ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη Web Application, για την ομαδική αγορά δώρων

Διπλωματική Εργασία

Μέμου Ελβίρα

Μπανταβάνης Ιωάννης-Χρήστος

Επιβλέπων:

Σταμούλης Γεώργιος

Ιανουάριος 2022

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Development of a web application for buying presents

Diploma Thesis

Memou Elvira

Bantavanis Ioannis-Christos

Supervisor: George Stamoulis

January 2022

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπων	**Σταμούλης Γεώργιος** Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Δαδαλιάρης Αντώνιος** Π.Δ. 407, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Ξενάκης Απόστολος** Επίκουρος Καθηγητής, Τμήμα Ψηφιακών Συστημάτων, Πανεπιστήμιο Θεσσαλίας

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελούν αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλουν οποιασδήποτε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχουν έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Δηλώνω επίσης ότι τα αποτελέσματα της εργασίας δεν έχουν χρησιμοποιηθεί για την απόκτηση άλλου πτυχίου. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Η Δηλούσα,
Μέμου Ελβίρα
27/01/2022

Ο Δηλών,
Μπανταβάνης Ιωάννης- Χρήστος
27/01/2022

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

The Declarant,
Memou Elvira
27/01/2022

The Declarant,
Bantavanis Ioannis-Christos
27/01/2022

Διπλωματική Εργασία

Ανάπτυξη Web Application, για την ομαδική αγορά δώρων

Μέμου Ελβίρα

Μπανταβάνης Ιωάννης-Χρήστος

Περίληψη

Με την ραγδαία εξέλιξη της τεχνολογίας τα τελευταία χρόνια καθώς και με την εξοικείωση μεγάλου ποσοστού του πληθυσμού με αυτήν, πολλές πλατφόρμες/εφαρμογές κοινωνικής δικτύωσης και ηλεκτρονικού εμπορίου, έχουν μπει στην καθημερινότητα μας. Ο σκοπός της παρούσας διπλωματικής διατριβής είναι η δημιουργία μιας πλατφόρμας που θα συνδυάζει βασικά στοιχεία εφαρμογών κοινωνικής δικτύωσης και ηλεκτρονικού εμπορίου, με αποτέλεσμα την πραγματοποίηση ομαδικών δώρων επιθυμίας του κάθε χρήστη. Η εφαρμογή αναπτύχθηκε με τη χρήση MERN Stack (MongoDB, Express, React, Node). Στην παρούσα εργασία αναφέρονται και αναλύονται οι τεχνολογίες που χρησιμοποιήθηκαν για την διεκπεραίωση της, καθώς και η ανάλυση των επιμέρους σταδίων της εφαρμογής. Τέλος εκθέτουμε κάποιες ιδέες για την εξέλιξη της εφαρμογής.

Λέξεις-κλειδιά:

MERN STACK, MongoDB, Express, React, Node, Web-App, JavaScript, Axios.

Diploma Thesis

Development of a web application for buying presents

Memou Elvira

Bantavanis Ioannis-Christos

Abstract

Due to the rapid development of technology in recent years as well as the familiarization of a large percentage of the population with it, a lot of platforms/applications of social network and e-commerce are a serious part of our life. The purpose of this thesis is the development of a platform which combines basic elements of social network and e-commerce applications, resulting in realization of group gifts of desire of each user. The application developed using the MERN Stack (MongoDB, Express, React, Node). In the present thesis the technologies used to process it are reported and analyzed, as well as the analysis of the individual stages of the application. Finally, we present some ideas for the additional progress of the application.

Keywords:

MERN STACK, MongoDB, Express, React, Node, Web-App, JavaScript, Axios.

Πίνακας Περιεχομένων

Συντομογραφίες

JSON	JavaScript Object Notation
MERN	MongoDB Express React Nodejs
API	Application Programming Interface
JWT	JSON Web Token
I/O	Input/Output
JS	JavaScript
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
PHP	PHP: Hypertext Preprocessor
ASP	Active Server Pages
HTTP	HyperText Transfer Protocol
npm	Node Package Manager
URL	Uniform Resource Locator
SQL	Structured Query Language
NoSQL	Not only SQL
DOM	Document Object Model
κ.λπ.	και λοιπά

Κεφάλαιο 1 Εισαγωγή

Την τελευταία δεκαετία έχουν αναπτυχθεί ραγδαία οι ηλεκτρονικές αγορές. Για το λόγο αυτό όλο και περισσότερες επιχειρήσεις έχουν στραφεί στο ηλεκτρονικό εμπόριο επενδύοντας στην δημιουργία του δικού τους e-shop. Με σκοπό την έκθεση των προϊόντων τους σε ένα ευρύτερο κοινό, επιλέγουν πλατφόρμες που δραστηριοποιούνται στον χώρο του online e-commerce marketplace (amazon, eBay, skroutz, κ.λπ.). Αυτές οι πλατφόρμες δίνουν την δυνατότητα αναζήτησης προϊόντων, δημιουργίας Wishlist και πραγματοποίησης δώρων. Καμία από αυτές τις πλατφόρμες όμως δεν παρέχει την δυνατότητα ομαδικών δώρων.

1.1 Αντικείμενο της διπλωματικής

Λόγω του προαναφερθέντος προβλήματος δημιουργήσαμε μια Web εφαρμογή, όπου από τη μία τα συμβεβλημένα καταστήματα έχουν τη δυνατότητα να ανεβάζουν τα προϊόντα τους στην εφαρμογή, ενώ από την άλλη οι χρήστες δημιουργούν την δική τους Wishlist και συμβάλουν στην αγορά δώρων μόνο για τους φίλους τους. Οι χρήστες δεν έχουν την δυνατότητα να συνεισφέρουν στο δικό τους δώρο, ώστε να μην χάνεται το ευχάριστο συναίσθημα της έκπληξης.

Στην παρούσα διπλωματική εργασία δίνεται έμφαση στο Web Development χρησιμοποιώντας τη σύγχρονη και ανερχόμενη τεχνολογία MERN Stack. Συγκεκριμένα αναλύονται οι τεχνολογίες που χρησιμοποιήσαμε καθώς και ο λόγος που τις επιλέξαμε. Ασχοληθήκαμε με αυτό το θέμα για να εξοικειωθούμε με την συγκεκριμένη τεχνολογία, αφού αποτελεί μια σπουδαία αφετηρία για την επαγγελματική μας ανέλιξη.

1.2 Οργάνωση του τόμου

Στο Κεφάλαιο 2 γίνεται αναφορά στο Back-End και στις επιμέρους τεχνολογίες που χρησιμοποιήθηκαν.

Στο Κεφάλαιο 3 αναφέρονται οι τύποι βάσεων δεδομένων που θα μπορούσαν να χρησιμοποιηθούν και αναλύεται η βάση δεδομένων που επιλέχθηκε.

Στο Κεφάλαιο 4 παρουσιάζονται οι τεχνολογίες που χρησιμοποιήθηκαν για το Front-End της εφαρμογής.

Το Κεφάλαιο 5 περιέχει τις βασικές λειτουργίες της εφαρμογή μας, καθώς και τα πιο σημαντικά κομμάτια κώδικα.

Τέλος, στο Κεφάλαιο 6 παρουσιάζονται κάποιες προτάσεις για την επέκταση και υλοποίηση της εφαρμογής σε πραγματικό χρόνο.

Κεφάλαιο 2 Back-End

Το Web Development είναι βασισμένο στο front-end και back-end development. Και τα δύο είναι απαραίτητα για την λειτουργικότητα ενός website.

Με τον όρο ”back-end” αναφερόμαστε στις διαδικασίες που υλοποιούνται στο παρασκήνιο, το οποίο δεν είναι ορατό από το χρήστη. Με λίγα λόγια, αφορά όλες τις ενέργειες οι οποίες είναι απαραίτητες έτσι ώστε μια εφαρμογή να είναι λειτουργική και βοηθάει το browser να επικοινωνεί με τη βάση δεδομένων [1] .

Κάποιες τυπικές ενέργειες του back-end είναι:

- Λήψη και μεταφόρτωση δεδομένων.
- Ανάκτηση εικόνων από τη βάση δεδομένων.
- Ενημέρωση βάσης δεδομένων.
- Απάντηση στα αιτήματα των χρηστών.
- Αποθήκευση δεδομένων σε κατακερματισμένη μορφή (hashed version) για λόγους ασφάλειας.
- Έλεγχος και παροχή εξουσιοδότησης στον εκάστοτε χρήστη.

Figure 1: Τα δεδομένα από τη μεριά του browser είναι γνωστά ως front-end (client-side), ενώ οτιδήποτε σχετίζεται με τον web server και τη βάση δεδομένων είναι γνωστά ως back-end [1].

2.1 JavaScript

Η JavaScript είναι μια γλώσσα προγραμματισμού, η οποία χρησιμοποιείται τόσο στην δημιουργία του front-end όσο και στου back-end, κάτι που εύκολα δίνει την δυνατότητα στους web developers να αναπτύξουν full-stack (front-end και back-end) μια εφαρμογή. Τα τελευταία 9 χρόνια είναι η πιο διαδεδομένη γλώσσα προγραμματισμού στον επαγγελματικό τομέα και όχι μόνο. Πιο συγκεκριμένα, η JS είναι μια lightweight, interpreted, just-in-time compiled, δυναμική, αντικειμενοστραφής γλώσσα προγραμματισμού [2].

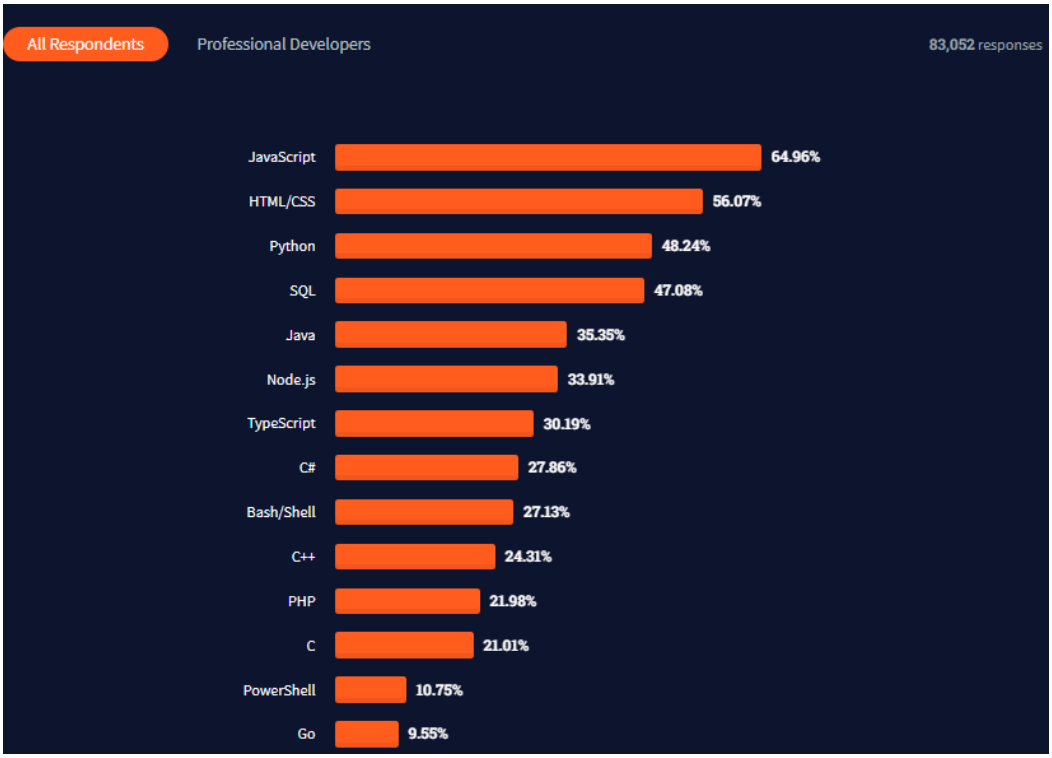

Figure 2: Most used programming languages (2021) [2].

Για να δημιουργηθεί μια ιστοσελίδα χρειάζονται 3 βασικά στοιχεία: HTML, CSS, JS. Η HTML είναι υπεύθυνη για την δομή της σελίδας, η CSS για την παρουσίασή της και η JS για την λειτουργικότητα, την συμπεριφορά και την διαδραστικότητα με τον χρήστη. Παρόλο που είναι ευρέως γνωστή ως scripting language για ιστοσελίδες, χρησιμοποιείται επίσης και για non-browser περιβάλλοντα, όπως τα Apache CouchDB, Adobe Acrobat, Node.js, κ.λπ. [3] [4].

2.2 Node.js

Το Node.js είναι ένα εργαλείο ανοιχτού κώδικα της JS το οποίο δημιουργήθηκε στο Google Chrome JavaScript Engine. Τρέχει σε διάφορες πλατφόρμες (Windows, Linux, Unix, Mac OS X, κ.λπ.) και δίνει την δυνατότητα σε κώδικα JS να εκτελείται στον server, εκτός δηλαδή του web browser. Με το Node.js μπορούμε να δημιουργήσουμε δυναμικές σελίδες, να δημιουργήσουμε/ανοίξουμε/διαβάσουμε/γράψουμε/διαγράψουμε/κλείσουμε αρχεία στον διακομιστή. Επίσης, μπορούμε να συλλέξουμε Form Data και να επεξεργαστούμε τα δεδομένα της βάσης δεδομένων (εισαγωγή, διαγραφή και επεξεργασία δεδομένων) [5].

Μια Node.js εφαρμογή τρέχει σε μία διεργασία, χωρίς να δημιουργεί καινούριο νήμα για κάθε request. Ένα από τα σημαντικότερα πλεονεκτήματα του Node.js είναι ότι χρησιμοποιεί non-blocking asynchronous I/O μοντέλο, με αποτέλεσμα να είναι lightweight και αποδοτικό. Αυτό σημαίνει ότι δεν χρειάζεται να τελειώσει μια διεργασία (επικοινωνία με τη βάση δεδομένων, κ.λπ.) και να μπλοκάρει το νήμα σπαταλώντας κύκλους της CPU, ενώ βρίσκεται σε αναμονή [6] [7]. Σε αντίθεση με την PHP ή την ASP που ενώ στέλνουν μια διεργασία στο file system πρέπει να περιμένουν μέχρι αυτό να ανοίξει και να διαβάσει το αρχείο και να επιστρέψει στον χρήστη τα δεδομένα, ώστε να είναι έτοιμο να διαχειριστεί το επόμενο request. Στην περίπτωση μας όμως, ενώ στέλνεται η διεργασία στο file system, είναι έτοιμο να διαχειριστεί το επόμενο request, όταν το file system τελειώσει με την πρώτη διεργασία.

Ένα αρχείο Node.js περιέχει κώδικα που θα εκτελεστεί έπειτα από συγκεκριμένες ενέργειες. Μια τυπική ενέργεια είναι η προσπάθεια πρόσβασης σε μια πύλη του server. Για να έχει οποιαδήποτε δυνατότητα λειτουργίας θα πρέπει να είναι πρώτα ορισμένο στο server. Τα αρχεία Node.js έχουν κατάληξη ".js" [5] [8].

```
const http = require("http");

const port = 5000;

const app = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello World');
});

app.listen(port, ()=> console.log("Server started on port " + port));
```

Figure 3: Example of a simple web server

Στο παραπάνω παράδειγμα βλέπουμε την δημιουργία ενός απλού διακομιστή. Αρχικά δημιουργείται ένας HTTP server που συνδέεται στις συγκεκριμένες πύλες του server και στέλνει ένα response πίσω στο χρήστη. Αφού εκτελεστεί ο κώδικας θα εκτυπωθεί το μήνυμα “Server started on port 5000” στο terminal και θα περιμένει μέχρις ότου ένας χρήστης επισκεφτεί την εφαρμογή (localhost:5000), όπου θα εμφανίζεται το μήνυμα “Hello World”.

2.2.1 Npm

Το Node.js επίσης παρέχει εργαλεία και συναρτήσεις όπως το Npm, τα οποία μας δίνουν τη δυνατότητα να χρησιμοποιούμε third-party libraries και frameworks (Express, Mongoose, κ.λπ.) διευκολύνοντας και κάνοντας πιο γρήγορη την υλοποίηση της εφαρμογής.

Πιο συγκεκριμένα συμβάλει στην :

- Προσαρμογή πακέτων κώδικα στην εφαρμογή μας.
- Χρήση αυτόνομων εργαλείων κώδικα.
- Δυνατότητα κοινοποίησης κώδικα σε άλλους χρήστες.
- Κοινοποίηση κώδικα σε συγκεκριμένη ομάδα developers.
- Ενημέρωση εφαρμογής όταν ο κώδικας που χρησιμοποιήθηκε έχει ενημερωθεί.
- Εύρεση άλλων developers που ενδέχεται να έχουν ίδια προβλήματα.

Τα 3 βασικά μέρη του Npm είναι:

- Το website
- Το CLI
- Το registry

Η χρήση του website βοηθά στην ανακάλυψη πακέτων κώδικα , δημιουργία προσωπικού προφίλ και περιέχει και άλλες δυνατότητες, όπως η δημιουργία οργανισμών για διαχείριση εισόδου σε public ή private πακέτα κώδικα.

Το CLI (Command Line Interface) είναι ο τρόπος με τον οποίο οι developers αλληλοεπιδρούν με το npm.

Τέλος, το registry αφορά την public βάση δεδομένων λογισμικού JS και meta-information που τα περικλείουν [9].

2.3 Express

Παραπάνω έγινε αναφορά στη δυνατότητα χρήσης third-party libraries και frameworks του Node.js μέσω του Npm, για την διευκόλυνση και επιτάχυνση της υλοποίησης μιας εφαρμογής. Το πιο ευρέως διαδεδομένο Node.js framework είναι το Express.js, το οποίο προσφέρει μηχανισμούς όπως:

- Δημιουργία handlers για requests με διαφορετικούς HTTP τύπους (GET, POST, DELETE, κ.λπ.) σε διάφορα URL paths (routes).
- Ορισμός ρυθμίσεων της εφαρμογής, όπως της πύλης που χρησιμοποιείται για τη σύνδεση ή την τοποθεσία των templates που χρησιμοποιούνται για την απόδοση του response.
- Χρήση για την εξυπηρέτηση στατικών αρχείων.
- Προσθήκη middleware (ενδιάμεσο λογισμικό) για επιπλέον επεξεργασία των request.

Το Express.js έχει την δυνατότητα να λύσει πολλά προβλήματα του Web Development, παρόλο που φαίνεται εκ πρώτης όψεως ως ένα πολύ απλό framework, γεγονός που οφείλεται στα

πακέτα middleware, τα οποία είτε προέρχονται από αυτό, είτε από διάφορους developers. Τα ήδη υπάρχοντα πακέτα middleware μπορούν να χειριστούν θέματα όσων αφορά cookies, User logins, URL parameters, security headers, POST data κ.λπ. Κάθε διαφορετικό βέβαια έχει τα πλεονεκτήματα και τα μειονεκτήματα του κάνοντας την επιλογή του κατάλληλου για την εφαρμογή μας μια δύσκολη διαδικασία, η οποία αν γίνει με το σωστό τρόπο μπορεί να επιλύσει πρακτικά οποιοδήποτε πρόβλημα. Στην πράξη υπάρχει η δυνατότητα χρήσης πολλών middleware ταυτόχρονα.

Ένα από τα δημοφιλέστερα και πιο χρήσιμα πακέτα middleware είναι το json-parser. Και αυτό γιατί παίρνει raw data από τα requests τα οποία είναι αποθηκευμένα στο request object, το αναλύει-μετατρέπει σε JS object και με την ιδιότητα *body* το τοποθετεί στο request object ξανά. Η εντολή για να είναι διαθέσιμο ένα πακέτο middleware είναι η *app.use().* Στη περίπτωσή του json-parser είναι η *app.use(express.json())* [10].

```
const express = require("express");

const app = express();
const port = 5000;

app.get("/", (request, response)=>{
    response.send("Hello World");
})

app.listen(port, ()=> console.log("Server started on port " + port));
```

Figure 4: Example of a server built with Express.

Για να δημιουργήσουμε μια Express εφαρμογή, αρχικά πρέπει να εισάγουμε το express το οποίο πλέον είναι μια συνάρτηση, η οποία χρησιμοποιείται για την δημιουργία της εφαρμογής και είναι αποθηκευμένη στην μεταβλητή app. Έπειτα ορίζουμε ένα route που διαχειρίζεται ένα HTTP GET request στο '/' root της εφαρμογής. Ο event handler παίρνει δύο παραμέτρους, το request και το response. Στο request είναι αποθηκευμένες όλες οι πληροφορίες από το HTTP request, ενώ στο response ορίζεται η απόκριση του request. Χρησιμοποιώντας την συνάρτηση

sent του response object ο server αποκρίνεται στο request στέλνοντας ένα string με το περιεχόμενο ”Hello World”. Όταν η παράμετρος είναι string το express θέτει αυτόματα τον Content-Type header σε text/html. Στην περίπτωση που θέλουμε να στείλουμε ένα αντικείμενο με JSON format πρέπει να χρησιμοποιήσουμε τη json συνάρτηση του response object και ο Content-Type header να είναι application/json [11].

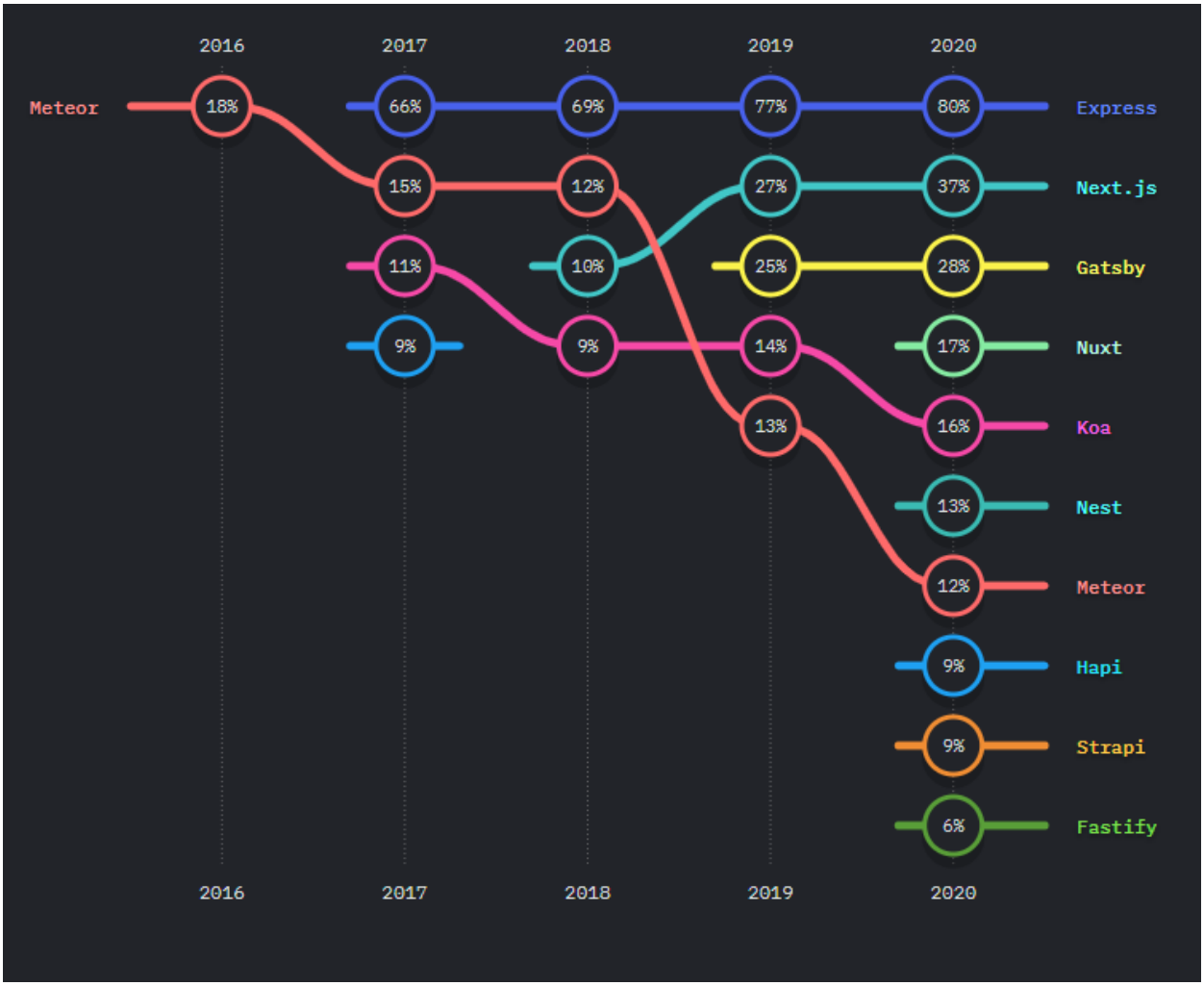

Figure 5: Usage of the most well-known back-end frameworks [12].

2.4 JSON

Η JSON είναι μια μορφή αναπαράστασης δεδομένων η οποία είναι βασισμένη σε κείμενο χωρίς συγκεκριμένο σχήμα. Παρόλο που προέρχεται από την JS, υποστηρίζεται από τις περισσότερες και πιο σημαντικές γλώσσες προγραμματισμού. Η κύρια χρήση της είναι η ανταλλαγή πληροφοριών μεταξύ διακομιστή και χρήστη. Επίσης, υποστηρίζεται πλέον από πολλές βάσεις δεδομένων είτε σχεσιακές (PostgreSQL, MySQL, κ.λπ.), είτε μη σχεσιακές (MongoDB, Neo4j, κ.λπ.).

(a) (b)

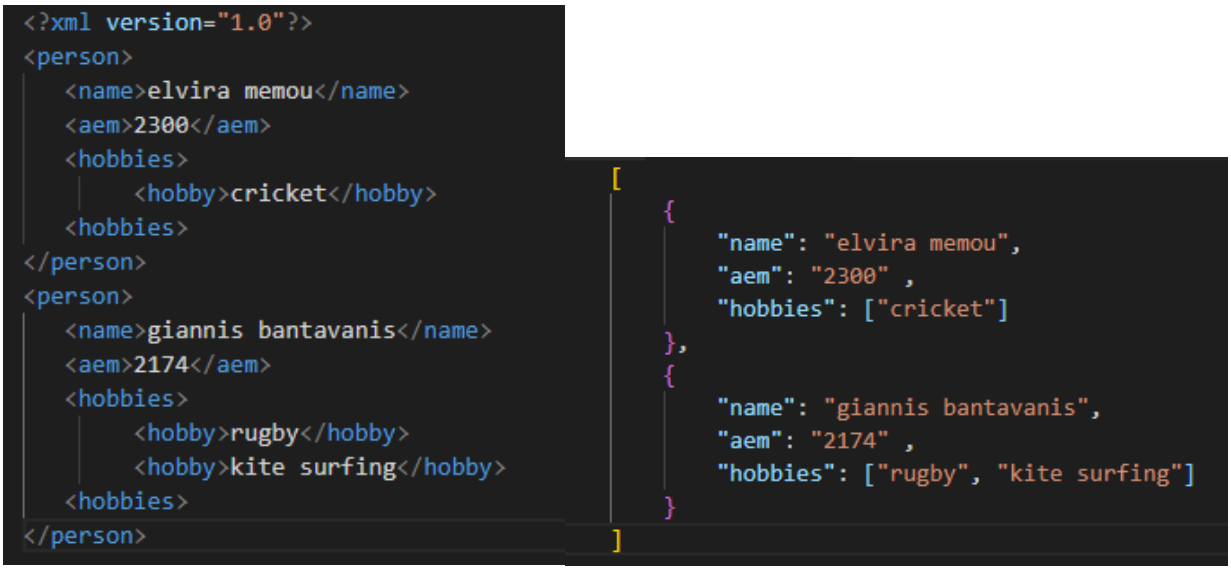

Figure 6: (a) XML file. (b) JSON file

Πριν καθιερωθεί η JSON, χρησιμοποιούνταν περισσότερο η XML. Όπως βλέπουμε και στις παραπάνω εικόνες το XML αρχείο έχει μεγαλύτερο μέγεθος. Επίσης, για να μετατραπεί ένα αντικείμενο XML σε JS, χρειάζεται πολύ δουλειά ανάλογα και με την πολυπλοκότητα του αντικειμένου, ενώ με τα JSON είναι μια γραμμή κώδικα. Τα μοναδικά μειονεκτήματα της JSON είναι ότι δεν υπάρχει σχήμα και ότι δεν μπορούν να συμπεριληφθούν σχόλια [13].

2.5 Asynchronous Programming

Όπως προαναφέραμε η JavaScript τρέχει κώδικα σε μοντέλο non-blocking. Δηλαδή κομμάτια κώδικα που χρειάζονται περισσότερο χρόνο για να τελειώσουν (διάβασμα δεδομένων από το local file system, επικοινωνία με τη βάση δεδομένων, κ.λπ.) εκτελούνται στο παρασκήνιο, επιτρέποντας παράλληλα τη συνέχιση της υλοποίησης του υπόλοιπου κώδικα. Αυτή η διαδικασία ονομάζεται asynchronous programming. Για να έχουμε τα επιθυμητά αποτελέσματα με τα συγκεκριμένα κομμάτια κώδικα, θα πρέπει να πάρουμε κάποια μέτρα πριν ξεκινήσει να εκτελείται το υπόλοιπο μέρος του.

```
const calculator = (num1, num2) => {
    setTimeout(() => {
        return {sum: num1+num2}
    }, 2000)
}

const calc = calculator(5,5);
console.log(calc.sum)
```

Figure 7: Asynchronous Programming error.

Όπως βλέπουμε στο παραπάνω παράδειγμα (Fig.7), το άθροισμα υπολογίζεται μετά από την εκτύπωσή του, επειδή η συνάρτηση calculator περιμένει 2 δευτερόλεπτα μέχρι να το επιστρέψει [14].

2.5.1 Callbacks

Ένας τρόπος να αποφευχθεί το παραπάνω πρόβλημα είναι οι callbacks συναρτήσεις. Οι callbacks είναι απλές συναρτήσεις, οι οποίες καλούνται μόλις έχει εκτελεστεί ένα κομμάτι ασύγχρονου κώδικα και το αποτέλεσμα του είναι διαθέσιμο.

```
const calculator = (num1, num2, callback) => {
    setTimeout(() => {
        callback({sum: num1+num2})
    }, 2000)
}

calculator(5,5,calc=>{
    console.log(calc.sum)
})
```

Figure 8: Callback example.

Στο παραπάνω παράδειγμα η calculator παίρνει ως τρίτη παράμετρο μια callback function, η οποία εκτελείται μετά το πέρας των 2 δευτερολέπτων, δηλαδή αφού έχει υπολογιστεί το άθροισμα. Αν υπήρχε μια εκτύπωση μετά την κλήση της συνάρτησης calculator, αυτή θα εκτυπωνόταν πριν το άθροισμα, αφού οι callback θα καθυστερούσε να εκτελεστεί [14].

2.5.2 Promises

Το ίδιο πρόβλημα μπορεί να επιλυθεί και με Promises, τα οποία είναι μια ενσωματωμένη δυνατότητα της JS για την διαχείριση ασύγχρονου κώδικα.

```
const calculator = (num1,num2) => {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            let error = false;
            if(!error)
                resolve({sum: num1+num2})
            else
                reject()
        }, 2000)
    })
}

calculator(5,5).then(calc => {
    console.log(calc.sum)
}).catch(error => {
    console.log("Error: " + error)
})
```

Figure 9: Promise example.

Η calculator (Fig.9) πρέπει να επιστρέφει ένα new Promise, το οποίο παίρνει ως παράμετρο μια ασύγχρονη συνάρτηση. Αυτή η συνάρτηση παίρνει 2 παραμέτρους, την resolve και την reject, και περιέχει το κώδικα που περιμένουμε να εκτελεστεί. Στην περίπτωση που ο κώδικας εκτελεστεί επιτυχημένα, καλείται η συνάρτηση resolve. Αντίθετα, αν υπάρξει κάποιο error, καλείται η reject. Όταν καλείται η συνάρτηση calculator είμαστε σε θέση να καλέσουμε το then, το οποίο παίρνει σαν παράμετρο μια συνάρτηση και θα εκτελεστεί σε περίπτωση που έχει κληθεί το resolve. Στην περίπτωση που υπάρξει error θα εκτελεστεί το catch, ώστε το error να εκτυπωθεί [14].

2.5.3 Async/Await

Χρησιμοποιώντας async και await μπορούμε να διαχειριστούμε πιο εύκολα τα promises. Όπως βλέπουμε παρακάτω η συνάρτηση calculator δεν έχει αλλάξει καθόλου.

```
const calculator = (num1,num2) => {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            let error = false;
            if(!error)
                resolve({sum: num1+num2})
            else
                reject()
        }, 2000)
    })
}

async function fetchCal(){
    return await calculator(5,5);
}

fetchCal().then(calc => {
    console.log(calc.sum)
})
```

Figure 10: Async/Await example.

Αυτό που αλλάζει είναι ο τρόπος που την καλούμε. Δημιουργούμε μια νέα async συνάρτηση την *fetchCal.* Με τον όρο async η συνάρτηση περιμένει να εκτελέσει ασύγχρονο κώδικα βασισμένο

σε promise. Στην περίπτωση μας αυτός είναι η κλήση της συνάρτησης calculator, χρησιμοποιώντας το await. Το await υποδηλώνει ότι θα περιμένουμε μέχρι η calculator να μας επιστρέψει ένα promise, resolve ή reject. Στη συνέχεια, καλούμε τη συνάρτηση fetchCal και είμαστε ξανά σε θέση να καλέσουμε το then, το οποίο παίρνει σαν παράμετρο μια συνάρτηση και θα εκτελεστεί σε περίπτωση που έχει κληθεί το resolve. Στην περίπτωση που υπήρξε error θα έπρεπε να υπάρχει ένα try-catch block μέσα στη συνάρτηση fetchCal, το οποίο θα εκτύπωνε το error αν το promise ήταν reject [14].

2.6 JWT

Το JWT είναι ένα ανοιχτό πρότυπο (RFC 7519), το οποίο διασφαλίζει την ασφαλή μετάδοση πληροφοριών μέσω JSON αντικειμένων. Οι πληροφορίες που μεταφέρονται είναι ασφαλείς, διότι είναι ψηφιακά υπογεγραμμένες. Η υπογραφή μπορεί να γίνει είτε χρησιμοποιώντας κρυφό κλειδί (με τον αλγόριθμο HMAC), είτε με το συνδυασμό ενός ζευγαριού private και public κλειδιών χρησιμοποιώντας RSA ή ECDSA.

Το JWT αποτελείται από 3 κύρια μέρη: header, payload και signature που χωρίζονται μεταξύ τους με μία τελεία.

- Ο header αποτελείται από 2 μέρη. Τον τύπο του token, που είναι JWT, και τον αλγόριθμο υπογραφής που χρησιμοποιείται.
- Το payload περιέχει τα claims, που είναι πληροφορίες οι οποίες αφορούν το χρήστη και επιπλέον δεδομένα. Υπάρχουν 3 κατηγορίες claims: register, public και private.
- Το signature δημιουργείται από την συμβολή των κρυπτογραφημένων header, του αλγορίθμου που αναφέρεται στον header, του κρυπτογραφημένου payload και ενός secret.

Όλα τα παραπάνω δημιουργούν ένα JWT, το οποίο αποτελείται από 3 Base64-URL strings που χωρίζονται από τελείες. Μπορούν να μεταβιβαστούν με ευκολία σε HTML και HTTP περιβάλλοντα, ενώ ταυτόχρονα είναι πιο σταθερά και ασφαλή από XML πρότυπα [15].

Figure 11: JWT with red header, purple payload and cyan signature divided by dots [15].

Κεφάλαιο 3 Database

Μια βάση δεδομένων (Database) περιέχει πληροφορίες με σκοπό την εύκολη πρόσβαση, διαχείριση και ενημέρωση τους. Ουσιαστικά αποθηκεύει σύνολα δεδομένων ή αρχείων, τα οποία περιέχουν πληροφορίες ανάλογα με το τι πραγματεύεται μια εφαρμογή. Οι πληροφορίες που περιέχονται σε μια βάση δεδομένων, οργανώνονται κατάλληλα έτσι ώστε να είναι διαθέσιμες για σκοπούς διαχείρισης, παρατήρησης και ανάλυσης.

Οι πρώτες databases δημιουργήθηκαν τη δεκαετία του 1960, οι οποίες ήταν διαδικτυακά μοντέλα όπου κάθε record σχετίζεται με πολλά κύρια και δευτερεύοντα records. Οι σχεσιακές databases αναπτύχθηκαν στη δεκαετία του 1970 και έπειτα από μια δεκαετία ακολούθησαν οι Object-oriented databases. Σήμερα οι πιο διαδεδομένοι τύποι είναι SQL, NoSQL και Cloud databases [16].

3.1 SQL vs NoSQL

Για την ανάπτυξη μιας εφαρμογής απαιτείται η επιλογή του κατάλληλου τύπου database, ανάμεσα σε σχεσιακές (SQL) και μη-σχεσιακές (NoSQL). Ανάλογα με τις ανάγκες της κάθε εφαρμογής επιλέγεται και η αντίστοιχη βάση δεδομένων, μιας και έχουν σημαντικές διαφορές μεταξύ τους.

Αυτές είναι:

- *Αρχιτεκτονική Database:* Η σημαντικότερη διαφορά τους είναι ότι οι SQL είναι σχεσιακές ενώ οι NoSQL μη-σχεσιακές.
- *Σχήμα Database:* Οι SQL βάσεις έχουν προκαθορισμένο σχήμα για να ορίζουν και να διαχειρίζονται δεδομένα. Είναι κατάλληλες για περίπλοκα queries, αλλά μπορεί να είναι και πολύ περιοριστικές. Όλα τα δεδομένα πρέπει να ακολουθούν την ίδια δομή. Αυτό προϋποθέτει αρκετά καλή προετοιμασία, γιατί σε περίπτωση που πρέπει να αλλάξει η δομή, θα είναι δύσκολο και περίπλοκο να προσαρμοστεί όλο το σύστημα. Από την άλλη, οι NoSQL έχουν δυναμικά σχήματα για μη-δομημένα δεδομένα, τα οποία μπορούν να αποθηκευτούν με διάφορους τρόπους. Αυτό σημαίνει ότι μπορούμε να δημιουργήσουμε

documents χωρίς να καθορίσουμε πρώτα το σχήμα τους. Το κάθε document μπορεί να έχει τη δική του δομή και μπορούμε να προσθέτουμε πεδία στην πορεία.

- *Database Scaling:* Οι SQL βάσεις είναι κάθετα επεκτάσιμες. Για να αυξηθεί το μέγεθος ενός server πρέπει να προστεθεί επιπλέον CPU, RAM ή SSD χωρητικότητα. Οι NoSQL όμως είναι οριζόντια επεκτάσιμες. Δηλαδή, σε περίπτωση μέγιστης επισκεψιμότητας προστίθενται περισσότεροι server στη βάση, ώστε να διαμοιραστούν τα δεδομένα. Η οριζόντια επέκταση έχει μεγαλύτερη χωρητικότητα από την κάθετη, κάνοντας έτσι τις NoSQL databases προτιμότερες για μεγάλα και συχνά μεταβαλλόμενα data sets [17].
- *Δομή δεδομένων:* Οι SQL databases είναι table-based, ενώ οι NoSQL είναι δομημένες σε JSON documents, key-value pairs, πίνακες με γραμμές και δυναμικές στήλες ή σε γραφήματα [18].

Συνοψίζοντας οι SQL είναι κατάλληλες για περιπτώσεις όπου τα δεδομένα είναι περιορισμένα σε όγκο, μοντελοποιημένα σε πίνακα και κυρίως για συστήματα που ως πρώτο μέλημα έχουν την συνέπεια. Ενώ οι NoSQL είναι κατάλληλες για ιεραρχημένα δεδομένα, που είναι μεγάλα σε όγκο και συχνά μεταβαλλόμενα και χρησιμοποιούνται σε συστήματα που αναπτύσσονται ραγδαία χωρίς data schema.

3.2 MongoDB

Η MongoDB είναι μια βάση δεδομένων η οποία είναι ανοιχτού κώδικα, cross-platform, document-oriented και αυτή τη στιγμή θεωρείται η δημοφιλέστερη NoSQL βάση. Βασίζεται σε συλλογές από documents, τα οποία αποτελούνται από key-value χαρακτηριστικά. Συγκριτικά με τις σχεσιακές βάσεις, κάθε document μπορεί να θεωρηθεί ως μια γραμμή ενός πίνακα και κάθε key-value ως μία στήλη, η τιμή του οποίου αντιστοιχεί στο συγκεκριμένο κελί του πίνακα. Η βασική διαφορά είναι ότι κάθε γραμμή (document) δεν περιορίζεται από συγκεκριμένο αριθμό στηλών (key-value), δηλαδή δύο documents μπορεί να αποτελούνται και από διαφορετικά πεδία. Τα file-schemas είναι τύπου JSON, που σημαίνει ότι τα δεδομένα μπορούν να αποθηκευτούν σε documents ή collections και μπορούν να είναι strings, numbers, float ακόμα και πίνακες ή αντικείμενα [19].

Όσον αφορά τις προδιαγραφές υποστηρίζεται από πολλά λειτουργικά συστήματα (Linux, MAC OS X, Solaris, Windows) και από πολλές γλώσσες προγραμματισμού (JS, PHP, Java, C, C++, Python, κ.λπ.). Είναι ειδικά σχεδιασμένη για μοντέρνα project ανάπτυξης εφαρμογών, δίνοντας κυρίως τη δυνατότητα στους developers να χρησιμοποιούν δεδομένα όπου και αν βρίσκονται (database, device, search engine, data lake) [20].

3.3 Mongoose

Η Mongoose είναι μια query γλώσσα, η οποία χρησιμοποιείται για την επικοινωνία μεταξύ του server και της βάσης δεδομένων. Η MongoDB όπως και το Node.js χρησιμοποιούν το ίδιο ODM (Object Data Modeling), που στην περίπτωση αυτή είναι η Mongoose. Αρμοδιότητές της είναι η διαχείριση συσχετίσεων δεδομένων και η δημιουργία σχημάτων (schemas) για την μοντελοποίηση των δεδομένων, έτσι ώστε να συνδέονται με τη βάση.

Βασικές λειτουργίες της Mongoose είναι:

- Δημιουργία εγγραφών.
- Αναζήτηση και ανάκτηση εγγραφών.
- Ενημέρωση εγγραφών.
- Διαγραφή εγγραφών.

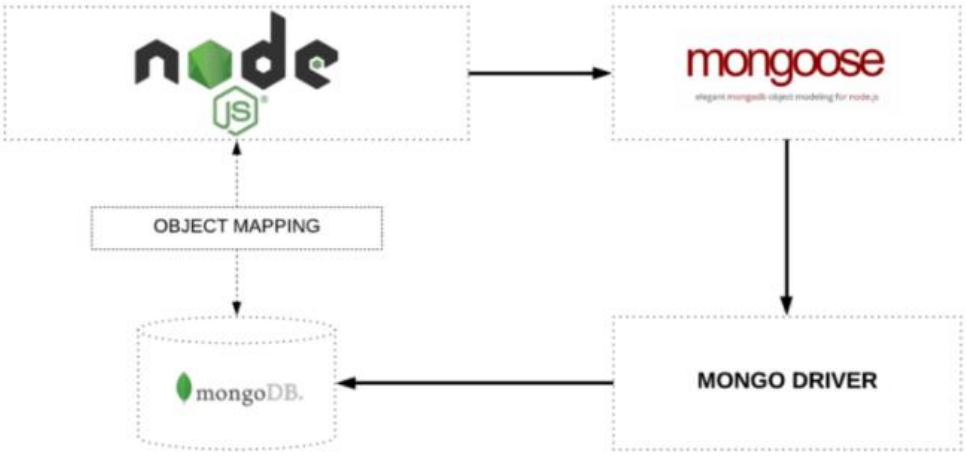

Figure 12: The relationship between Node.js, Mongoose, MongoDB [21].

Κεφάλαιο 4 Front-End

Οτιδήποτε βλέπει ένας χρήστης σε ένα website αποτελεί μέρος του Front-end. Οι 3 βασικές γλώσσες που πρέπει να γνωρίζει ένας front-end developer είναι η HTML, η CSS και η JS. Από κει και πέρα για τη καλύτερη εμβάθυνση στον τομέα του front-end development, απαιτείται η χρήση frameworks και βιβλιοθηκών. Για παράδειγμα όσον αφορά το CSS, υπάρχουν εργαλεία που επιταχύνουν την διαδικασία προσθέτοντας έτοιμες κλάσεις. Τέτοια εργαλεία είναι τα frameworks όπως το Tailwind CSS και το Bootstrap. Αντίστοιχα και για την JavaScript υπάρχουν βοηθητικά frameworks και βιβλιοθήκες όπως το Angular, Vue και React [22]. Τα frameworks ή οι βιβλιοθήκες παίρνουν τα δεδομένα από τον server με σκοπό να τα αναπαραστήσουν σε ένα διαδραστικό user interface.

4.1 React

Το React είναι μια front-end βιβλιοθήκη της JavaScript, η οποία χρησιμοποιείται για την δημιουργία του user interface. Δημιουργήθηκε από software-engineers του Facebook και αργότερα έγινε ανοιχτού κώδικα. Χρησιμοποιεί JavaScript γλώσσα για την κατασκευή επαναλαμβανόμενων DOM elements [24]. Όπως βλέπουμε στο παρακάτω διάγραμμα πρόκειται για το πιο δημοφιλές framework με βάση τα στατιστικά του 2021.

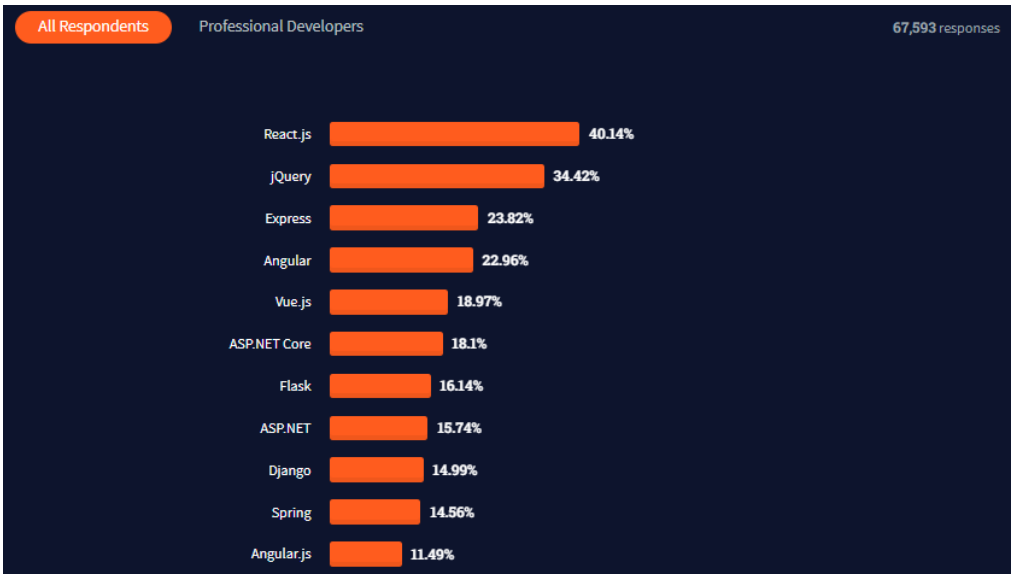

Figure 13: Most used web frameworks (2021) [2].

Αρχικά για να δημιουργήσουμε μια εφαρμογή React χρησιμοποιούμε την εντολή “create-react-app”, που απαιτεί το Node.js. Αφού το εγκαταστήσουμε, με την εντολή “npm start”, τρέχουμε την εφαρμογή η οποία εμφανίζεται στη διεύθυνση localhost:3000 του browser. Το React δημιουργεί ένα Virtual Dom στην μνήμη, έτσι ώστε να μην κάνει απευθείας αλλαγές στο DOM του browser. Δηλαδή, βρίσκει ποιες αλλαγές έχουν γίνει, και αλλάζει ότι είναι απαραίτητο.

Ο βασικός σκοπός του React είναι να εμφανίσει HTML στον browser (rendering), χρησιμοποιώντας την συνάρτηση “ReactDOM.render()”. Όπως βλέπουμε στην επόμενη εικόνα το rendering γίνεται στο body του αρχικού index.html αρχείου και πιο συγκεκριμένα σε ένα div με id = “root”.

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>React App</title>
  </head>
  <body>
    <div id="root"></div>
    <script src="../src/index.js" type="text/jsx"></script>
  </body>
</html>
```

Figure 14: Index.html

Στην εικόνα 15, βλέπουμε ότι το “ReactDOM.render()” παίρνει 2 ορίσματα. Το δεύτερο όρισμα όπως προαναφέραμε αφορά το που θα γίνει το rendering. Ενώ, το πρώτο όρισμα είναι ένα component που περιέχει τον κώδικα που θα γίνει rendering.

```jsx
import React from "react";
import ReactDOM from "react-dom";
import App from "./App"

ReactDOM.render(<App />, document.getElementById("root"));
```

Figure 15: ReactDOM method to render the React element to the DOM.

Τα components (<App />) είναι αρχεία τύπου JSX (JavaScript XML), που μας επιτρέπουν εύκολα να γράφουμε HTML κώδικα ενδιάμεσα σε JS. Το μεγάλο πλεονέκτημα των components είναι ότι τα χρησιμοποιούμε σε πολλά σημεία του κώδικα, αποφεύγοντας έτσι επαναλαμβανόμενα κομμάτια κώδικα.

```jsx
import React from "react";
import NameDetails from "./NameDetails"
import contacts from "./contacts"

function App() {
    return (
        <div>
            <h1>Hello </h1>
            {
                contacts.map(contact =>{
                    return(
                        <NameDetails
                            key={contact.id}
                            firstName={contact.firstName}
                            lastName={contact.lastName}/>);
                })
            }
        </div>
    );
}
export default App;
```

Figure 16: App.jsx returns HTML and contains a mapping method which returns another component.

Στην παραπάνω εικόνα αναπαρίσταται μια λίστα δεδομένων, όπου για κάθε δεδομένο καλείται το component NameDetails που παίρνει ως ορίσματα 2 μεταβλητές, firstName και lastName. Το "key" δεν αποτελεί μεταβλητή, αλλά ένα γνώρισμα που βοηθάει το React να αναγνωρίζει την ταυτότητα κάθε στοιχείου. Για αυτό το λόγο κάθε "key" πρέπει να είναι μοναδικό.

```jsx
import React from "react";

function NameDetails(props) {
    return (
        <div>
            <h4>{props.firstName} {props.lastName}</h4>
        </div>
    );
}

export default NameDetails;
```

Figure 17: NameDetails.jsx

Τα React Props είναι σαν function arguments στην JS ή σαν attributes στην HTML. Για παράδειγμα, το component NameDetails λαμβάνει τα ορίσματα σαν "props" object, που έχει ως properties τις μεταβλητές που έχουν οριστεί όταν καλούμε το συγκεκριμένο component [23].

4.1.1 React Hooks

Τα React Hooks επιτρέπουν στα function components να έχουν πρόσβαση στη κατάσταση και σε άλλες δυνατότητες του React. Δύο από τα πιο σημαντικά είναι το *useState* και το *useEffect*. Και τα 2 ανήκουν στην βιβλιοθήκη του React.

Το React useState Hook μας επιτρέπει να ελέγχουμε ανά πάσα στιγμή την κατάσταση ενός function component. Πρώτα, αρχικοποιούμε την κατάστασή του καλώντας μέσα σε αυτό, το useState. Το useState επιστρέφει 2 τιμές, την τωρινή κατάσταση και μια συνάρτηση η οποία την ενημερώνει.

```
import React, { useState } from "react";

function Text() {
    const [text, setText] =  useState("");

    return (
        <div>
            <input type="text" placeholder="Enter text" onChange={e=>setText(e.target.value)}/>
            <h2>You texted: {text}</h2>
        </div>
    );
}

export default Text;
```

Figure 18: UseState example.

Στο παραπάνω παράδειγμα αρχικοποιούμε το text σε κενό string. Όταν κάποιος αλλάζει το κείμενο του input, καλείται η συνάρτηση setText, η οποία απευθείας αλλάζει το περιεχόμενο του text.

Το React useEffect Hook μας επιτρέπει να πραγματοποιούμε side effects σε components. Κάποια χαρακτηριστικά side effects είναι η απευθείας ενημέρωση του DOM, η μεταφόρτωση δεδομένων από το back-end κ.λπ. Παίρνει 2 ορίσματα όπου το πρώτο είναι μια συνάρτηση. Το δεύτερο όρισμα είναι προαιρετικό και μας δείχνει πότε θα εκτελείται το useEffect, δηλαδή η συνάρτηση του πρώτου ορίσματος. Αν δεν υπάρχει δεύτερο όρισμα, η συνάρτηση του useEffect εκτελείται κάθε φορά που γίνεται rendering. Αν είναι ένας κενός πίνακας, εκτελείται μόνο την πρώτη φορά που γίνεται rendering, ενώ αν ο πίνακας περιέχει τιμές, εκτελείται την πρώτη φορά και κάθε φορά που αλλάζει το περιεχόμενο των τιμών που έχει ο πίνακας [26].

4.2 Axios

Το Axios είναι μια HTTP client library που μας επιτρέπει να κάνουμε requests σε ένα endpoint είτε σε ένα εξωτερικό API, είτε στο back-end του server. Είναι promise-based, το οποίο μας δίνει την δυνατότητα να χρησιμοποιήσουμε το async/await της JavaScript, όπως αναλύσαμε σε προηγούμενο κεφάλαιο.

Κάποια από τα πλεονεκτήματα του Axios είναι ότι έχει πολύ καλές default προδιαγραφές για να διαχειρίζεται δεδομένα τύπου JSON. Έχει ονόματα συναρτήσεων, τα οποία ταιριάζουν με όλες τις HTTP μεθόδους. Παραδείγματος χάριν, για να κάνει ένα GET request χρησιμοποιεί την μέθοδο ".get()". Επίσης έχει άριστη διαχείριση των errors και μπορεί να χρησιμοποιηθεί τόσο στο server-side όσο και στο client-side.

```
import axios from 'axios'
import API from './config'

axios.defaults.baseURL = API;
axios.defaults.headers.post['Content-Type'] = 'application/json';
```

Figure 19: Set default global headers and base URL.

```
const [user, setUser] = useState({});

useEffect(() => {
  axios.get('user/auth')
    .then(res=>{
      setUser(res.data);
    }).catch(err=>{
      console.log(err);
    });
}, []);
```

Figure 20: Example of axios get request.

Στην εικόνα 19 αρχικοποιούμε το base URL, ώστε να μην χρειάζεται κάθε φορά που κάνουμε ένα request να επισημαίνουμε ολόκληρο το endpoint του API. Στην επόμενη εικόνα γίνεται ένα GET request στο συγκεκριμένο endpoint (baseURL/user/auth). Στην περίπτωση που το request

είναι επιτυχημένο (status 200), εκτελείται το ".then()" που μας επιστρέφει τα response data. Στην περίπτωση που το request αποτύχει (status 400-500), θα τρέξει το catch block.

```
const [email, setEmail] = useState("");
const [password, setPassword] = useState("");

const handleSubmit = (e)=>{
  e.preventDefault();

  const body = {email, password}

  axios.post(`/login`, body)
       .then(res=> {
         console.log(res.data);
       })
       .catch(err=>{
           console.log(err);
       });
}
```

Figure 31: Example of axios post request.

Στην παραπάνω εικόνα γίνεται ένα POST request. Σε αντίθεση με τα GET που έχουν ως παράμετρο μόνο το endpoint, τα POST request έχουν και μια δεύτερη παράμετρο που περιέχει ένα αντικείμενο με τα properties που απαιτούνται για το συγκεκριμένο request. Στην περίπτωσή μας γίνεται ένα POST request στο endpoint "baseURL/login" και στέλνεται ένα αντικείμενο που περιέχει τις πληροφορίες για το email και το password [27].

Με παρόμοιο τρόπο λειτουργούν και τα DELETE, PUT, PATCH requests.

Κεφάλαιο 5 Η Εφαρμογή μας

Πρόκειται για μια εφαρμογή όπου μπορεί κάποιος να συνδεθεί είτε σαν χρήστης είτε σαν συνεργάτης. Από την πλευρά του ο χρήστης μπορεί να δημιουργήσει το προφίλ του, να κάνει φίλους, να αναζητήσει προϊόντα και να τα προσθέσει στη λίστα του (Wishlist) καθώς και να προσθέσει χρήματα στα δώρα των φίλων του. Από την άλλη, ο συνεργάτης (συνεργαζόμενα καταστήματα) μπορεί να ανεβάσει ένα αρχείο JSON με την λίστα των προϊόντων που επιθυμεί να διαθέσει προς πώληση μέσω της εφαρμογής μας.

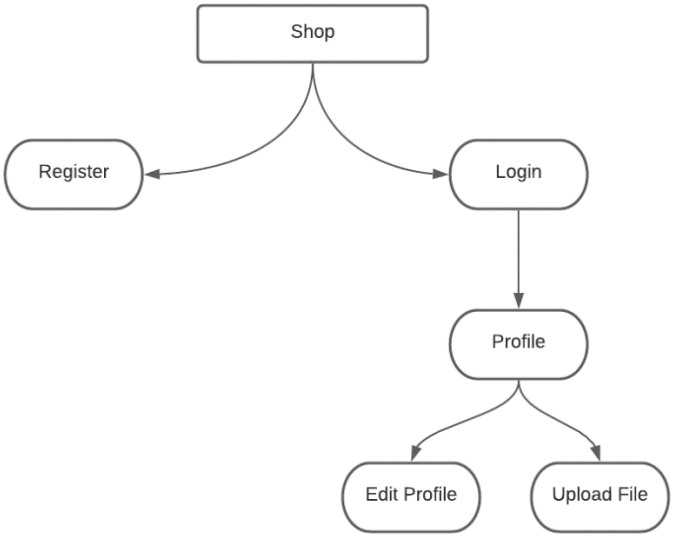

Figure 22: The Shop-side architecture of project.

Πιο συγκεκριμένα ο χρήστης μπορεί να κάνει sign up και έπειτα αφού τα στοιχεία του είναι στη βάση δεδομένων, να συνδέεται με Login. Στην συνέχεια βρίσκεται στην αρχική σελίδα (Home), όπου μπορεί να αναζητήσει άλλους χρήστες και να τους στείλει αιτήματα φιλίας. Στην αρχική σελίδα βλέπει τα δώρα τα οποία έχουν οι φίλοι του στις αντίστοιχες Wishlist τους, τα οποία μπορούν να ταξινομηθούν είτε με χρονολογική σειρά είτε με βάση τα χρήματα που έχουν συγκεντρωθεί για το κάθε δώρο. Σε κάθε δώρο ο χρήστης μπορεί είτε να προσθέσει χρήματα αν δεν το έχει ήδη κάνει είτε να επεξεργαστεί το ποσό που έχει ήδη προσθέσει. Εκτός από το home ο χρήστης μπορεί να επισκεφθεί και την σελίδα Gifts, όπου υπάρχουν τα προϊόντα που είναι διαθέσιμα από τα καταστήματα που συνεργάζονται με την εφαρμογή μας. Σε αυτή τη σελίδα ο

χρήστης μπορεί να προσθέσει προϊόντα στην Wishlist του μέσω της keyword αναζήτησης ή των κατηγοριών που χωρίζονται τα διαθέσιμα προϊόντα. Στο προφίλ εκτός από την Wishlist του και την επεξεργασία του προφίλ του, ο χρήστης μπορεί να δει τους φίλους του, τα δώρα που έχει λάβει και ποιοι συνέβαλαν στην αγορά του κάθε δώρου χωρίς να βλέπει το ποσό που έχει προσθέσει ο καθένας (My Gifts), τα δώρα στα οποία έχει συνεισφέρει με το ποσό που έχει συνεισφέρει και έχουν λάβει οι φίλοι του (Donated to) καθώς και τα δώρα που έχει συνεισφέρει αλλά δεν έχουν ολοκληρωθεί (Pending). Όσον αφορά την Wishlist του δεν μπορεί να δει το ποσό που έχει προστεθεί στο κάθε δώρο του. Μπορεί να προσθέσει όμως μια περιγραφή, ώστε να πείσει τους φίλους του να συνδράμουν στην αγορά του. Στα προφίλ των φίλων του μπορεί να δει μόνο την Wishlist και τη λίστα φίλων του, ενώ σε προφίλ που δεν είναι φίλου του μπορεί να στείλει απλά αίτημα φιλίας. Τέλος υπάρχουν και οι ειδοποιήσεις που εμφανίζονται αν κάποιος δέχτηκε το αίτημα φιλίας του, αν κάποιος έχει διαγράψει από την λίστα του δώρο που έχει συνεισφέρει (επιστροφή του πόσου), αν διαγραφεί ένα προϊόν από τη λίστα του επειδή δεν είναι πλέον διαθέσιμο από το κατάστημα και αν έχει γίνει κάποια αλλαγή στην τιμή ενός προϊόντος που έχει προσθέσει ο χρήστης στην λίστα του.

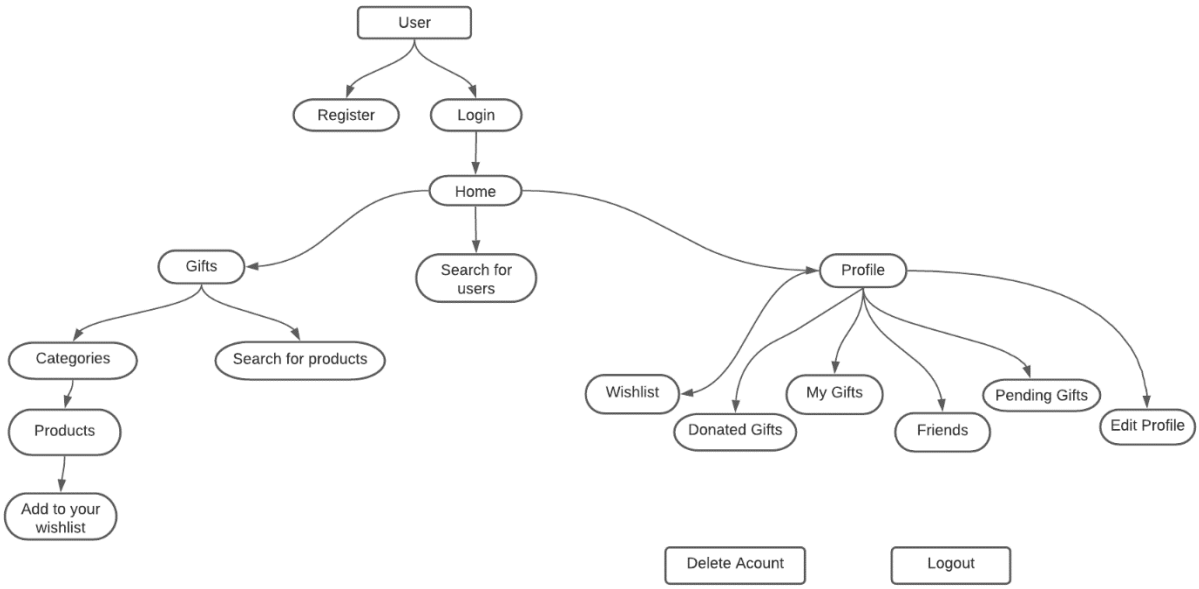

Figure 23: The User-side architecture of project.

5.1 Τεχνολογίες και εργαλεία

Για την ανάπτυξη της εφαρμογής μας επιλέξαμε μια σειρά τεχνολογιών η οποία είναι γνωστή ως MERN stack. Η λέξη MERN (MongoDB, Express, React, Node.js) αναφέρεται στις τεχνολογίες που χρησιμοποιήσαμε και αναλύσαμε στα παραπάνω κεφάλαια. Με λίγα λόγια, τα δεδομένα μας αποθηκεύονται στην MongoDB βάση δεδομένων, η οποία θεωρείται τα τελευταία χρόνια η πιο διαδεδομένη και ευρέως χρησιμοποιούμενη NoSQL βάση δεδομένων. Το React χρησιμοποιείται για διαδικασίες του front-end, δηλαδή για τη δημιουργία ενός user-friendly interface. Τα Express.js και Node.js αντίστοιχα χρησιμοποιούνται για συγκεκριμένες λειτουργίες και αλληλεπιδράσεις του back-end.

Για την σύνθεσή της, χρειαζόμαστε και έναν browser, έναν code editor, το Postman και τη MongoDB Atlas. Σαν code editor χρησιμοποιήσαμε Visual Studio Code.

Με τη βοήθεια του Postman επαληθεύσαμε την λειτουργία του API που δημιουργήσαμε. Με λίγα λόγια, αν θέλουμε να επαληθεύσουμε ένα request, αρχικά επιλέγουμε τον σωστό τύπο request (GET, POST, κ.λπ.) που θέλουμε να στείλουμε, προσθέτουμε το κατάλληλο URL, ορίζουμε τους κατάλληλους headers και όποια πρόσθετα δεδομένα είναι απαραίτητα για το request. Αφού σταλθεί το request, το response αποτελείται από πληροφορίες για το body, τους headers, το status code, το χρόνο και το μέγεθος του [28].

Η MongoDB Atlas μας δίνει τη δυνατότητα δημιουργίας της βάσης μας στο internet με την μορφή cloud, έτσι ώστε να είναι προσβάσιμη από οποιοδήποτε υπολογιστή. Εξυπηρετεί τον ίδιο σκοπό με την MongoDB, δημιουργώντας απλά έναν λογαριασμό και έπειτα ένα καινούριο project (cluster), το οποίο συνδέουμε με τον server μας.

5.2 Back-end

5.2.1 Setup

Αρχικά εγκαθιστούμε το Node.js. Όποτε θέλουμε να αναπτύξουμε οποιαδήποτε εφαρμογή με τη βοήθεια του Node.js αρχικοποιούμε το project μας με την βοήθεια του Npm και πιο συγκεκριμένα με την εντολή *"npm init",* η οποία προσθέτει μερικές χρήσιμες πληροφορίες για το project μας, οι οποίες αναγράφονται στο αρχείο package.json, όπως βλέπουμε στην παρακάτω εικόνα. Έτσι από το σημείο αυτό οποιαδήποτε καινούρια βιβλιοθήκη προστίθεται στο project μας θα αναγράφεται στο αρχείο αυτό, όπως για παράδειγμα γίνεται με το Express.js το οποίο εισάγουμε μέσω του Terminal με την εντολή " npm install express --save".

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "Backend",
  "main": "server.js",
  ▷ Debug
  "scripts": {
    "start": "nodemon server.js"
  },
  "author": "Bantavanis-Memou",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.0.1",
    "cors": "^2.8.5",
    "crypto": "^1.0.1",
    "dotenv": "^10.0.0",
    "express": "^4.17.1",
    "fs": "0.0.1-security",
    "jsonwebtoken": "^8.5.1",
    "moment": "^2.29.1",
    "mongoose": "^5.13.7",
    "multer": "^1.4.3",
    "multer-gridfs-storage": "^5.0.2",
    "path": "^0.12.7"
  }
}
```

Figure 24: package.json file.

Όπως προαναφέραμε τα δεδομένα αποθηκεύονται στην MongoDB, οπότε είναι απαραίτητο να εδραιώσουμε την σύνδεση μεταξύ της βάσης δεδομένων (MongoDB) και του server μας. Το εργαλείο που μας βοηθάει σε αυτή τη περίπτωση είναι το Mongoose, το οποίο πέρα από την εξασφάλιση της σύνδεσης μεταξύ της βάσης και του server, περιέχει όλες τις απαραίτητες μεθόδους για την διαχείριση της MongoDB. Το παρακάτω σχήμα δείχνει τη σύνδεση της βάσης με τον server στην εφαρμογή μας.

```
require('dotenv').config();
const mongoose = require("mongoose");

mongoose.set('useNewUrlParser', true);
mongoose.set('useFindAndModify', false);
mongoose.set('useCreateIndex', true);
mongoose.set('useUnifiedTopology', true);

const connectToDB = async () =>{
    try {
        await mongoose.connect(process.env.MONGO_URL)
        console.log("Connected to DB");
    } catch (error) {
        console.log("Error: connectToDB: " + error);
        process.exit(1);
    }
}
module.exports = connectToDB;
```

Figure 25: Mongoose used to connect.

Τέλος, ιδιαίτερα σημαντική είναι η σωστή οργάνωση του Project, δηλαδή ο καταμερισμός του κάθε αρχείου κώδικα σε σωστά ομαδοποιημένους φακέλους. Τέτοιοι φάκελοι είναι οι Handlers, Models και Routes. Στα Models έχουμε αρχεία κώδικα με συναρτήσεις που περιέχουν σχήματα (schemas) ενώ τα Routes περιέχουν διάφορα URL που χρησιμεύουν σε συναρτήσεις που περιέχονται στους Handlers.

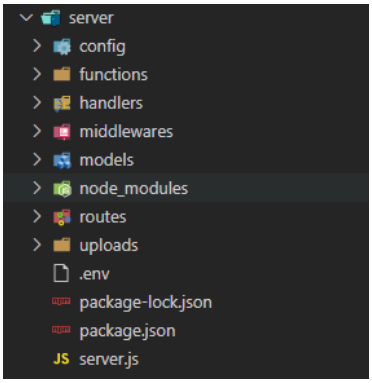

Figure 26: Grouping files.

5.2.2 Authentication Handling

Authentication είναι η διαδικασία κατά την οποία γίνεται η ταυτοποίηση ενός χρήστη για το αν έχει πρόσβαση στην εφαρμογή, όταν αυτός προσπαθεί να συνδεθεί. Υπάρχουν διάφοροι τρόποι που επιτυγχάνεται αυτή η διαδικασία. Στην εφαρμογή μας, αυτή γίνεται μέσω του login form με τη βοήθεια του JWT. Αρχικά, ο χρήστης πρέπει να δημιουργήσει έναν λογαριασμό προσθέτοντας κάποια βασικά στοιχεία (email, username, password), τα οποία αποθηκεύονται στη βάση και συγκεκριμένα ο κωδικός κρυπτογραφείται με τον τρόπο που αναφέραμε στο κεφάλαιο 2.6. Κάθε φορά που ο χρήστης επιχειρεί να κάνει Login, αν τα στοιχεία του είναι σωστά, του επιστρέφεται ένα JWT το οποίο αποθηκεύεται στο Local Storage. Στην συνέχεια, κάθε φορά που ο χρήστης επιχειρεί να επισκεφθεί ένα Private/Protected Route, πρέπει να έχει έναν Authorization Header με ένα JWT, το οποίο περιέχει τα στοιχεία του χρήστη που έκανε το request και ελέγχει αν πρέπει να του δοθεί πρόσβαση. Αυτό γίνεται μέσω ενός Middleware (fig.27), το οποίο παρεμβάλλεται του request και ελέγχει αν ισχύουν οι προϋποθέσεις έτσι ώστε να υπάρξει response από τον server. Αν δεν υπάρχει token ή το token υπάρχει αλλά δεν είναι έγκυρο, δεν δίνεται πρόσβαση στον χρήστη και επιστρέφεται ένα error message. Από την άλλη, αν το token είναι έγκυρο αποκωδικοποιείται και αποθηκεύεται στη μεταβλητή “req.tokenUser” που μπορεί να χρησιμοποιηθεί σε οποιονδήποτε Private/Protected Route. Το token παύει να ισχύει όταν ο χρήστης κάνει Logout.

```
require('dotenv').config();
const {verify} = require("jsonwebtoken");

exports.auth =  async(req, res, next)=>{
    try {
        let token;
        if(req.headers.authorization && req.headers.authorization.startsWith('Bearer ')) token = req.headers.authorization.split('Bearer ')[1];
        else return res.status(401).json({msg: 'No token, authorization denied'});

        verify(token, process.env.JWT_SECRET, (error, token)=>{
            if(error) throw error;
            req.tokenUser = token;
            next();
        });
    } catch (err) {
        res.status(400).json({msg: "Token is not valid"});
    }
}
```

Figure 27: Validation of JWT.

5.2.3 Models

Τα Models καθορίζουν την δομή των αντικειμένων που αποθηκεύουμε στη βάση δεδομένων. Η δική μας εφαρμογή αποτελείται από 8 Models (User, Product, Gift, Transaction, FriendRequest, Notification, Order, Shop), τα οποία αποθηκεύονται σε collections που περιέχουν τα αντίστοιχα documents.

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const giftSchema = new mongoose.Schema(
    {
        recipient: {
            type: Schema.Types.ObjectId,
            required: true,
            ref: 'User'
        },
        product:{
            type: Schema.Types.ObjectId,
            required: true,
            ref: 'Product'
        },
        description: {
            type: String,
            default: "",
            max: 150
        },
        transactions: {
            type: [Schema.Types.ObjectId],
            ref: 'Transactions'
        },
        totalAmount:{type: Number, default: 0}
    },
    { timestamps: true }
);

module.exports = Gift =  mongoose.model("Gift", giftSchema);
```

Figure 28: Gift model.

Στη παραπάνω εικόνα έχουμε ένα παράδειγμα ενός mongoose schema, το οποίο ονομάζουμε “giftSchema” και αναπαριστά τη δομή ενός document του collection των gifts. Όπως βλέπουμε, περιέχει έναν recipient, ο οποίος είναι τύπου ObjectId και αναφέρεται σε ένα document του collection users (userSchema), που περιέχει τις πληροφορίες του κάθε χρήστη (email, username, password, bio, profilePic, κ.λπ.). Ένα product τύπου ObjectId που αναφέρεται σε ένα document του collection products (productSchema), που περιέχει τις πληροφορίες του προϊόντος (productName, productLink, price, imageLink, countInStock, shop, κ.λπ.). Το κάθε προϊόν περιέχει ένα property shop, το οποίο αναφέρεται στο κατάστημα στο οποίο ανήκει (ObjectId του shopSchema), και το οποίο με τη σειρά του περιέχει τα στοιχεία του καταστήματος (brand, email, phone, shopPic, κ.λπ.). Ο πίνακας transactions περιέχει τα id των transactions document

που αφορούν αυτό το δώρο, το description που είναι η περιγραφή που βάζει ο χρήστης για το δώρο του και το totalAmount το συνολικό ποσό που έχει μαζευτεί για το συγκεκριμένο δώρο (είναι το άθροισμα των ποσών που περιέχει κάθε transaction).

Το transactionSchema έχει τρία properties, το gift και το donator που είναι τύπου ObjectId και έχουν reference τα giftSchema και userSchema αντίστοιχα, και το amount που αφορά το ποσό που έχει προσθέσει ο donator στο συγκεκριμένο δώρο.

Όταν το totalAmount ενός δώρου γίνει ίσο με το price του product στο οποίο αναφέρεται, ολοκληρώνεται η παραγγελία και το δώρο διαγράφεται από το collection των gifts. Στη συνέχεια εισέρχεται στο collection των orders (orderSchema), όπου περιέχονται οι πληροφορίες για τα δώρα που έχουν ολοκληρωθεί, όπως για παράδειγμα ο ιδιοκτήτης του δώρου και τα μέλη που συνέβαλαν στην αγορά του.

Το notificationSchema έχει τρία properties, τον παραλήπτη (ObjectId user), το κείμενο της ειδοποίησης και αν έχει διαβαστεί. Τέλος το friendRequestSchema, έχει δύο properties τύπου ObjectId, που αφορούν τον χρήστη που έστειλε το αίτημα και τον παραλήπτη καθώς και μία για το αν το αίτημα φιλίας έχει διαβαστεί. Όταν ένα αίτημα γίνει αποδεκτό από τον παραλήπτη, εισέρχεται στο userSchema του παραλήπτη και του αποστολέα στο property friendList, που είναι ένας πίνακας με ObjectId των user. Αν απορριφθεί ή ακυρωθεί, διαγράφεται από το collection των friendRequests.

Στην παρακάτω εικόνα βλέπουμε ένα document που δημιουργήθηκε και αποθηκεύτηκε στο collection gifts σε μορφή JSON.

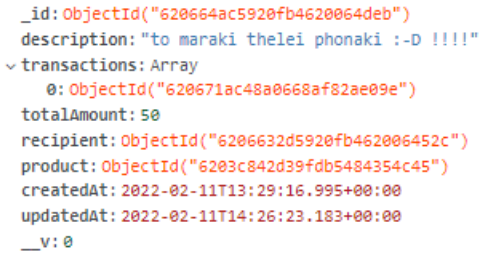

Figure 29: Example of Gift document in JSON.

5.2.4 Routes

Τα Routes αφορούν όλη τη λογική της λειτουργίας του server-side της εφαρμογής μας. Τα routes τα οποία είναι private είναι αυτά στα οποία ο χρήστης έχει πρόσβαση, αφού συνδεθεί μετά το Login. Εκτός από τα “/register”, ”/login”, ”/image/:id”, ”/registerAdmin”, ”/loginAdmin”, ”/imageShop/:id”, τα υπόλοιπα routes είναι private, δηλαδή απαιτείται έγκυρο JWT.

User Routes:

Route	**Purpose**
/register	POST: Αποθήκευση email, username, password στη βάση και αποστολή token.
/login	POST: Έλεγχος email και password και αποστολή token.
/user/auth	GET: Επιστρέφει τα στοιχεία του συνδεδεμένου χρήστη.
/user/edit	POST: Ενημέρωση των στοιχείων που προσθέτει ο χρήστης.
/user/:username	GET: Επιστρέφει τα στοιχεία του χρήστη με το συγκεκριμένο username.
/user/friends	POST: Επιστρέφει τους φίλους του χρήστη που το id του στέλνεται ως body.
/user/notifications	GET: Επιστρέφει τις ειδοποιήσεις του συνδεδεμένου χρήστη.
/user/setReadedNotifications	GET: Θέτει τις ειδοποιήσεις ως διαβασμένες.
/user/delete	DELETE: Διαγραφή προφίλ και όλων των αντικειμένων στα οποία υπάρχει ως reference.
/user/sendfriendrequest	POST: Αποστολή friend request στον χρήστη με id που στέλνεται στο body.
/user/cancelfriendrequest	POST: Ακύρωση friend request στον χρήστη με id που στέλνεται στο body.
/user/getfriendrequests	GET: Επιστρέφει τα friend request στα οποία δεν έχει απαντήσει ο συνδεδεμένος χρήστης.
/user/setReadedRequests	GET: Θέτει τα αιτήματα φιλίας ως διαβασμένα.
/user/getpendingrequest	POST: Ελέγχει αν υπάρχει αίτημα φιλίας το οποίο εκκρεμεί ανάμεσα στον συνδεδεμένο

	χρήστη και στον χρήστη με id που στέλνεται στο body.
/user/acceptfriendrequest	POST: Αποδοχή αιτήματος φιλίας.
/user/rejectfriendrequest	POST: Απόρριψη αιτήματος φιλίας.
/user/unfriend	POST: Διαγραφή φίλου.
/image/upload	POST: Προσθήκη/ενημέρωση εικόνας.
/image/:id	GET: Επιστροφή εικόνας με το συγκεκριμένο id.
/image/delete	DELETE: Διαγραφή εικόνας.
/product/product	POST: Επιστροφή προϊόντος με το id που στέλνεται ως body.
/product/addtoyourlist	POST: Προσθήκη προϊόντος στην Wishlist.
/product/deletefromyourlist	POST: Διαγραφή προϊόντος από τη Wishlist και ενημέρωση αντικειμένων όπου αυτό υπάρχει ως reference.
/product/existtomylist	POST: Έλεγχος για το αν υπάρχει το προϊόν στην Wishlist του συνδεδεμένου χρήστη.
/product/editgift	POST: Ενημέρωση της περιγραφής ενός προϊόντος.
/product/getwishlist	POST: Επιστροφή της Wishlist του χρήστη που το id του στέλνεται ως body.
/product/productsCategory	POST: Επιστροφή των υποκατηγοριών που υπάγονται στην κατηγορία που περνάει ως body
/product/category	POST: Επιστροφή των προϊόντων που ανήκουν στην συγκεκριμένη κατηγορία.
/product/search	POST: Επιστροφή προϊόντων που περιέχουν στο όνομά τους την λέξη της αναζήτησης.
/transactions/addmoney	POST: Δημιουργία transaction με amount και id (giftId) που στέλνονται ως body.
/transactions/deletemoney	POST: Διαγραφή transaction και ενημέρωση του δώρου στο οποίο υπάρχει ως reference.
/transactions/editmoney	POST: Ενημέρωση του ποσού του transaction και του totalAmount του δώρου.
/transactions/mytransactions	GET: Επιστροφή δώρων στα οποία έχει προσθέσει ποσό ο συνδεδεμένος χρήστης.

/home/gethomeDate	GET: Επιστροφή δώρων των φίλων του συνδεδεμένου χρήστη, ταξινομημένα χρονολογικά.
/home/gethomeAmount	GET: Επιστροφή δώρων των φίλων του συνδεδεμένου χρήστη, ταξινομημένα με βάση το totalAmount.
/home/search	POST: Επιστροφή χρηστών που περιέχουν στο username τους την λέξη της αναζήτησης.
/order/getOwnerData	GET: Επιστροφή ολοκληρωμένων παραγγελιών όπου παραλήπτης του δώρου είναι ο συνδεδεμένος χρήστης.
/order/getDonatorData	GET: Επιστροφή ολοκληρωμένων παραγγελιών όπου έχει προσθέσει ποσό ο συνδεδεμένος χρήστης.

Τα καταστήματα που είναι οι συνεργάτες μας μπορούν να ανεβάσουν ένα αρχείο JSON στην εφαρμογή. Το αρχείο αυτό πρέπει να ακολουθεί μια συγκεκριμένη μορφή και να έχει κάποια συγκεκριμένα πεδία. Τα βασικά πεδία που πρέπει να περιέχει κάθε αντικείμενο του αρχείου είναι:

- productid: αφορά το id του προϊόντος στη βάση του καταστήματος,
- productName,
- productLink: το link του προϊόντος για το e-shop του καταστήματος,
- imageLink: είναι η κύρια φωτογραφία του προϊόντος και πρέπει να έχει διαστάσεις minimum “500 X 500”,
- price: Η τιμή του προϊόντος,
- shippingCost,
- categoryPath: πρέπει να συνάδει με τα Path των προϊόντων που είναι συμβατά με την εφαρμογή μας,
- description: υποχρεωτικές τεχνικές λεπτομέρειες ανάλογα με τον τύπο του προϊόντος.

Όταν γίνεται upload ενός αρχείου από συνεργάτη, ελέγχεται αρχικά αν ο τύπος του αρχείου ακολουθεί τις παραπάνω προδιαγραφές. Έπειτα ελέγχεται αν το productId για το συγκεκριμένο

κατάστημα υπάρχει ήδη στη βάση μας. Αν υπάρχει, σημαίνει ότι το κατάστημα θέλει να ανανεώσει κάποιες πληροφορίες για το συγκεκριμένο προϊόν, οπότε ενημερώνουμε τα πεδία του συγκεκριμένου προϊόντος στη βάση μας. Αν δεν υπάρχει το προϊόν προστίθεται. Τέλος, αν η βάση μας περιέχει προϊόντα από το συγκεκριμένο κατάστημα, τα οποία δεν περιέχονται στο τρέχων αρχείο, διαγράφονται και ενημερώνονται όλοι οι χρήστες (μέσω ειδοποιήσεων) για τις αλλαγές που έχουν γίνει.

Shop Routes:

Route	**Purpose**
/registerAdmin	POST: Αποθήκευση απαιτούμενων στοιχείων στη βάση και αποστολή token.
/loginAdmin	POST: Έλεγχος email και password και αποστολή token.
/shop/auth	GET: Επιστρέφει τα στοιχεία του συνδεδεμένου καταστήματος.
/shop/edit	POST: Ενημέρωση των στοιχείων που προσθέτει το κατάστημα.
/shop/uploadFile	POST: Εισαγωγή στη βάση των προϊόντων που περιέχονται στο JSON αρχείο.
/shop/delete	DELETE: Διαγραφή προφίλ και όλων των προϊόντων του. Ενημέρωση όλων των αντικειμένων που περιέχουν ως reference τα συγκεκριμένα προϊόντα.
/imageShop/upload	POST: Προσθήκη/ενημέρωση εικόνας.
/imageShop/:id	GET: Επιστροφή εικόνας με το συγκεκριμένο id.
/imageShop/delete	DELETE: Διαγραφή εικόνας.

5.3 Front-end

Σε αυτό το κεφάλαιο θα αναφερθούμε στο UI (User Interface) της εφαρμογής μας, παραθέτοντας κάποια στιγμιότυπα. Αρχικά όποιος επισκεφτεί την εφαρμογή μας έχει τη δυνατότητα να συνδεθεί ή να δημιουργήσει λογαριασμό με 2 τρόπους (user ή partner).

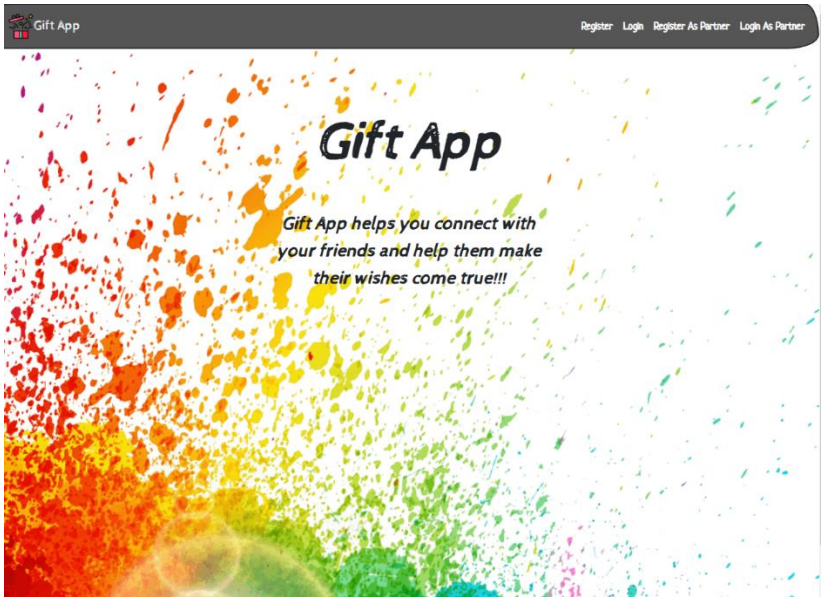

Figure 30: Landing page of application.

5.3.1 As Partner

Σελίδες για την εγγραφή/σύνδεση ως συνεργάτης.

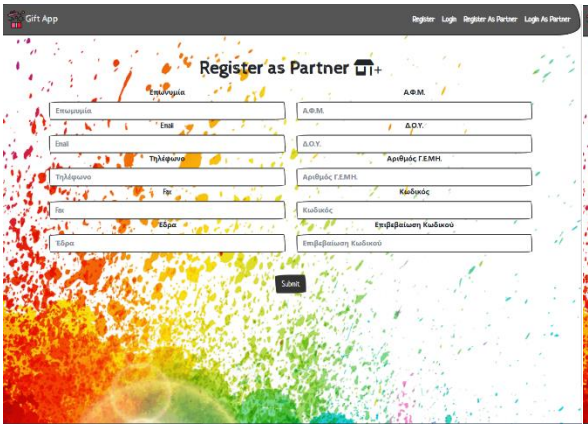

Figure 31: Register as partner.

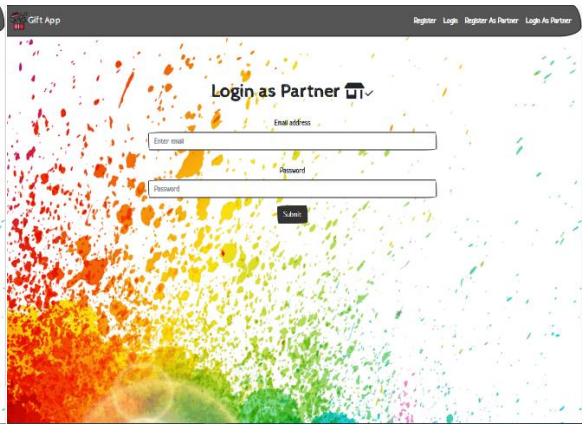

Figure 31: Login as partner.

Έπειτα από την επιτυχή σύνδεση ως συνεργάτης, το κατάστημα είναι σε θέση να περιηγηθεί στην εφαρμογή. Όπως βλέπουμε στην παρακάτω εικόνα βρίσκεται στο Home, όπου οι μόνες του επιλογές είναι να ανεβάσει ένα αρχείο με προϊόντα όπως προαναφέραμε, να κάνει edit των στοιχείων του και να αποσυνδεθεί ή να διαγράψει τον λογαριασμό του. (οι επιλογές Statistics και Stock είναι ανενεργές για την ώρα).

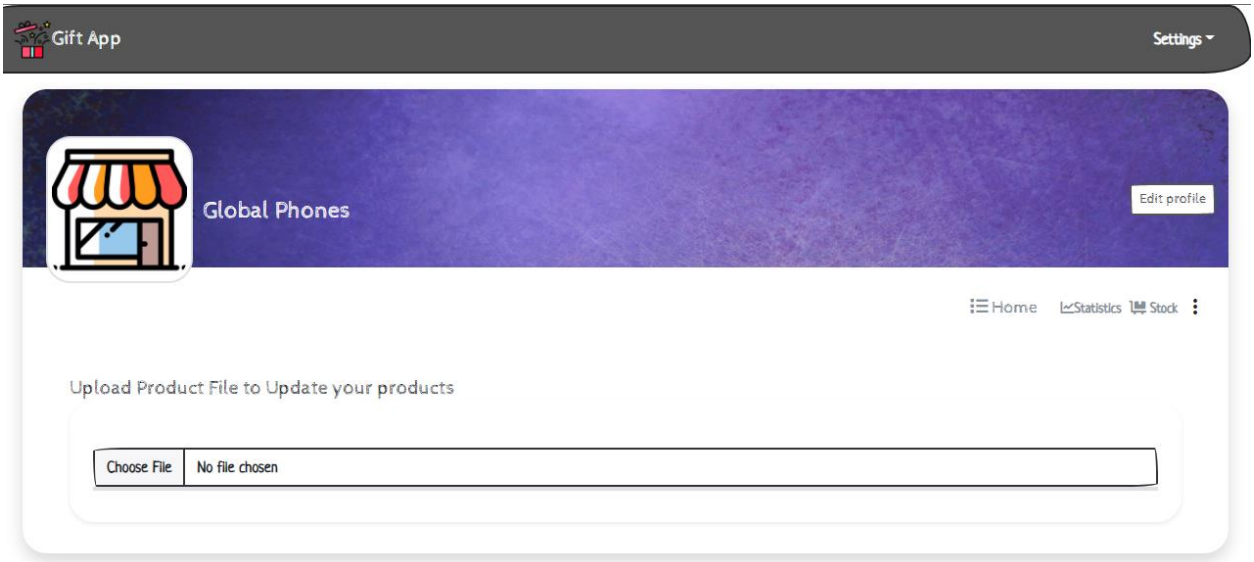

Figure 33: Home page as partner.

5.3.2 As User

Σελίδες για την εγγραφή/σύνδεση ως χρήστης.

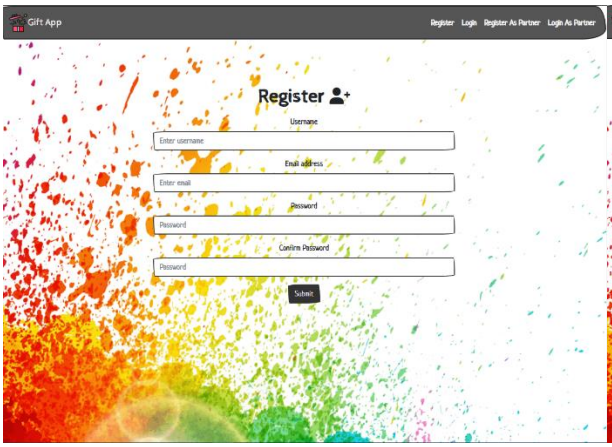

Figure 34: Register as user.

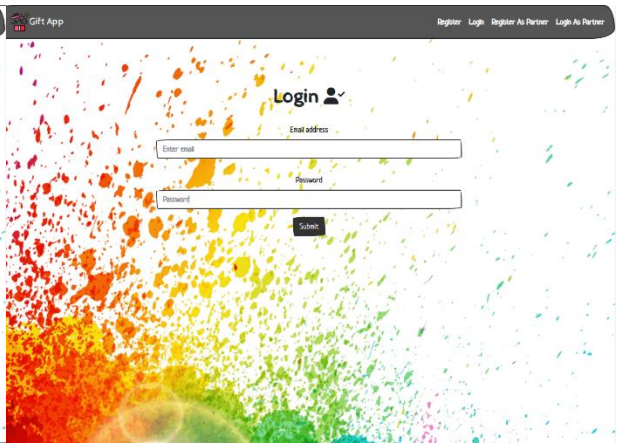

Figure 35: Login as user.

Έπειτα από την επιτυχή σύνδεση, ο χρήστης βρίσκεται στο Home, όπου βλέπει την ροή των δώρων των φίλων του. Όπως βλέπουμε παρακάτω, όσον αφορά τα δώρα μπορεί είτε να συνεισφέρει είτε να επεξεργαστεί το ποσό που έχει ήδη προσθέσει.

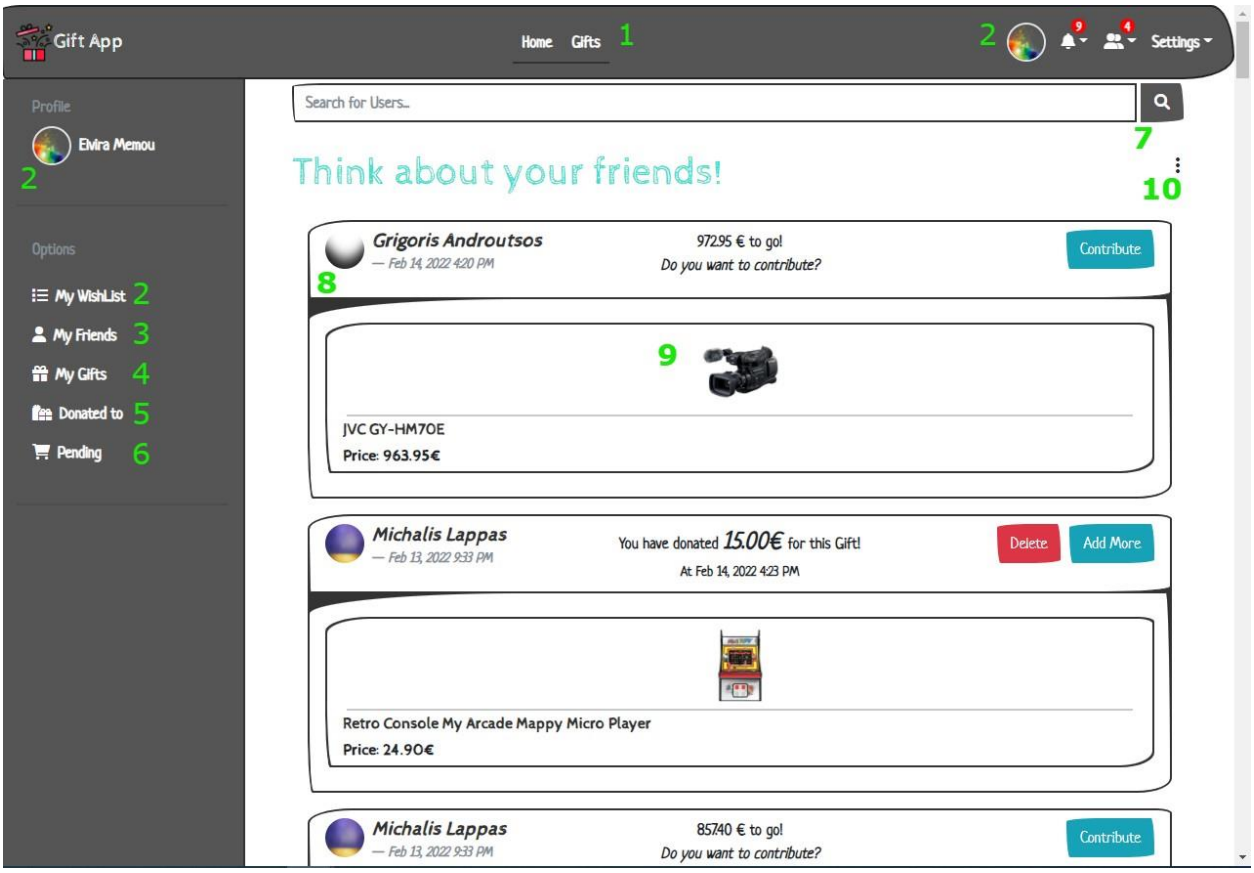

Figure 36: Home page as user.

Επίσης ο χρήστης μπορεί να περιηγηθεί:

- στη σελίδα με τα προϊόντα (1),
- στο προφίλ του και συγκεκριμένα στη Wishlist του (2),
- στο προφίλ του και συγκεκριμένα στη λίστα των φίλων του (3),
- στο προφίλ του και συγκεκριμένα στα δώρα που έχει παραλάβει (4),
- στο προφίλ του και συγκεκριμένα στα δώρα που έχουν παραλάβει οι φίλοι του και έχει συνεισφέρει (5),

- στο προφίλ του και συγκεκριμένα στα δώρα που έχει συνεισφέρει και δεν έχουν ολοκληρωθεί ακόμα (6),
- στην αναζήτηση άλλων χρηστών (7),
- στα προφίλ των φίλων του (8),
- στις σελίδες των προϊόντων που επιθυμούν οι φίλοι του (9).

Επιπλέον, μπορεί να αλλάξει την ταξινόμηση της ροής (10), με βάση τα δώρα που έχουν συγκεντρώσει το μεγαλύτερο ποσό. Τέλος, στο navbar μπορεί να δει τις ειδοποιήσεις του, τα αιτήματα φιλίας που του έχουν σταλθεί και να αποσυνδεθεί (settings).

Όσον αφορά την σελίδα των προϊόντων ο χρήστης αρχικά βλέπει τις κατηγορίες προϊόντων που υπάρχουν στην εφαρμογή μας και μπορεί να περιηγηθεί σε αυτές μέσω του κατάλληλου path.

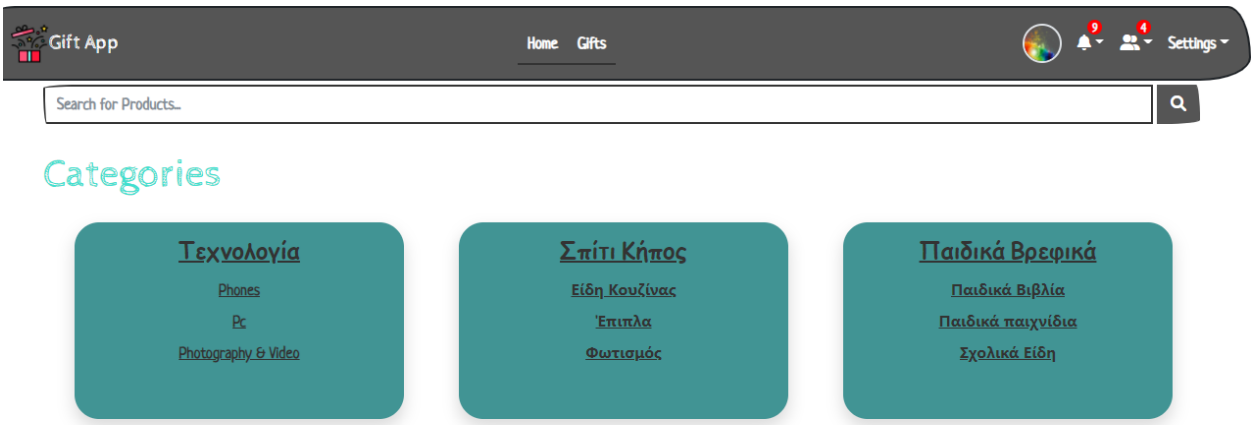

Figure 37: Basic Categories.

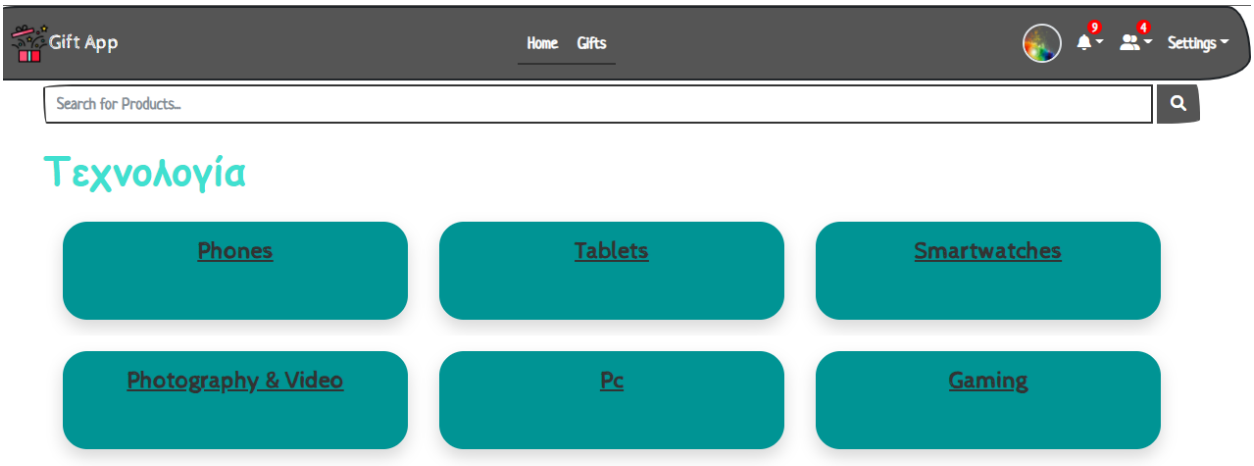

Figure 38: Sub-categories.

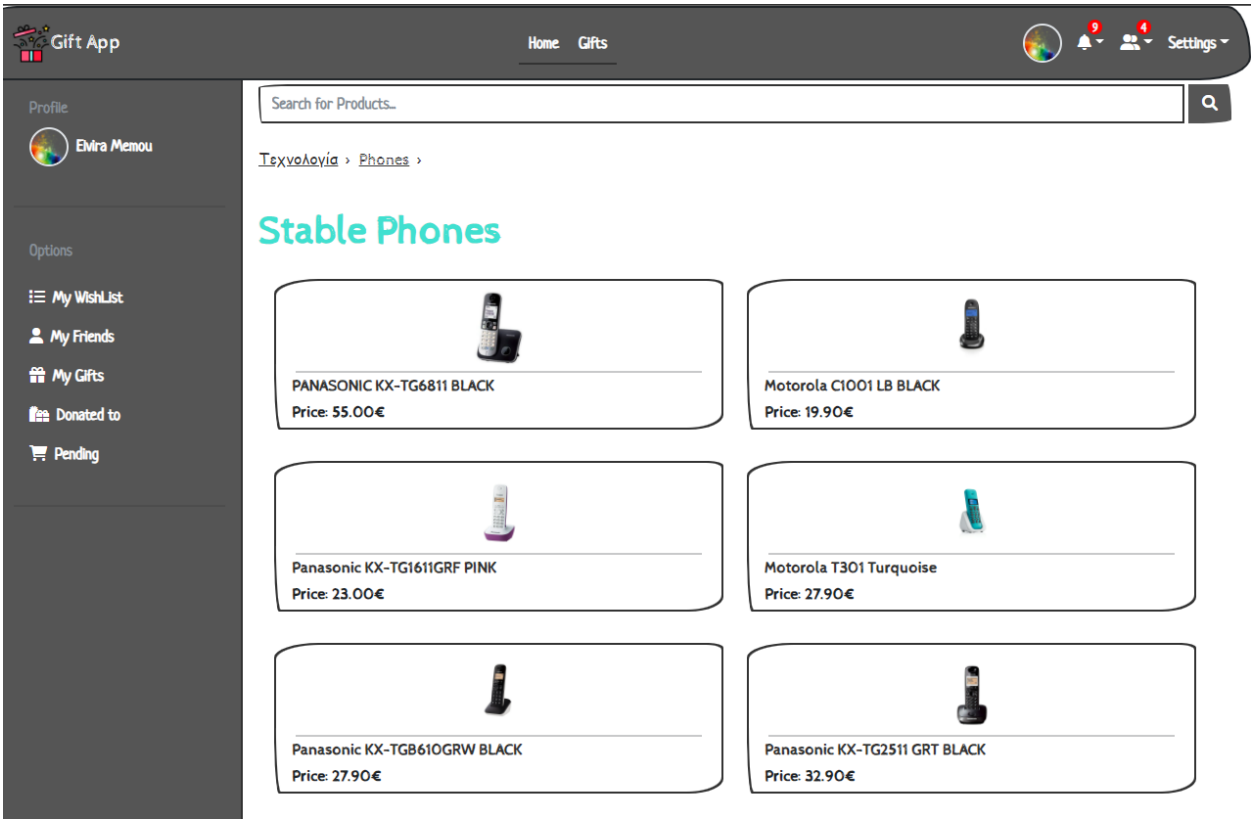

Figure 39: Page of products.

Στο προφίλ του ο χρήστης αρχικά βλέπει την Wishlist του, χωρίς στα δώρα να αναγράφεται το ποσό που έχει συλλεχθεί για να μην χάνεται το στοιχείο της έκπληξης και έχει την δυνατότητα να προσθέσει περιγραφή σε αυτά ή να τα διαγράψει.

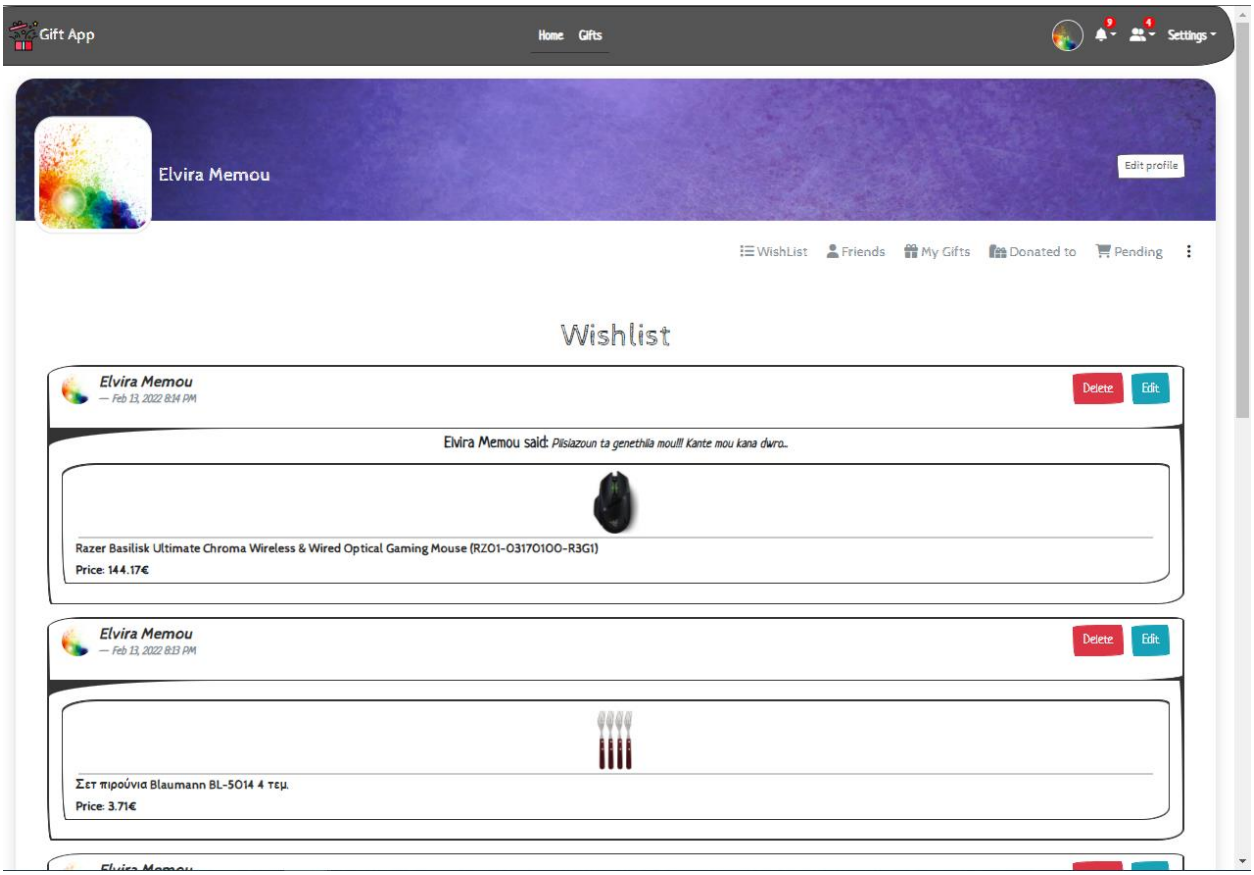

Figure 40: Profile page.

Εκτός από την Wishlist, στο προφίλ του έχει και τις υπόλοιπες δυνατότητες που αναφέραμε προηγουμένως (3, 4, 5, 6). Αυτές οι σελίδες παρουσιάζονται με την σειρά παρακάτω.

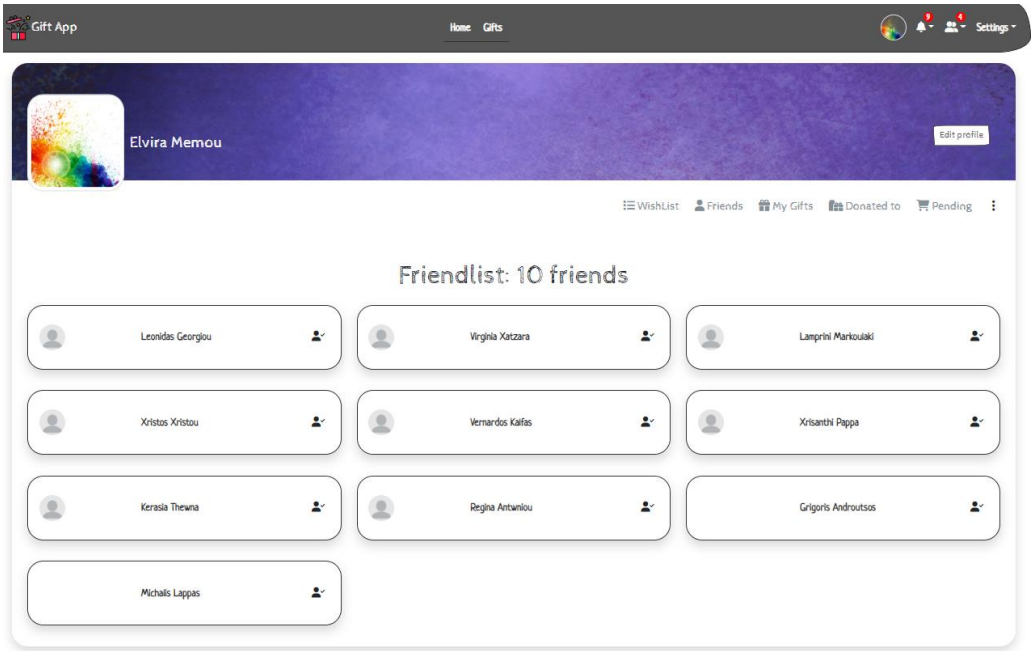

Figure 41: Friend-list page.

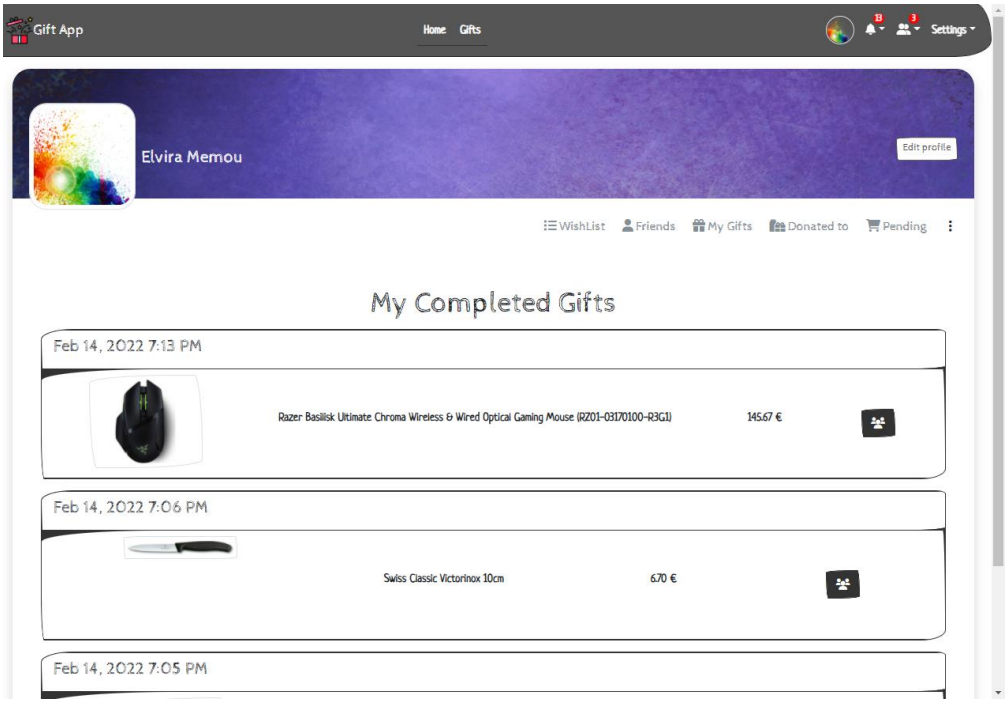

Figure 42: Completed Gifts page.

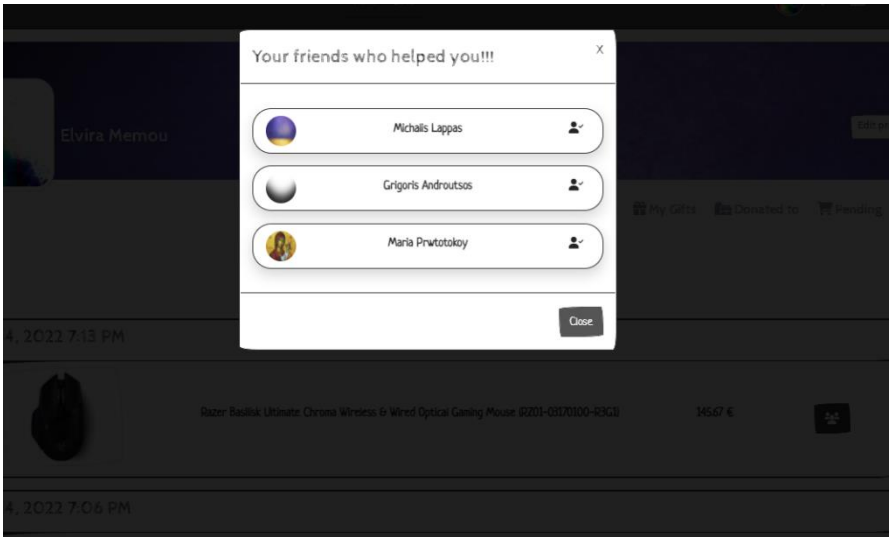

Figure 43: Completed Gifts page (donators of a gift).

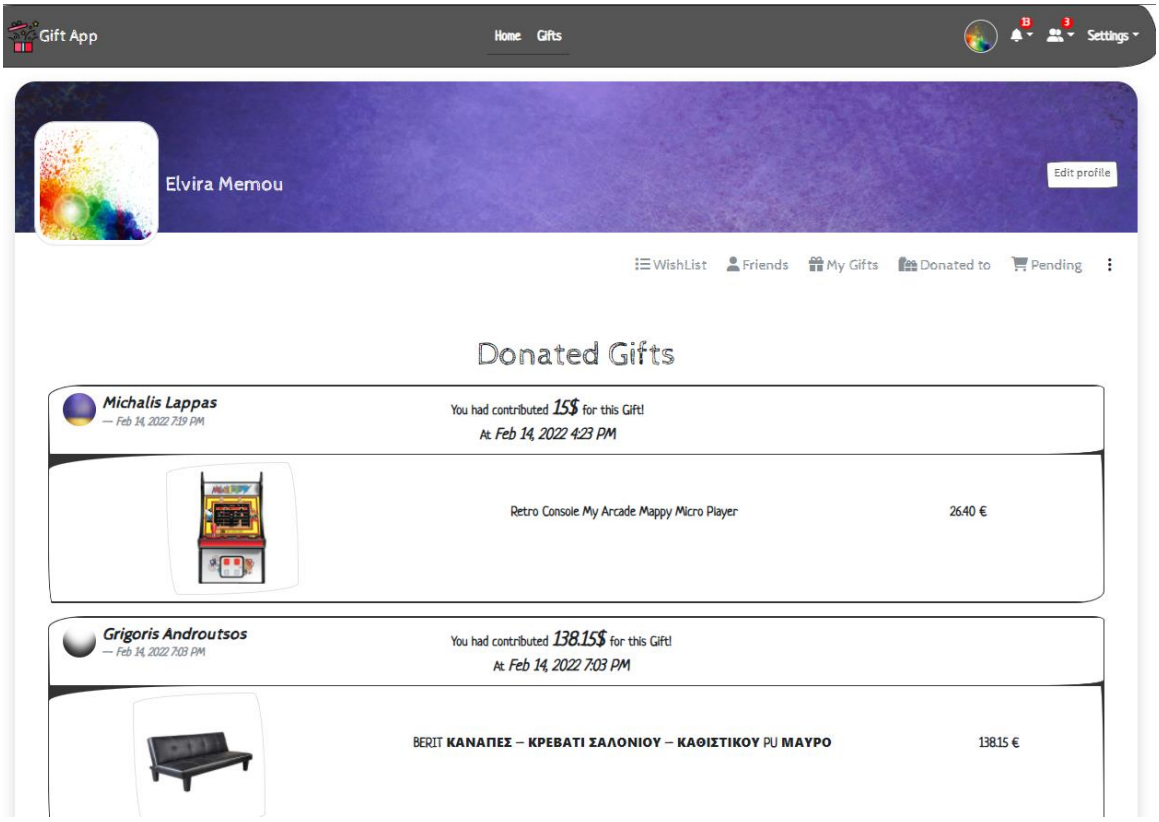

Figure 44: Donated to page.

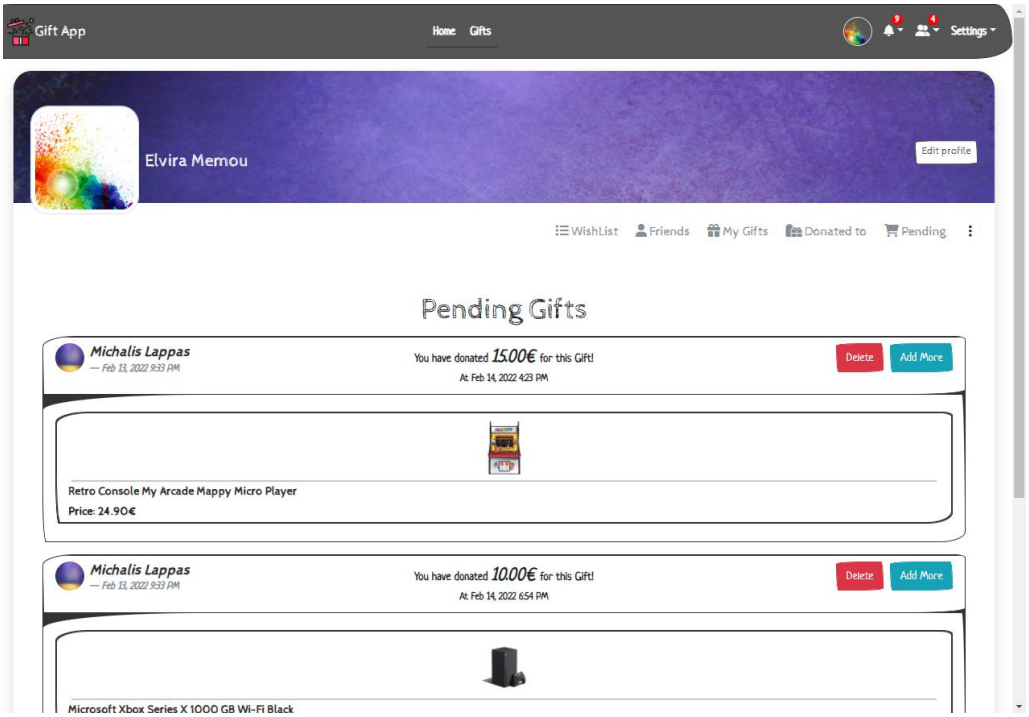

Figure 45: Pending Gifts page.

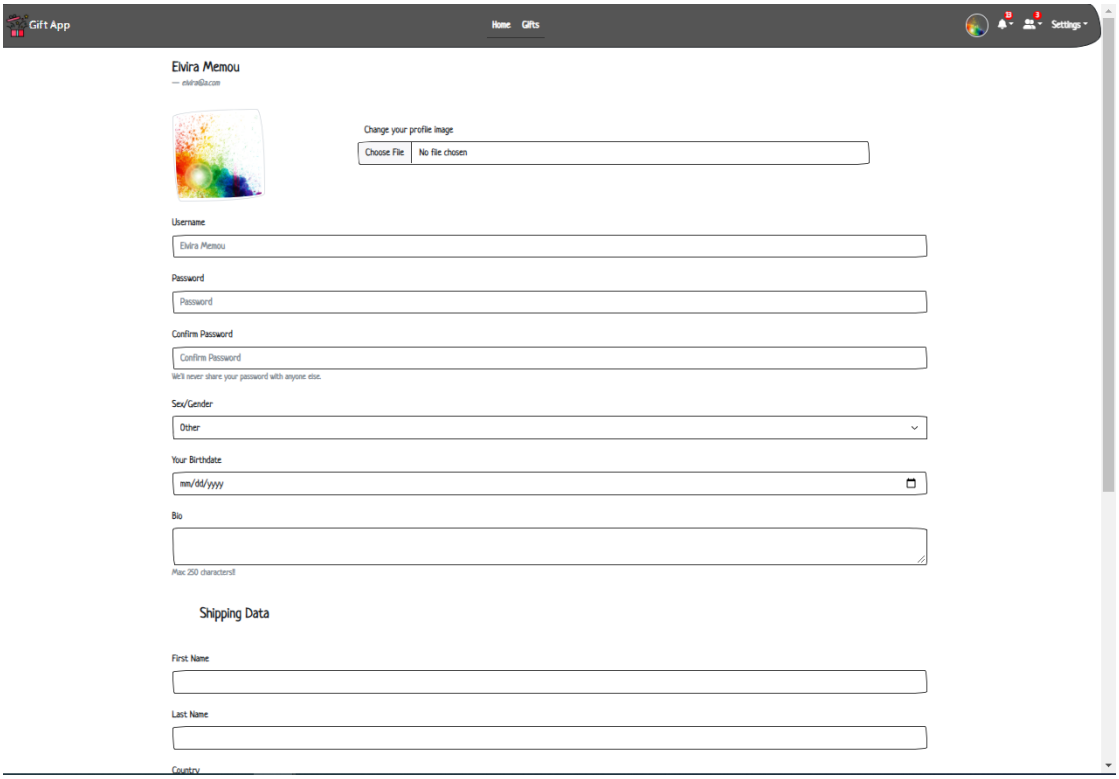

Figure 46: Edit profile page.

Στη συνέχεια βλέπουμε τα αποτελέσματα μιας αναζήτησης χρηστών και προϊόντων, τα προφίλ φίλου και μη. Τέλος παρουσιάζεται η σελίδα ενός μεμονωμένου προϊόντος.

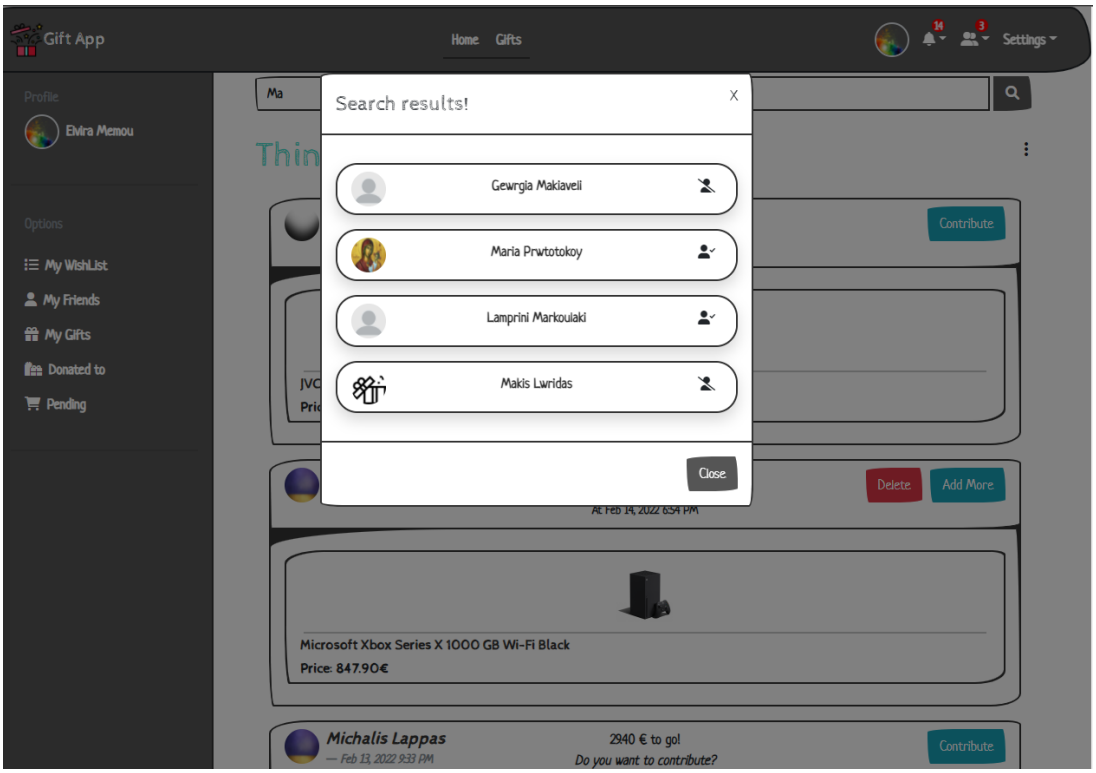

Figure 47: Search results for users.

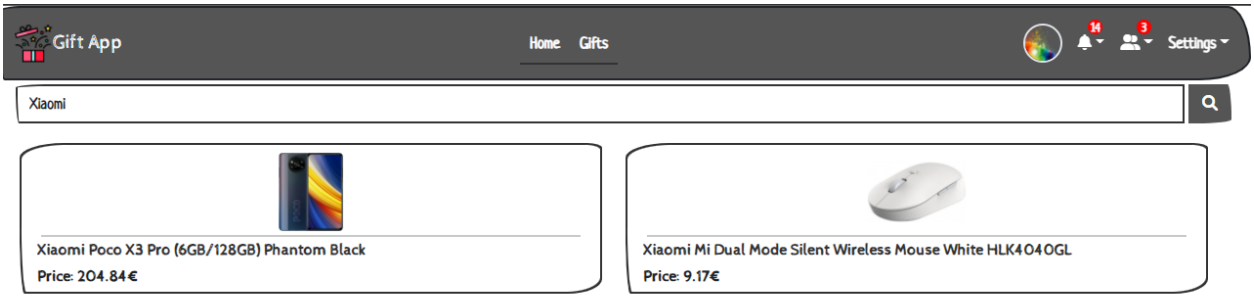

Figure 48: Search results for products.

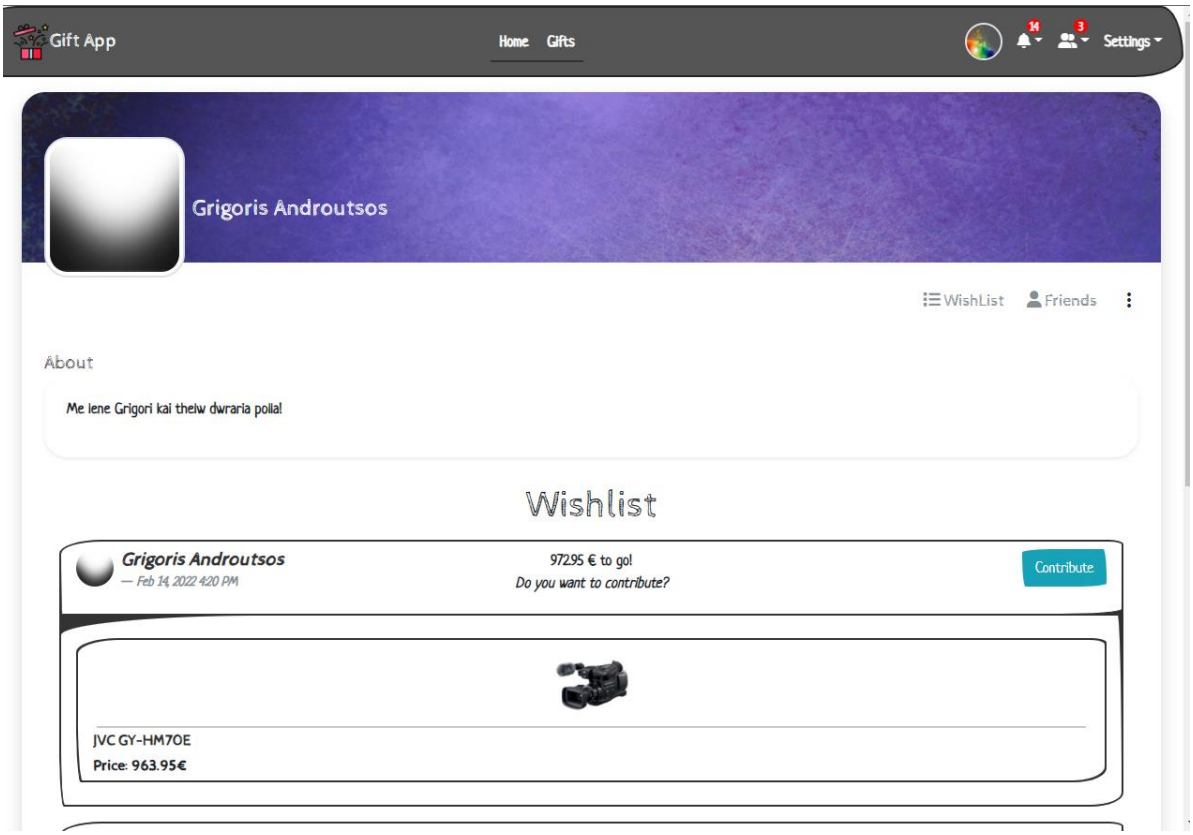

Figure 49: Friend profile.

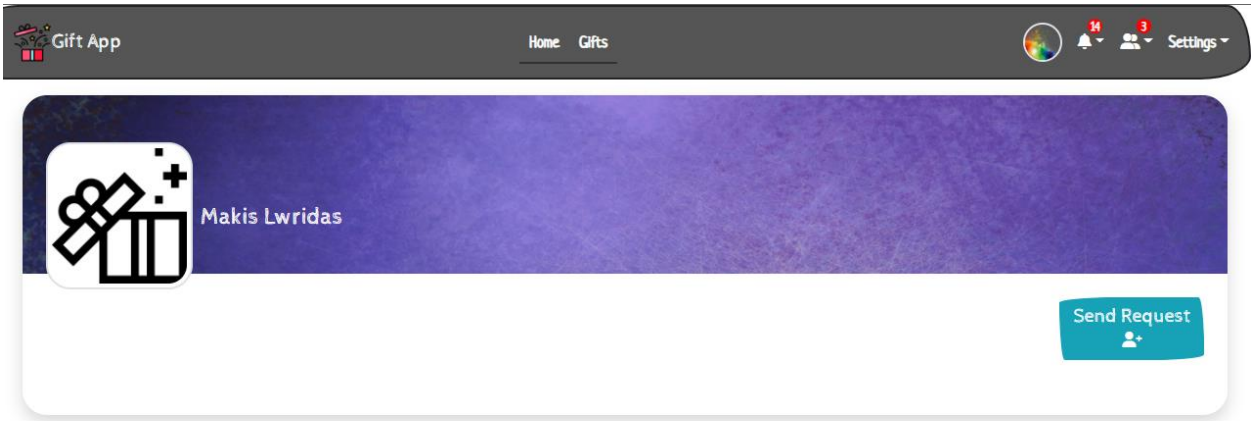

Figure 49: Non-friend profile.

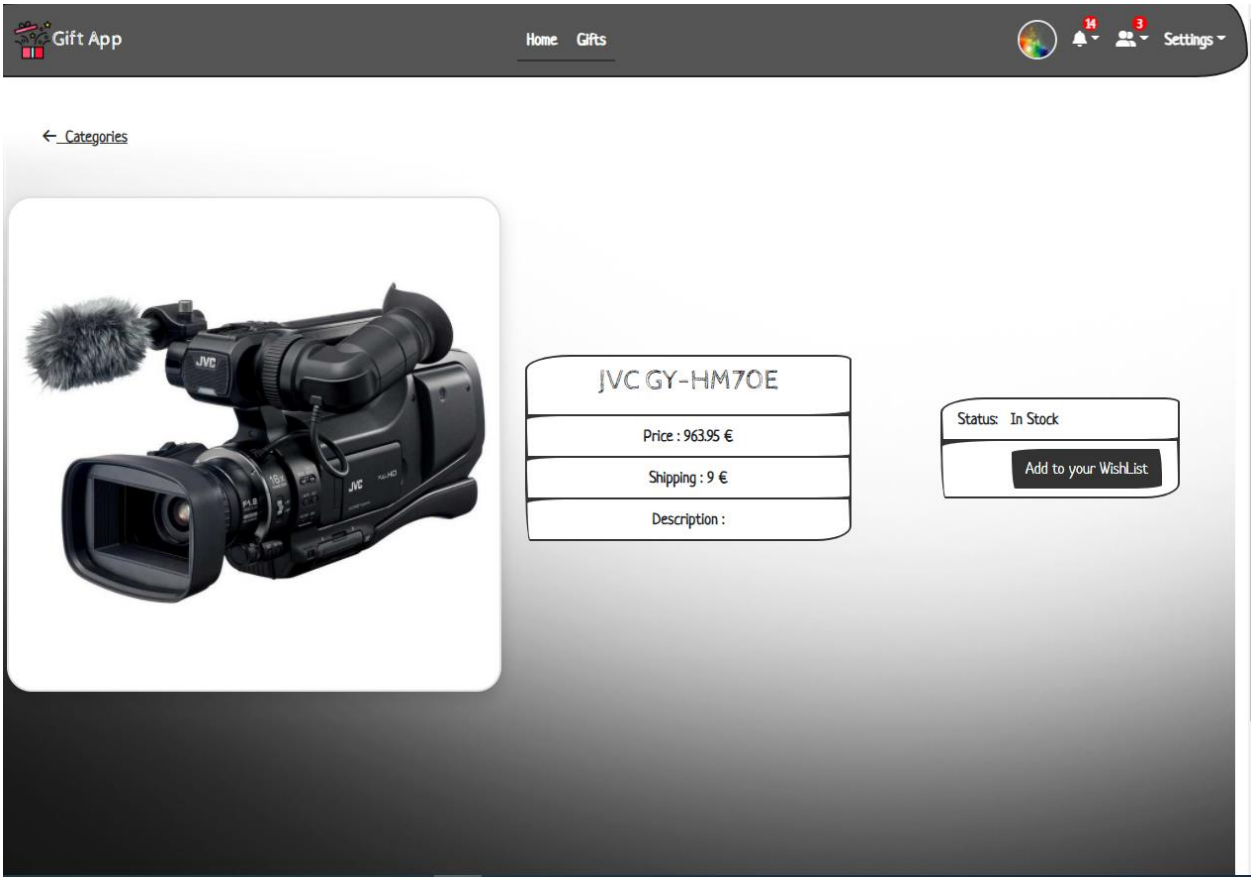

Figure 49: Product Page.

Κεφάλαιο 6 Σύνοψη

6.1 Συμπεράσματα

Ο στόχος της παρούσας διπλωματικής εργασίας ήταν η ανάπτυξη μιας σύνθετης εφαρμογής με την τεχνολογία MERN stack η οποία περιλαμβάνει τα Node.js, Express.js, MongoDB ως βάση δεδομένων και React ως την βιβλιοθήκη μας στο front-end. Όπως φαίνεται ο στόχος μας επιτεύχθηκε. Οι λόγοι που χρησιμοποιούμε τις συγκεκριμένες τεχνολογίες περιγράφονται στα κεφάλαια θεωρητικού περιεχομένου. Επίσης παρουσιάζεται διεξοδικά και αναλυτικά η λειτουργία της εφαρμογής μας και πιο συγκεκριμένα οι αλληλεπιδράσεις των χρηστών, ο τρόπος λειτουργίας της βάσης μας και γενικά η αρχιτεκτονική του back-end που έχουμε δημιουργήσει. Τέλος γίνεται αναφορά στο front-end, δηλαδή κυρίως στο εμφανισιακό κομμάτι.

6.2 Μελλοντικές επεκτάσεις

Προφανώς βέβαια υπάρχει χώρος και για προσθήκες. Η εφαρμογή αυτή χτίστηκε για να προβάλει τα σημαντικότερα ζητήματα που λύνονται με τη χρήση MERN stack. Υπάρχουν αρκετά χαρακτηριστικά τα οποία θα καθιστούσαν την εφαρμογή μας πιο ολοκληρωμένη και ελκυστική έτσι ώστε να μπορεί να σταθεί σε πραγματικό περιβάλλον. Για παράδειγμα όταν ολοκληρώνεται η αγορά ενός προϊόντος, με την βοήθεια ενός λογισμικού, όπως το stripe, θα μπορούσε να γίνει πιο ρεαλιστική η αναπαράσταση του συστήματος πληρωμών με την χρήση εικονικών πιστωτικών καρτών. Όσον αφορά τα καταστήματα που είναι και οι συνεργάτες μας, θα μπορούσαμε να προσθέσουμε κάποιους δείκτες στατιστικών για τις πωλήσεις, όπως για παράδειγμα ποια προϊόντα είναι πιο περιζήτητα, τις πωλήσεις ανά μήνα, τις ηλικιακές ομάδες σε σχέση με τα προϊόντα, κ.λπ. Τέλος, για τους χρήστες υπάρχουν κάποια περιθώρια βελτιώσεων όσον αφορά το UI (User Interface) και το UX (User Experience).

Βιβλιογραφία

[1] David Max, “Ultimate Guide To Backend Web Development: Details and Required Skill Set”, 2020.

[2] Stack Overflow, “2021 Developer Survey”, 2021.

[3] T.J. DeGroat, "The History of JavaScript: Everything You Need to Know”, 2019.

[4] MDN Web Docs, “Web technology for developers: JavaScript”, 2021.

[5] W3Schools, “Node.js Introduction”, 2021.

[6] Node.js, “Introduction Node.js”, 2021.

[7] Priyesh Patel, “What exactly is Node.js”, 2018.

[8] Martin Heller, “What is Node.js? The JavaScript runtime explained”, 2020.

[9] Npm Docs, “About npm”, 2022.

[10] MDN Web Docs, “Express/Node introduction”, 2021.

[11] Fullstackopen.com, “Fullstack Part 3 Node.js and Express”, 2021.

[12] State of JS, “Back-end Frameworks”, 2020.

[13] Jonathan Freeman, “What is JSON? A better format for data exchange”, 2019.

[14] Sebastian Eschweiler, “Async Programming With JavaScript- Callbacks, Promises and Async/Await”, 2019.

[15] Auth0, “Introduction to JSON Web Tokens”, 2021.

[16] Ben Lutkevich and Adam Hughes, “database (DB)”, 2021.

[17] MongoDB, “NoSQL vs SQL Databases”, 2021.

[18] Mark Smallcombe, “SQL vs NoSQL: 5 Critical Differences”, 2021.

[19] Michael Wilson, “MongoDB Explained in 5 Minutes or Less”, 2014.

[20] DB-ENGINES, “MongoDB System Properties”, 2021.

[21] Nick Karnik, “Introduction to Mongoose for MongoDB”, 2018.

[22] Jessica Wilkins, “Front End Developer – What is Front End Development, Explained in Plain English”, 2021.

[23] W3Schools, “React Tutorial”, 2021.

[24] Skillcrush.com, “Tech 101: What is React JS?”, 2022.

[25] Becky Newborn, “MEAN and MERN Stacks”, 2017.

[26] W3Schools, “React Hooks”, 2021.

[27] Reed Barger, “How To Use Axios With React: The Definitive Guide (2021)”, 2021.

[28] Postman, “Learning Center”, 2021