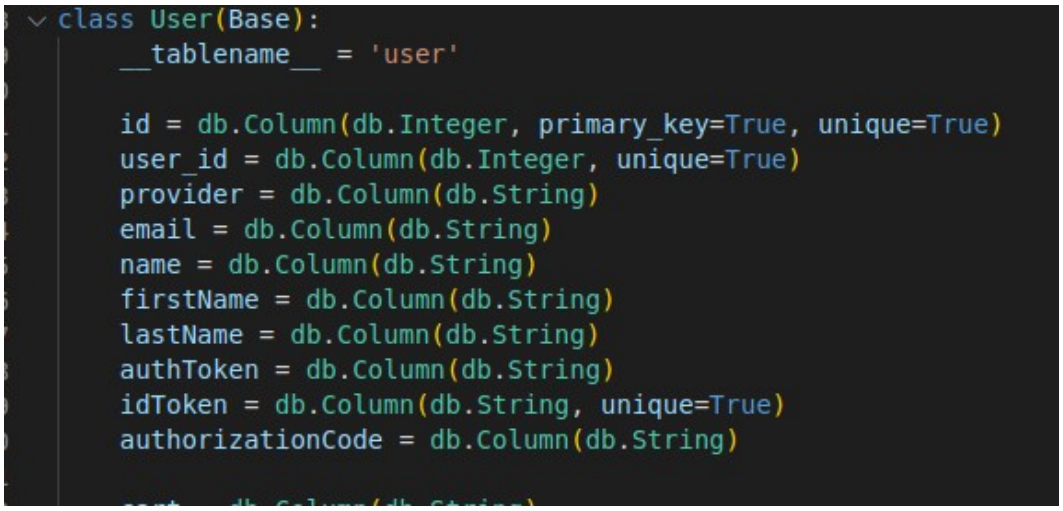
After scraping new data and storing it in the database, I invoke this API that calculates the dropPrice, which is a class property of the Product. only products with discounts are retained in the database. This process allows for efficient management of product data and ensures that only relevant and discounted products are made available to users.

```
495 def findBestDeals():
496     session = Session()
497     try:
498         session.query(Deal).delete()
499         session.commit()
500     except Exception as e:
501         print(e)
502
503     ret = [prod for prod in session.query(Product).filter(Product.exist == True).all() if prod.dropPrice < -0.1]
504
505     ret.sort(key=lambda x: x.dropPrice, reverse=False)
506
507     from datetime import date
508     today = date.today().strftime("%d/%m/%Y")
509
510     for p in ret:
511         deal = Deal()
512         deal.parent.append(p)
513         session.add(deal)
514
515     session.commit()
516
517
518     session.close()
```

Image 2.1.2.2 Find best Deals function

2.1.3 Users Table

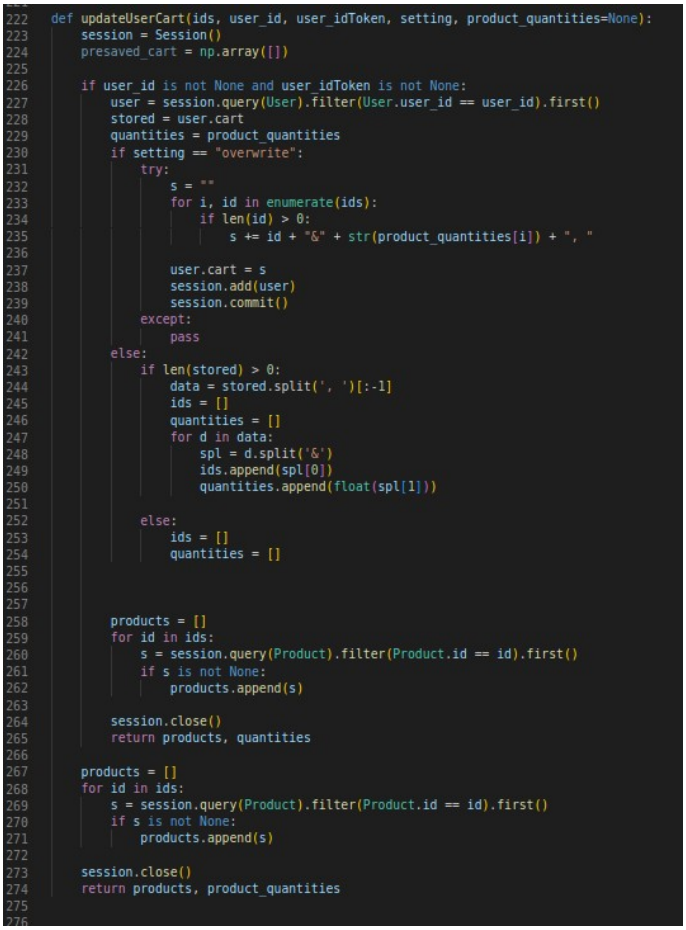
As is the case with any application, the need to store user data arises. To facilitate this, users have the option to login with their Google account and keep track of their shopping cart. Specifically, a cart property was implemented which holds a list of product IDs and their corresponding quantity in a single string. This allows for easy management and retrieval of the user's cart data.



```python
class User(Base):
    __tablename__ = 'user'

    id = db.Column(db.Integer, primary_key=True, unique=True)
    user_id = db.Column(db.Integer, unique=True)
    provider = db.Column(db.String)
    email = db.Column(db.String)
    name = db.Column(db.String)
    firstName = db.Column(db.String)
    lastName = db.Column(db.String)
    authToken = db.Column(db.String)
    idToken = db.Column(db.String, unique=True)
    authorizationCode = db.Column(db.String)
```

Image 2.1.3.1 User Class

The API for updating a user's cart has two modes of operation: "overwrite" and "update." In the "overwrite" mode, the user completely replaces the existing cart with a new one, as specified in the "product_quantities" parameter. Prior to this update, a check is performed to ensure that the products being added to the cart exist. The API then returns the results of this update. In the "update" mode, the user can add or remove products from their cart as specified in the "product_quantities" parameter. A check is also performed to ensure that the products being added to or removed from the cart exist. The API then returns the results of this update.



```python
def updateUserCart(ids, user_id, user_idToken, setting, product_quantities=None):
    session = Session()
    presaved_cart = np.array([])

    if user_id is not None and user_idToken is not None:
        user = session.query(User).filter(User.user_id == user_id).first()
        stored = user.cart
        quantities = product_quantities
        if setting == "overwrite":
            try:
                s = ""
                for i, id in enumerate(ids):
                    if len(id) > 0:
                        s += id + "&" + str(product_quantities[i]) + ", "

                user.cart = s
                session.add(user)
                session.commit()
            except:
                pass
        else:
            if len(stored) > 0:
                data = stored.split(', ')[:-1]
                ids = []
                quantities = []
                for d in data:
                    spl = d.split('&')
                    ids.append(spl[0])
                    quantities.append(float(spl[1]))

            else:
                ids = []
                quantities = []



        products = []
        for id in ids:
            s = session.query(Product).filter(Product.id == id).first()
            if s is not None:
                products.append(s)

        session.close()
        return products, quantities

    products = []
    for id in ids:
        s = session.query(Product).filter(Product.id == id).first()
        if s is not None:
            products.append(s)

    session.close()
    return products, product_quantities
```

Image 2.1.3.2 How to update the users cart

2.1.4 User’s Messages Table

Furthermore, there has been implemented a message table that essentially store the messages from the users.

```python
from .base import Base

class Message(Base):
    __tablename__ = 'message'

    id = db.Column(db.Integer, primary_key=True, unique=True, nullable=False, autoincrement=True)

    viewed = db.Column(db.Boolean)
    name = db.Column(db.String)
    email = db.Column(db.String)
    date = db.Column(db.Date)
    message = db.Column(db.String)
    answer = db.Column(db.String)

    def serialize(self):
        return {
            "id": self.id,
            "viewed": self.viewed,
            "name": self.name,
            "email": self.email,
            "date": self.date,
            "message": self.message,
            "answer": self.answer,
        }
```

Image 2.1.4.1 Message Table

2.1.5 Suggestions Tables

The structure of the suggestions table is more intricate in comparison to the previous ones. In order to establish a many-to-many relationship using SQLAlchemy, it is necessary to create a temporary table to store the unique relationships. Additionally, a parent-child association must also be established, with the children being stored in a list format. This approach is required in order to effectively implement the desired relationship within the suggestions table. Important to note is that we place only deals in this table.

```python
suggestions_list_deals = db.Table('suggestions_list_deal', Base.metadata,
    db.Column('id', db.Integer, primary_key=True),
    db.Column('child1_id', db.Integer, db.ForeignKey('deal.id')),
    db.Column('child2_id', db.Integer, db.ForeignKey('suggestion.primary_id'))
)
```

Image 2.1.5.1 Temporary Table

```python
class Suggestion(Base):
    __tablename__ = 'suggestion'

    primary_id = db.Column(db.Integer, primary_key=True, unique=True)
    id = db.Column(db.Integer, db.ForeignKey('product.id'))  # this is 1-1 with parent class

    parent = relationship("Product", uselist=True) # this is 1-1 with parent class
    associations = relationship('Deal', secondary=suggestions_list_deals) # , backref=db.backref('child', lazy='dynamic'))
```

Image 2.1.5.2 Suggestion Table

As outlined in Chapter 3, the execution of the K-means algorithm generates a cluster label file. This file is structured such that the row i corresponds to the product with an identifier of i-1. Each row in the file contains a numerical value that pertains to the cluster that the respective product belongs to, as determined by the algorithm.

```
def fillSuggestions():
    session = Session()

    try:
        session.query(Suggestion).delete()
        session.commit()
    except Exception as e:
        print(e)
    try:
        session.query(suggestions_list_deals).delete()
        session.commit()
    except Exception as e:
        print(e)

    for p in session.query(Product).all():
        sug = Suggestion()
        sug.parent.append(p)
        session.add(sug)

    session.commit()

    print("Done with deletion")
    import numpy as np

    suggestions = session.query(Suggestion).all()

    K = 1800
    with open("clust_labels_K1800.txt", "r") as f:
        cluster_labels = np.array([int(line.replace('\n', '')) for line in f.readlines() ])

        for i in range(K):
            # print(i)
            new_line = list([int(l+1) for l in np.where(cluster_labels == i)[0]])

            results = session.query(Deal).filter(Deal.id.in_(new_line))

            ids = [r.id for r in results.all()]
            print(i, len(ids))
            for idx, id in enumerate(ids):
                for j in range(idx):
                    suggestions[id - 1].associations.append(results[j])
                for j in range(idx + 1, len(ids), 1):
                    suggestions[id - 1].associations.append(results[j])

    session.commit()
        # for line in f.readlines():
        #     print(line.replace('\n'))
    session.close()
```

Image 2.1.5.3 Fill Suggestions implementation

2.2 Scraped data: Data immunity

Web scraping, as a method of data collection, can sometimes produce unreliable or inaccurate results. This is often due to the lack of proper data formatting on the websites being scraped. To mitigate this issue, I have implemented an algorithm for validating and correcting basic data such as price, value ratio, and weight. Specifically, the algorithm compares the product value ratio multiplied by weight to the price, and corrects any discrepancies that may be the result of errors on the source website. This approach allows me to improve the accuracy and reliability of the data collected through web scraping.

```
def fixPrices():
    session = Session()
    f = open("temp.txt", "w")
    for prod in session.query(Product).all():
        price = prod.price
        weight = prod.weight
        value_ratio = prod.value_ratio

        if abs(float(round(value_ratio * weight, 4)) - price) > 0.5  :
            if value_ratio == 0.0 or value_ratio == -1.0:
                prod.value_ratio = float(round(price / weight, 2))
            elif weight == 0.0 or weight == -1.0:
                prod.weight = float(round(price / value_ratio, 4))
            elif price == -1.0:
                prod.exist = False
            else:
                f.write(str(prod.id) + " " + str(price) + " " + str(weight) + " " + str(value_ratio) + " " + str( value_ratio * weight) + " " + str(prod.product) + "\n")

    session.commit()
    session.close()
```

Image 2.2.1 Fix Prices

2.3 VPS setup

In the context of this thesis, a virtual private server (VPS) is deemed necessary for the uninterrupted operation of the application. However, certain challenges must be addressed in order to ensure the security of the VPS and the smooth distribution of network calls, as well as the sustained running of the flask application. These challenges include, but are not limited to, protecting the VPS from potential attacks, effectively distributing network calls, and maintaining the continuous operation of the flask application.

2.3.1 Nginx

NGINX is a web server software that is often used as a reverse proxy, load balancer, and HTTP cache. It is known for its high performance and low resource consumption, making it a popular choice for high-traffic websites and web applications. It can also be used to serve static content, as a proxy for other web servers, and as a load balancer to distribute incoming requests among multiple servers. Additionally, it can be used to terminate SSL/TLS connections, and to serve as an origin server for the HTTP/2 protocol [12]. The configuration of the NGINX server in this scenario is designed to redirect all incoming HTTP requests to their corresponding HTTPS equivalents. This is accomplished by utilizing the appropriate redirect directives within the NGINX configuration file. Furthermore, the configuration also manages the inclusion of the "www." subdomain within the domain, ensuring that all relevant network requests are properly directed to the intended server. In the case of HTTPS requests, the configuration utilizes the Certbot utility to generate SSL certificates, and redirects these requests to the local running server on port 8080 for further processing. Overall, the NGINX configuration is designed to provide optimal security and performance, while also ensuring that all network requests are properly handled and directed.

```
server {

        server_name bestcart.gr www.bestcart.gr;

        root /home/ntua/projects/Super-Market-App/backend/frontend;

#        index index.html;

        location / {
                proxy_pass http://localhost:8080/;
        }
#        location / {
#                try_files $uri $uri/ =404;
#        }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/bestcart.gr/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/bestcart.gr/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot


}

server {
    if ($host = www.bestcart.gr) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    if ($host = bestcart.gr) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80;

        server_name bestcart.gr www.bestcart.gr;
    return 404; # managed by Certbot
```

Image 2.3.1.1 NGINX Configurations

2.3.2 Docker with Gunicorn

Docker is a platform that uses containerization to package and distribute applications in a format that can run consistently across different environments. It allows for easy creation, testing, and deployment of applications, and enables developers to focus on writing code without worrying about the system it will run on. It also helps IT operations teams to quickly move applications between environments and simplify scaling and managing applications using orchestration tools. Additionally, it provides a large collection of pre-built images that can be used to quickly start an application or service, saving time and effort.

Gunicorn (short for Greenlet-based Worker for UNIX) is a Python Web Server Gateway Interface (WSGI) HTTP server. It is used to serve Python web applications, and is often used as a production-ready web server for applications built using frameworks such as Django, Flask, and Pyramid. Gunicorn is designed to be lightweight, fast, and efficient, and can handle multiple concurrent connections using a pre-fork worker model, allowing it to handle large numbers of incoming requests without a significant increase in resource usage.
It's also known for its ability to handle slow clients without blocking the worker processes, it can also daemonize itself, and it can also be run behind a reverse proxy like nginx or Apache.

Gunicorn is typically used in combination with a reverse proxy and a front-end web server like nginx or Apache, which can handle tasks such as SSL termination, load balancing, and caching.

To run a Dockerfile, first, make sure you have Docker installed on your system. Once you have Docker up and running, navigate to the directory where your Dockerfile is located using the command line. Then, you can use the command "docker build" to build an image from your Dockerfile. The basic syntax for the build command is "docker build -t [image_name] [path_to_dockerfile]". For example, if your Dockerfile is located in the local directory "myapp" and you want to name the image "myapp-image", you would use the command "docker build -t myapp-image myapp/".

Once the image is built, you can use the command "docker run" to start a new container using the image. The basic syntax for the run command is "docker run [image_name] [command_to_run]". For example, to start a container using the "myapp-image" image and run the command "python app.py", you would use the command "docker run myapp-image python app.py".

You can also use the option -p in the run command to map the ports of the container to the host. This option is useful if you want to access the application running on the container from the host. For example, to map the port 8080 of the container to port 80 of the host, you would use the command "docker run -p 80:8080 myapp-image python app.py"

Additionally, you may also use the option -d to run the container in detached mode, which allows the container to run in the background, it also allows you to view logs and stop the container.

It's important to keep in mind that when a container is stopped all the data is lost, if you want to persist the data you should use volumes or bind mounts with the -v option. [13]

Below, I included my Dockerfile and the command to run the build container.

```
# syntax=docker/dockerfile:1

FROM python:3.8

WORKDIR /app
COPY . .

RUN pip install -r requirements.txt

ENTRYPOINT [ "gunicorn" ]
CMD ["--bind", "0.0.0.0:8080", "--workers=2", "--access-logfile", "-", "app:app"]

# Run as
# docker run -p 8080:8080 --name test --rm -v /mnt/AEBA35E3BA35A8AB/Super-Market-App/backend/databases2:/app/databases/ -d supermarket
# docker run -p 8080:8080 -v /mnt/AEBA35E3BA35A8AB/Super-Market-App/backend/:/app --rm -d --name supermarket_image supermarket
```

Image 2.3.2.1 Dockerfile

Chapter 3: Recommender System

3.1 The need for a good recommendation algorithm

The need for a good recommendation algorithm is paramount in today's fast-paced retail environment. With the plethora of options available to consumers, it is often difficult for them to make informed decisions about which products to purchase. A good recommendation algorithm addresses this issue by suggesting products to consumers based on their preferences and past behavior. This not only helps consumers to remember products they may have forgotten, but also helps them discover new products that may be a more suitable choice for them. Additionally, a good recommendation algorithm can also help to map identical products across different supermarkets, providing consumers with a more comprehensive view of the products and prices available to them. This mapping can also be a great benefit for retailers, as it allows them to have a more accurate understanding of the products that are available in the market, which in turn can help them make more informed decisions about the products they carry and their prices. Overall, a good recommendation algorithm can greatly enhance the shopping experience for both consumers and retailers.

3.2 TF-IDF

3.2.1 Basic Theory

Started with the simple binary bag of words of model where each document is represented as a fixed size vector of 0s and 1s where if a word appears in a document it gets a 1 and if it doesn't then it gets a 0. As an example, consider these four document below:

D1: the movie was a very indulging cinematic experience.
D2: standard of this movie is above its contemporaries.
D3: director brought out the best of the pair.
D4: moviegoers won't mind seeing the pair again. [14]

The binary bag of words representation for these four documents using 8 frequently occurring words is shown in the table below.

Docs/ Words	the	movie	of	pair	was	a	wont	mind
D1	1	1	0	0	1	1	0	0
D2	0	1	1	0	0	0	0	0
D3	0	0	1	1	0	0	0	0
D4	0	0	0	1	0	0	1	1

Table Binary Bag of Words Model [14]

In this model, we only represent the existence of each word, but we do not take into consideration the importance of a word. For example, the word "indulging" is an import one.

TF-IDF, short for term frequency-inverse document frequency, is a statistical measure used to understand the importance of a word in a given document or corpus. It is a widely used technique in

natural language processing and information retrieval, and it is particularly useful for text classification and search engines.

The TF-IDF measure is composed of two parts: term frequency (TF) and inverse document frequency (IDF). The term frequency is a measure of the number of times a word appears in a document, and it is used to gauge the relevance of a word to a particular document. There are several ways to define term frequency, such as raw count, term frequency adjusted for the length of the document, logarithmically scaled frequency, and boolean frequency.

On the other hand, the inverse document frequency is a measure of how common or uncommon a word is across a corpus of documents. It is used to gauge the importance of a word in the corpus as a whole. The inverse document frequency is calculated by taking the logarithm of the ratio of the total number of documents in the corpus to the number of documents in which the word appears [14].

$$idf\,(t, D) = log\,(\ \frac{N}{count\,(d \in D{:}t \in d)}\ )$$

Image 3.2.1.1 [14]

Scikit-Learn

- $\mathrm{IDF}(t) = \log\frac{1+n}{1+\mathrm{df}(t)} + 1$

Standard notation

- $\mathrm{IDF}(t) = \log\frac{n}{\mathrm{df}(t)}$

Image 3.2.1.2 [14]

3.2.2 Preprocessing

It is crucial to preprocess the product description prior to utilizing the tf-idf algorithm, as it is highly sensitive to new words. One method of preprocessing is to employ the use of a python library called "greek_stemmer," which stems Greek words. The purpose of stemming is to reduce multiple variations of a word, such as "Μπανάνες" and "Μπανάνα," to a single, common form, in this case "μπαναν." Additionally, upon examination of the dataset, it was noted that there were numerous instances of mistypings, such as the combination of multiple words into a single word, as seen in the example "ΠΟΡΤΟΚΑΛΙ/ΜΗΛΟ/ΒΕΡΙΚΟΚΟ." As a result, further modifications were necessary before proceeding with the use of any algorithms.

After, running this algorithm, the unique words and the tf-idf values were better.

```python
def applyWeightChanges(s):
    '''
        gets a string in greek finds and fixes weird
    '''
    upper = removeAccent(s.upper())
    upper = upper.replace(',', '.')

    if upper.endswith(" GR"):
        upper = upper[:-3] + "GR"

    '''
        Starting with kg, gr etc
    '''
    # Search of "{number}G" if it exist and after the G is a space character or the string ends
    # add R to make it GR
    # # remove redundant end spaces
    upper = re.sub(r'(\d+G)(\s|$)', r'\1R ', upper)
    while upper[-1] == " ":
        upper = upper[:-1]

    # Search of "{number}ΓΡ." , relpace it with GR to make it GR
    # # remove redundant end spaces
    upper = re.sub(r'(\d+)ΓΡ\.', r'\1GR ', upper)
    while upper[-1] == " ":
        upper = upper[:-1]

    upper = upper.replace(' KG', 'KG').replace(' KG.', 'KG').replace('KG.', 'KG').replace(' ΚΙΛΩΝ', 'KG')

    upper = upper.replace(' ML', 'ML').replace(' ML.', 'ML').replace('ML.', 'ML')
    upper = upper.replace(' LT', 'LT').replace(' LT.', 'LT').replace('LT.', 'LT')

    upper = upper.replace(' ΤΕΜΑΧΙΟ', 'ΤΕΜΑΧ').replace('ΤΕΜΑΧΙΟ', 'ΤΕΜΑΧ').replace(' ΤΕΜΑΧΙΑ', 'ΤΕΜΑΧ').replace('ΤΕΜΑΧΙΑ', 'ΤΕΜΑΧ')
    upper = upper.replace('ΚΑΡΑΜΕΛΑ-ΜΠΑΝΑΝΑ', 'ΚΟΤΟΠΟΥΛΟ ΛΑΧΑΝΙΚΑ').replace('ΚΟΥΝΕΛΙ&ΛΑΧΑΝΙΚΑ', 'ΚΟΥΝΕΛΙ ΛΑΧΑΝΙΚΑ')
    upper = upper.replace('ΜΗΛΟ-ΑΧΛΑΔΙ-ΣΤΑΦΥΛΙ-ΜΠΑΝΑΝΑ', 'ΜΗΛΟ ΑΧΛΑΔΙ ΣΤΑΦΥΛΙ ΜΠΑΝΑΝΑ')
    upper = upper.replace('ΜΗΛΟ-ΜΠΑΝΑΝΑ-ΒΑΤΟΜΟΥΡΟ-ΔΗΜΗΤΡΙΑΚΑ', 'ΜΗΛΟ ΜΠΑΝΑΝΑ ΒΑΤΟΜΟΥΡΟ ΔΗΜΗΤΡΙΑΚΑ')
    upper = upper.replace('ΚΑΡΑΜΕΛΑ-ΦΙΣΤΙΚΙ-ΜΠΑΝΑΝΑ', 'ΚΑΡΑΜΕΛΑ ΦΙΣΤΙΚΙ ΜΠΑΝΑΝΑ')
    upper = upper.replace('ΜΠΑΝΑΝΑ-ΒΑΝΙΛΙΑ', 'ΜΠΑΝΑΝΑ ΒΑΝΙΛΙΑ')
    upper = upper.replace('ΜΠΑΝΑΝΑ-ΣΟΚΟΛΑΤΑ', 'ΜΠΑΝΑΝΑ ΣΟΚΟΛΑΤΑ')
    upper = upper.replace('ΣΟΚΟΛΑΤΑ&ΜΠΑΝΑΝΑ', 'ΣΟΚΟΛΑΤΑ ΜΠΑΝΑΝΑ')
    upper = upper.replace('ΜΠΑΝΑΝΑ&ΣΟΚΟΛΑΤΑ', 'ΜΠΑΝΑΝΑ ΣΟΚΟΛΑΤΑ')
    upper = upper.replace('ΜΠΑΝΑΝΑ-ΜΗΛΟ-ΒΡΩΜΗ', 'ΜΠΑΝΑΝΑ ΜΗΛΟ ΒΡΩΜΗ')
    upper = upper.replace('ΜΗΛΟ-ΒΕΡΙΚΟΚΟ-ΜΠΑΝΑΝΑ', 'ΜΗΛΟ ΒΕΡΙΚΟΚΟ ΜΠΑΝΑΝΑ')
    upper = upper.replace('ΜΠΑΝΑΝΑ+ΣΟΚΟΜΠΙΛ', 'ΜΠΑΝΑΝΑ ΣΟΚΟΜΠΙΛ')
    upper = upper.replace('ΦΡΑΟΥΛΑ-ΜΠΑΝΑΝΑ', 'ΦΡΑΟΥΛΑ ΜΠΑΝΑΝΑ')
    upper = upper.replace('ΜΗΛΟ/ΣΤΑΦΥΛΙ/ΦΡΑΟΥΛΑ', 'ΜΗΛΟ ΣΤΑΦΥΛΙ ΦΡΑΟΥΛΑ')
    upper = upper.replace('ΜΗΛΟ/ΑΧΛΑΔΙ/ΣΤΑΦΥΛΙ/ΜΠΑΝΑΝΑ', 'ΜΗΛΟ ΑΧΛΑΔΙ ΣΤΑΦΥΛΙ ΜΠΑΝΑΝΑ')
    upper = upper.replace('ΜΗΛΟ/ΠΟΡΤΟΚΑΛΙ/ΡΟΔΑΚΙΝΟ/ΒΕΡΙΚΟΚΟ', 'ΜΗΛΟ ΠΟΡΤΟΚΑΛΙ ΡΟΔΑΚΙΝΟ ΒΕΡΙΚΟΚΟ')
    upper = upper.replace('ΠΟΡΤΟΚΑΛΙ/ΜΗΛΟ/ΒΕΡΙΚΟΚΟ', 'ΠΟΡΤΟΚΑΛΙ ΜΗΛΟ ΒΕΡΙΚΟΚΟ')

    return upper

def stemLine(stemmer, line):
    return " ".join([stemmer.stem(removeAccent(l.upper())) for l in line.split(' ')])
```

Image 3.2.3.1 Line Preprocessing

```
12889   μιγ
12890   μιγμ
12891   μικ
12892   μικρ
12893   μικρα
12894   μικροβιοκτον
12895   μικρογκοφρ
12896   μικροι
12897   μικροιν
12898   μικροκαρπ
12899   μικροκυμ
12900   μικροσπερμ
12901   μικροσωμ
12902   μικτ
12903   μικτες
12904   μικτη
12905   μικτο
12906   μικτσικελ
12907   μιλαν
12908   μιλανεζ
12909   μιλφει
12910   μιλφειγ
12911   μιμ
```

```
12912   μιμοζ
12913   μιν
12914   μινερβ
12915   μινερβιν
12916   μινεστρον
12917   μινθ
12918   μινι
12919   μινιι
12920   μινιατουρ
12921   μιξ
12922   μιξερ
12923   μιξερακ
12924   μιρ
12925   μιραντ
12926   μις
12927   μισκοτ
12928   μισκοτιν
12929   μισο
12930   μιστ
12931   μιστικανζ
12932   μιτατ
12933   μιτατοτυρ
12934   μιτσικελ
12935   μιχ
12936   μιχαηλιδ
12937   μιχαλ
12938   μιχαλακ
```

Image 3.2.3.2 Example of the unique words

3.3 Suggestion Algorithms

The question of recommending new products to users from a vast array of options is a complex one. However, it is indeed possible to develop recommendation systems using the available data.

In this thesis, there are proposed two recommendation systems with different methods: K means, and Cosine similarity.

3.3.1 K-Means

One potential solution is to utilize the K-Means algorithm, a unsupervised method that only requires the specification of the number of clusters, denoted as K, to be created. This approach is a viable option for making personalized recommendations to users.

The k-means method is a popular way to group data points into clusters by trying to minimize the distance between points in the same cluster. It is simple and fast, but it does not provide any guarantees for accuracy. By adding a random element to the initial placement of clusters, the method can be made more efficient and accurate. Furthermore, besides the selection of K, we cannot influence the number in each cluster. That's why in some cases, there are clusters with many products, while other have 1 or 2 products. Initial tests have shown significant improvements in both speed and accuracy [17].

For the `k-means` problem, we are given an integer k and a set of n data points $\mathcal{X} \subset \mathbb{R}^d$. We wish to choose k centers $\mathcal{C}$ so as to minimize the potential function,

$$\phi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} \|x - c\|^2.$$

Image 3.3.1.1 Definition of k-means [17]

In this thesis, an evaluation of the k-means algorithm is conducted through the use of two methods: the Silhouette Coefficient and optical representations.
The Silhouette Coefficient, also known as the silhouette score, is a metric used to measure the effectiveness of a clustering technique, with a value range of -1 to 1. A value of 1 indicates that the clusters are well separated, a value of 0 suggests that the clusters are indifferent, and a value of -1 implies that the clusters have been incorrectly assigned [19].

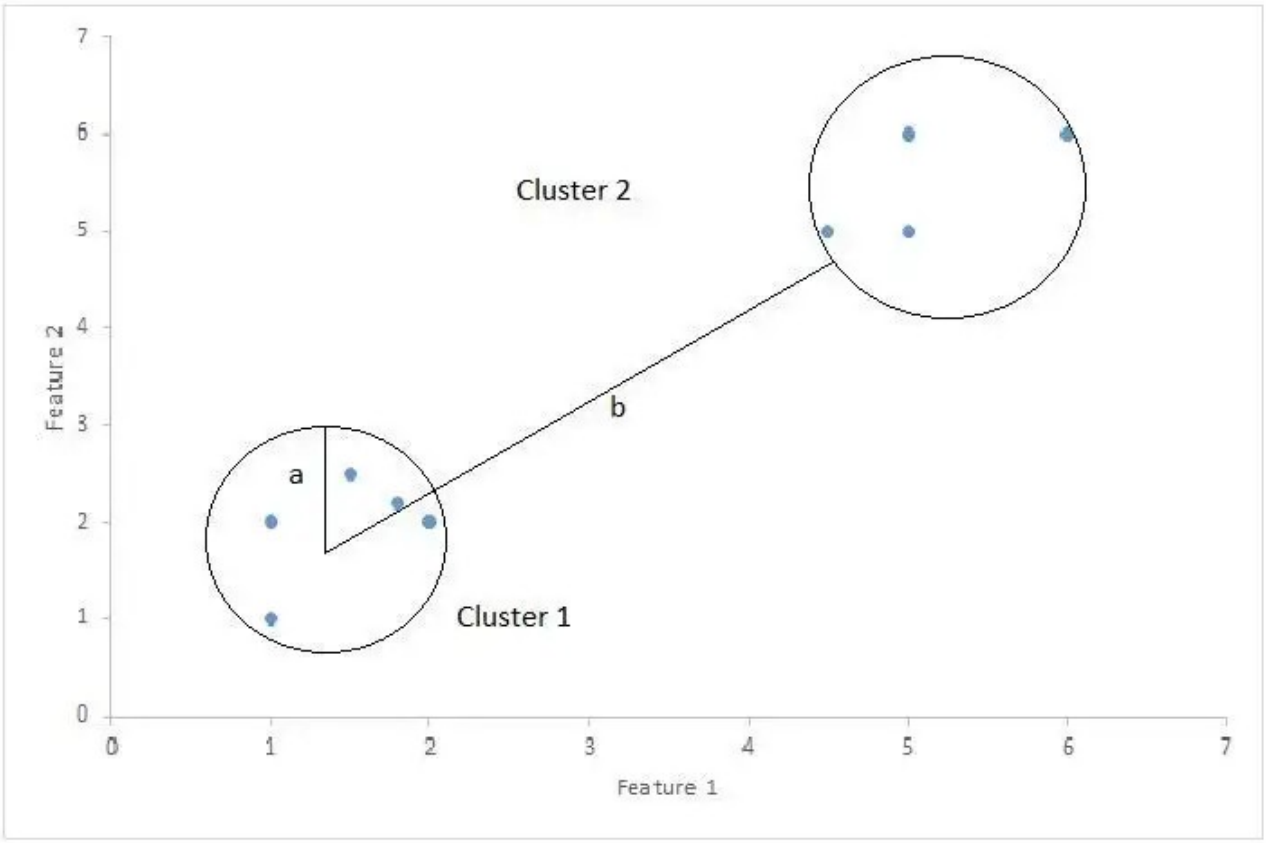


Image 3.3.1.2 Silhouette Score Representation

Silhouette Score = $\frac{b-a}{max(a,b)}$, where

a: average intra-cluster distance i.e the average distance between each point within a cluster, and
b: average inter-cluster distance i.e the average distance between all clusters.

Table 1. Silhouette Score for various K values

K Value	Silhouette Score
500	0.064123910053892
1000	0.083192436793098
3000	0.114632552705606
5000	0.143830595692552
10000	0.201307073686309
15000	0.237640750096125
20000	0.257919619180028

Upon running the k-means algorithm on the dataset for various values of K, it was found that as the value of K increases, the silhouette score also increases. However, as K increases, the number of products within each category decreases. In instances where similar products are sought, this often results in finding zero products. Therefore, a K value between 5000 and 10000 is deemed optimal for this use case, and a value of K=5000 is ultimately chosen [18].

[7]:

```
features = vectorizer.get_feature_names_out()

from sklearn.metrics import silhouette_samples, silhouette_score

import cuml

from sklearn.cluster import KMeans

range_n_clusters = [500] # 18000, 21000, 25000, 27500] # , 12000, 15000, 20000, 25000, 30000]
import numpy as np

X = X.todense()# X = np.array(X)
silhouette_scores = []
for n_clusters in range_n_clusters:
    clusterer = cuml.KMeans(n_clusters=n_clusters)
    cluster_labels = clusterer.fit_predict(X)

    with open("/kaggle/working/cluster_labels_K" + str(n_clusters) + ".txt", "w") as f:
        for c in cluster_labels:
            f.write(str(c) +"\n")

    s = silhouette_score(X, cluster_labels)
    silhouette_scores.append(s)
    print(n_clusters, s)


print(silhouette_scores)
```

Image 3.3.1.3 Implementation of the k-means.

[18]:

```
import numpy as np

def getLabels(path="clust_labels_K13000_6k_columns.txt"):
    with open(path, "r") as f:
        cluster_labels = [int(line.replace('\n', '')) for line in f.readlines() ]

    cluster_labels = np.array(cluster_labels)
    return cluster_labels


file_data = np.array(product_data)

def writeNewCategories(file_data, cluster_labels, K, columns="6k"):
    with open("new_categories_K" + str(K) + "_" + columns + ".txt", "w") as f:
        for i in range(K):
            new_line = [(l, file_data[l]) for l in np.where(cluster_labels == i)[0]]
            for l, prod in new_line:
                f.write(str(l) + ": " + prod + " |&|\t")
            f.write("\n")

# print(cluster_labels)
```

+ Code + Markdown

Image 3.3.1.4 Write new categories in a file

After, the kmeans execution is complete, I can then run the two functions to get a visual representation of how my categories are formed.

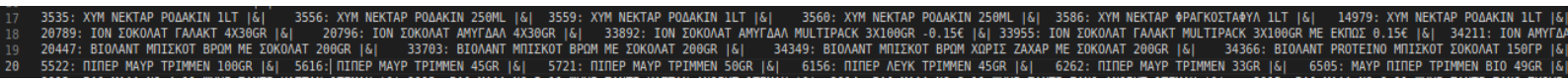


Image 3.3.1.5 Example of categories formed with K = 3000

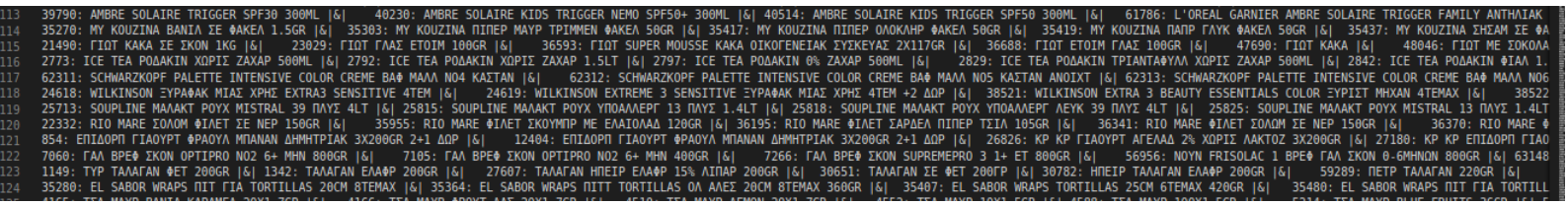


Image 3.3.1.6 Example of categories formed with K = 5000

3.3.2 Cosine Similarity

The solution for recommending products through assigning similarity using cosine distance involved several steps to finalize. Initially, Jaccard similarity was considered as the primary option for determining similarity between products. However, it was determined that Jaccard similarity did not accurately reflect the similarity between products, thus leading to the implementation of cosine distance as an alternative method. Cosine distance is a measure of similarity between two non-zero vectors, and is calculated as the dot product of the vectors divided by the product of their magnitudes. One advantage of using cosine distance over other methods such as k-means is the ability to regulate the maximum number of recommended products. By choosing a low threshold, the total number of results can be narrowed down, and the threshold can then be adjusted accordingly based on the desired number of recommended products [20]. The cosine distance is calculated as: $S_c(A,B) = \cos(\theta) = \frac{\boldsymbol{A}\cdot\boldsymbol{B}}{\|\boldsymbol{A}\|\cdot\|\boldsymbol{B}\|}$

Table 2. Example of similar products through cosine similarity

Product' Id	Product Name	Similar Product	Similar's id	Cosine Similarity
6479	ΜΟΥΣΤΑΡΔ DIJONNAISE 200GR	ΜΟΥΣΤΑΡΔ DIJONNAISE 240GR	5672	0.83
		ΜΟΥΣΤΑΡΔ 200GR	6019	0.63
		MAILLE ΜΟΥΣΤΑΡΔ DIJONNAISE 200GR	36829	0.8
6480	ΕΛΑΙΟΛΑΔ ΕΞΑΙΡΕΤ ΠΑΡΘΕΝ ΠΑΡΑΔΟΣΙΑΚ 1LT 9.22	ΔΩΡ ΕΞΑΙΡΕΤ ΠΑΡΘΕΝ ΕΛΑΙΟΛΑΔ 1LT	37053	0.7
		ΕΛΑΙΟΛΑΔ ΕΞΑΙΡΕΤ ΠΑΡΘΕΝ 1LT	6064	0.74
16479	ΚΑΡΑΜΟΛΕΓΚ ΖΩΜ ΣΕ ΦΕΤ 500GR	ΖΚΑΡΑΜΟΛΕΓΚ ΨΩΜ Ζ ΣΕ ΦΕΤ 500GR	56748	1
		ΚΑΡΑΜΟΛΕΓΚ ΟΚΤΑΣΠΟΡ ΨΩΜ ΣΕ ΦΕΤ 500GR	16577	0.78
16480	ΚΑΤΣΕΛ ΨΩΜ	ΨΩΜ ΤΟΣΤ	1952	0.88

ΤΟΣΤ ΣΤΑΡΕΝΣΤΑΡΕΝ 720GR
720GR

It is important to note that the reason the descriptions in question do not contain complete words is due to the fact that these are stemmed words.

Specifically, in order to obtain these results, I implemented the following code.

```python
from numba import jit, cuda

# @jit(target_backend='cuda')
def similarity_check_for_list(tfIdfX, product_data, id):
    # print("Starting", product_data[id])

    len_words = tfIdfX.getrow(id).data.shape[0]
    minimum_similarity = (len_words - 1)/(2 * len_words + 1) + 0.1

    # print(minimum_similarity)

    # minimum_similarity = 0.6

    ids = []

    a = list(zip(list(X.getrow(id).data), list(X.getrow(id).indices)))

    for i in range(product_data):
        if i == id:
            continue

        # and_ = len(set(a_indices) & set(b_indices))
        # union_ = len(set(a_indices) | set(b_indices))

        b = list(zip(list(X.getrow(i).data), list(X.getrow(i).indices)))

        and_ = list(set(X.getrow(id).indices) & set(X.getrow(i).indices))
        temp1 = {}

        product = 0.0
        a_l2, b_l2 = 0.0, 0.0

        for q in a:
            a_l2 += q[0] ** 2

            if q[1] in and_:
                temp1[q[1]] = q[0]
        for q in b:
            b_l2 += q[0] ** 2

            if q[1] in and_:
                product += temp1[q[1]] * q[0]

        div = (a_l2 * b_l2) ** 0.5

        cosine_similarity = product / div

        if cosine_similarity > minimum_similarity:
            # print(i, cosine_similarity, product_data[id], "&", product_data[i])
            ids.append((i, cosine_similarity))

    return ids
```

Image 3.3.2.1 Cosine Similarity check

In terms of time, it takes approximately 0.25 seconds per product to find all similarities on a laptop with 12GB RAM and an Intel i7 processor. Additionally, when tested on a second laptop with 8GB RAM and an Intel i5 processor, the process required 0.35 seconds to complete. It is worth noting that I also

attempted to utilize numba, a tool that aims to improve performance by utilizing the GPU, in order to optimize the process. However, this did not have the desired effect. On the 12GB RAM laptop, the process still required 8-9 seconds per product, and on the 8GB laptop, it required 14-15 seconds to complete. This lack of improvement is likely due to the fact that numba is not well-suited for handling sparse matrixes (X).

3.4 Store the results in the database

Before concluding the chapter on recommenders, it is important to discuss the integration of the recommendations into the database. To accomplish this, I have implemented a function within the backend folder that is specifically designed for this purpose.

```
def fillSuggestions():
    session = Session()

    try:
        session.query(Suggestion).delete()
        session.commit()
    except Exception as e:
        print(e)
    try:
        session.query(suggestions_list_deals).delete()
        session.commit()
    except Exception as e:
        print(e)

    for p in session.query(Product).all():
        sug = Suggestion()
        sug.parent.append(p)
        session.add(sug)

    session.commit()

    print("Done with deletion")
    import numpy as np

    suggestions = session.query(Suggestion).all()

    K = 5000
    with open("clust_labels_K5000.txt", "r") as f:
        cluster_labels = np.array([int(line.replace('\n', '')) for line in f.readlines() ])

        for i in range(K):
            # print(i)
            new_line = list([int(l+1) for l in np.where(cluster_labels == i)[0]])

            results = session.query(Deal).filter(Deal.id.in_(new_line))

            ids = [r.id for r in results.all()]
            print(i, len(ids))
            for idx, id in enumerate(ids):
                for j in range(idx):
                    suggestions[id - 1].associations.append(results[
                for j in range(idx + 1, len(ids), 1):
                    suggestions[id - 1].associations.append(results[j])

    session.commit()
        # for line in f.readlines():
        #     print(line.replace('\n'))
    session.close()
```

(variable) j: int

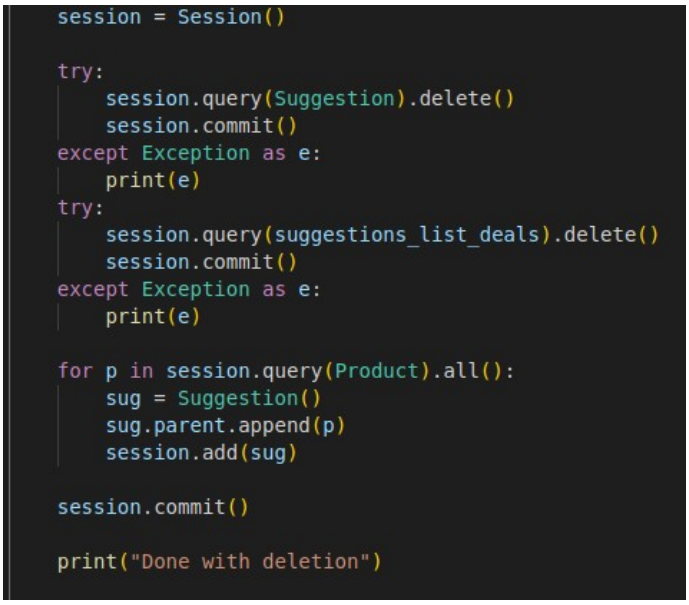
Image 3.4.1 Fill Suggestions implementation with K-Means results

```python
with open("similarity_output.txt", "r") as file:
    data = file.read()

# Convert the string data into a Python object
data = data.split('\n')

corellations = []

for index, line in enumerate(data):
    l = line.replace('(', '').replace(')', '').replace('[', '').replace(']', '')
    l = l.split(', ')
    i = 0
    corellations_temp = []
    while 2 * i + 1 < len(l):
        corellations_temp.append((int(l[2 * i]), float(l[2 * i + 1])))
        i += 1

    corellations_temp.sort(key=lambda a: a[1])

    corellations.append(corellations_temp)
```

```python
suggestions = session.query(Suggestion).all()

for i, cor in enumerate(corellations):
    ids = [c[0] for c in cor]
    results = session.query(Deal).filter(Deal.id.in_(ids)).all(

    if i % 100 == 0:
        print(len(ids), len(results))
        print(i, len(results))

    for res in results:
        suggestions[i].associations.append(res)

session.commit()
session.close()
```

Image 3.4.2 Cosine similarity update

The process of adding the suggestions into the database involves several steps. Firstly, it is necessary to delete all previous suggestions, as the deals have changed. Then, for each cluster, all products assigned to that cluster are identified and added to a list. For each product in that list, a separate list is created, with the exception of the product itself. Finally, the changes are committed to the database.



```
session = Session()

try:
    session.query(Suggestion).delete()
    session.commit()
except Exception as e:
    print(e)
try:
    session.query(suggestions_list_deals).delete()
    session.commit()
except Exception as e:
    print(e)

for p in session.query(Product).all():
    sug = Suggestion()
    sug.parent.append(p)
    session.add(sug)

session.commit()

print("Done with deletion")
```

Image 3.4.3 Delete previous data

Chapter 4: Automate the whole process

4.1 Why automate?

During the development of this application, it became apparent that performing each process manually was a significant challenge. As a result, creating automated scripts to manage all processes became an imperative.

4.2 Automate the web scraping

4.2.1 Web Scraping Process Automation

Launching the web crawlers is simple and include a single command in the terminal. Specifically from the root folder of the project, we can execute “python3 scraper/”. Internally, in the scraper directory, there is a __main__.py file that reads some system inputs and creates the basic threads.

```
import sys
threads = []

if len(sys.argv) <= 1:
    threads.append(threading.Thread(target=AB, args=(1,)))
    threads.append(threading.Thread(target=AB, args=(2,)))

    threads.append(threading.Thread(target=sklavenitis, args=(1,)))
    threads.append(threading.Thread(target=sklavenitis, args=(2,)))

    threads.append(threading.Thread(target=kritikos))
    threads.append(threading.Thread(target=MyMarket))
    threads.append(threading.Thread(target=masoutis))
else:
    for i in range(1,len(sys.argv)):
        if sys.argv[i] == 'AB':
            threads.append(threading.Thread(target=AB, args=(1,)))
            threads.append(threading.Thread(target=AB, args=(2,)))
        elif sys.argv[i] == 'Sklavenitis':
            threads.append(threading.Thread(target=sklavenitis, args=(1,)))
            threads.append(threading.Thread(target=sklavenitis, args=(2,)))
        elif sys.argv[i] == 'Kritikos':
            threads.append(threading.Thread(target=kritikos))
        elif sys.argv[i] == 'MyMarket':
            threads.append(threading.Thread(target=MyMarket))
        elif sys.argv[i] == 'Masoutis':
            threads.append(threading.Thread(target=masoutis))

for thread in threads:
    thread.start()

for thread in threads:
    thread.join()

print("Done")
```

Image 4.2.1.1 Automate web scraping

4.2.2 Back-end Automation

Another sequence that requires automation is the updating of the back-end and database. Once the web crawler completes its task, the backend/backend.py file can be executed. This file includes several steps. The first step is to clear the previous product information. This is accomplished by setting the 'exist' variable, which indicates whether a product currently exists in the store, to false.

```
861     session = Session()
862
863     for prod in session.query(Product).all():
864         prod.discount = False
865         prod.exist = False
866         session.add(prod)
867
868     session.commit()
869     session.close()
```

Image 4.2.2.1 Clearing Products

Next, the files acquired by the web crawler are read and stored in the database.

```
871     import os
872
873     os.chdir("..")
874     main_dir = os.getcwd() # /home/.../Super-Market-app
875
876     insertFromInputFile(main_dir + "/scraper/ab/ab.txt", "AB")
877     insertFromInputFile(main_dir + "/scraper/sklavenitis/sklavenitis.txt", "Sklavenitis")
878     insertFromInputFile(main_dir + "/scraper/kritikos/kritikos.txt", "Kritikos")
879     insertFromInputFile(main_dir + "/scraper/my_market/my-market.txt", "My Market")
880     insertFromInputFile(main_dir + "/scraper/masoutis/masoutis.txt", "Masoutis")
881
882     os.chdir("backend")
```

Image 4.2.2.2 Data update

Subsequently, any necessary fixes are applied to the data, such as updating English categories and correcting prices. The script then identifies the best deals and populates the suggestions.

```
884     updateEnglishCategories()
885
886     fixPrices()
887
888     findBestDeals()
889
890     fillSuggestions()
```

Image 4.2.2.3 Fixes

4.2.3 User data perseverance

It is important to note that before updating the products, user data, specifically the cart, must be preserved. To achieve this, the script logs in to the VPS using SSH credentials, utilizing the pexpect

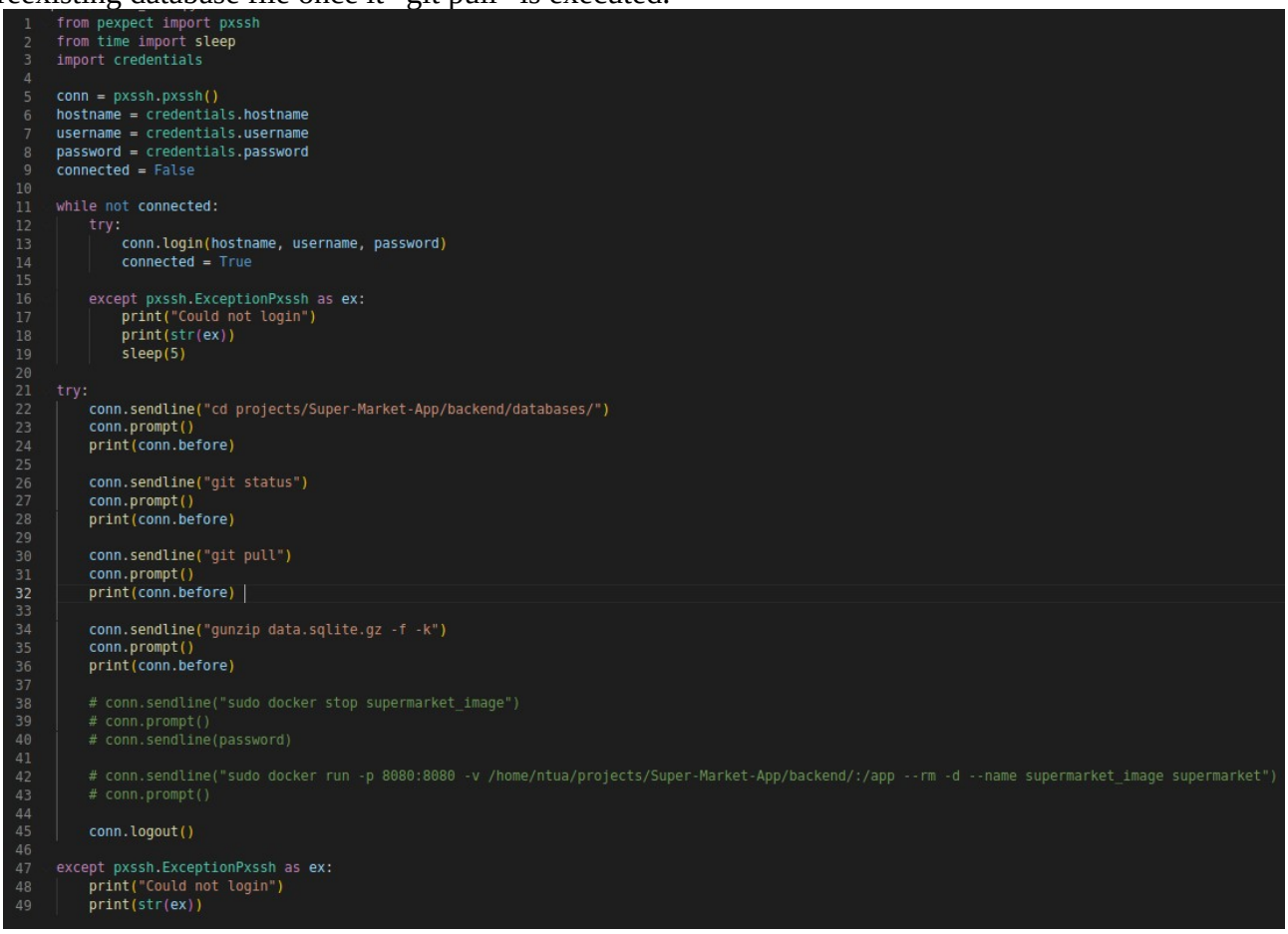
Python library to provide a direct SSH API interface. Additionally, because I am using git as the update system, I have included a compression process to reduce the file size of the database, as the alternative would be to use git-lfs, a paid system.

```python
from pexpect import pxssh
import credentials

try:
    conn = pxssh.pxssh()
    hostname = credentials.hostname
    username = credentials.username
    password = credentials.password

    conn.login(hostname, username, password)

    conn.sendline("cd projects/Super-Market-App/backend/databases")
    conn.prompt()
    print(conn.before)

    conn.sendline("git pull")
    conn.prompt()
    print(conn.before)

    conn.sendline("gzip data.sqlite -k -f")
    conn.prompt()
    print(conn.before)

    conn.sendline("git status")
    conn.prompt()
    print(conn.before)

    conn.sendline("git add .")
    conn.prompt()
    print(conn.before)

    conn.sendline("git commit -m 'update users'")
    conn.prompt()
    print(conn.before)

    conn.sendline("git push")
    conn.prompt()
    print(conn.before)

    conn.logout()

except pxssh.ExceptionPxssh as ex:
    print("Could not login")
    print(str(ex))
```

Image 4.2.3.1 User update script

4.2.4 Update the server

Another aspect that requires automation is the updating of the server. Specifically, I have developed a script that compresses the database into a ".gz" file, commits it to git, logs in to the server and pulls the data. The script utilizes the volume option of the Docker container, which automatically overrides the preexisting database file once it "git pull" is executed.



```python
from pexpect import pxssh
from time import sleep
import credentials

conn = pxssh.pxssh()
hostname = credentials.hostname
username = credentials.username
password = credentials.password
connected = False

while not connected:
    try:
        conn.login(hostname, username, password)
        connected = True

    except pxssh.ExceptionPxssh as ex:
        print("Could not login")
        print(str(ex))
        sleep(5)

try:
    conn.sendline("cd projects/Super-Market-App/backend/databases/")
    conn.prompt()
    print(conn.before)

    conn.sendline("git status")
    conn.prompt()
    print(conn.before)

    conn.sendline("git pull")
    conn.prompt()
    print(conn.before)

    conn.sendline("gunzip data.sqlite.gz -f -k")
    conn.prompt()
    print(conn.before)

    # conn.sendline("sudo docker stop supermarket_image")
    # conn.prompt()
    # conn.sendline(password)

    # conn.sendline("sudo docker run -p 8080:8080 -v /home/ntua/projects/Super-Market-App/backend/:/app --rm -d --name supermarket_image supermarket")
    # conn.prompt()

    conn.logout()

except pxssh.ExceptionPxssh as ex:
    print("Could not login")
    print(str(ex))
```

Image 4.2.4.1 Server Update

4.2.5 Combine Everything together

It is also crucial to have a way to automatically store the data after each scraping and to ensure that new data files are not lost. To achieve this, there is a data_new folder in each supermarket directory. The script moves the {supermarket}/{supermaket}.txt file into this folder with the respective date, and also executes the compression commands. By importing the ssh_update_users script, it automatically updates the users. The script then pulls the changes, decompresses them, and executes the backend script. It then forms the new files in the data folders.

```python
import subprocess
import time
import os

import ssh_update_users

result = subprocess.run(['git', 'pull'], stdout=subprocess.PIPE)
print(result.stdout.decode('utf-8'))

main_dir = os.getcwd() # /home/.../Super-Market-app


os.chdir(main_dir + "/backend/databases/")

result = subprocess.run(['gunzip', 'data.sqlite.gz', '-f', '-k'], stdout=subprocess.PIPE)
print(result.stdout.decode('utf-8'))

os.chdir(main_dir + "/backend/")

# run backend.py
result = subprocess.run(['python3', 'backend.py'], stdout=subprocess.PIPE)
print(result.stdout.decode('utf-8'))

print("Waiting 10 seconds before continuing...")
time.sleep(10)

# Get list of files on scraper/ab/data and scraper/sklavenitis

os.chdir(main_dir) # back in the main dir

from datetime import datetime
current_date = datetime.now()
formatted_date = current_date.strftime("%Y_%m_%d")

def copy_new_file(main_dir, old_file, path, date):
    new_file = main_dir + path + date + ".txt"
    result = subprocess.run(['touch', new_file], stdout=subprocess.PIPE)
    print(result.stdout.decode('utf-8'))

    result = subprocess.run(['cp', main_dir + old_file, new_file], stdout=subprocess.PIPE)
    print(result.stdout.decode('utf-8'))

copy_new_file(main_dir, "/scraper/ab/ab.txt", "/scraper/ab/data_new/", formatted_date)
copy_new_file(main_dir, "/scraper/sklavenitis/sklavenitis.txt", "/scraper/sklavenitis/data_new/", formatted_date)
copy_new_file(main_dir, "/scraper/kritikos/kritikos.txt", "/scraper/kritikos/data_new/", formatted_date)
copy_new_file(main_dir, "/scraper/my_market/my-market.txt", "/scraper/my_market/data_new/", formatted_date)
copy_new_file(main_dir, "/scraper/masoutis/masoutis.txt", "/scraper/masoutis/data_new/", formatted_date)


print("ab.txt and sklavenitis.txt, kritikos.txt, my_market.txt, masoutis.txt were copied to the new files")
```

Image 4.3.5.1 Initial __main__.py script in the server-update folder

The script also clears the initial files for each supermarket, compresses the database, and commits the new data to git. Lastly, it runs the ssh_server script that updates the server with the new data.

```
52
53  # Clear ab.txt and sklavenitis.txt
54
55  os.chdir(main_dir) # back in the main dir
56
57  result = subprocess.run(['rm', main_dir + "/scraper/ab/ab.txt"], stdout=subprocess.PIPE)
58  print(result.stdout.decode('utf-8'))
59  result = subprocess.run(['touch', main_dir + "/scraper/ab/ab.txt"], stdout=subprocess.PIPE)
60  print(result.stdout.decode('utf-8'))
61
62  result = subprocess.run(['rm', main_dir + "/scraper/sklavenitis/sklavenitis.txt"], stdout=subprocess.PIPE)
63  print(result.stdout.decode('utf-8'))
64  result = subprocess.run(['touch', main_dir + "/scraper/sklavenitis/sklavenitis.txt"], stdout=subprocess.PIPE)
65  print(result.stdout.decode('utf-8'))
66
67  result = subprocess.run(['rm', main_dir + "/scraper/kritikos/kritikos.txt"], stdout=subprocess.PIPE)
68  print(result.stdout.decode('utf-8'))
69  result = subprocess.run(['touch', main_dir + "/scraper/kritikos/kritikos.txt"], stdout=subprocess.PIPE)
70  print(result.stdout.decode('utf-8'))
71
72  result = subprocess.run(['rm', main_dir + "/scraper/my_market/my-market.txt"], stdout=subprocess.PIPE)
73  print(result.stdout.decode('utf-8'))
74  result = subprocess.run(['touch', main_dir + "/scraper/my_market/my-market.txt"], stdout=subprocess.PIPE)
75  print(result.stdout.decode('utf-8'))
76
77  result = subprocess.run(['rm', main_dir + "/scraper/masoutis/masoutis.txt"], stdout=subprocess.PIPE)
78  print(result.stdout.decode('utf-8'))
79  result = subprocess.run(['touch', main_dir + "/scraper/masoutis/masoutis.txt"], stdout=subprocess.PIPE)
80  print(result.stdout.decode('utf-8'))
81
82  print("Files were cleared")
83
84  # submit changes to server
85  os.chdir(main_dir + "/backend/databases/")
86  result = subprocess.run(['gzip', 'data.sqlite', '-k', '-f'], stdout=subprocess.PIPE)
87  print(result.stdout.decode('utf-8'))
88
89  os.chdir(main_dir) # back in the main dir
90
91  result = subprocess.run(['git', 'add', '.'], stdout=subprocess.PIPE)
92  print(result.stdout.decode('utf-8'))
93
94  result = subprocess.run(['git', 'commit', '-m' '"update new values"'], stdout=subprocess.PIPE)
95  print(result.stdout.decode('utf-8'))
96
97
98  result = subprocess.run(['git', 'push'], stdout=subprocess.PIPE)
99  print(result.stdout.decode('utf-8'))
100
101
102 # possibly add docker update
103 # docker build --tag supermarket .
104 # docker stop supermarket_image
105 # docker run -p 8080:8080 -v /home/thodoris/projects/Super-Market-App/backend/:/app --rm -d --name supermarket_image supermarket
106
107 import ssh_server
```

Image 4.3.5.2 The rest

To run all the sequence, it requires the execution of "python3 scraper/ ; python3 server-update/", a two-command terminal sequence.

Chapter 5: Frontend, a way to visualize my data

A recent study shows that 60% of all Internet traffic is made up of web traffic, indicating that websites have become a primary means of organizing and transmitting data. Web communication is a crucial intermediary for transmitting Internet data. (Cantelon, 2015).

In my thesis, I argue that a frontend user interface (UI) is crucial for the success of any web-based system. The frontend UI serves as the bridge between the user and the backend system, allowing the user to interact with and navigate through the system's functionality and data. A well-designed UI ensures that the user experience is intuitive and seamless, making it easy for users to access and utilize the system's features. Additionally, a frontend UI can also play a key role in the overall aesthetic and branding of a web-based system, making it more visually appealing and professional. In short, a frontend UI is an essential component of any web-based system, as it allows for an efficient and user-friendly interaction with the system's functionality and data [16].

5.1 Frameworks, libraries and technologies

Following the year 2000, as web browser technology advanced rapidly, JavaScript emerged as a leading language in web development. Additionally, web front-end technologies also saw significant advancements [16].

Following the proliferation of JavaScript, a plethora of JavaScript frameworks and libraries have been developed to aid in the structuring of basic code. These include popular options such as Angular, React js, Vue js, Vanilla js, and Node js. After careful consideration, I have elected to utilize Angular for the development of my project. While it may require a steep learning curve initially, Angular offers a plethora of built-in tools that enable the developer to implement a wide range of features. Furthermore, Angular's component-based structure allows for the creation, combination, and manipulation of individual components to construct a cohesive whole [16].

5.2 Available features

To commence, allow me to provide a general overview of my web application.

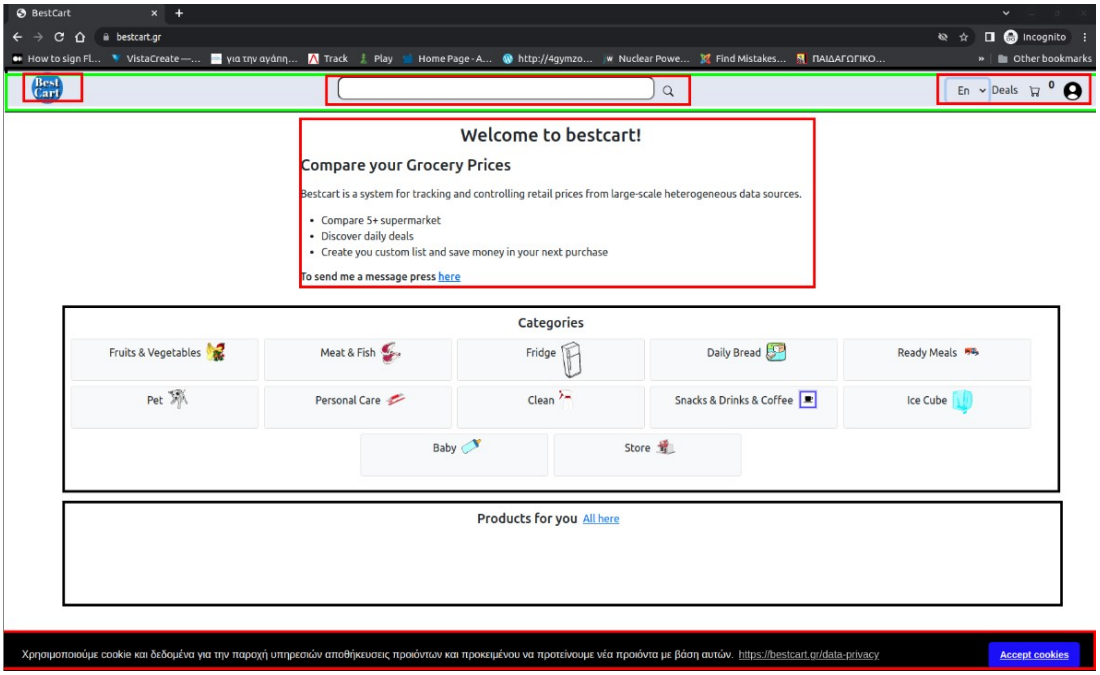


Image 5.2.1 Home page with all the components
In addition to the standard HTML code, it is worth highlighting the approach I have taken in regards to routing. I have chosen to implement a specific number of routing points, each of which loads a distinct component.

```
const routes: Routes = [
  {path: '', pathMatch: 'full', component: HomeComponent},
  {path: 'cart', component: CheckoutPageComponent},
  {path: 'search', component: ItemsListComponent},
  {path: 'hot-deals', component: HotDealsComponent},
  {path: 'login', component: LoginComponent},
  {path: 'write-message', component: WriteMessageComponent},
  {path: 'root', component: RootComponent},
  {path: 'personalized-deals', component: PersonalizedDealsComponent},
  {path: 'data-privacy', component: DataPrivacyComponent},
];
```

Image 5.2.2 Routes

Sitting above every component is the Header component, which includes my logo on the left, a search bar in the center, and various buttons such as language button, deals button, personal cart, and login button to the right. The Header component is responsible for managing routing changes initiated from the header. Specifically, the onNavigate() function is called on every button click.

```
35  onNavigate(input: any) {
36    if (input === 'main') {
37      this.router.navigate(['']);
38      this.search = '';
39    } else if (input === 'hot-deals') {
40      this.router.navigate(['hot-deals']);
41    }else if (input === 'cart') {
42      this.router.navigate(['cart']);
43      this.search = '';
44    } else if (input === 'search' && this.search.length > 0) {
45      this.router.navigate(['search'],
46        {
47          queryParams: {
48            'search': this.search,
49            'page': 1
50          },
51        }
52      );
53    } else if (input === 'login') {
54      // this.router.navigate(['login']);
55      if (this.loggingService.userValue == null) {
56        this.loggingService.googleLogin();
57      }
58    } else if (input === 'profile') {
59      this.router.navigate(['login']);
```

Image 5.2.3 Above code ov onNavigate()

```
19    constructor(
20      private router: Router,
21      public cartService: CartService,
22      public loggingService: LoggingService,
23      private dataService: DataService
24    ) { }
```

Image 5.2.4 Header Components Services

Depending on the button pressed, the corresponding routing point is altered. Furthermore, to optimize the use of space on mobile screens, I have incorporated CSS animations to ensure that the search bar only appears once the user clicks the search button on mobile devices.

```
60      } else if (input === 'search-full') {
61        let small_header = document.querySelectorAll<HTMLElement>('.small-header')[0].className;
62        let inside_header = document.querySelectorAll<HTMLElement>('.inside-header')[0].className;
63
64        let btns: string[] = [];
65        let b = document.getElementsByTagName("button");
66        for (let i = 0; i < b.length; ++i) {
67          btns.push(b[i].className);
68          b[i].className += " btn-disabled disabled"
69        }
70
71        // document.querySelectorAll<HTMLElement>('.remove-header-button')[0].className += " btn-disa
72
73
74        document.querySelectorAll<HTMLElement>('.small-header')[0].style.display = 'inline';
75
76        document.querySelectorAll<HTMLElement>('.inside-header')[0].className += " disappear";
77        document.querySelectorAll<HTMLElement>('.small-header')[0].className += " appear";
78
79        setTimeout( () => {
80          for (let i = 0; i < btns.length; ++i) {
81            b[i].className += btns[i];
82          }
83
84          document.querySelectorAll<HTMLElement>('.inside-header')[0].style.display = 'none';
85
86          document.querySelectorAll<HTMLElement>('.small-header')[0].className = small_header;
87          document.querySelectorAll<HTMLElement>('.inside-header')[0].className = inside_header;
88        }, 2000);
```

Image 5.2.5 Open search on mobile

To achieve this, I have implemented a small-header CSS class, an inside-header class, and an appear and disappear class that I can toggle each time the user clicks the button. One important consideration is that to ensure the animation functions correctly, a timeout (a JavaScript delay) was added to remove the display of the remaining header. Similarly, this approach was also used to implement the closure of the search bar.

```
90      } else if (input === 'close-search') {
91        let small_header = document.querySelectorAll<HTMLElement>('.small-header')[0].className;
92        let inside_header = document.querySelectorAll<HTMLElement>('.inside-header')[0].className;
93
94        let btns: string[] = [];
95        let b =
96        for (le
97          btns.
98          b[i].className += " btn-disabled disabled"
99        }
100
101       document.querySelectorAll<HTMLElement>('.inside-header')[0].style.display = 'inline';
102
103       document.querySelectorAll<HTMLElement>('.small-header')[0].className += " disappear";
104       document.querySelectorAll<HTMLElement>('.inside-header')[0].className += " appear";
105
106       setTimeout( () => {
107         for (let i = 0; i < btns.length; ++i) {
108           b[i].className += btns[i];
109         }
110
111         document.querySelectorAll<HTMLElement>('.small-header')[0].style.display = 'none';
112
113         document.querySelectorAll<HTMLElement>('.small-header')[0].className = small_header;
114         document.querySelectorAll<HTMLElement>('.inside-header')[0].className = inside_header;
115       }, 2000);
116     }
117
118   }
```

(property) Element.className: string

Returns the value of element's class content attribute. Can be set to change it.

Image 5.2.6 Search bar closure

Before delving into the other components, it is imperative to explain my custom-made services as they appear in the constructor. One such service is the LoggingService, which is responsible for managing user data, handling the Google sign-in API, and maintaining a subject to enable any component that wishes to keep track of changes to the user variable to subscribe to it. This way, any component can be notified when the user value is modified.

Image 5.2.7 User data

Moreover, to subscribe to my database, there is a loginServer function that is called once a user is logged in.

```
10      private _user = new BehaviorSubject<SocialUser | null>(null);
11      public userValue !: SocialUser | null;
12      public readonly user: Observable<SocialUser | null> = this._user.asObservable();
```

```
50    loginServer(userValue: SocialUser) {
51
52        let params = new HttpParams();
53        // console.log("login server: ", userValue.id)
54        if (userValue) {
55            params = params
56                .set('user_id', userValue.id)
57                .set('user_firstName', userValue.firstName)
58                .set('user_lastName', userValue.lastName)
59                .set('user_email', userValue.email)
60                .set('user_name', userValue.name)
61                .set('user_authToken', userValue.authToken)
62                .set('user_idToken', userValue.idToken)
63                .set('user_authorizationCode', userValue.authorizationCode)
64                .set('user_provider', userValue.provider);
65        }
66        // console.log(params);
67
68        return this.httpClient.get<void>(environment.path + 'login', { params });
69    }
```

Image 5.2.8 login Server

Another such service is the CartService, which is responsible for managing the cart data. Specifically, it has a subject (as the user) in which all the product data transfer objects (DTO) arrive from the backend. This time, we need two variables, one for the data and one for the size.

```
12      private _data = new BehaviorSubject<ProductDto[]>([]);
13      private _dataSize = new BehaviorSubject<number>(0);
14
15      public readonly data: Observable<ProductDto[]> = this._data.asObservable();
16      public readonly dataSize: Observable<number> = this._dataSize.asObservable();
```

Image 5.2.9 data and datasize Subjects

The LoggingService includes a variety of functionalities, such as the retrieval of personalized products for the home page, updating the personal cart in the database, and reorganizing the cart.

```
// products: ProductDto[],
    getPersonalizedProducts(page: number = 1) {
        let params = new HttpParams();

        let data = this._data.getValue();
        let p = [];
        for (let d in data){
            p.push(data[d].id);
        }
        params = params.set("cart", JSON.stringify(p));
        params = params.set("page", page);
        // console.log(params);
        return this.httpClient.get<ListDto>(environment.path + "personalized-products", {params});
    }

    updateAfterLogin(user: SocialUser | null, setting: string = "nothing") {
        const prev = localStorage.getItem('cart');
        if (prev === null) {
            localStorage.setItem('cart', JSON.stringify([]));
        } else {
            // console.log("Here?")
            const products: ProductDto[] = JSON.parse(prev);
            this.updateCartWithNewData(products, user, setting).pipe(take(1)).subscribe(res => {
                localStorage.setItem('cart', JSON.stringify(res.products));
                // console.log("products: ", res.products);

                this._data.next(res.products);
                this._dataSize.next(this._data.getValue().length);
            });
        }
    }
    updateCartWithNewData(products: ProductDto[], user: SocialUser | null, setting: string) {
        let params = new HttpParams();
        products = products.filter(p => p != null);

        if (user != null) {
            params = params.set("user_id", user.id);
            params = params.set("user_idToken", user.idToken);
        }
        // console.log(products);

        params = params.set('products', products.map((product: ProductDto) => product.id).join(', '));
        // console.log(products);
        params = params.set('quantities', products.map((product: ProductDto) => product.quantity).join(', '));
        params = params.set('setting', setting);

        return this.httpClient.get<ProductListDto>(environment.path + "cart/update", {params});
    }

    getSuperMarket(superMarket: string) {
        return this._data.getValue().filter(x => x?.supermarket === superMarket);
    }

    getProducts() {
        return this._data.getValue();
    }
```

Image 5.2.10 Basic functions

```
updateProduct(product: ProductDto) {
    let list = this._data.getValue().filter(x => x.product != product.product);

    let v = this._data.getValue();
    for (let i = 0; i < v.length; i++) {
        if (v[i].product == product.product) {
            list.splice(i, 0, product);

            this._data.next(v);
            this._dataSize.next(v.length);

            this.updateLocalStorage("overwrite");

            return ;
        }
    }
}
updateProducts(products: ProductDto[]) {
    this._data.next(products);
    this._dataSize.next(products.length);

    this.updateLocalStorage("overwrite");
}

deleteProduct(product: ProductDto) {
    const list = this._data.getValue().filter(item => item != product);

    this._data.next(list);
    this._dataSize.next(list.length);
    this.updateLocalStorage("overwrite");
}

moveProduct(supermarket: string, startIndex: number, endIndex: number) {
    const list = this._data.getValue();

    // Find the 2 item indexes
    let index1 = -1, index2 = -1;
    let j = 0;
    for (let i = 0; i < list.length; i++) {
        if (j === startIndex) {
            index1 = i;
        }
        if (j === endIndex) {
            index2 = i;
        }

        if (list[i].supermarket == supermarket) {
            j++;
        }
    }
    let p_rem = list[index1];
    let list2 = this._data.getValue().filter(item => item != p_rem);

    list2.splice(index2, 0, p_rem);

    this._data.next(list2);
```

Image 5.2.11 Basic functions

```
export interface ProductDto {
    id: number;

    product : string;
    product_english: string;

    price: number;
    metric: string;

    supermarket: string;
    supermarket_english: string;

    quantity: number;

    company ?: string;
    company_english ?: string;

    category ?: string;
    category_english : string;

    img ?: string;
    url ?: string;

    ratio ?: number;
    weight ?: number;

    discount ?: boolean;
    discount_value : number;
    lastUpdated?: string;

    dropPrice: number;
}
```

Image 5.2.12 The ProductDto object

Another such service is the DataService, which is responsible for managing the visual data. Specifically, it handles the search load, the deals load, the the prices list in a single product.

```
    constructor(private httpClient: HttpClient){}

    search(search: string, page: number) {
        let params = new HttpParams();
        params = params.set('search', search);
        params = params.set('page', page);
        // params = params.set('filters', filters);

        return this.httpClient.get<ListDto>(environment.path + 'search', { params }); // pass DTO HERE
    }
    getHotDeals(page: number) {
        let params = new HttpParams();
        params = params.set('page', page);

        return this.httpClient.get<ListDto>(environment.path + 'bestdeals', {params});
    }
    getPrices(id: number) {
        return this.httpClient.get<{prices: number[], dates: string[]}>(environment.path + 'prices/' + id);
    }

}
```

Image 5.2.13 DataService API

In addition, I designed a list module that handles the UI update of a list of products. Note that on each search it loads only 25 products for the best efficiency.

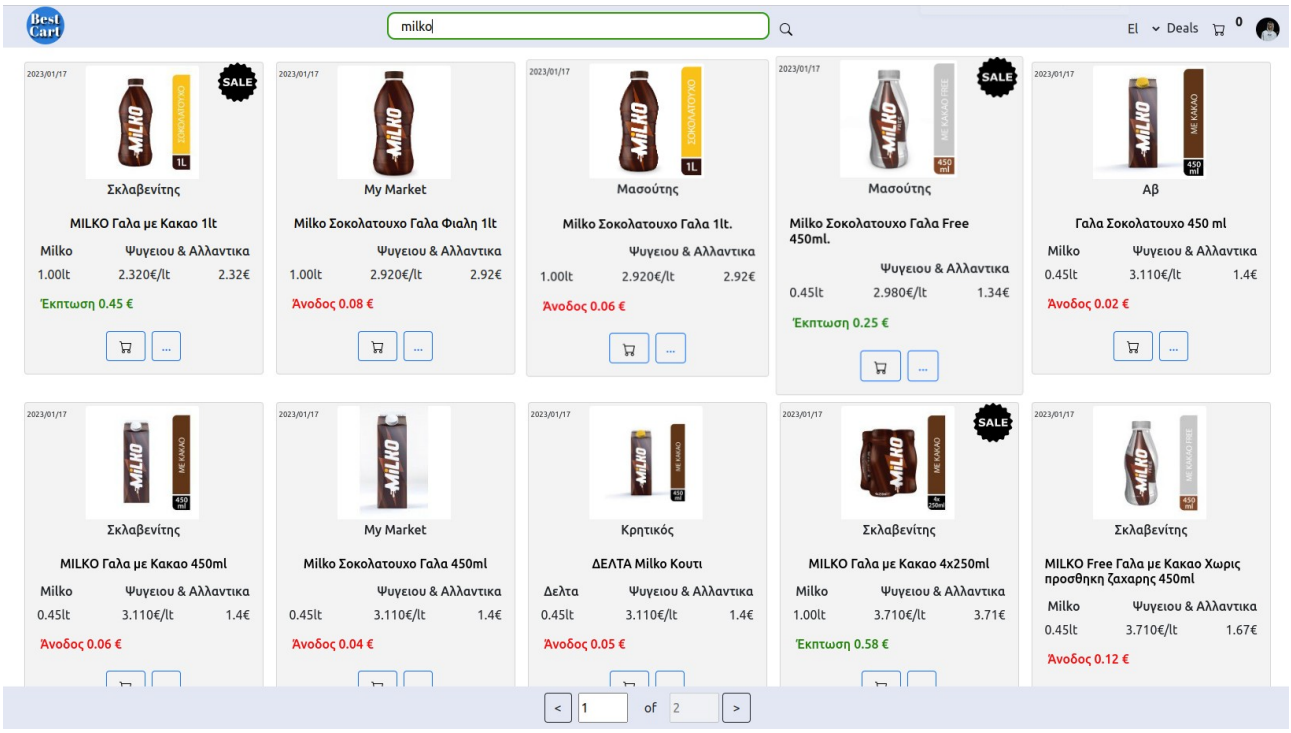


Image 5.2.14 Custom search.

Next adding a product to the cart it appears as

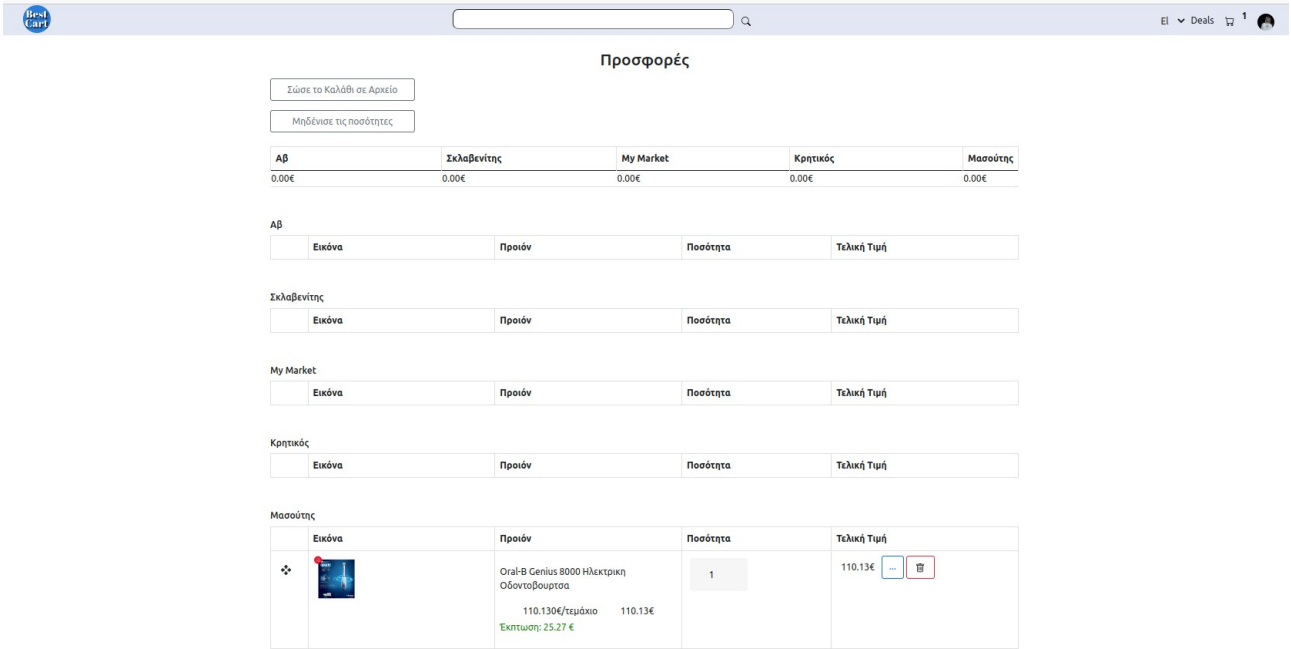


Image 5.2.15 Cart

This loads suggestions in the home page.

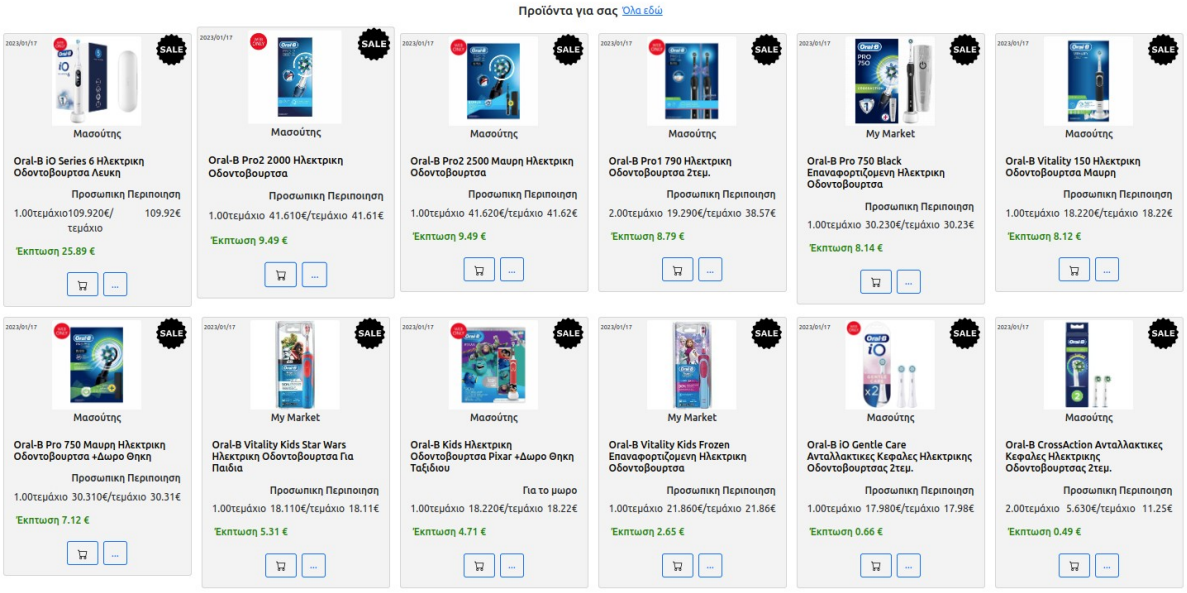
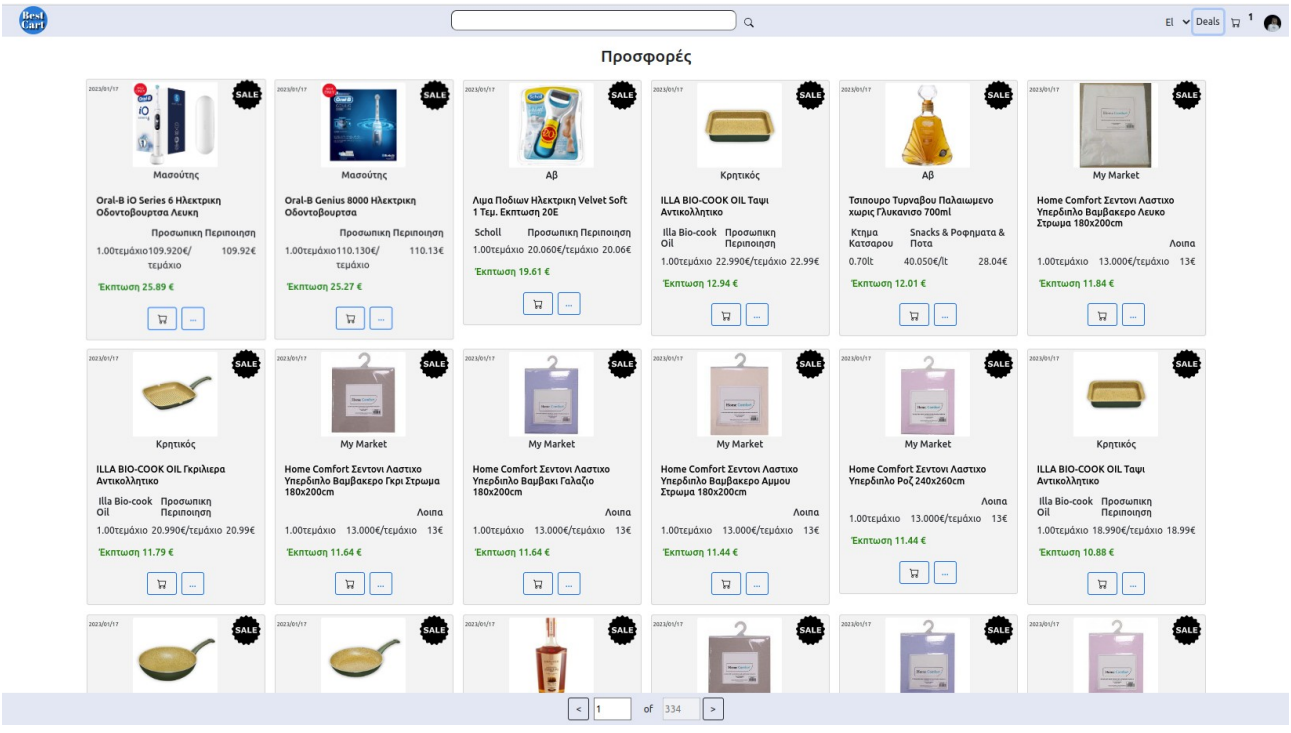


Image 5.2.16 Suggestions

Lastly, we have the hot-deals Image



5.2.17 Hot deals

5.3 Results and performance

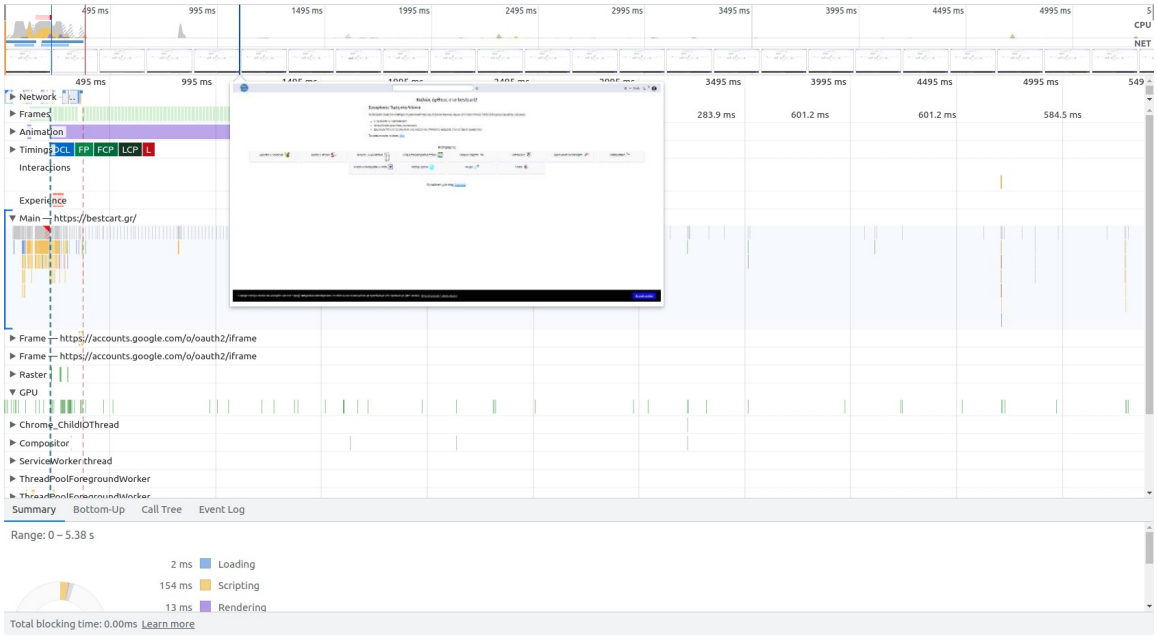
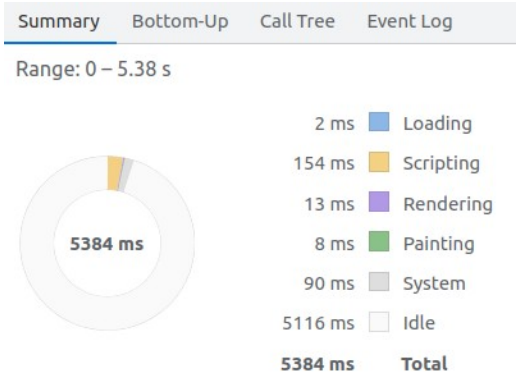


Image 5.3.1 Application load on a new browser



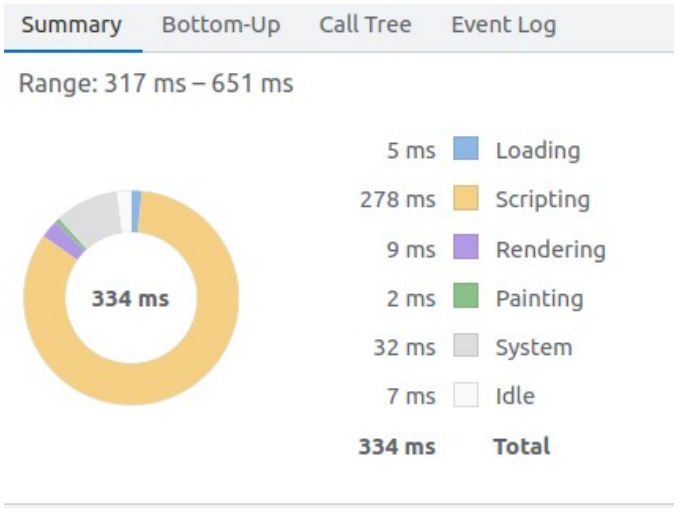


Image 5.3.2 Summary

Image 5.3.3 Application load on a previously logged in device

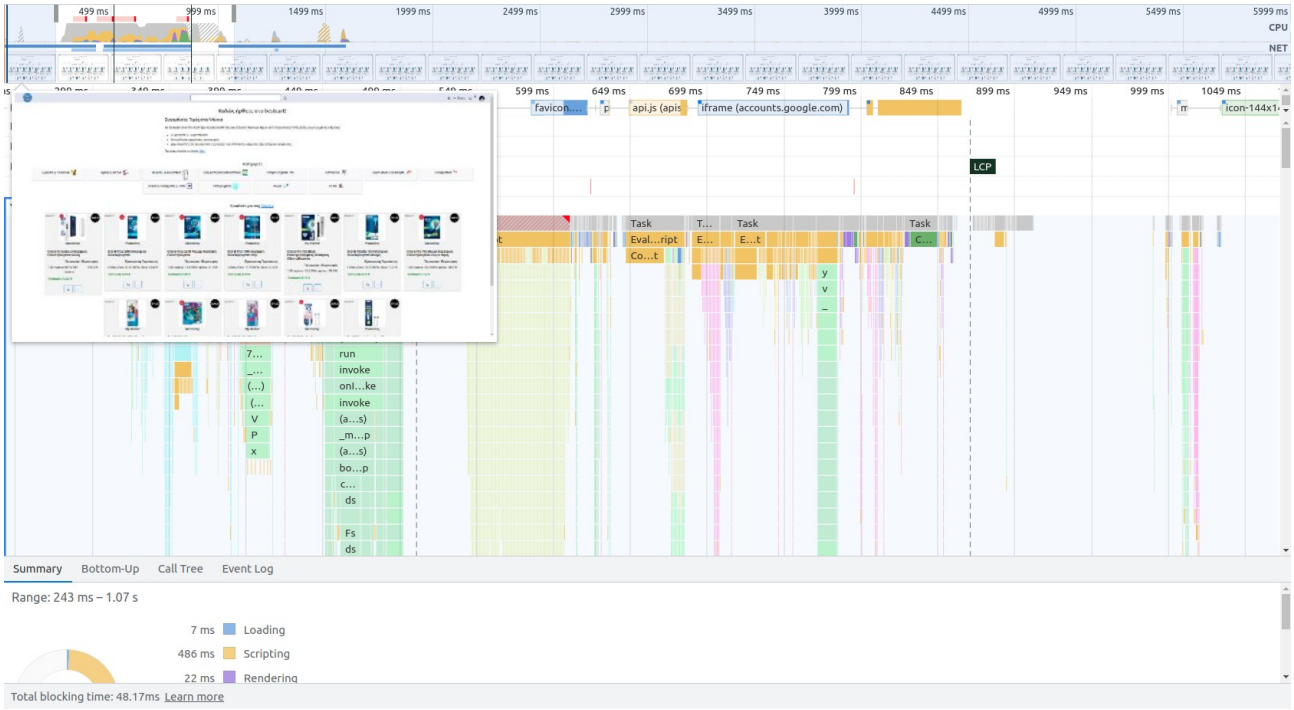


Image 5.3.4 Summary

We can see the noticable difference in speed. That is to be expected. The appication is progressive (PWA). PWA stands for Progressive Web App. PWAs are web applications that use modern web technologies such as service workers and web app manifests to provide a user experience similar to that of a native app on a mobile device. PWAs can be accessed through a web browser like traditional websites, but they can also be installed on a user's device like a native app and can work offline or with low-quality network. PWAs are designed to be fast, reliable, and engaging, and they can be used across multiple platforms such as Android and iOS.

PWAs can also be added to the home screen of a user's device and can receive push notifications. They can also be indexed by search engines, making it easier for users to find them. For all those functionalities, it needs to load more files that just the webpage initially. Hence, the initial delay is plausible.

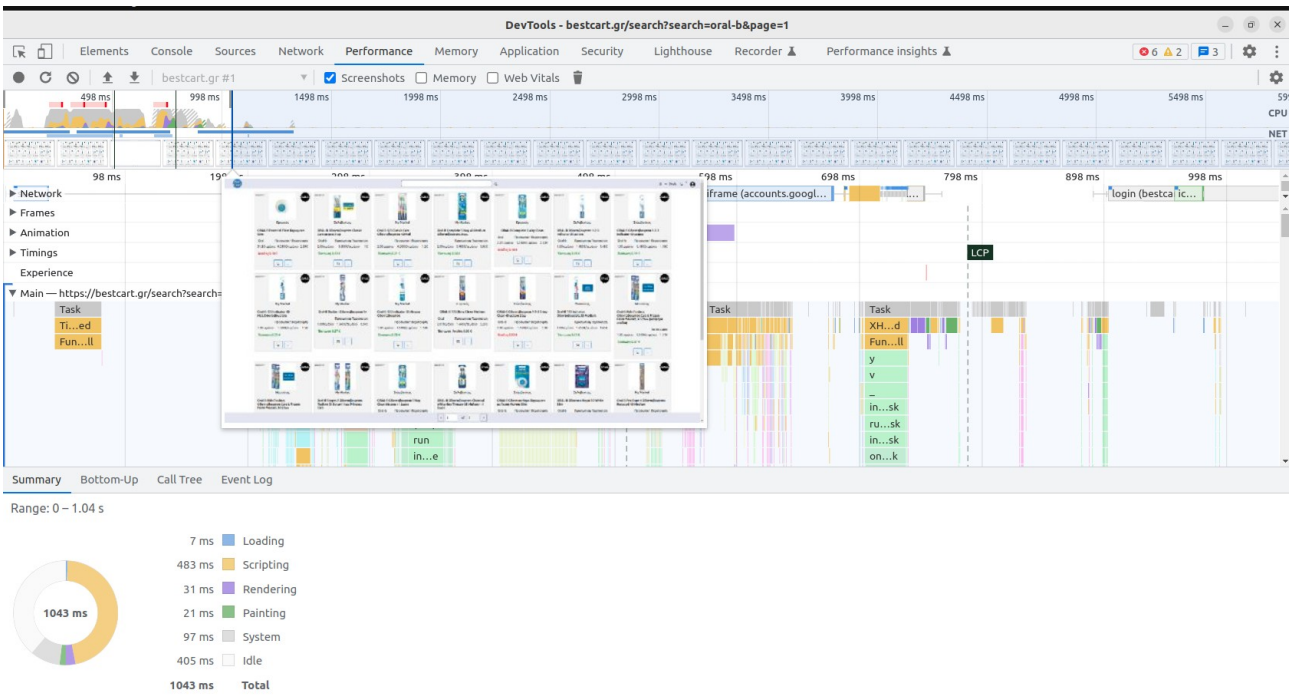


Image 5.3.5 “Oral-B” search results

As we see, this is the api response time adding to the ui delays.

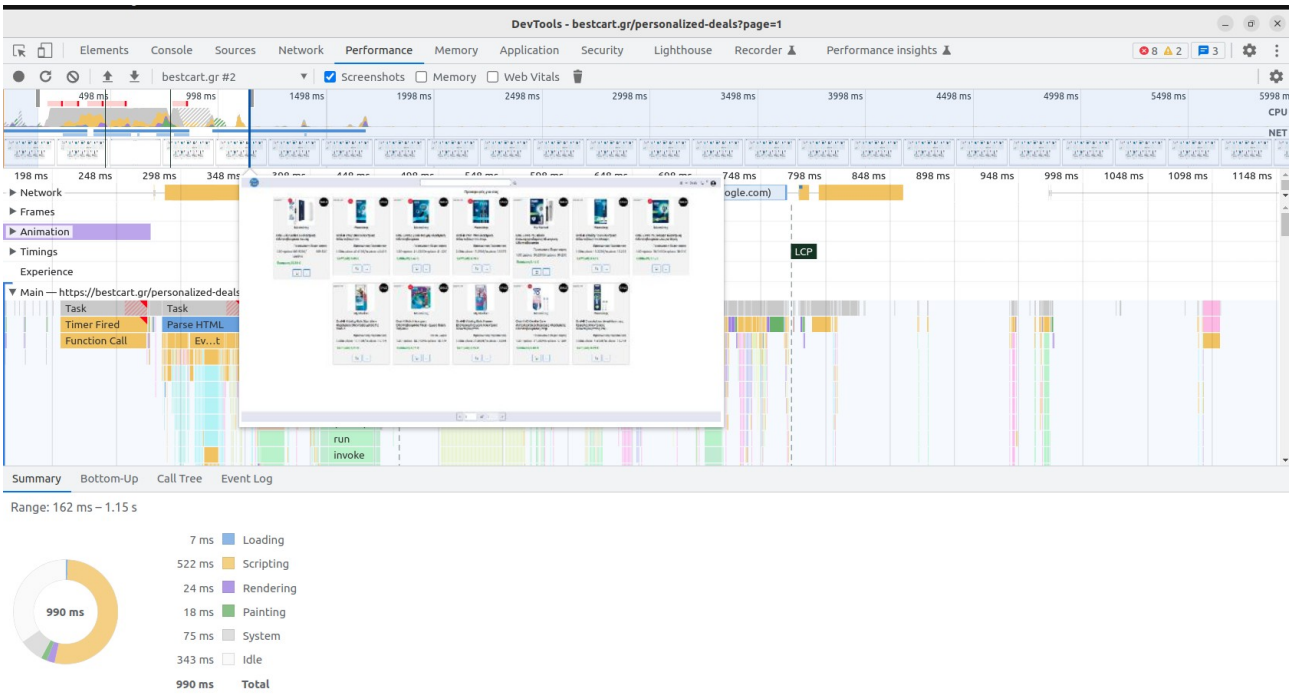


Image 5.3.6 Personalized deals result

In conclusion, the results of the user interface testing indicate that on average, the page fully loads within 1 second. It is worth noting that these results were obtained using a 8GB RAM VPS, which likely contributed to the efficient loading time. Overall, the user interface design and implementation have been successful in providing a fast and responsive experience for users. It is important to continue monitoring and optimizing the UI performance to ensure that the web application remains efficient and user-friendly.

Βιβλιογραφία

[1] Mitchell, Ryan. Web Scraping with Python. 2nd ed., O'Reilly Media, Inc., 2018, https://edu.anarcho-copy.org/Programming%20Languages/Python/Web%20Scraping%20with%20Python,%202nd%20Edition.pdf.

[2] Azevedo, Alisson. "Web Scraping with Python and Selenium." IOSR Journal of Computer Engineering, vol. 23, no. 3, May-June 2021, pp. 1-8.

[3] "Scrapy | An open-source and collaborative web crawling framework." Scrapy.org, Scrapinghub Ltd., n.d., https://docs.scrapy.org/en/latest/index.html.

[4] "Beautiful Soup Documentation." Beautifulsoup.crummy.com, Leonard Richardson, n.d., https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[5] "Selenium with Python." Selenium.dev, Selenium, n.d., https://www.selenium.dev/documentation/en/.

[6] "Requests-HTML: HTML Parsing for Humans™." Html.python-requests.org, Kenneth Reitz, n.d., https://html.python-requests.org/.

[7] Picot, Julien. "Python Libraries for Web Scraping." ProjectPro.io, ProjectPro, 8 March 2016, https://www.oncrawl.com/technical-seo/introduction-web-crawler/#:~:text=Web%20crawling%20started%20as%20mapping,if%20any%20issue%20was%20spotted.

[8] Han, Ray. "Python Libraries for Web Scraping." ProjectPro.io, ProjectPro, 12 January 2023, https://www.projectpro.io/article/python-libraries-for-web-scraping/625.

[9] Nechytailo, Yelyzaveta. "How to Make Web Scraping Faster." Oxylabs.io, 7 June 2022, https://oxylabs.io/blog/how-to-make-web-scraping-faster.

[10] Grinberg, Miguel. Flask Web Development. O'Reilly Media, 2014.

[11] Myers, Jason. Essential SQLAlchemy: Mapping Python to Databases. O'Reilly Media, Inc., 2009.

[12] Nedelcu, Clement. Nginx: From Beginner to Pro. Apress, 2017.

[13] Turnbull, James. "The Docker Book" 2 Setpember 2014, http://lsi.vc.ehu.es/pablogn/docencia/manuales/The%20Docker%20Book.pdf

[14] Bijoyan, Das, and Sarit, Chakraborty. "An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation", pp. 1-6.

[15] Simha, Anirudha. "Understanding TF-IDF." Capital One, Capital One, 6 Oct. 2021, www.capitalone.com/tech/machine-learning/understanding-tf-idf/.

[16] Yang, Xiao. "WEB FRONT-END ARCHITECTURE WITH NODE.JS PLATFORM", pp 1-50.

[17] David, Arthur, and Sergei, Vassilvitskii. “k-means++: The Advantages of Careful Seeding”, https://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf

[18] D. T. Pham, S. S. Dimov, and C. D. Nguen. “Selection of K in K-means clustering”, 26 May 2004, https://www.ee.columbia.edu/~dpwe/papers/PhamDN05-kmeans.pdf

[19] Bhardwaj, Ashutosh. "Silhouette Coefficient: Validating Clustering Techniques." Towards Data Science, 26 May 2020, https://towardsdatascience.com/silhouette-coefficient-validating-clustering-techniques-e976bb81d10c.

[20] Garcia, Edel. (2015). Cosine Similarity Tutorial, https://www.researchgate.net/publication/327248753_Cosine_Similarity_Tutorial.