ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΔΙΑΔΙΚΤΥΑΚΗ ΕΦΑΡΜΟΓΗ ΗΛΕΚΤΡΟΝΙΚΗΣ ΤΙΜΟΛΟΓΗΣΗΣ»

Του φοιτητή
Βασιλείου Γεωργίου
Αρ. Μητρώου: 154602

Επιβλέπων
Τεκτονίδης Δημήτριος
Διδάσκων Καθηγητής

Θεσσαλονίκη, Αύγουστος 2023

Τίτλος Δ.Ε.: Διαδικτυακή Εφαρμογή Ηλεκτρονικής Τιμολόγησης
Ονοματεπώνυμο φοιτητή: Βασιλείου Γεώργιος
Ονοματεπώνυμο εισηγητή: Τεκτονίδης Δημήτριος
Ημερομηνία ανάληψης Δ.Ε.: 24/01/2023
Ημερομηνία περάτωσης Δ.Ε.: 20/08/2023

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

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

Πρόλογος

Ο προσδιορισμός της τιμής, ή κοινώς η τιμολόγηση, ενός προϊόντος ή μίας υπηρεσίας, καθώς και η έκδοση διαφόρων παραστατικών αποτελούν το πλέον βαρυσήμαντο κομμάτι για την εύρυθμη λειτουργία των επιχειρήσεων κάθε λογής κλάδου. Η ανάγκη δε ψηφιοποίησης και απευθείας διαβίβασης των εν λόγω παραστατικών στην Ανεξάρτητη Αρχή Δημοσίων Εσόδων (Α.Α.Δ.Ε.) σύμφωνα με την απόφαση της Ελληνικής κυβέρνησης, έχει δημιουργήσει την ανάγκη ανάπτυξης ηλεκτρονικών εφαρμογών και λογισμικών προγραμμάτων τιμολόγησης.

Στόχος της παρούσας πτυχιακής εργασίας είναι η ανάπτυξη μίας διαδικτυακής εφαρμογής ηλεκτρονικής τιμολόγησης και έκδοσης παραστατικών, όπως αποδείξεις λιανικής πώλησης και τιμολόγια πώλησης και παροχής υπηρεσιών, με περιβάλλον φιλικό, εύπεπτο, εύκολο και απλό στη χρήση, περιήγηση και διαχείρισή του.

Περίληψη

Η παρούσα πτυχιακή εργασία έχει ως στόχο τη δημιουργία και ανάπτυξη μίας Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης και – κατ' επέκταση – έκδοσης βασικών παραστατικών, χρησιμοποιώντας σύγχρονα εργαλεία και τεχνολογίες.

Αρχικά, για το διαδραστικό και εμφανές περιβάλλον που προσφέρει η εφαρμογή μας στον εκάστοτε χρήστη, δηλαδή το Front – End μέρος, χρησιμοποιείται η Βιβλιοθήκη React της Javascript, η οποία βοηθάει στη δημιουργία όμορφων Διεπαφών Χρήστη. Εν συνεχεία, για τη λειτουργικότητα της εφαρμογής μας, δηλαδή το Back – End μέρος, χρησιμοποιείται το Framework Node της Javascript, καθώς και το Framework Express του Node Framework. Τέλος, για την αποθήκευση των δεδομένων της εφαρμογής μας χρησιμοποιείται η Βάση Δεδομένων MongoDB ανοιχτού κώδικα.

Η εν λόγω εφαρμογή απευθύνεται σε επιχειρήσεις, αποσκοπεί στην ηλεκτρονική τιμολόγηση προϊόντων και υπηρεσιών, καθώς και στη μεταβίβαση των αντίστοιχων ηλεκτρονικών εγγράφων στην Ανεξάρτητη Αρχή Δημοσίων Εσόδων (Α.Α.Δ.Ε.).

Λέξεις – Κλειδιά: Προγραμματισμός, Ηλεκτρονική Τιμολόγηση, Μεταβίβαση Εγγράφων, React, Node, REST API, MongoDB.

Web Application for Electronic Invoicing

VASILEIOU GEORGIOS

Abstract

This thesis' aim is to create and develop a web application for electronic invoicing and – by extension – to issue basic documents, using modern tools and technologies.

First, for the interactive and visible environment that our application offers to each user, as known as the Front-End part, the React Javascript library is being used, which helps to create beautiful user interfaces. Then, for the functionality of our application, as known as the Back-End part, the Javascript Node Framework is being used, as well as the Node Framework Express. Finally, to store our application's usage data, the open source MongoDB database is being used.

This application is being addressed to businesses and it aims at the electronic invoicing of products and services, as well as the transition of electronic document correspondences to the Independent Public Revenue Authority (I.P.R.A.).

Key – Words: Programming, Electronic Invoicing, Documents' Transition, React, Node, REST API, MongoDB.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, τον κύριο Τεκτονίδη Δημήτριο, για τη πολύτιμη βοήθεια που μού παρείχε και την εμπιστοσύνη που μού έδειξε, αναθέτοντάς μου την εν λόγω πτυχιακή εργασία.

Περιεχόμενα

Πρόλογος i
Περίληψη ii
Abstract iii
Ευχαριστίες iv
Περιεχόμενα v
Κατάλογος Σχημάτων vi
Κεφάλαιο 1ο: Εισαγωγή 1
1.1 Αντικείμενο Εργασίας 1
1.2 Στόχος Εργασίας 1
1.3 Δομή Εργασίας 2
Κεφάλαιο 2ο: Μέθοδοι Ανάπτυξης Διαδικτυακών Εφαρμογών 3
2.1 Διαδικτυακές Εφαρμογές 3
2.1.1 Ορισμός και Δομή Αρχιτεκτονικής Διαδικτυακής Εφαρμογής 3
2.1.2 Αρχιτεκτονική Επιπέδων 4
2.1.2.1 Αρχιτεκτονική Δύο Επιπέδων 4
2.1.2.2 Αρχιτεκτονική Τριών Επιπέδων 4
2.1.3 Αρχιτεκτονική MVC 6
2.2 Τεχνολογίες Ανάπτυξης Front – End Μέρους Διαδικτυακής Εφαρμογής 7
2.2.1 HTML 8
2.2.2 CSS 9
2.2.3 JavaScript 10
2.2.4 AngularJS 10
2.2.5 React 12
2.2.6 VueJS 15
2.3 Τεχνολογίες Ανάπτυξης Back – End Μέρους Διαδικτυακής Εφαρμογής 16
2.3.1 NodeJS 18
2.3.2 Java 18
2.3.3 Python 19
2.4 Διεπαφές Προγραμματισμού Εφαρμογών (APIs) 20
2.4.1 Τρόπος Λειτουργίας των API 20
2.4.2 Πλεονεκτήματα των APIs 21
2.4.3 Αρχιτεκτονικές API 21

2.4.3.1 Αρχιτεκτονική REST 22
2.4.3.2 Αρχιτεκτονική SOAP 22
2.5 Βάσεις Δεδομένων Διαδικτυακών Εφαρμογών 23
2.6 Έλεγχος Ταυτότητας και Εξουσιοδότηση 23
Κεφάλαιο 3ο: Υλοποίηση Front – End Μέρους Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης 25
3.1 Επιλογή Τεχνολογίας Ανάπτυξης Front – End 25
3.2 Δημιουργία Front – End Μέρους Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης 25
3.3 Σελίδα «Login Χρήστη» 31
3.4 Αρχική Σελίδα Χρήστη 33
3.5 Δημιουργία Νέου Πελάτη 33
3.6 Προβολή Πελάτη 38
3.7 Προβολή Λίστας Πελατών 42
3.8 Δημιουργία Απόδειξης Λιανικής Πώλησης 45
3.9 Δημιουργία Τιμολογίου Πώλησης 53
3.10 Δημιουργία Τιμολογίου Παροχής Υπηρεσιών 58
3.11 Σελίδα Παραστατικών 63
3.12 Σελίδα Παραστατικού 65
Κεφάλαιο 4ο: Υλοποίηση Back – End Μέρους Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης 70
4.1 Επιλογή Τεχνολογίας Ανάπτυξης Back – End 70
4.2 Δημιουργία Back – End Μέρους Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης 71
4.2.1 Πακέτα 71
4.2.2 Δομή Back – End 72
4.2.3 Σύνδεση σε Βάση Δεδομένων 74
4.2.4 Models 77
4.2.5 Διαχείριση Σφαλμάτων 78
4.2.6 Authentication & Authorization 78
4.2.7 Routes & Controllers 79
4.3 Ηλεκτρονική Πλατφόρμα myData 86
4.3.1 Προσφερόμενες Λειτουργίες & Μέθοδοι 86
4.3.2 Τεχνολογικές Απαιτήσεις 87
4.3.2.1 Εγγραφή Χρήστη & Ταυτοποίηση 87
4.3.2.2 HTTPS Αιτήματα 87
4.4 Χρήση του Δοκιμαστικού Περιβάλλοντος του myData 89

Κεφάλαιο 5ο: Συμπεράσματα και Πρόταση Βελτίωσης.....95
Βιβλιογραφία.....96

Κατάλογος Σχημάτων

Σχήμα 2.1: Αρχιτεκτονική Διαδικτυακής Εφαρμογής Τριών (3) Επιπέδων [12]……4
Σχήμα 2.2: Βασική Αρχιτεκτονική Διαδικτυακής Εφαρμογής [11]…….5
Σχήμα 2.3: Αρχιτεκτονική MVC [15]……..6
Σχήμα 2.4: Τεχνολογίες και Γλώσσες Ανάπτυξης Front – End μέρους Διαδικτυακής Εφαρμογής [11]……7
Σχήμα 2.5: Δομή σελίδας HTML [33]……..9
Σχήμα 2.6: Παράδειγμα κώδικα HTML χρησιμοποιώντας το CSS [38]……..9
Σχήμα 2.7: Σύγκριση Πραγματικού και Εικονικού DOM [22]……..13
Σχήμα 2.8: Πολλαπλά Στοιχεία (Components) [23]……...13
Σχήμα 2.9: Τεχνολογίες ανάπτυξης Back – End μέρους [11]……17
Σχήμα 2.10: Παράδειγμα ροής αιτημάτων (requests) [41]…….21
Σχήμα 3.1: Χρήση του Provider της Βιβλιοθήκης Redux……..26
Σχήμα 3.2: Παρακολούθηση της κατάστασης του State……27
Σχήμα 3.3: Δημιουργία σταθερού Layout……..27
Σχήμα 3.4: Χρήση του Component “BrowserRouter” (1)…….28
Σχήμα 3.5: Χρήση του Component “BrowserRouter” (2)…….29
Σχήμα 3.6: Χρήση του Component “BrowserRouter” (3)…….30
Σχήμα 3.7: Χρήση του Component “PrivateRoute”……..30
Σχήμα 3.8: Απεικόνιση Σελίδας «Login Χρήστη» σε Η/Υ……31
Σχήμα 3.9: Δημιουργία Σελίδας «Login Χρήστη»…….31
Σχήμα 3.10: Δημιουργία φόρμας συμπλήρωσης στοιχείων Χρήστη και σύνταξη συνάρτησης “onChange()”……..32
Σχήμα 3.11: Σύνταξη συνάρτησης “onSubmit()”……...32
Σχήμα 3.12: Απεικόνιση Σελίδας «Αρχική Σελίδα»……...33
Σχήμα 3.13: Δημιουργία Σελίδας «Αρχική Σελίδα»……...34
Σχήμα 3.14: Κλικ στο κουμπί «Πελάτες» για εμφάνιση της λίστας των καταχωρημένων πελατών….34
Σχήμα 3.15: Λίστα καταχωρημένων πελατών Χρήστη……..35
Σχήμα 3.16: Κλικ στο κουμπί «Δημιουργία Νέου Πελάτη» για τη δημιουργία νέου πελάτη……35
Σχήμα 3.17: Σελίδα «Δημιουργία Νέου Πελάτη»……..36
Σχήμα 3.18: Δημιουργία του κουμπιού «Δημιουργία Νέου Πελάτη» στη σελίδα «Λίστα Πελατών»..36
Σχήμα 3.19: Δημιουργία του form στη Σελίδα «Δημιουργία Νέου Πελάτη» (1)……..36
Σχήμα 3.20: Δημιουργία του form στη Σελίδα «Δημιουργία Νέου Πελάτη» (2)……..37
Σχήμα 3.21: Σύνταξη της συνάρτησης “SearchCustomer()”…….37
Σχήμα 3.22: Σύνταξη της συνάρτησης “createCostumer()”……...38
Σχήμα 3.23: Σελίδα προβολής στοιχείων νεοεισαχθέντος πελάτη……39
Σχήμα 3.24: Σύνταξη συνάρτησης “getCostumer()”……..39
Σχήμα 3.25: Δημιουργία Σελίδας προβολής στοιχείων νεοεισαχθέντος πελάτη (1)……..40
Σχήμα 3.26: Δημιουργία Σελίδας προβολής στοιχείων νεοεισαχθέντος πελάτη (2)……..40
Σχήμα 3.27: Σύνταξη συνάρτησης “DeleteButton()”…..……41
Σχήμα 3.28: Form ενημέρωσης στοιχείων πελάτη……41
Σχήμα 3.29: Δημιουργία form ενημέρωσης στοιχείων πελάτη……..41
Σχήμα 3.30: Σύνταξη συνάρτησης “updateCustomer()”……42
Σχήμα 3.31: Εμφάνιση Σελίδας «Λίστα Πελατών» σε οθόνη ενός ηλεκτρονικού υπολογιστή…….43
Σχήμα 3.32: Εμφάνιση Σελίδας «Λίστα Πελατών» σε οθόνη ενός κινητού τηλεφώνου……...43

Σχήμα 3.33: Δημιουργία Σελίδας «Λίστα Πελατών»…….44
Σχήμα 3.34: Σύνταξη Hook “useEffect()”……..44
Σχήμα 3.35: Σελίδα «Έκδοση Απόδειξης»…….45
Σχήμα 3.36: Δημιουργία Σελίδας «Έκδοση Απόδειξης»……45
Σχήμα 3.37: Σύνταξη συνάρτησης “changePaymentType()”……46
Σχήμα 3.38: Σύνταξη συνάρτησης “addProducts()”……...46
Σχήμα 3.39: Σύνταξη συνάρτησης “prevIsValid()”……47
Σχήμα 3.40: Σύνταξη Hook “useEffect()”……..48
Σχήμα 3.41: Σύνταξη Hook “useState()”……48
Σχήμα 3.42: Ορισμός και αρχικοποίηση μεταβλητών……48
Σχήμα 3.43: Σύνταξη Hook “useEffect()” με for loop……...49
Σχήμα 3.44: Σύνταξη Hook “useEffect()”……..50
Σχήμα 3.45: Σύνταξη του Hook “useState()”……50
Σχήμα 3.46: Σύνταξη Hook “useSelector()”……..……50
Σχήμα 3.47: Σύνταξη συνάρτησης “onSubmit()”……...51
Σχήμα 3.48: Ορισμός σταθεράς “invoiceSummary”……..51
Σχήμα 3.49: Κλήση της συνάρτησης “postAadeInvoice”……..52
Σχήμα 3.50: Ορισμός του Hook “useEffect()”……...53
Σχήμα 3.51: Σελίδα «Έκδοση Τιμολογίου Πώλησης»……...54
Σχήμα 3.52: Αναζήτηση πελάτη βάσει Α.Φ.Μ……...54
Σχήμα 3.53: Query με το Α.Φ.Μ. πελάτη……...54
Σχήμα 3.54: Κώδικας δημιουργίας της Σελίδας «Έκδοση Τιμολογίου Πώλησης»……...55
Σχήμα 3.55: Ορισμός των Hooks “useEffect()” και “useLocation()”……55
Σχήμα 3.56: Ορισμός του Hook “useEffect()”……...56
Σχήμα 3.57: Κώδικας εμφάνισης των αποτελεσμάτων της αναζήτησης……56
Σχήμα 3.58: Ορισμός συνάρτησης “FindSingleuser()”……..57
Σχήμα 3.59: Εντοπισμός αλλαγής και ανανέωση στοιχείων πελάτη με χρήση του Hook “useEffect()”……...57
Σχήμα 3.60: Δημιουργία συνάρτησης τύπου “arrow”……57
Σχήμα 3.61: Μεταβλητές για τη δημιουργία Τιμολογίου Πώλησης……..58
Σχήμα 3.62: Εμφάνιση πεδίου στοιχείων πελάτη……..58
Σχήμα 3.63: Κώδικας δημιουργίας Σελίδας «Δημιουργία Τιμολογίου Παροχής Υπηρεσιών»..……...59
Σχήμα 3.64: Έλεγχος συνολικής καθαρής αξίας……59
Σχήμα 3.65: Σύνταξη συνάρτησης “addTaxes()”……...60
Σχήμα 3.66: Σύνταξη συνάρτησης “prevIsValid()”……60
Σχήμα 3.67: Σύνταξη συνάρτησης “handleRemoveProduct()”……..61
Σχήμα 3.68: Σύνταξη συνάρτησης “onChangeTax()”……61
Σχήμα 3.69: Σύνταξη Hook “useEffect()” με τη βοήθεια της συνάρτησης “map” (1)……..62
Σχήμα 3.70: Σύνταξη Hook “useEffect()” με τη βοήθεια της συνάρτησης “map” (2)……..62
Σχήμα 3.71: Έλεγχος ύπαρξης φόρων και υπολογισμός των τιμών τους……..63
Σχήμα 3.72: Εμφάνιση της λίστας παραστατικών υπό μορφή πίνακα σε οθόνες ηλεκτρονικών υπολογιστών.
Σχήμα 3.73: Εμφάνιση της λίστας παραστατικών υπό μορφή πίνακα σε οθόνες κινητών τηλεφώνων……..64
Σχήμα 3.74: Κώδικας δημιουργίας της Σελίδας «Λίστα Παραστατικών»……65
Σχήμα 3.75:Σύνταξη Hook “useEffect()”……...65

Σχήμα 3.76: Σελίδα προβολής παραστατικού……66
Σχήμα 3.77: Σελίδα αποθήκευσης παραστατικού υπό τη μορφή αρχείου PDF……66
Σχήμα 3.78: Κώδικας για τη προβολή παραστατικού (1)……..67
Σχήμα 3.79: Κώδικας για τη προβολή παραστατικού (2)……..67
Σχήμα 3.80: Κώδικας για τη προβολή παραστατικού (3)……..68
Σχήμα 3.81: Εντοπισμός μη υπάρχοντος παραστατικού……68
Σχήμα 3.82: Κώδικας δημιουργίας αρχείο PDF παραστατικού……69
Σχήμα 4.1: Χρησιμοποιηθέντα πακέτα……...71
Σχήμα 4.2: Αρχιτεκτονική MVC……72
Σχήμα 4.3: Δομή Back – End……73
Σχήμα 4.4: Δημιουργία project Βάσης Δεδομένων……74
Σχήμα 4.5: Δημιουργία χρήστη και πρόσθεση διεύθυνσης IP……...75
Σχήμα 4.6: Σύνδεση με τη Βάση Δεδομένων και εισαγωγή password……..75
Σχήμα 4.7: Διαχείριση δημιουργηθείσας Βάσης Δεδομένων……76
Σχήμα 4.8: Σύνδεση Εφαρμογής και Βάσης Δεδομένων……76
Σχήμα 4.9: Κλήση της συνάρτησης σύνδεσης……76
Σχήμα 4.10: Δημιουργία models……77
Σχήμα 4.11: Σύνταξη συνάρτησης “errorHandler()”……..78
Σχήμα 4.12: Εισαγωγή του ”middleware” στο αρχείο “server.js”……78
Σχήμα 4.13: Σύνταξη συνάρτησης “protect()”……...80
Σχήμα 4.14: Παράδειγμα δήλωσης ενός Route……..80
Σχήμα 4.15: Ορισμός “path” για κάθε Route……81
Σχήμα 4.16: Σύνταξη “postInvoice()”……82
Σχήμα 4.17: Σύνταξη “postCancelInvoice()”……82
Σχήμα 4.18: Σύνταξη “createInvoice()”……83
Σχήμα 4.19: Σύνταξη “getAllInvoice()”……84
Σχήμα 4.20: Σύνταξη “getInvoice()”……..84
Σχήμα 4.21: Σύνταξη “postAfm()” (1)……...85
Σχήμα 4.22: Σύνταξη “postAfm()” (2)……...86
Σχήμα 4.23: Επιλογή προς εγγραφή ενός Χρήστη στην ηλεκτρονική πλατφόρμα myData [30]……...88
Σχήμα 4.24: Εγγραφή Χρήστη στο myData [30]……...88
Σχήμα 4.25: Χρήση των διαπιστευτηρίων του Χρήστη ως κεφαλίδες των HTTP αιτημάτων προς ταυτοποίηση [30]……89
Σχήμα 4.26: Φόρμα εγγραφής στο δοκιμαστικό περιβάλλον του myData [30]……90
Σχήμα 4.27: Πραγματοποίηση εγγραφής στο δοκιμαστικό περιβάλλον του myData……90
Σχήμα 4.28: Δομή του περιεχομένου του παραστατικού [30]……91
Σχήμα 4.29: Πεδία παραστατικού [30]……...91
Σχήμα 4.30: Χρησιμοποιούμενα είδη παραστατικών της Α.Α.Δ.Ε. [30]……...92
Σχήμα 4.31: Δομή πεδίου paymentMethods [30]……...92
Σχήμα 4.32: Σώμα του αιτήματος της μεθόδου /SendInvoice (1)……..93
Σχήμα 4.33: Σώμα του αιτήματος της μεθόδου /SendInvoice (2)……..94

Κεφάλαιο 1ο: Εισαγωγή

Η ηλεκτρονική τιμολόγηση προϊόντων και υπηρεσιών αποτελεί πλέον αναπόσπαστο κομμάτι της εύρυθμης λειτουργίας των επιχειρήσεων και των επαγγελματιών κάθε κλάδου. Πρόκειται για μία διαδικασία ανταλλαγής ψηφιακών παραστατικών μεταξύ δύο ή ακόμη και περισσότερων οντοτήτων που συμμετέχουν σε μία εμπορική συναλλαγή [1].

Η ηλεκτρονική τιμολόγηση, πέρα από το γεγονός ότι αποτελεί ισχυρό εργαλείο απλοποίησης των συναλλαγών, συμβάλλει και στη πάταξη της φοροδιαφυγής, καθώς με σχετική απόφαση της Ελληνικής Κυβέρνησης, τα εκδοθέντα παραστατικά πρέπει να διαβιβάζονται απευθείας στην Ανεξάρτητη Αρχή Δημοσίων Εσόδων (Α.Α.Δ.Ε.) [2].

Έτσι, για τη ψηφιοποίηση των συναλλαγών και της απευθείας διαβίβασής τους στην αρμόδια Αρχή, δημιουργείται η ανάγκη ανάπτυξης ειδικών διαδικτυακών εφαρμογών και λογισμικών, εύκολων ως προς τη χρήση και διαχείρισή τους από την εκάστοτε επιχείρηση ή τον εκάστοτε επαγγελματία χρήστη.

1.1 Αντικείμενο Εργασίας

Η παρούσα πτυχιακή εργασία έχει ως αντικείμενό της τη δημιουργία και ανάπτυξη μίας διαδικτυακής εφαρμογής τιμολόγησης. Η εκάστοτε επιχείρηση ή ο εκάστοτε επαγγελματίας χρήστης θα έχει τη δυνατότητα με την εφαρμογή αυτή:

1. Να καταχωρεί νέους πελάτες και να διαγράφει υπάρχοντες πελάτες.
2. Να ανανεώνει τα στοιχεία πελατών.
3. Να αναζητά πελάτες.
4. Να εκδίδει Τιμολόγια Απόδειξης Λιανικής Πώλησης, Πώλησης και Παροχής Υπηρεσιών.
5. Να βλέπει όλα τα εκδοθέντα παραστατικά σε μία λίστα.
6. Να προβάλει παραστατικά και να τα εκτυπώνει.
7. Να ακυρώνει εκδοθέντα παραστατικά.

1.2 Στόχος Εργασίας

Στόχο της εργασίας αυτής αποτελεί η δημιουργία μίας διαδικτυακής εφαρμογής, εύκολης στη χρήση και στη διαχείριση από τον εκάστοτε χρήστη για τη πραγματοποίηση των απλών καθημερινών διεργασιών. Χωρίς περίπλοκες διαδικασίες, μεταβάσεις από σελίδα σε σελίδα και μη λειτουργικό περιβάλλον.

Το επιθυμητό είναι να προσφέρεται και να παρέχεται στον χρήστη ένα εύχρηστο, απλό, γρήγορο, εύκολα προσβάσιμο και διαχειρίσιμο περιβάλλον, το οποίο θα διευκολύνει τη λειτουργία της επιχείρησής του, θα τού επιτρέπει να έχει τον έλεγχό της, καθώς και θα φροντίζει να τηρούνται όλα τα νόμιμα βάσει νομοθεσίας.

1.3 Δομή Εργασίας

Η εν λόγω πτυχιακή εργασία αποτελείται από έξι (6) κύρια κεφάλαια, το κάθε ένα από τα οποία πραγματεύεται και μία διαφορετική της πτυχή. Πιο συγκεκριμένα, τα κεφάλαια είναι τα εξής παρακάτω:

Κεφάλαιο 1ο: Εισαγωγή
Το πρώτο κεφάλαιο, όπως και βλέπουμε, εξηγεί στους αναγνώστες του το αντικείμενο, τον στόχο και τη δομή της πτυχιακής εργασίας.

Κεφάλαιο 2ο: Μέθοδοι Ανάπτυξης Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης
Το δεύτερο κεφάλαιο αρχικά παρουσιάζει συνοπτικά τις μεθόδους που χρησιμοποιούνται για την ανάπτυξη μίας διαδικτυακής εφαρμογής γενικότερα.

Κεφάλαιο 3ο: Υλοποίηση Front – End Μέρους Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης
Το τρίτο κεφάλαιο υποδεικνύει τη χρησιμοποιούμενη τεχνολογία ανάπτυξης του Front – End μέρους, όπως και περιγράφει τη διαδικασία υλοποίησης του Front – End μέρους της Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης και όλων των προσφερόμενων δυνατοτήτων.

Κεφάλαιο 4ο: Υλοποίηση Back – End Μέρους Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης
Το τέταρτο κεφάλαιο υποδεικνύει τη χρησιμοποιούμενη τεχνολογία ανάπτυξης του Back – End μέρους, όπως και περιγράφει τη διαδικασία υλοποίησης του Back – End μέρους της Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης και όλων των λειτουργιών του.

Κεφάλαιο 5ο: Συμπεράσματα και Πρόταση Βελτίωσης
Το πέμπτο κεφάλαιο παρουσιάζει τα συμπεράσματα στα οποία έχουμε καταλήξει μετά την υλοποίηση και τη χρήση της Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης, καθώς και υποδεικνύει μία πρόταση βελτίωσής της.

Κεφάλαιο 2ο: Μέθοδοι Ανάπτυξης Διαδικτυακών Εφαρμογών

Στο παρόν κεφάλαιο θα παρουσιάσουμε συνοπτικά όλες τις διαθέσιμες μεθόδους που μπορούν να χρησιμοποιηθούν για την ανάπτυξη μίας διαδικτυακής εφαρμογής, προετοιμάζοντας το έδαφος για το επόμενο κεφάλαιο, στο οποίο και θα επιλέξουμε τις καταλληλότερες για την ανάπτυξη της εφαρμογής της παρούσας πτυχιακής εργασίας.

2.1 Διαδικτυακές Εφαρμογές

Ως Διαδικτυακή Εφαρμογή (Web Application) ορίζεται ένα διαδραστικό πρόγραμμα, που εκτελείται σε έναν Διακομιστή Ιστού (Web Server) και είναι προσβάσιμο μέσω ενός Προγράμματος Περιήγησης Ιστού (Web Browser). Κατασκευάζεται έτσι, ώστε η Διεπαφή Χρήστη (User Interface) να παρέχει δεδομένα – για τη χρήση της ή για τη βελτιστοποίησή της – στον ειδικό που την έχει σχεδιάσει [3].

Η ανάπτυξη μίας Διαδικτυακής Εφαρμογής και – κατ' επέκταση – ο σχεδιασμός της Διεπαφής Χρήστη πρέπει πάντοτε να ξεκινάνε έχοντας κατά νου τους στόχους και τον σκοπό της, όπως και τις ανάγκες που αυτή πρέπει να καλύπτει. Η επιλογή του σωστού σχεδιασμού καθορίζει την περαιτέρω ανάπτυξη της εφαρμογής, της αξιοπιστίας και της διαλειτουργικότητάς της [3].

2.1.1 Ορισμός και Δομή Αρχιτεκτονικής Διαδικτυακής Εφαρμογής

Γενικότερα, ως Αρχιτεκτονική Λογισμικού ορίζεται η απεικόνιση της δομής ενός συστήματος, των κύριων στοιχείων και της διάταξής τους, το σύνολο των οποίων καθορίζει τον τρόπο με τον οποίο συνδέονται και επικοινωνούν μεταξύ τους [12].

Αντιστοίχως, ως Αρχιτεκτονική Διαδικτυακής Εφαρμογής ορίζεται η απεικόνιση όλων των στοιχείων που την απαρτίζουν, όπως Βάσεις Δεδομένων και Ενδιάμεσο Λογισμικό (Middleware), και του τρόπου που αυτά αλληλεπιδρούν μεταξύ τους. Πιο συγκεκριμένα:

- Καθορίζει τον τρόπο παράδοσης των δεδομένων μέσω HTTP, διασφαλίζοντας ότι οι Διακομιστές Πελάτη (Client – Side Server) και Υποστήριξης (Back – End Server) τα κατανοούν.
- Διασφαλίζει την εγκυρότητα των δεδομένων σε όλα τα αιτήματα των χρηστών.
- Δημιουργεί και διαχειρίζεται εγγραφές, παρέχοντας πρόσβαση και έλεγχο ταυτότητας βάσει αδειών [11].

Συνήθως, η Αρχιτεκτονική μίας Διαδικτυακής Εφαρμογής περιλαμβάνει τα εξής βασικά στοιχεία:

- **Το Πρόγραμμα Περιήγησης Ιστού** (Web Browser).
  - Αποτελεί το κυριότερο στοιχείο με το οποίο αλληλεπιδρά ο χρήστης.
  - Λαμβάνει τα δεδομένα εισόδου.
  - Ελέγχει τις αλληλεπιδράσεις του χρήστη με την εφαρμογή.
- **Τον Διακομιστή Ιστού** (Web Server).
  - Επεξεργάζεται τα αιτήματα των χρηστών, δρομολογώντας τα στο σωστό στοιχείο και έχοντας τη διαχείριση του συνόλου των λειτουργιών της εφαρμογής.

- **Τον Διακομιστή Βάσης Δεδομένων** (Database Server).
  - Αποθηκεύει και παρέχει τα απαιτούμενα – για την εφαρμογή – δεδομένα.
- **Τη Διεπαφή Προγραμματισμού Εφαρμογών** (Application Programming Interface – API).
  - Αποστέλλει τα δεδομένα που καταχωρεί ο χρήστης στον Διακομιστή Ιστού (Web Server) και τα αποστέλλει πίσω σε αυτόν, με τα επιθυμητά – από τον χρήστη – αποτελέσματα [11–13].

2.1.2 Αρχιτεκτονική Επιπέδων

Οι Διαδικτυακές Εφαρμογές αποτελούνται από πολλαπλά επίπεδα, κάθε ένα από τα οποία είναι υπεύθυνο για συγκεκριμένες λειτουργίες. Ανάλογα με τη δομή μίας Διαδικτυακής Εφαρμογής και των επιπέδων από τα οποία αυτή αποτελείται, η ακολουθούμενη Αρχιτεκτονική μπορεί να είναι Δύο (2), Τριών (3) ή Πολλαπλών Επιπέδων [12].

2.1.2.1 Αρχιτεκτονική Δύο Επιπέδων

Σε μία Διαδικτυακή Εφαρμογή Αρχιτεκτονικής Δύο (2) Επιπέδων, υπάρχουν δύο στοιχεία: το Πρόγραμμα Περιήγησης Ιστού (Web Browser) και ένας Διακομιστής Βάσης Δεδομένων (Database Server). Στη περίπτωση αυτή, η επεξεργασία και ανακατεύθυνση των αιτημάτων του χρήστη ενσωματώνεται σε ένα από τα δύο ανωτέρω στοιχεία. Το κυριότερο μειονέκτημα της Αρχιτεκτονικής αυτής είναι ότι με αυξημένο αριθμό χρηστών, η απόδοση μειώνεται, ενώ και η άμεση αλληλεπίδραση της Βάσης Δεδομένων και της εκάστοτε συσκευής του εκάστοτε χρήστη εγείρει ορισμένες ανησυχίες για την ασφάλεια. Γι’ αυτό δε προτιμάται η χρήση της [11].

2.1.2.2 Αρχιτεκτονική Τριών Επιπέδων

Αντιθέτως, όπως βλέπουμε και στο Σχήμα 2.1, μία τυπική Διαδικτυακή Εφαρμογή αποτελείται από τρία (3) επίπεδα:

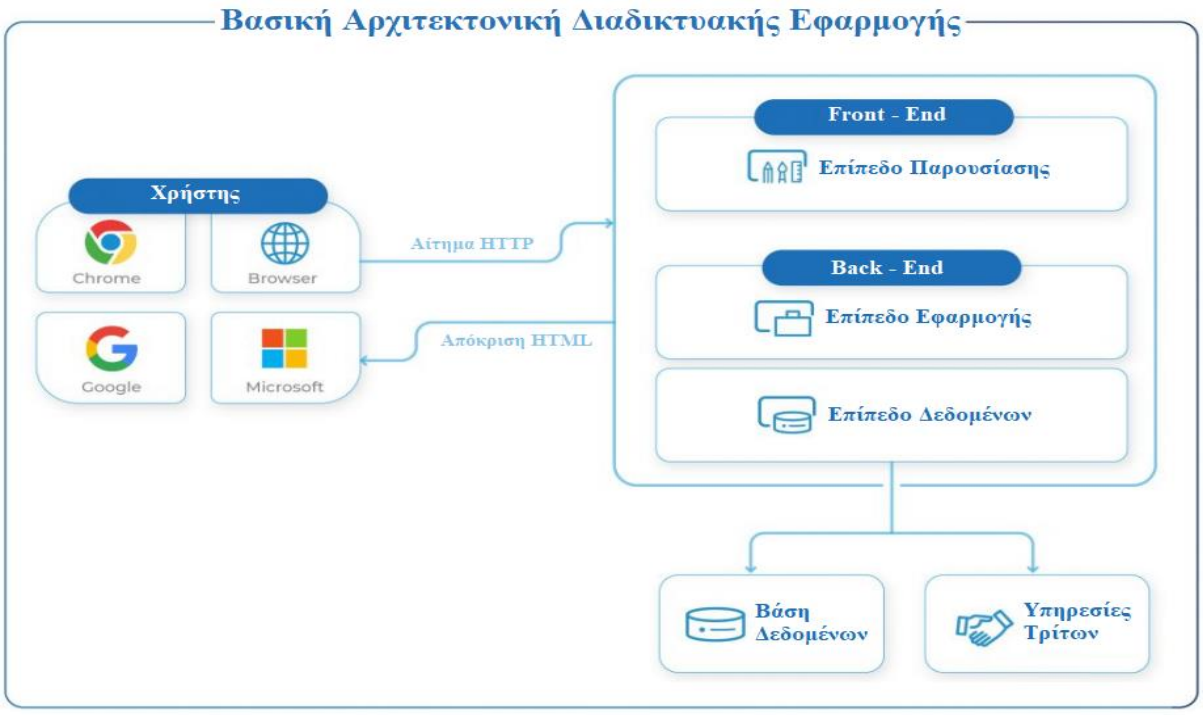
1. Επίπεδο Παρουσίασης,
2. Επίπεδο Εφαρμογής, και
3. Επίπεδο Δεδομένων [12].

Σχήμα 2.1: Αρχιτεκτονική Διαδικτυακής Εφαρμογής Τριών (3) Επιπέδων [12].

Μία Διαδικτυακή Εφαρμογή αποτελείται από το Front – End μέρος, το οποίο περιλαμβάνει όλες τις οπτικές πτυχές μίας Διαδικτυακής Εφαρμογής, με τις οποίες μπορεί να αλληλεπιδράσει ο χρήστης, όπως τα χρώματα, τη διάταξη και τις γραμματοσειρές, και το Back – End μέρος, το οποίο δημιουργεί την «αόρατη» δομή που καθιστά δυνατή την ορθή λειτουργία του Front – End μέρους και του συνόλου των λειτουργιών της Διαδικτυακής Εφαρμογής. Όπως φαίνεται στο Σχήμα 2.2, το Front – End μέρος υλοποιείται στο Επίπεδο Παρουσίασης, το οποίο χειρίζεται τις αλληλεπιδράσεις των χρηστών με την εφαρμογή. Το Back – End μέρος υλοποιείται στο Επίπεδο Εφαρμογής, το οποίο ενορχηστρώνει το σύνολο των εργασιών της εφαρμογής και διαχειρίζεται όλα τα αιτήματα των χρηστών, και στο Επίπεδο Δεδομένων, το οποίο διαχειρίζεται την αποθήκευση και τη διαχείριση των δεδομένων της εφαρμογής [4][11][12].

Η Αρχιτεκτονική Τριών (3) Επιπέδων είναι ασφαλέστερη, καθώς:

- Ο χρήστης δεν έχει άμεση πρόσβαση στα δεδομένα.
- Η ακεραιότητα των δεδομένων βελτιώνεται, καθώς όλα τα δεδομένα περνούν μέσω του Διακομιστή Ιστού (Web Server), ο οποίος αποφασίζει πώς και από ποιον πρέπει να έχει πρόσβαση στα δεδομένα.
- Μπορεί να αναπτύσσεται περαιτέρω, αναπτύσσοντας κάθε στοιχείο και επίπεδο της εφαρμογής ανεξάρτητα από τα υπόλοιπα.
- Παρέχει υψηλότερη επεκτασιμότητα, καλύτερη απόδοση και επαναχρησιμοποίηση των στοιχείων της εφαρμογής [11].



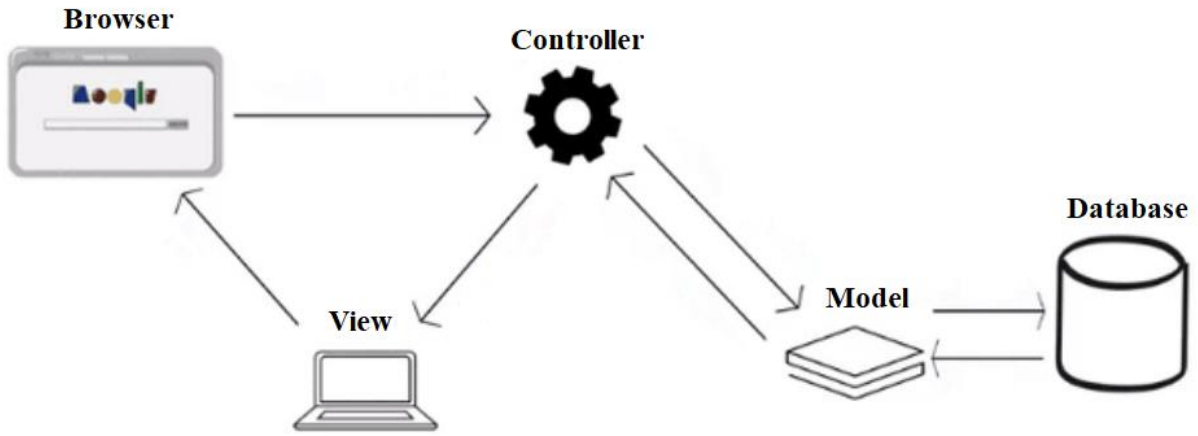
Σχήμα 2.2: Βασική Αρχιτεκτονική Διαδικτυακής Εφαρμογής [11].

2.1.3 Αρχιτεκτονική MVC

Η Αρχιτεκτονική MVC αποτελείται από τρία (3) στοιχεία:

1. Model (M),
2. View (V), και
3. Controller (C) [15].

Το *στοιχείο Model* αποτελείται από μία Βάση Δεδομένων και είναι υπεύθυνο για τη διαχείριση των δεδομένων, καθώς και του συνόλου των κανόνων και λειτουργιών της Διαδικτυακής Εφαρμογής. Το στοιχείο Model απαντά στο αίτημα που γίνεται από το στοιχείο View και την οδηγία που δίνεται από το στοιχείο Controller, προκειμένου αυτό να ενημερωθεί εκ νέου [15]. Το *στοιχείο View* είναι υπεύθυνο για την εμφάνιση των δεδομένων της Διαδικτυακής Εφαρμογής στους χρήστες. Είναι, ουσιαστικά, η Διεπαφή Χρήστη (User Interface, UI) που βοηθά στην απόδοση των απαιτούμενων δεδομένων στον χρήστη. Το *στοιχείο Controller* είναι υπεύθυνο για την επικοινωνία και την αλληλεπίδραση μεταξύ των στοιχείων Model και View. Παρέχει διάφορες λειτουργίες βάσει συμβάντων που μπορούν να ενεργοποιηθούν. Εν συνέχεια, έρχεται σε επαφή με το στοιχείο Model προς ανάγνωση ή/και ενημέρωση δεδομένων. Τέλος, παρουσιάζει τις νέες πληροφορίες στο στοιχείο View, ούτως ώστε να τις εμφανίσει στον χρήστη [15].



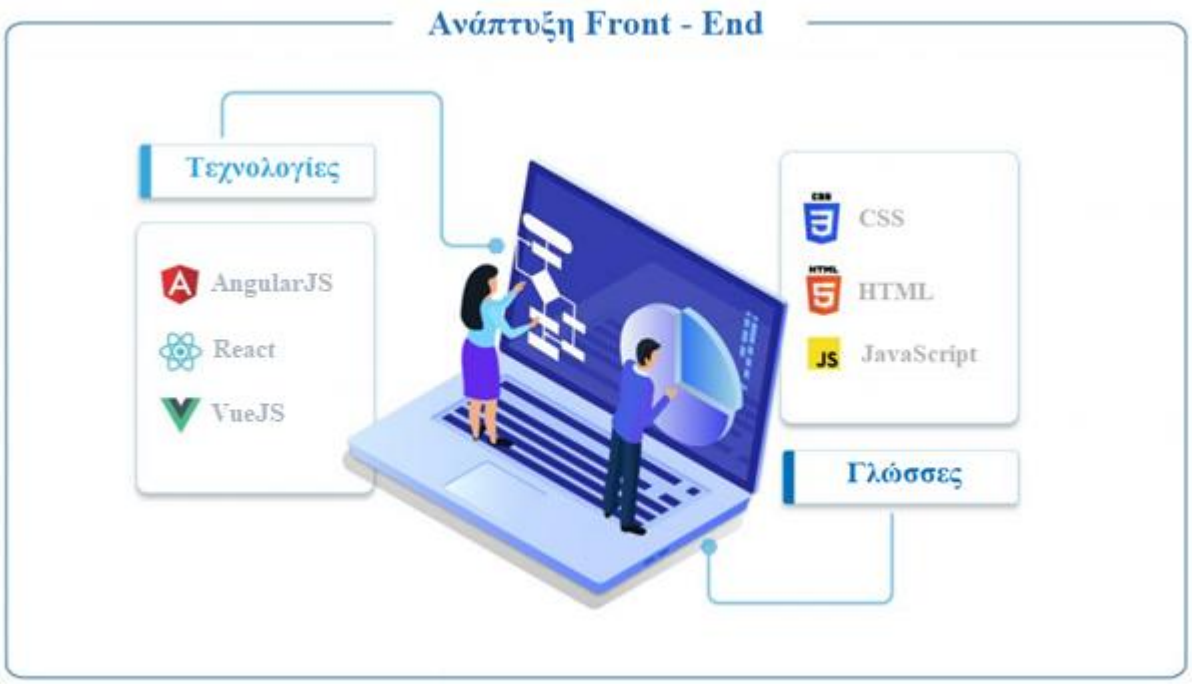
Σχήμα 2.3: Αρχιτεκτονική MVC [15].

Η ροή λειτουργίας της Αρχιτεκτονικής MVC είναι η εξής:

- Ο χρήστης αλληλεπιδρά με το Πρόγραμμα Περιήγησης Ιστού (Web Browser).
- Ενεργοποιείται ένα συμβάν.
- Το Πρόγραμμα Περιήγησης Ιστού αποστέλλει ένα αίτημα HTTP στον Διακομιστή Ιστού (Web Server).
- Το αίτημα HTTP από το Πρόγραμμα Περιήγησης Ιστού λαμβάνεται από τον Δρομολογητή (Router), ο οποίος και το κατευθύνει στο στοιχείο Controller.
- Το στοιχείο Controller αλληλεπιδρά με το στοιχείο Model για να αναγνώσει ή/και να τροποποιήσει δεδομένα από τη Βάση Δεδομένων (Database).
- Οι νέες πληροφορίες μεταβιβάζονται στο στοιχείο View, το οποίο τροποποιεί αναλόγως τη κατάστασή του, και τις στέλνει πίσω στο Πρόγραμμα Περιήγησης Ιστού για να τις δει ο χρήστης [15].

2.2 Τεχνολογίες Ανάπτυξης Front – End Μέρους Διαδικτυακής Εφαρμογής

Το Front – End μέρος μίας Διαδικτυακής Εφαρμογής επιτρέπει στους χρήστες να αλληλεπιδρούν με τον Διακομιστή Ιστού (Web Server) και την Υπηρεσία Υποστήριξης (Back – End Service) μέσω ενός Προγράμματος Περιήγησης (Web Browser). Η ανάπτυξή του εστιάζει στη πλευρά την οποία βλέπει ο χρήστης. Υψίστης σημασίας για την εμπειρία του χρήστη είναι να διασφαλίζεται ότι μπορεί να αλληλεπιδρά και να περιηγείται εύκολα σε αυτήν, χρησιμοποιώντας τις κατάλληλες γλώσσες προγραμματισμού, διατάξεις και άλλα χρήσιμα εργαλεία, όπως αναπτυσσόμενα μενού και σχέδια. Στο Σχήμα 2.4 που ακολουθεί, απεικονίζονται οι τεχνολογίες και οι γλώσσες προγραμματισμού που χρησιμοποιούνται για την ανάπτυξη του Front – End μέρους, οι οποίες και αναλύονται παρακάτω [4][11].



Σχήμα 2.4: Τεχνολογίες και Γλώσσες Ανάπτυξης Front – End μέρους Διαδικτυακής Εφαρμογής [11].

Frameworks Διαδικτυακών Εφαρμογών

Ένα ουσιαστικό βήμα για την ανάπτυξη μίας Διαδικτυακής Εφαρμογής είναι η επιλογή του κατάλληλου Front – End Πλαισίου (Framework). Πρόκειται για ένα σύνολο προ – γραμμένου κώδικα, το οποίο παρέχει μία επεκτάσιμη και διατηρήσιμη δομή για τη δημιουργία αποτελεσματικών Διεπαφών Χρήστη, και περιέχει στοιχεία HTML, CSS και JavaScript, τα οποία μπορούν να επαναχρησιμοποιούνται, διατηρώντας μία συνεπή και οργανωμένη βάση κώδικα [5].

Για τη δημιουργία Διαδικτυακών Εφαρμογών υπάρχουν τρία Πλαίσια: το AngularJS, η React και το VueJS. Αυτά περιέχουν τυποποιημένο και προ – γραμμένο κώδικα, ο οποίος διευκολύνει, επιταχύνει και αναπτύσσει τη Διαδικτυακή Εφαρμογή σε ελκυστική. Έχουν, ωστόσο, διαφορετική δομή και αρχιτεκτονική. Σημαντικό είναι να διαχωρίσουμε ότι η React είναι μία Front – End Βιβλιοθήκη JavaScript, ενώ τα Vue και Angular είναι Πλαίσια JavaScript. Ο διαχωρισμός έγκειται στην ύπαρξη ελέγχου ή μη των περιεχομένων του Πλαισίου από τον χρήστη. Μία Βιβλιοθήκη

(Library) είναι ένα σύνολο από κλάσεις και συναρτήσεις, τον πλήρη έλεγχο του οποίου έχει ο χρήστης και μπορεί να το καλεί. Αντιθέτως, ένα Πλαίσιο (Frame) ορίζεται ως ένα μοντέλο, με περιορισμένες επιλογές και προκαθορισμένο σχέδιο, με αποτέλεσμα ο χρήστης να έχει μερικό μόνο έλεγχο [5].

2.2.1 HTML

Η γλώσσα HTML (HyperText Markup Language), δηλαδή Γλώσσα Χαρακτηρισμού Υπερ – Κειμένου, αποτελεί τη βασική γλώσσα με την οποία πραγματοποιείται η δόμηση των σελίδων του Παγκόσμιου Ιστού. Αναπτύχθηκε το 1990 από τον Tim Berners – Lee από το Ευρωπαϊκό Επιστημονικό Εργαστήριο Μοριακής Φυσικής της Γενεύης CERN, όταν δημιούργησε το πρωτόκολλο HTTP (HyperText Transfer Protocol), με το οποίο μπορούμε να μεταφέρουμε κάθε είδους πληροφορία μέσα στο Διαδίκτυο [32].

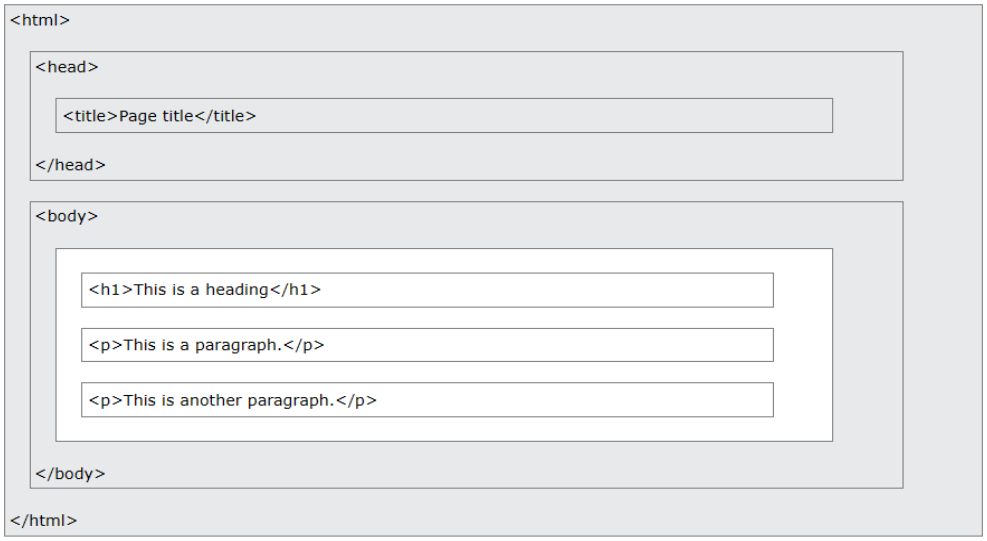
Η HTML δεν αποτελεί μία γλώσσα προγραμματισμού, αλλά μία περιγραφική γλώσσα. Είναι ένας ειδικός τρόπος γραφής κειμένου. Ορίζει ένα σύνολο στοιχείων για τις Web σελίδες, όπως:

- τίτλους (titles),
- επικεφαλίδες (headings),
- παραγράφους (paragraphs),
- λίστες (lists), και
- πίνακες (tables) [32].

Κάθε στοιχείο έχει ένα όνομα και περιέχεται μέσα στα σύμβολα < >, τα οποία αποκαλούνται ετικέτες (tags). Κάθε αρχείο HTML αποτελείται από δύο ενότητες: τη κεφαλή (head) και το σώμα (body) της σελίδας. Η κεφαλή (head) εμπεριέχει την ετικέτα που καθορίζει τον τίτλο της σελίδας, η οποία εμφανίζεται στο επάνω μέρος του παραθύρου του Φυλλομετρητή, ενώ το σώμα (body) περιλαμβάνει το κυρίως περιεχόμενο, μέσα στο οποίο γράφουμε το κείμενο που θέλουμε να εμφανίζεται στη σελίδα, μαζί με τις όποιες ετικέτες (tags) που μορφοποιούν το εν λόγω κείμενο. Όταν, λοιπόν, ο Φυλλομετρητής (Browser) ανακτά μία ιστοσελίδα, ανακτά ουσιαστικά τον κώδικα HTML αυτής της ιστοσελίδας. Έπειτα «διαβάζει» και «διερμηνεύει» τις ετικέτες της HTML, δημιουργεί την ιστοσελίδα και – μορφοποιώντας κατάλληλα το κείμενο και τις εικόνες της – την εμφανίζει στην οθόνη [32].

Η HTML είναι μία ευρέως χρησιμοποιούμενη γλώσσα στη δημιουργία δομής ενός ιστότοπου, καθώς:

- Είναι εύκολη στην εκμάθηση, σύνταξη και υλοποίηση.
- Επεξεργάζεται ως απλό κείμενο.
- Την υποστηρίζει κάθε πρόγραμμα περιήγησης.
- Επιτρέπει την αποθήκευση μεγάλων αρχείων.
- Συνεργάζεται εύκολα με άλλες γλώσσες, όπως CSS και JavaScript.
- Διαθέτει πολλές ετικέτες (tags) και χαρακτηριστικά (attributes), που μπορούν να συντομεύσουν τον συνολικό κώδικα μίας ιστοσελίδας [34].

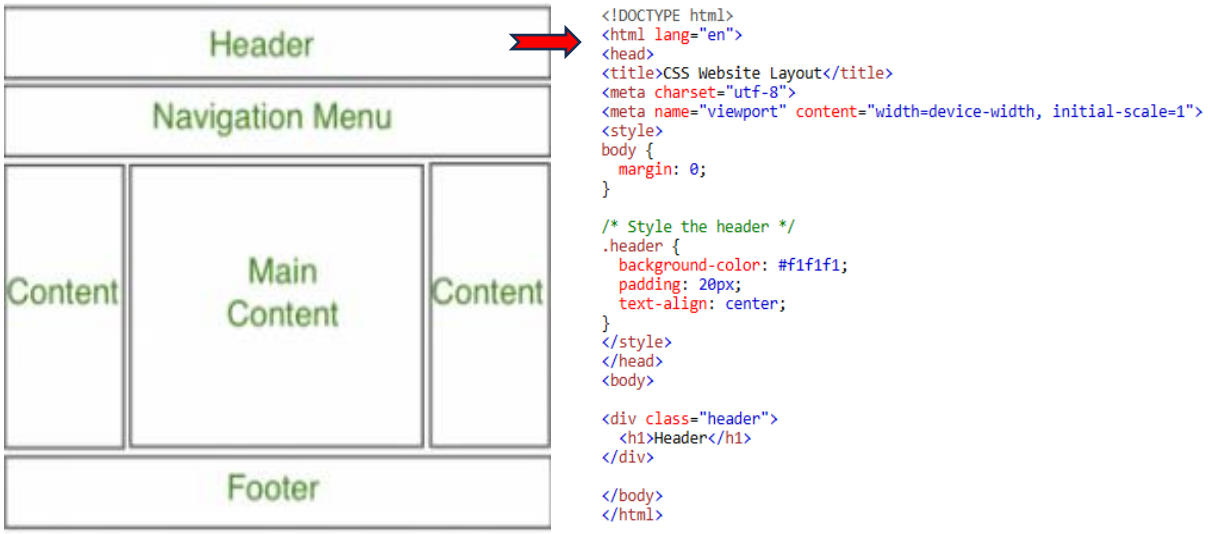


Σχήμα 2.5: Δομή σελίδας HTML [33].

2.2.2 CSS

Το CSS (Cascading Style Sheets, Διαδοχικά Φύλλα Στυλ) είναι μία γλώσσα, η οποία χρησιμοποιείται για τη περιγραφή της παρουσίασης ενός εγγράφου γραμμένου σε μία γλώσσα σήμανσης, όπως είναι η HTML ή η XML. [1] Αποτελεί μία τεχνολογία ακρογωνιαίο λίθο του Παγκόσμιου Ιστού, μαζί με την HTML που είδαμε παραπάνω και τη JavaScript που ακολουθεί παρακάτω [35].

Το CSS χειρίζεται το κομμάτι της εμφάνισης και της αισθητικής μίας ιστοσελίδας. Χρησιμοποιώντας το, μπορούμε να ελέγξουμε το χρώμα του κειμένου, το στυλ των γραμματοσειρών, την απόσταση μεταξύ των παραγράφων, το μέγεθος και τη διάταξη των στηλών, ποιες εικόνες φόντου ή χρώματα χρησιμοποιούνται, σχέδια διάταξης, παραλλαγές στην οθόνη για διαφορετικές συσκευές και μεγέθη οθόνης, καθώς και μία ποικιλία άλλων εφέ [37].



Σχήμα 2.6: Παράδειγμα κώδικα HTML χρησιμοποιώντας το CSS [38].

Το CSS χρησιμοποιείται κατά κόρον για τη δημιουργία ιστοσελίδων και εφαρμογών, αφού:

- Είναι μία απλή sheet style γλώσσα.
- Προσφέρει μεγαλύτερη ευελιξία και καλύτερα αποτελέσματα, καθώς καθίστανται πλέον δυνατές μορφοποιήσεις, οι οποίες με τη χρήση μόνο της HTML ήτο δύσκολες ή και αδύνατες.
- Καθιστά πιο ευανάγνωστο, συντομότερο και λιγότερο πολύπλοκο τον κώδικα HTML.
- Δίνει μεγαλύτερο έλεγχο στον χρήστη σε ό,τι αφορά τη σχεδίαση του στυλ της ιστοσελίδας.
- Προσφέρει τη δυνατότητα χρήσης πολλαπλών CSS αρχείων για μία ιστοσελίδα, καθιστώντας τη συντήρηση της ιστοσελίδας ευκολότερη σε περίπτωση επιθυμίας εφαρμογής αλλαγών [36].

2.2.3 JavaScript

Η JavaScript είναι μία δυναμική γλώσσα προγραμματισμού υψηλού επιπέδου, η οποία χρησιμοποιείται κυρίως για τη δημιουργία διαδραστικών Web Εφαρμογών Διεπαφών Χρήστη (UI). Δημιουργήθηκε τη δεκαετία του 1990 από τον Brendan Eich και αποτελεί μία από τις δημοφιλέστερες και ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού. Ενώ οι HTML και CSS δομούν και σχεδιάζουν τη δομή μίας ιστοσελίδας, η JavaScript προσφέρει διαδραστικά στοιχεία, ώστε να μπορεί ο χρήστης να αλληλεπιδρά με αυτή. Παρέχει σε προγραμματιστές και σχεδιαστές πάρα πολλές δυνατότητες, όπως να δημιουργούν κινούμενες εικόνες ή να ενημερώνουν αυτόματα το περιεχόμενο σε μία ιστοσελίδα [39].

Η JavaScript μπορεί να χρησιμοποιηθεί τόσο από τη πλευρά του Πελάτη (Client – Side) όσο και από τη πλευρά του Διακομιστή (Server – Side). Υπάρχει, λοιπόν, πληθώρα γνωστών Frameworks (Πλαισίων) και Βιβλιοθηκών (Libraries) της JavaScript, που συμβάλλουν στη δημιουργία των Front – End και Back – End κομματιών. Πιο συγκεκριμένα, χρησιμοποιούμενα Frameworks για την ανάπτυξη Front – End είναι οι VueJS και AngularJS και Βιβλιοθήκη η React, ενώ για την ανάπτυξη Back – End χρησιμοποιείται η πλατφόρμα NodeJS, τα οποία και θα δούμε παρακάτω [39].

Χρησιμοποιώντας τη JavaScript:

- Βελτιστοποιείται η εμπειρία των χρηστών ως προς την αλληλεπίδρασή τους με κάποια ιστοσελίδα ή εφαρμογή με τη χρήση κινούμενων εικόνων, αναδυόμενων παραθύρων και άλλων πολλών διαδραστικών λειτουργιών.
- Προσφέρεται πρόσβαση σε πόρους, Frameworks και Βιβλιοθήκες που υπάρχουν διαθέσιμα μέσω της τεράστιας κοινότητας προγραμματιστών που έχει αναπτυχθεί και συνεχίζει να αναπτύσσεται [39].

2.2.4 AngularJS

Το AngularJS είναι ένα δημοφιλές Πλαίσιο ανοιχτού κώδικα JavaScript, το οποίο αναπτύχθηκε το 2008 – 2009 από τους Misko Hevery και Adam Abrons και τώρα υποστηρίζεται και συντηρείται από την Google. Χρησιμοποιείται κυρίως για την ανάπτυξη ισχυρών Διαδικτυακών Εφαρμογών Μονής Σελίδας (Single – Page Applications, SPAs), βασιζόμενων στην Αρχιτεκτονική MVC. Αναπτύσσεται και διευρύνεται συνεχώς από χιλιάδες προγραμματιστές ανά τον κόσμο, παρέχοντας

πολύ καλούς τρόπους ανάπτυξης Διαδικτυακών Εφαρμογών. Είναι δωρεάν και μπορεί να χρησιμοποιηθεί και να τροποποιηθεί από οποιονδήποτε [19].

Το Πλαίσιο AngularJS είναι εύκολο στη χρήση, καθώς χρησιμοποιεί βασικά στοιχεία των γλωσσών HTML, CSS και Javascript. Ειδικότερα, χρησιμοποιεί το απλό πρότυπο HTML, το οποίο περνά στο DOM, και έπειτα στον μεταγλωττιστή AngularJS. Διασχίζει τα πρότυπα, τα οποία τελικά είναι έτοιμα για χρήση. Επιτρέπει οι προγραμματιστές να εργάζονται με στοιχεία, τα οποία μπορούν να επαναχρησιμοποιήσουν, εξοικονομώντας έτσι και χρόνο και κώδικα. Αυτά είναι μόνο ορισμένα από τα χαρακτηριστικά του εν λόγω Πλαισίου [19].

Τα κυριότερα χαρακτηριστικά που κάνουν ελκυστική τη χρήση του Πλαισίου AngularJS στην ανάπτυξη Διαδικτυακών Εφαρμογών είναι τα εξής:

Αρχιτεκτονική MVC

Το Πλαίσιο AngularJS χρησιμοποιεί την Αρχιτεκτονική MVC (Model – View – Controller), η οποία – όπως προείπαμε – αποτελείται από τα στοιχεία Model, View και Controller, που αφορούν τα δεδομένα, τη Διεπαφή Χρήστη και τις διεργασίες της εκάστοτε Διαδικτυακής Εφαρμογής, αντίστοιχα. Για τη σύνδεσή τους, ωστόσο, απαιτείται σύνταξη κατάλληλου κώδικα από τον εκάστοτε προγραμματιστή. Η ιδιαιτερότητα του Πλαισίου AngularJS έγκειται στην ικανότητά του να διαχειρίζεται ανεξάρτητα καθένα από τα τρία (3) στοιχεία της Αρχιτεκτονικής MVC, όπως και να φροντίζει τη σύνδεσή τους χωρίς να χρειάζεται να επέμβει ο προγραμματιστής. Έτσι εξοικονομείται πολύς χρόνος και έκταση κώδικα [16][19].

Αμφίδρομη Σύνδεση Δεδομένων

Τυχόν πραγματοποιηθείσες αλλαγές στο στοιχείο Model της Αρχιτεκτονικής MVC αντικατοπτρίζονται αμέσως στο στοιχείο View και το αντίστροφο. Το εν λόγω χαρακτηριστικό εξαλείφει την ανάγκη των προγραμματιστών να γράψουν εκτεταμένο κώδικα για να συνδέσουν τα δύο προαναφερθέντα στοιχεία μεταξύ τους, ενώ βοηθά επίσης στη διατήρηση του συγχρονισμού μεταξύ του DOM (Document Object Model) και του στοιχείου Model, οδηγώντας σε βελτιωμένη απόδοση της Διαδικτυακής Εφαρμογής και μία απρόσκοπτη εμπειρία χρήστη [16].

Εισαγωγή Εξαρτήσεων

Πρόκειται για ένα ενσωματωμένο μοτίβο σχεδίασης, όπου μια Τάξη (Class) λαμβάνει τις Εξαρτήσεις (Dependencies) της από εξωτερικές πηγές αντί να τις δημιουργεί η ίδια. Έτσι, οι προγραμματιστές μπορούν να αναπτύξουν στοιχεία ως ανεξάρτητες οντότητες, τα οποία μπορούν έπειτα να συνδεθούν. Αυτό απλοποιεί σημαντικά τη διαδικασία ανάπτυξης Διαδικτυακών Εφαρμογών, καθιστώντας τον κώδικα ευκολότερο διαχειρίσιμο, δοκιμαζόμενο και επαναχρησιμοποιήσιμο [16].

Λειτουργικές Μονάδες

Οι Λειτουργικές Μονάδες (Modules) αποτελούν «αποθηκευτικό χώρο» για διαφορετικά μέρη της εκάστοτε Διαδικτυακής Εφαρμογής, συμβάλλοντας στην οργάνωση και αποτελεσματικότερη διαχείριση του κώδικα του Πλαισίου AngularJS. Κάθε λειτουργική Μονάδα προσδιορίζεται με ένα μοναδικό όνομα και μπορεί να εξαρτάται από άλλες Λειτουργικές Μονάδες. Η χρήση Λειτουργικών Μονάδων επιτρέπει την ανάπτυξη και δοκιμή μεμονωμένων στοιχείων της εκάστοτε Διαδικτυακής Εφαρμογής χωρίς να επηρεάζονται τα υπόλοιπα, όπως υποστηρίζει και προωθεί την

επαναχρησιμοποίηση κώδικα. Παραδείγματα Λειτουργικών Μονάδων αποτελούν οι Ελεγκτές (Controllers), οι Οδηγίες (Directives) και τα Φίλτρα (Filters) [16].

Ενοποίηση – Επεκτασιμότητα

Το Πλαίσιο AngularJS μπορεί εύκολα να ενσωματωθεί με άλλα Πλαίσια και άλλες Βιβλιοθήκες. Επίσης, παρέχει υποστήριξη για αλληλεπίδραση με API (Application Programming Interface) από τη πλευρά του Διακομιστή, χειρισμό αιτημάτων HTTP και ενσωμάτωση με εργαλεία τρίτων, ενώ επιτρέπει την επέκταση της λειτουργικότητάς του, δημιουργώντας προσαρμοσμένες Λειτουργικές Μονάδες, δίνοντας έτσι την ευελιξία να προσαρμόσει ο εκάστοτε προγραμματιστής το Πλαίσιο στις επιθυμητές για αυτόν ανάγκες [19].

Πλεονεκτήματα και Μειονεκτήματα

Τα πλεονεκτήματα χρήσης του Πλαισίου AngularJS στην ανάπτυξη Διαδικτυακών Εφαρμογών είναι τα εξής:

1. Αποτελεί δωρεάν Πλαίσιο ανοιχτού κώδικα JavaScript.
2. Χρησιμοποιεί και ελέγχει πλήρως την Αρχιτεκτονική MVC.
3. Υποστηρίζει την αμφίδρομη ανανέωση των στοιχείων Model και View της Αρχιτεκτονικής MVC.
4. Είναι εύκολα επεκτάσιμη, προσαρμοζόμενη και δοκιμαζόμενη.
5. Δεν απαιτείται εκμάθηση νέας γλώσσας, καθώς χρησιμοποιούνται οι γνωστές γλώσσες JavaScript και HTML.
6. Συμβάλει στην ανάπτυξη μίας αποκριτικής Διαδικτυακής Εφαρμογής.
7. Υποστηρίζει την ανάπτυξη Εφαρμογών Μονής Σελίδας (SAPs).
8. Υποστηρίζει την ανάπτυξη Εφαρμογών REST (Representational State Transfer) [18][19].

Ωστόσο, επιφέρει και σημαντικά μειονεκτήματα, επίσης:

1. Καθώς είναι Πλαίσιο της JavaScript, στηρίζεται αποκλειστικά στη γλώσσα JavaScript.
2. Δεν ακολουθεί συγκεκριμένο μοτίβο σχεδίασης και ανάπτυξης, κάνοντάς το δύσκολο στη παρακολούθηση στην εκμάθηση, καθώς και στον εντοπισμό σφαλμάτων.
3. Δεν υποστηρίζεται ευρέως, μη διαθέτοντας πλήρως ενεργή κοινότητα προγραμματιστών [19].

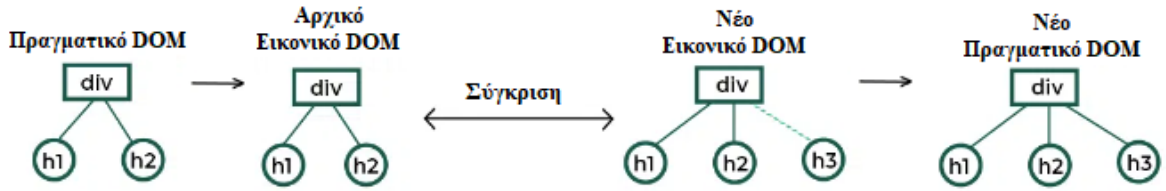
2.2.5 React

Η React, ή ReactJS, είναι μία δηλωτική, αποτελεσματική και ευέλικτη Βιβλιοθήκη JavaScript για τη δημιουργία Διεπαφών Χρήστη. Είναι ανοιχτού κώδικα, βασισμένη σε Στοιχεία (Components) και είναι υπεύθυνη μόνο για το στοιχείο View της Αρχιτεκτονικής MVC. Δεν αποτελεί Πλαίσιο, αλλά μία Βιβλιοθήκη της JavaScript που αναπτύχθηκε και συντηρείται από τη Meta (πρώην Facebook), καθώς και μία ενεργή κοινότητα ανεξάρτητων εταιρειών και προγραμματιστών. Η Βιβλιοθήκη React χρησιμοποιεί στοιχεία των γλωσσών HTML, CSS και JavaScript [21].

Τα κυριότερα χαρακτηριστικά που κάνουν τη React τη δημοφιλέστερη και περισσότερο χρησιμοποιούμενη Βιβλιοθήκη στην ανάπτυξη Διαδικτυακών Εφαρμογών είναι τα παρακάτω:

Εικονικό DOM

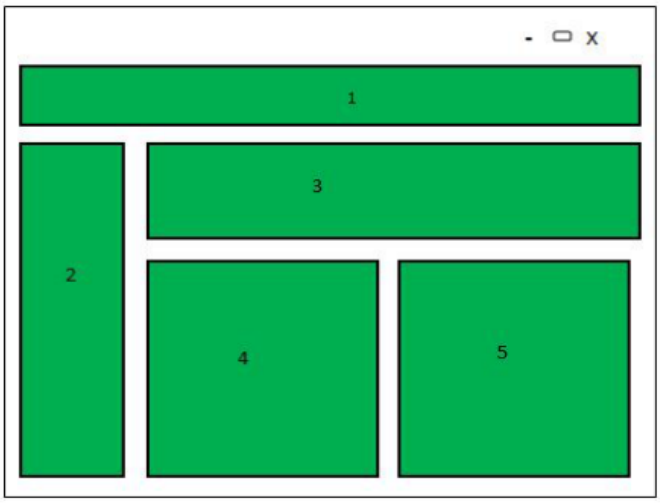
Το DOM (Document Object Model) αποτελεί το σημαντικότερο μέρος του Ιστού, το οποίο χωρίζεται σε λειτουργικές μονάδες και εκτελεί τον κώδικα. Συνήθως, τα Πλαίσια JavaScript ενημερώνει ολόκληρο το DOM ταυτόχρονα σε περίπτωση τροποποιήσεων στην εκάστοτε Διαδικτυακή Εφαρμογή, γεγονός που την καθιστά αργή. Η React, όμως, χρησιμοποιεί Εικονικό (Virtual) DOM, το οποίο αποτελεί ακριβές αντίγραφο του Πραγματικού (Actual) DOM. Κάθε φορά, λοιπόν, που υπάρχει μία τροποποίηση στη Διαδικτυακή Εφαρμογή, ενημερώνεται αρχικά ολόκληρο το Εικονικό DOM (New Virtual DOM). Έπειτα το Πραγματικό DOM βρίσκει τη διαφορά μεταξύ αυτού και του Εικονικού DOM και μόλις τη βρει, ενημερώνει μόνο το τμήμα που υπέστη τροποποίηση πρόσφατα, κρατώντας όλα τα υπόλοιπα μέρη ίδια (New Actual DOM) [22][23].



Σχήμα 2.7: Σύγκριση Πραγματικού και Εικονικού DOM [22].

Στοιχεία

Η React βασίζεται σε Στοιχεία (Components) για τη δημιουργία Διεπαφών Χρήστη. Επομένως, διαιρεί τη Διαδικτυακή Ιστοσελίδα σε πολλαπλά Στοιχεία (Components). Κάθε Στοιχείο έχει τον δικό του σχεδιασμό και τη δική του λογική, η οποία γράφεται σε JavaScript. Το γεγονός αυτό καθιστά τη λογική εύκολη και ταχύτερη, όπως και επαναχρησιμοποιήσιμο ολόκληρο το Στοιχείο [23].



Σχήμα 2.8: Πολλαπλά Στοιχεία (Components) [23].

Μονόδρομη Σύνδεση Δεδομένων

Στη React τα δεδομένα ρέουν μόνο από τα Γονικά Στοιχεία (Parent Components) στα Θυγατρικά Στοιχεία (Child Components), δηλαδή προς μία κατεύθυνση. Μπορούν, ωστόσο, τα Θυγατρικά

Στοιχεία να επικοινωνούν με τα Γονικά, ούτως ώστε να τροποποιήσουν τη κατάστασή τους σύμφωνα με τις τροποποιήσεις που πραγματοποιούνται, γεγονός που καθιστά τη συνολική λειτουργία της Διαδικτυακής Εφαρμογής γρήγορη [23].

Απόδοση

Όπως αναφέραμε παραπάνω, η React χρησιμοποιεί Εικονικό DOM και ενημερώνει μόνο τα τροποποιημένα μέρη του. Αυτό κάνει το DOM, το οποίο εκτελείται στη μνήμη, να λειτουργεί γρηγορότερα [23].

Απλότητα

Η Βιβλιοθήκη React χρησιμοποιεί τη JSX, έναν συνδυασμό των HTML και JavaScript, καθιστώντας εύκολη την κατανόηση του κώδικα και τον εντοπισμό σφαλμάτων, και εξασφαλίζοντας μικρότερη έκταση κώδικα [23].

Επεκτασιμότητα

Η React διαθέτει ποικίλες επεκτάσεις, όπως είναι οι Redux, React Native κ.ά., που μπορούν να χρησιμοποιηθούν για τη δημιουργία μίας πλήρους εφαρμογής Διεπαφής Χρήστη, καθώς υποστηρίζει την ανάπτυξη εφαρμογών για κινητές συσκευές και παρέχει απόδοση από την πλευρά του Διακομιστή (Server – Side Rendering) [23].

Πλεονεκτήματα και Μειονεκτήματα

Τα πλεονεκτήματα χρήσης της Βιβλιοθήκης React στην ανάπτυξη Διαδικτυακών Εφαρμογών είναι τα εξής:

1. Είναι συνθετική, καθώς παρέχει τη δυνατότητα «διαχωρισμού» του κώδικα σε προσαρμοσμένα ανεξάρτητα Στοιχεία (Components) και τη μετέπειτα χρήση και ενσωμάτωσή τους σε ένα μέρος.
2. Είναι δηλωτική, καθώς αλλάζοντας τη κατάσταση των Στοιχείων, η React αναλαμβάνει να ενημερώνει το DOM.
3. Είναι εύκολη στην εκμάθηση, αφού απαιτείται βασική γνώση των θεμελιωδών αρχών των γλωσσών HTML, CSS και JavaScript.
4. Είναι φιλική προς την ανάπτυξη Εφαρμογών Μονής Σελίδας (SAPs).
5. Διαθέτει μία τεράστια κοινότητα προγραμματιστών και εταιρειών που συμβάλουν στη περαιτέρω ανάπτυξή της, αλλά δημιουργούν βιβλιοθήκες, και εργαλεία τρίτων για τη διευκόλυνση των χρηστών παγκοσμίως [21].

Επιφέρει, όμως, και σημαντικά μειονεκτήματα:

1. Καθώς καλύπτει μόνο το Στοιχείο View της Αρχιτεκτονικής MVC, οι προγραμματιστές θα πρέπει να επιλέξουν και ορισμένες ακόμη τεχνολογίες, ούτως ώστε να αποκτήσουν ένα πλήρες σύνολο εργαλείων για την ανάπτυξη Διαδικτυακών Εφαρμογών.
2. Χρησιμοποιεί την JSX, έναν συνδυασμό των HTML και JavaScript, προσέγγιση την οποία ορισμένα μέλη της κοινότητας ανάπτυξης θεωρούν πολύπλοκη ως προς την εκμάθηση και κατανόηση, ειδικά για τους νέους προγραμματιστές.

3. Ο υψηλός ρυθμός ανάπτυξης αναγκάζει τους προγραμματιστές να υιοθετούν τις όποιες νέες αλλαγές και να μαθαίνουν νέους τρόπους σχεδίασης και ανάπτυξης, κάνοντας δύσκολη τη παρακολούθηση των συνεχώς εξελίξεων, καθώς και τον εντοπισμό σφαλμάτων.
4. Η έλλειψη επίσημης βιβλιογραφίας – εξαιτίας του πολύ υψηλού ρυθμού ανάπτυξης των τεχνολογιών React – οδηγεί τους προγραμματιστές στο να συντάσσουν οι ίδιοι οδηγίες [24].

2.2.6 VueJS

Το VueJS είναι ένα προοδευτικό Πλαίσιο JavaScript ανοιχτού κώδικα, το οποίο εστιάζει κυρίως στο Στοιχείο View και χρησιμοποιείται για την ανάπτυξη διαδραστικών Διεπαφών Χρήστη και Εφαρμογών Μονής Σελίδας. Δημιουργήθηκε από τον Evan You, εργαζόμενο της Google, και έγινε γνωστό το 2014. Χτίζεται πάνω στις τυπικές HTML, CSS και JavaScript και γίνεται δημοφιλές μέρα με τη μέρα, καθώς είναι πολύ εύκολο να ενσωματωθεί σε άλλα έργα και βιβλιοθήκες, αλλά και πολύ απλό στην εγκατάσταση και χρήση [25][27].

Τα κυριότερα χαρακτηριστικά που κάνουν το VueJS να γίνεται όλο και δημοφιλέστερο και στην ανάπτυξη Διαδικτυακών Εφαρμογών είναι τα παρακάτω:

Στοιχεία

Τα Στοιχεία (Components) αποτελούν ένα από τα σημαντικότερα χαρακτηριστικά του Πλαισίου VueJS. Χρησιμοποιούνται για την επέκταση βασικών HTML στοιχείων προς ενθυλάκωση επαναχρησιμοποιήσιμου κώδικα. Έτσι, οι προγραμματιστές μπορούν να δημιουργούν επαναχρησιμοποιήσιμα προσαρμοσμένα στοιχεία, τα οποία μπορούν αργότερα να χρησιμοποιηθούν ξανά σε HTML [25].

Εικονικό DOM

Το Πλαίσιο VueJS παρέχει πρότυπα που βασίζονται σε HTML που μπορούν να χρησιμοποιηθούν για τη σύνδεση του DOM που έχει αποδοθεί με τα δεδομένα παρουσίας Vue. Το VueJS μεταγλωττίζει τα πρότυπα σε συναρτήσεις απόδοσης Εικονικού DOM. Το VueJS αποδίδει Στοιχεία στη μνήμη του Εικονικού DOM πριν από την ενημέρωση του Προγράμματος Περιήγησης και υπολογίζει τον ελάχιστο αριθμό Στοιχείων για εκ νέου απόδοση, ώστε να εφαρμόζει τον ελάχιστο αριθμό χειρισμών DOM σε αλλαγή της κατάστασης της εφαρμογής [27].

Αντιδραστικότητα

Το VueJS παρέχει ένα σύστημα αντιδραστικότητας, καθώς κάθε στοιχείο παρακολουθεί τις εξαρτήσεις του και έτσι το σύστημα γνωρίζει ακριβώς πότε και ποια στοιχεία πρέπει να αποδώσει ξανά [27].

Δρομολόγηση

Το Πλαίσιο VueJS παρέχει μία Διεπαφή για την αλλαγή τού τι εμφανίζεται στη σελίδα της εφαρμογής βάσει της τρέχουσας διαδρομής διεύθυνσης URL, ανεξάρτητα από το πώς αυτό άλλαξε (μέσω συνδέσμου με email, ανανέωσης ή συνδέσμων εντός σελίδας). Η Διεπαφή επιτρέπει τη σκόπιμη μετάβαση της διαδρομής του Προγράμματος Περιήγησης όταν λαμβάνουν χώρα κάποια συμβάντα του Προγράμματος Περιήγησης, όπως είναι το κλικ, σε υπάρχοντα κουμπιά ή συνδέσμους. Η παροχή

της εν λόγω Διεπαφής από το Πλαίσιο VueJS δίνει λύση στο μειονέκτημα των Εφαρμογών Μονής Σελίδας (SPAs), το οποίο είναι η αδυναμία κοινής χρήσης συνδέσμων προς την ακριβή «υποσελίδα» σε μία συγκεκριμένη ιστοσελίδα, καθώς «προσφέρουν» στους χρήστες τους μόνο μία απόκριση βάσει URL από τον Διακομιστή. Και έτσι, η προσθήκη σελιδοδείκτη ή η κοινή χρήση συνδέσμων είναι δύσκολη, εάν όχι και αδύνατη [27].

Μεταβάσεις

Το Πλαίσιο VueJS παρέχει ποικίλους τρόπους για την εφαρμογή εφέ μετάβασης όταν εισάγονται, ενημερώνονται ή αφαιρούνται Στοιχεία από το DOM, κάνοντας πιο ευφάνταστη την εμπειρία του χρήστη. Πιο συγκεκριμένα, καθιστά δυνατή:

1. Την αυτόματη εφαρμογή κλάσεων για μεταβάσεις και κινούμενα σχέδια CSS.
2. Την ενσωμάτωση βιβλιοθηκών κινούμενων σχεδίων CSS τρίτων, όπως η Animate.css.
3. Την ενσωμάτωση βιβλιοθηκών κινούμενων σχεδίων JavaScript τρίτων, όπως η Velocity.js [27].

Πλεονεκτήματα και Μειονεκτήματα

Το Πλαίσιο VueJS επιφέρει σημαντικά πλεονεκτήματα ως προς τη χρήση του στην ανάπτυξη Διαδικτυακών Εφαρμογών, όπως:

1. Είναι εύκολο στην εκμάθηση και χρήση, καθώς βασίζεται στις HTML, CSS και JavaScript.
2. Είναι απλό, καθώς αποτελείται από Στοιχεία (Components) τα οποία μπορούν να επαναχρησιμοποιούνται.
3. Προσφέρει βέλτιστη ταχύτητα στη Διαδικτυακή Εφαρμογή, καθώς χρησιμοποιεί Εικονικό DOM, στο οποίο εντοπίζει και ανανεώνει μόνο τα τροποποιημένα Στοιχεία.
4. Είναι φιλικό προς την ανάπτυξη και βελτιστοποίηση Εφαρμογών Μονής Σελίδας (SAPs) [25][27].

Από την άλλη, όμως, επιφέρει και ορισμένα εξίσου σημαντικά μειονεκτήματα:

1. Διαθέτει περιορισμένη κοινότητα.
2. Αναβαθμίζεται συνεχώς και ο αριθμός των προγραμματιστών που το χρησιμοποιούν και το δουλεύουν αυξάνεται, γεγονός που μπορεί να αποτελεί πρόβλημα, καθώς παρέχονται διαρκώς πολλαπλές νέες μέθοδοι, καθιστώντας ολοένα και δυσκολότερη την επιλογή της καταλληλότερης.
3. Παρά τη σημαντική ανάπτυξή του, το Πλαίσιο VueJS υστερεί σε ό,τι αφορά τον αριθμό των προσθηκών που διατίθενται στη κοινότητά του [27].

2.3 Τεχνολογίες Ανάπτυξης Back – End Μέρους Διαδικτυακής Εφαρμογής

Το Back – End μέρος αποτελεί το βασικό στοιχείο της Αρχιτεκτονικής μίας Διαδικτυακής Εφαρμογής, καθώς λαμβάνει τα αιτήματα χρηστών, εκτελεί το απαραίτητο σύνολο λειτουργιών και παρέχει τα απαιτούμενα δεδομένα στο Front – End μέρος της. Επικεντρώνεται στη διαχείριση των

λειτουργιών από τη πλευρά του Διακομιστή (Server – Side) – αυτών δηλαδή που δεν είναι ορατές από τον χρήστη – και την υλοποίηση λειτουργιών [9] [11] [29].

Στο Σχήμα 2.9 απεικονίζονται οι τεχνολογίες που χρησιμοποιούνται για την ανάπτυξη του Back – End μέρους μίας Διαδικτυακής Εφαρμογής, το οποίο περιλαμβάνει τρία (3) βασικά στοιχεία:

- *Μία Βάση Δεδομένων* (Database), η οποία αποθηκεύει δομημένα έναν μεγάλο όγκο πληροφοριών και δεδομένων.
- *Έναν Διακομιστή* (Server), ένας υπολογιστής δηλαδή, ο οποίος λαμβάνει αιτήματα (requests) των Χρηστών μέσω του Front – End μέρους και ανταποκρίνεται με αποτελέσματα που υπολογίζει η Εφαρμογή.
- *Μία Εφαρμογή* (Application), ένα πρόγραμμα δηλαδή, το οποίο αφουγκράζεται τα αιτήματα του Διακομιστή, ανακτά πληροφορίες και δεδομένα από τη Βάση Δεδομένων και αποστέλλει αποτελέσματα στον Χρήστη μέσω του Διακομιστή [11][29].

Σχήμα 2.9: Τεχνολογίες ανάπτυξης Back – End μέρους [11].

Χαρακτηριστικό παράδειγμα μίας Back – End υλοποίησης αποτελεί ο σχεδιασμός API (Application Programming Interface, Διεπαφή Προγραμματισμού Εφαρμογών). Πιο συγκεκριμένα, για να επιτρέπεται να επικοινωνεί το λογισμικό μίας Διαδικτυακής Εφαρμογής με άλλα κομμάτια λογισμικού – εσωτερικά ή εξωτερικά – με συνέπεια, πρέπει να υλοποιηθεί μία Διεπαφή API. Αυτή αναφέρεται στις εντολές των προγραμματιστικών διαδικασιών, τις οποίες παρέχει ένα λειτουργικό σύστημα, μία βιβλιοθήκη ή μία εφαρμογή με τρόπο τέτοιο, ώστε να επιτρέπει από άλλα προγράμματα να κάνουν αιτήματα (requests) προς αυτά, προς ανταλλαγή ή επεξεργασία δεδομένων, μη επιτρέποντας όμως να αναγνωρίζεται ο εκτελούμενος κώδικας [29].

Από τις εικονιζόμενες – στο Σχήμα 2.9 – τεχνολογίες ανάπτυξης του Back – End μέρους μίας Διαδικτυακής Εφαρμογής, πιο δημοφιλείς και χρησιμοποιούμενες είναι οι NodeJS, Java και Python.

2.3.1 NodeJS

Το NodeJS είναι μια πλατφόρμα ανάπτυξης λογισμικού – και κυρίως Διακομιστών (Servers) – χτισμένη σε περιβάλλον Javascript. Δημιουργήθηκε το 2009 από τον Ryan Dahl και στόχος του είναι να παρέχει έναν εύκολο τρόπο ανάπτυξης του Back – End μέρους για τη δημιουργία κλιμακωτών και επεκτάσιμων Διαδικτυακών Εφαρμογών, καθώς και Back – End APIs [42][43].

Το NodeJS βασίζεται σε συμβάντα (events) και νήματα (threads). Ως νήμα ορίζεται μία σειρά λειτουργιών που πρέπει να εκτελέσει ο εκάστοτε Διακομιστής. Κάθε φορά που πραγματοποιείται ένα αίτημα από τον Πελάτη, αυτό το χειρίζεται Διακομιστής NodeJS, δίνοντάς του απάντηση. Η χρήση του NodeJS καθιστά δυνατή την απάντηση σε πολλούς Πελάτες ταυτόχρονα [43].

Τα χαρακτηριστικά που κάνουν το NodeJS ελκυστικό στη χρήση και αποτελούν πλεονεκτήματά του είναι τα παρακάτω:

1. Είναι εύκολο στην εκμάθηση και σύνταξη, καθώς βασίζεται στη γλώσσα JavaScript.
2. Διευκολύνει τον χειρισμό πολλαπλών αιτημάτων που υποβάλλονται από τον Πελάτη, επιτρέποντας έτσι τη κοινή χρήση κώδικα και την επαναχρησιμοποίηση των κωδίκων της εκάστοτε βιβλιοθήκης του.
3. Δεν αποκλείει κανένα αίτημα όταν ανταποκρίνεται σε άλλο.
4. Επεξεργάζεται γρήγορα και αποτελεσματικά τις ροές δεδομένων από και προς τους Πελάτες.
5. Είναι ευέλικτο και υποστηρίζει τη δημιουργία εφαρμογών που δε περιλαμβάνουν Διακομιστή, καθιστώντας τες ελαφριές.
6. Υποστηρίζει την ανάπτυξη Εφαρμογών Μονής Σελίδας (SAPs), οι οποίες φορτώνουν μία μεμονωμένη σελίδα HTML και την ενημερώνουν δυναμικά, καθώς ο χρήστης αλληλεπιδρά με την εκάστοτε εφαρμογή, μειώνοντας έτσι τον χρόνο απόκρισής της.
7. Υποστηρίζει τη δημιουργία REST APIs για τη δημιουργία αποτελεσματικής και απρόσκοπτης επικοινωνίας του Front – End μέρους με το Back – End μέρος μίας Διαδικτυακής Εφαρμογής.
8. Διαθέτει μεγάλη και ενεργή κοινότητα προγραμματιστών που το αναπτύσσουν και εμπλουτίζουν περαιτέρω διαρκώς [43].

Ωστόσο, χρησιμοποιώντας το NodeJS χρήζει προσοχής το γεγονός της απόδοσής του, η οποία μειώνεται με βαριές υπολογιστικές εργασίες, αφού δε μπορεί να επεξεργαστεί απαιτητικές εργασίες που συνδέονται με τη CPU (Central Processing Unit, Κεντρική Μονάδα Επεξεργασίας). Αυτό έχει ως αποτέλεσμα τη μείωση της απόδοσης και της Εφαρμογής, αλλά και της συνολικής εμπειρίας του Χρήστη με αυτήν [43].

2.3.2 Java

Η Java, η οποία εισήχθη στο κόσμο του προγραμματισμού το 1995, παραμένει μέχρι και σήμερα μία από τις πιο δημοφιλείς και ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού. Ως γλώσσα αντικειμενοστραφής και υψηλού επιπέδου, η Java χρησιμοποιείται σε πολλούς τομείς, όπως στην ανάπτυξη βάσεων δεδομένων, βιντεοπαιχνιδιών και Διακομιστών Ιστού (Web Servers), ενώ είναι δημοφιλής για την ανώτερη ικανότητά της να αναπτύσσει τα Back – End συστήματα των εκάστοτε Διαδικτυακών Εφαρμογών [44].

Η χρήση της γλώσσας Java για την ανάπτυξη του Back – End μέρους μίας Διαδικτυακής Εφαρμογής επιφέρει αξιοσημείωτα πλεονεκτήματα, όπως:

1. Σχεδιάστηκε έτσι, ώστε να είναι ανεξάρτητη από τη πλατφόρμα στην οποία εφαρμόζεται, όπως να ενσωματώνεται και να προσαρμόζεται στα διαφορετικά λειτουργικά συστήματα και αρχιτεκτονικές υπολογιστών.
2. Είναι αντικειμενοστραφής, καθώς αναλύεται σε αρθρωτά στοιχεία, τα οποία μπορούν να επαναχρησιμοποιούνται. Έτσι, ο κώδικας γίνεται συντομότερος, ενώ ταυτόχρονα απλοποιείται σημαντικά η αντιμετώπιση προβλημάτων και ο εντοπισμός σφαλμάτων.
3. Είναι επεκτάσιμη και ευέλικτη, επιτρέποντας τη κατασκευή Διαδικτυακών Εφαρμογών που μπορούν να υποστηρίξουν πολλές ταυτόχρονες διεργασίες χωρίς συμβιβασμούς στην ταχύτητα [44].

Παρουσιάζει και ορισμένα βαρυσήμαντα μειονεκτήματα:

1. Είναι σημαντικά πιο αργή από εγγενώς μεταγλωττισμένες γλώσσες, καθώς προσθέτει ένα επιπλέον επίπεδο μεταγλώττισης, το οποίο απαιτεί επιπλέον χρόνο για τη μετατροπή του πηγαίου κώδικα σε κώδικα μηχανής.
2. Είναι πολύπλοκη και περιεκτική σύνταξη, καθώς χρησιμοποιεί πολλές λέξεις για να αναπαραστήσει ακόμη και τις βασικότερες λειτουργίες.
3. Υστερεί στη δημιουργία και υποστήριξη Γραφικών Διεπαφών Χρήστη (Graphic User Interface) [44].

2.3.3 Python

Μία ακόμη ευρέως χρησιμοποιούμενη γλώσσα προγραμματισμού σήμερα είναι η Python. Ιδρύθηκε το 1991 και βρίσκει εφαρμογές στην ανάπτυξη Διαδικτυακών Εφαρμογών, στην ανάλυση και οπτικοποίηση δεδομένων, όπως και στην αυτοματοποίηση εργασιών [45].

Τα κυριότερα χαρακτηριστικά και πλεονεκτήματά της είναι:

1. Η απλότητα σύνταξής της, καθιστώντας την εύκολη στην ανάγνωση, κατανόηση, εκμάθηση και χρήση.
2. Δε περιλαμβάνει πολύπλοκες δομές, όπως η Java.
3. Υποστηρίζει τις αρχές του αντικειμενοστραφή προγραμματισμού, διευκολύνοντας την επαναχρησιμοποίηση κώδικα, την επεκτασιμότητα και την απλοποίηση του εντοπισμού και της αντιμετώπισης ενδεχόμενων σφαλμάτων.
4. Η εκτεταμένη πρότυπη βιβλιοθήκη της αποτελείται από πολλαπλά πακέτα και ενότητες για τη βελτίωση της λειτουργικότητας της εκάστοτε εφαρμογής που τη χρησιμοποιεί. Εισάγοντας την απαιτούμενη βιβλιοθήκη στον κώδικά της, καθίσταται δυνατή η ολοκλήρωση μίας συγκεκριμένης εργασία χωρίς την ανάγκη σύνταξης κώδικα για αυτήν.
5. Είναι συμβατή με διάφορα λειτουργικά συστήματα.
6. Η απλή σύνταξή της και η εκτεταμένη βιβλιοθήκη που παρέχει απλοποιούν την ανάπτυξη σύνθετων και πολύπλοκων λογισμικών προγραμμάτων.
7. Είναι μία γλώσσα ανοιχτού κώδικα, γεγονός το οποίο βοηθά τους προγραμματιστές στο να μειώσουν σημαντικά το κόστος ανάπτυξης μίας εκάστοτε εφαρμογής. Διαθέτει πολλά πλαίσια, εργαλεία ανάπτυξης και βιβλιοθήκες περιορίζοντας σημαντικά τον χρόνο ανάπτυξης [45].

Μαζί με τα ποικίλα πλεονεκτήματα, η Python παρουσιάζει ορισμένους περιορισμούς σε ό,τι αφορά απόδοση και την ασφάλεια της εκάστοτε υλοποίησης που την «ασπάζεται». Πιο συγκεκριμένα:

1. Παρουσιάζει αργή ταχύτητα εκτέλεσης, καθώς είναι γλώσσα που λειτουργεί με διερμηνέα (interpreter) και όχι μεταγλωττιστή (compiler).
2. Οι δομές δεδομένων της απαιτούν περισσότερο χώρο στη μνήμη, καθιστώντας τη γλώσσα ακατάλληλη για χρήση προς ανάπτυξη μίας εκάστοτε εφαρμογής υπό περιορισμούς μνήμης.
3. Πρόσβαση στη βάση δεδομένων
4. Θεωρείται εξαιρετικά ανασφαλής και ενέχει κινδύνους ασφαλείας [45].

2.4 Διεπαφές Προγραμματισμού Εφαρμογών (APIs)

Ως Διεπαφή Προγραμματισμού Εφαρμογών (Application Programming Interface, API) ορίζεται ένα σύνολο κανόνων, το οποίο καθορίζει τον τρόπο με τον οποίο ένα πρόγραμμα λογισμικού μπορεί να έχει πρόσβαση σε δεδομένα ή και λειτουργίες που παρέχονται από ένα άλλο πρόγραμμα λογισμικού. Αποτελεί ζωτικό κομμάτι της σύγχρονης ανάπτυξης λογισμικού, καθώς επιτρέπει σε διαφορετικά συστήματα και εφαρμογές να επικοινωνούν μεταξύ τους και να μοιράζονται τη λειτουργικότητα με τρόπο ευέλικτο και αποτελεσματικό. Εφαρμόζεται σε ποικίλες υλοποιήσεις, όπως στην ανάπτυξη εφαρμογών Ιστού, εφαρμογών για κινητά και Διαδικτύου των Πραγμάτων (Internet of Things, IoT) [40][41].

Τα API έρχονται υπό τη μορφή μίας βιβλιοθήκης, την οποία ο εκάστοτε προγραμματιστής και σχεδιαστής λογισμικού μπορεί να συμπεριλάβει στον κώδικά του. Η βιβλιοθήκη αυτή παρέχει ένα σύνολο λειτουργιών, οι οποίες μπορούν να κληθούν να εκτελέσουν διάφορες εργασίες. Το API καθορίζει τις κλήσεις συναρτήσεων, τις εισόδους που δέχονται, καθώς και τις εξόδους που επιστρέφουν [41].

2.4.1 Τρόπος Λειτουργίας των API

Ο τρόπος με τον οποίο λειτουργεί ένα API είναι ο εξής:

- Ένας Πελάτης (Client) στέλνει ένα αίτημα (request) στον Διακομιστή API (API Server) μέσω του Διαδικτύου ή ενός τοπικού δικτύου. Το αίτημα πραγματοποιείται με τη χρήση ενός συγκεκριμένου πρωτοκόλλου, όπως είναι το HTTP (HyperText Transfer Protocol, Πρωτόκολλο Μεταφοράς Υπερκειμένου), και περιλαμβάνει πληροφορίες σχετικά με τη λειτουργία που επιθυμεί να εκτελέσει ο Πελάτης, όπως είναι η ανάκτηση δεδομένων και πληροφοριών ή η ενημέρωση ενός πόρου.
- Ο Διακομιστής API λαμβάνει το αίτημα και το επεξεργάζεται. Συγκεκριμένα μπορεί να το επικυρώσει, να ελέγξει τη ταυτότητα του Πελάτη, να το εξουσιοδοτήσει ή να εκτελέσει άλλες απαραίτητες λειτουργίες.
- Ο Διακομιστής API στέλνει μία απάντηση στον Πελάτη, η οποία μπορεί να περιλαμβάνει πληροφορίες και δεδομένα, μήνυμα σφάλματος ή και έναν κωδικό κατάστασης, ο οποίος υποδεικνύει το αποτέλεσμα της ζητούμενης λειτουργίας [41].

Σχήμα 2.10: Παράδειγμα ροής αιτημάτων (requests) [41].

2.4.2 Πλεονεκτήματα των APIs

Τα API απλοποιούν το σχεδιασμό και την ανάπτυξη νέων εφαρμογών και υπηρεσιών, όπως επίσης την ενοποίηση και διαχείριση των ήδη υπαρχόντων. Η ανάπτυξη και χρήση τους επιφέρει τα εξής σημαντικά οφέλη:

- *Ευελιξία και βελτιωμένη συνεργασία*. Επιτρέπουν οι διάφορες πλατφόρμες και εφαρμογές να μπορούν να επικοινωνούν απρόσκοπτα μεταξύ τους. αυτοματοποιώντας τις ροές εργασίας.
- *Ασφάλεια συστήματος*. Διαχωρίζουν την αιτούσα εφαρμογή από την υποδομή της υπηρεσίας απόκρισης και προσφέρουν επίπεδα ασφάλειας μεταξύ των δύο κατά την επικοινωνία τους, καθώς τα HTTP αιτήματα API απαιτούν διαπιστευτήρια ελέγχου ταυτότητας.
- *Ασφάλεια και απόρρητο τελικού χρήστη*. Ακριβώς όπως τα API παρέχουν πρόσθετη προστασία σε ένα δίκτυο, μπορούν επίσης να παρέχουν ένα άλλο επίπεδο προστασίας για προσωπικούς χρήστες. Παραδείγματος χάρη, όταν μία ιστοσελίδα ζητά την τοποθεσία ενός χρήστη, η οποία παρέχεται μέσω ενός API τοποθεσίας, ο χρήστης μπορεί να αποφασίσει εάν θα επιτρέψει ή θα απορρίψει αυτό το αίτημα.
- *Επιταχυνόμενη καινοτομία*. Προσφέρουν ευελιξία, επιτρέποντας στις εκάστοτε εταιρείες να συνδέονται με νέους επιχειρηματικούς εταίρους, να προσφέρουν νέες υπηρεσίες στην αγορά τους και να έχουν πρόσβαση σε νέες αγορές που μπορούν να αποφέρουν τεράστιες αποδόσεις και να οδηγήσουν στον ψηφιακό μετασχηματισμό.
- *Δημιουργία εσόδων δεδομένων*. Δεν είναι λίγες οι εταιρείες που επιλέγουν να προσφέρουν δωρεάν API, ώστε να δημιουργήσουν ένα κοινό προγραμματιστών γύρω από την επωνυμία τους και να σφυρηλατήσουν σχέσεις με πιθανούς επιχειρηματικούς συνεργάτες. Εάν το API αυτό παραχωρεί πρόσβαση σε πολύτιμα ψηφιακά στοιχεία, η επιχείρηση μπορεί να δημιουργήσει έσοδα πουλώντας πρόσβαση [40].

2.4.3 Αρχιτεκτονικές API

Η αρχιτεκτονική API αναφέρεται στη συνολική σχεδίαση και δομή ενός API. Πιο συγκεκριμένα, στον τρόπο οργάνωσής του, στους τύπους αιτημάτων που υποστηρίζει, στις μορφών δεδομένων και στα πρωτόκολλα που χρησιμοποιεί, όπως και σε τυχόν μηχανισμούς ασφάλειας ή ελέγχους ταυτότητας που χρησιμοποιεί. Τέσσερις (4) κοινοί τύποι αρχιτεκτονικής API που χρησιμοποιούνται ευρέως είναι οι εξής:

1. Representational State Transfer (REST).
2. Simple Object Access Protocol (SOAP).
3. Remote Procedure Call (RPC).
4. GraphQL,

με τις REST και SOAP να προηγούνται [40].

2.4.3.1 Αρχιτεκτονική REST

Τα REST APIs είναι ένας τύπος Web API – δηλαδή η πρόσβαση σε αυτά γίνεται μέσω του Διαδικτύου – ο οποίος χρησιμοποιεί αιτήματα HTTP για χειρισμό δεδομένων. Είναι σχεδιασμένα να είναι ελαφριά και ευέλικτα και είναι ιδανικά για τη δημιουργία επεκτάσιμων και εύκολα συντηρήσιμων υπηρεσιών Διαδικτύου. Μπορούν να αναπτυχθούν χρησιμοποιώντας σχεδόν οποιαδήποτε γλώσσα προγραμματισμού, όπως και να υποστηρίζουν μία ποικιλία μορφών δεδομένων [40].

Χρησιμοποιούν ένα σταθερό σύνολο ρημάτων (verbs) HTTP, ούτως ώστε να εκτελούν τυπικές λειτουργίες βάσης δεδομένων, όπως δημιουργία, ανάγνωση, ενημέρωση και διαγραφή εγγραφών μέσα σε έναν πόρο. Πιο συγκεκριμένα, ένα REST API χρησιμοποιεί ένα αίτημα τύπου GET για την ανάκτηση μίας εγγραφής, ένα αίτημα τύπου POST για τη δημιουργία μίας εγγραφής, ένα αίτημα τύπου PUT για την ενημέρωση μίας εγγραφής και ένα αίτημα τύπου DELETE για τη διαγραφή μίας εγγραφής [40].

Τα πλέον ελκυστικά χαρακτηριστικά της δημοφιλούς Αρχιτεκτονικής REST είναι τα εξής:

- Χρησιμοποιεί μικρότερο εύρος ζώνης, καθιστώντας την κατάλληλη για την ανάπτυξη ολοκληρωμένων υπηρεσιών Ιστού.
- Χρησιμοποιεί το πρωτόκολλο HTTP για την ανάκτηση δεδομένων ή την εκτέλεση λειτουργιών σε διάφορες μορφές δεδομένων, όπως XML (Extensible Markup Language) και JSON (JavaScript Object Notation), επιτρέπει ταχύτερη διεκπεραίωση διαδικασιών.
- Τα REST APIs αποτελούν ανεξάρτητες μονάδες. Δεν επηρεάζουν τη λειτουργία των υπόλοιπων μονάδων του συστήματος, παρέχοντας ευελιξία και δυνατότητα επαναχρησιμοποίησης κατά την προσθήκη, αντικατάσταση ή προσαρμογή μονάδων.
- Τα REST APIs είναι εύκολα στην υλοποίηση, καθώς δεν απαιτούν ξεχωριστό επίπεδο ανταλλαγής μηνυμάτων για την επικοινωνία μεταξύ συστημάτων, καθιστώντας τα γρήγορα.
- Η Αρχιτεκτονική REST είναι προσβάσιμη από διαφορετικές γλώσσες προγραμματισμού [40][46].

2.4.3.2 Αρχιτεκτονική SOAP

Το SOAP είναι μία πλατφόρμα δικτύου που χρησιμοποιείται σε μία υπηρεσία Ιστού για την ανταλλαγή ή την επικοινωνία δεδομένων μεταξύ δύο διαφορετικών συστημάτων. Χρησιμοποιεί τη μορφή δεδομένων XML για τη μεταφορά μηνυμάτων μέσω του πρωτοκόλλου HTTP. Στις υπηρεσίες Ιστού, το SOAP επιτρέπει στο αίτημα του χρήστη να αλληλεπιδρά με άλλες γλώσσες

προγραμματισμού. παρέχοντας έναν τρόπο επικοινωνίας μεταξύ εφαρμογών που εκτελούνται σε διαφορετικά λειτουργικά συστήματα [47].

Τα χαρακτηριστικά της Αρχιτεκτονικής SOAP είναι τα παρακάτω:

- Είναι ένα ανοιχτό και ελαφρύ πρότυπο πρωτόκολλο που χρησιμοποιείται στην υπηρεσία Ιστού για την επικοινωνία μέσω Διαδικτύου.
- Χρησιμοποιείται για τη μετάδοση μηνύματος μέσω του δικτύου.
- Χρησιμοποιείται για τη κλήση απομακρυσμένων διαδικασιών και την ανταλλαγή εγγράφων.
- Είναι μία πλατφόρμα και γλώσσα ανεξάρτητη, καθώς μπορεί να χρησιμοποιηθεί σε οποιαδήποτε πλατφόρμα και μπορεί να υποστηρίξει πολλές γλώσσες.
- Χρησιμοποιεί μόνο τη μορφή XML για την αποστολή μηνυμάτων μέσω του πρωτοκόλλου HTTP.
- Δε περιλαμβάνει χαρακτηριστικά ασφαλείας [47].

2.5 Βάσεις Δεδομένων Διαδικτυακών Εφαρμογών

Ως «δεδομένα» ορίζεται μία συλλογή μικρών μονάδων πληροφοριών, η μορφή των οποίων μπορεί να είναι κείμενα, αριθμοί, bytes κ.λπ. και μπορούν να αποθηκευτούν σε ηλεκτρονική μνήμη. Ως «βάση δεδομένων» ορίζεται μία οργανωμένη συλλογή δεδομένων κατά τρόπο τέτοιο, ώστε να είναι εύκολη η πρόσβαση σε αυτήν και η διαχείρισή της. Τα εκάστοτε δεδομένα μπορούν να οργανώνονται σε πίνακες, σειρές και στήλες [6].

Οι βάσεις δεδομένων διαχωρίζονται σε SQL (Structured Query Language), οι οποίες οργανώνουν τα στοιχεία που περιλαμβάνουν βάσει ενός μοντέλου Οντοτήτων – Σχέσεων και σε NoSQL (Not Only SQL), οι οποίες δε χρησιμοποιούν κάποιο οργανωμένο σύστημα, όπως π.χ. έναν πίνακα, δε τηρούν κάποιο μοντέλο και «αδιαφορούν» για τη σχέση που μπορεί ή όχι να έχουν τα στοιχεία που περιλαμβάνουν [7].

Τα κριτήρια για την επιλογή της κατάλληλης βάσης δεδομένων για μία αναπτυσσόμενη διαδικτυακή εφαρμογή εξαρτάται από τις εκάστοτε ανάγκες και απαιτήσεις της. Εάν απαιτείται μία ευέλικτη βάση δεδομένων, η οποία να μπορεί να χειρίζεται πολλά δεδομένα, να κλιμακώνεται οριζόντια και να είναι λιγότερο δαπανηρή ως προς τη συντήρησή της, τότε μία βάση δεδομένων NoSQL αποτελεί μία καλή επιλογή. Οι MongoDB, CouchDB, CouchBase, Cassandra, HBase, Redis, Riak, Neo4J αποτελούν τα δημοφιλέστερα παραδείγματα βάσεων δεδομένων NoSQL. Εάν απαιτείται μία αξιόπιστη και συνεπής βάση δεδομένων, στην οποία είναι εύκολη η κατακόρυφη κλιμάκωση και η αναζήτηση στοιχείων, τότε μία βάση δεδομένων SQL αποτελεί την καλύτερη επιλογή. Οι MySQL, PostgresSQL, Oracle και Microsoft SQL Server αποτελούν τις πιο χρησιμοποιούμενες βάσεις δεδομένων SQL [8].

2.6 Έλεγχος Ταυτότητας και Εξουσιοδότηση

Δύο – ζωτικής σημασίας – διαδικασίες ασφάλειας των πληροφοριών που χρησιμοποιούνται για την προστασία των συστημάτων και των πληροφοριών μίας Διαδικτυακής Εφαρμογής, είναι ο Έλεγχος Ταυτότητας (Authentication – AuthN) και η Εξουσιοδότηση (Authorization – AuthZ).

Η διαδικασία «Έλεγχος Ταυτότητας» επαληθεύει τη ταυτότητα ενός χρήστη ή μίας υπηρεσίας, εάν δηλαδή κάποιος χρήστης ή κάποια εφαρμογή είναι αυτό που ισχυρίζεται ότι είναι, χρησιμοποιώντας κάποια μορφή ελέγχου ταυτότητας, ώστε να εξασφαλίσει τη πρόσβαση σε μία εφαρμογή ή στα δεδομένα αυτής [10].

Η διαδικασία «Εξουσιοδότηση» καθορίζει τα δικαιώματα πρόσβασης ενός χρήστη ή μίας υπηρεσίας, δηλαδή το επίπεδο πρόσβασής τους, χρησιμοποιώντας κάποιας μορφής εξουσιοδότηση, ώστε να δοθεί στους χρήστες ή τις υπηρεσίες άδεια πρόσβασης σε ορισμένα δεδομένα ή άδεια πρόσβασης εκτέλεσης μίας συγκεκριμένης ενέργειας [10].

Κεφάλαιο 3ο: Υλοποίηση Front – End Μέρους Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης

Έχοντας παρουσιάσει και διαχωρίσει τις διαθέσιμες μεθόδους και τεχνολογίες που χρησιμοποιούνται για την ανάπτυξη του Front – End μέρους μίας Διαδικτυακής Εφαρμογής, στο κεφάλαιο αυτό θα επιλέξουμε τη καταλληλότερη τεχνολογία για την υλοποίηση του Front – End μέρους της Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης. Επίσης, θα δούμε βήμα προς βήμα τη διαδικασία ανάπτυξής του.

3.1 Επιλογή Τεχνολογίας Ανάπτυξης Front – End

Όπως είδαμε στο προηγούμενο κεφάλαιο, τα Πλαίσια AngularJS και VueJS, καθώς και η Βιβλιοθήκη React της JavaScript, είναι οι τεχνολογίες που χρησιμοποιούνται ευρέως για την ανάπτυξη του Front – End μέρους των εκάστοτε Διαδικτυακών Εφαρμογών.

Για την επιλογή της πλέον καταλληλότερης – από τις προαναφερθείσες τρεις – για την ανάπτυξη του Front – End μέρους της Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης, θα πρέπει να πληροί ορισμένες προϋποθέσεις για τη διευκόλυνση και του σχεδιαστή – προγραμματιστή, αλλά και των τελικών χρηστών της Εφαρμογής. Οι προϋποθέσεις είναι:

- Να μπορεί να δημιουργεί διατηρήσιμο κώδικα, ο οποίος να είναι απλός και εύκολος στη σύνταξη, ανάγνωση, δοκιμή και τροποποίησή του.
- Να προσφέρει τη δυνατότητα επαναχρησιμοποίησης στοιχείων, και επομένως, να εξαλείφει τη σύνταξη εκτεταμένου κώδικα.
- Να δημιουργεί συνεπών, αποτελεσματικές, διαδραστικές και αισθητικά όμορφες Διεπαφές Χρήστη.
- Να διαθέτει ενεργή κοινότητα προγραμματιστών, η οποία βελτιώνει τα ήδη υπάρχοντα εργαλεία και τις βιβλιοθήκες της ή ακόμη αναπτύσσει και προσφέρει νέα.

Λαμβάνοντας υπόψη τα παραπάνω, επιλέξαμε τη Βιβλιοθήκη React της JavaScript, καθώς:

- Βασίζεται στις γλώσσες HTML, CSS και JavaScript.
- Βασίζεται σε Στοιχεία (Components), η λογική των οποίων γράφεται σε JavaScript, καθιστώντας επαναχρησιμοποιήσιμο το εκάστοτε Στοιχείο και τον τελικό κώδικα συντομότερο.
- Διαθέτει ποικίλες επεκτάσεις, οι οποίες μπορούν να χρησιμοποιηθούν για τη δημιουργία μίας πλήρως αποτελεσματικής εφαρμογής Διεπαφής Χρήστη.
- Διαθέτει μία τεράστια κοινότητα προγραμματιστών και εταιρειών παγκοσμίως, η οποία δημιουργεί βιβλιοθήκες και περαιτέρω εργαλεία.

3.2 Δημιουργία Front – End Μέρους Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης

Για τη δημιουργία του Front – End της Διαδικτυακής Εφαρμογής χρησιμοποιείται η Βιβλιοθήκη React της JavaScript. Έτσι, επιτρέπεται στους χρήστες να καταχωρούν, να ανανεώνουν και να

επεξεργάζονται στοιχεία, καθώς και να εκδίδουν σημαντικά – για την επιχείρησή τους – παραστατικά. Οι σελίδες που δημιουργούνται και χρησιμοποιούνται είναι οι εξής παρακάτω:

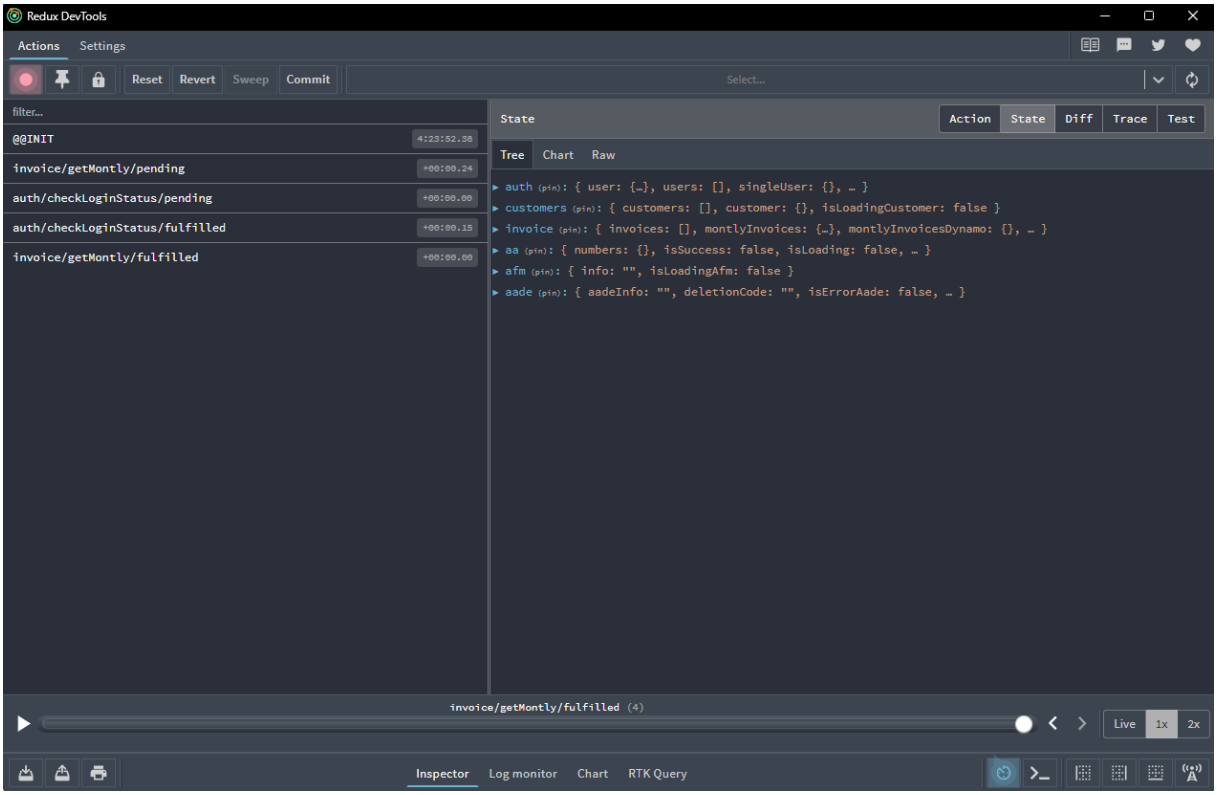
1. Login Χρήστη,
2. Αρχική Σελίδα,
3. Register Νέου Χρήστη (μόνο για τον Χρήστη Admin),
4. Λίστα Χρηστών (μόνο για τον Χρήστη Admin),
5. Προβολή στοιχείων χρήστη και ανανέωσή τους (Update) (μόνο για τον Χρήστη Admin),
6. Δημιουργία Νέου Πελάτη,
7. Προβολή Πελάτη και Ανανέωση των Στοιχείων του,
8. Προβολή Λίστας Πελατών με Αναζήτηση,
9. Έκδοση Απόδειξης Λιανικής Πώλησης,
10. Έκδοση Τιμολογίου Πώλησης,
11. Έκδοση Τιμολογίου Παροχής Υπηρεσιών,
12. Προβολή Λίστας Παραστατικών με Φόρμα Αναζήτησης και Εκτύπωσή τους σε Πίνακα,
13. Προβολή Παραστατικού.

Αρχικά, για τη δημιουργία της React εφαρμογής μας χρησιμοποιείται η εντολή “npx create-react-app”. Στη συνέχεια, στο αρχείο index.js «τυλίγουμε» (Wrap) την εφαρμογή μας με έναν Provider, τον οποίο μάς παρέχει η βιβλιοθήκη του Redux, περνώντας σαν παράμετρο το δημιουργηθέν Store, όπως φαίνεται στο Σχήμα 3.1. Έτσι, έχουμε πρόσβαση στο State του Redux από όλη την Εφαρμογή μας.

```
import React from 'react'
import ReactDOM from 'react-dom'
import { Provider } from 'react-redux'
import { store } from './app/store'
import './index.css'
import App from './App'

ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>,
  document.getElementById('root')
)
```

Σχήμα 3.1: Χρήση του Provider της Βιβλιοθήκης Redux.

Χρησιμοποιώντας το υπάρχον Extension του Redux για τον Chrome, μπορούμε να δούμε τη κατάσταση του State ανά πάσα στιγμή, όπως φαίνεται στο Σχήμα 3.2. Επίσης, χρησιμοποιώντας τη βιβλιοθήκη του Redux, και πιο συγκεκριμένα τα Hooks “useDispatch” και “useSelector”, μπορούμε να κάνουμε κλήσεις στο Back – End μας, όπως και να παίρνουμε τιμές από μεταβλητές που είναι αποθηκευμένες globally στο State του Redux.



Σχήμα 3.2: Παρακολούθηση της κατάστασης του State.

Για τη περιήγηση στη Διαδικτυακή Εφαρμογή, χρησιμοποιείται το πακέτο React – Router – DOM και, πιο συγκεκριμένα, η έκδοση 6.4. Το Layout της Διαδικτυακής Εφαρμογής Τιμολόγησης είναι ίδιο σε κάθε Σελίδα, πέρα από αυτή του «Login Χρήστη». Για να μη χρειάζεται, λοιπόν, να γίνεται import του Navbar σε κάθε σελίδα ξεχωριστά, δημιουργήθηκε ένα σταθερό Layout, όπως φαίνεται στο Σχήμα 3.3. Έτσι, κάθε φορά που αλλάζουμε Σελίδα, θα γίνεται Render μόνο το περιεχόμενο του <Outlet />.

```
const Layout = () => {
    return (
        <>
            <Navbar />
            <PrivateRoute>
                <Outlet />
            </PrivateRoute>
        </>
    )
}
```

Σχήμα 3.3: Δημιουργία σταθερού Layout.

Για να αλλάξουμε το περιεχόμενο του Outlet, κάνουμε import και χρησιμοποιούμε το Component “CreateBrowserRouter” από το React – DOM, το οποίο απεικονίζεται στα Σχήματα 3.4, 3.5 και 3.6.

```
const router = createBrowserRouter([
    {
      path: '/',
      element: <Layout />,
      children: [
        {
          path: '/',
          element: <Home />,
        },
        {
          path: '/invoices/:invoiceId',
          element: <Invoice />,
        },
        {
          path: '/invoices',
          element: <Invoices />,
        },
        {
          path: '/apodeixh',
          element: <Apodeixh />,
        },
        {
          path: '/timologio',
          element: <Timologio />,
        },
        {
          path: '/timologio-yphresiwn',
          element: <TimologioYphresiwn />,
        },
```

Σχήμα 3.4: Χρήση του Component “BrowserRouter” (1).

Αυτό σημαίνει πως κάθε Path που ορίσαμε είναι Παιδί (Child) του Outlet. Ανάλογα με το Path της Εφαρμογής στο οποίο βρισκόμαστε, θα γίνεται Render και το κατάλληλο Component. Παραδείγματος χάρη, εάν βρισκόμαστε στην «Αρχική Σελίδα», και συνεπώς στο Path “/”, θα γίνει Render το Component “Home”.

Στο δημιουργηθέν Layout, όπως φαίνεται στο Σχήμα 3.7, όλα τα ορισθέντα Παιδιά – Σελίδες έχουν ως Γονέα (Parent) μία συνιστώσα (Component) “PrivateRoute”. Βασική προϋπόθεση αποτελεί ένας Χρήστης να έχει πρωτίστως συνδεθεί, ούτως ώστε να μπορέσει να χρησιμοποιήσει την Εφαρμογή Τιμολόγησης. Έτσι, κάθε φορά που προσπαθούμε να κάνουμε Render μία εκ των Σελίδων, προτού γίνει το Render, το Component “PrivateRoute” ελέγχει εάν έχουμε συνδεθεί.

```
{
  path: '/users/:userId',
  element: <User />,
},
{
  path: '/users',
  element: <Users />,
},
{
  path: '/register',
  element: <Register />,
},
{
  path: '/customers/:afm',
  element: <Customer />,
},
{
  path: '/customers',
  element: <Customers />,
},
{
  path: '/customers/new-customer',
  element: <NewCustomer />,
},
{
  path: '/about',
  element: <About />,
},
{
  path: '/help',
  element: <Help />,
},
{
  path: '*',
  element: <NotFound />,
},
```

Σχήμα 3.5: Χρήση του Component “BrowserRouter” (2).

```
    ],
  },

  {
    path: '/login',
    element: <Login />,
  },
])
```

Σχήμα 3.6: Χρήση του Component “BrowserRouter” (3).

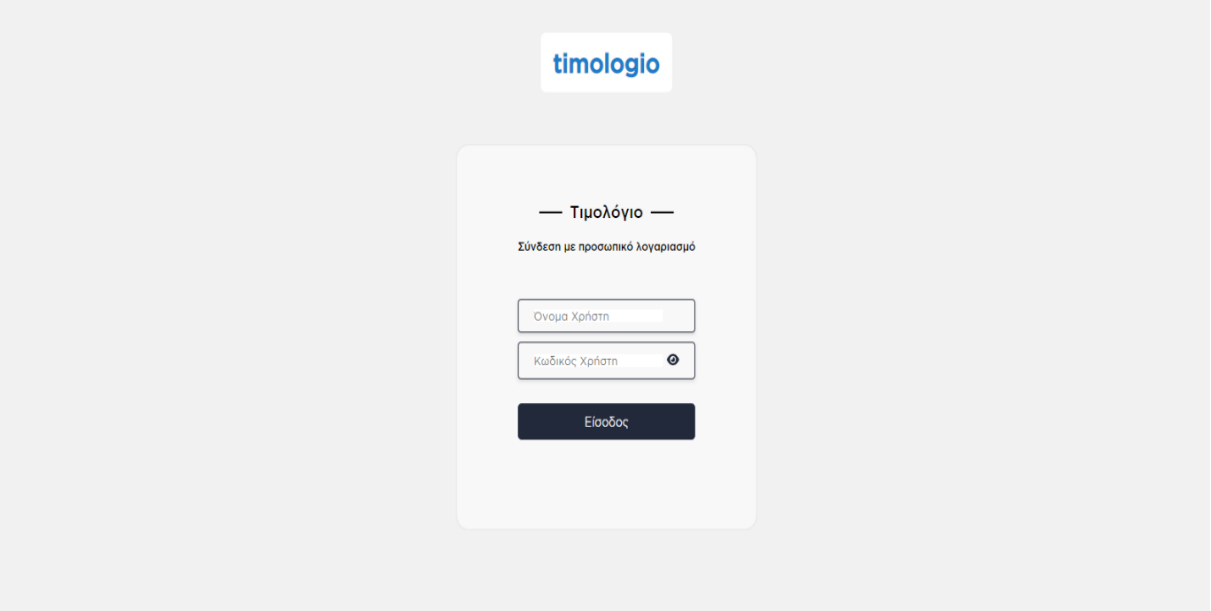
```
import { Navigate } from 'react-router-dom'
import { checkLoginStatus } from '../features/auth/authSlice'
import { useSelector, useDispatch } from 'react-redux'
import { useEffect } from 'react'

const PrivateRoute = ({ children }) => {
    const { user } = useSelector((state) => state.auth)

    const dispatch = useDispatch()

    useEffect(() => {
        const status = async () => {
            dispatch(checkLoginStatus())
        }

        status()
    })

    if (user) return children

    return <Navigate to='/login' />
}

export default PrivateRoute
```

Σχήμα 3.7: Χρήση του Component “PrivateRoute”.

Με τη χρήση του Hook “useEffect()” καλούμε τη συνάρτηση “status()”, η οποία είναι ασύγχρονη, και με τη βοήθεια του Hook “useDispatch()”, γίνεται ένα αίτημα “GET” στο Route “/api/users/status”. Εάν το αίτημα γίνει “Fulfilled”, τότε επιστρέφεται το Παιδί (Child) και βλέπουμε τη σελίδα που ζητήσαμε. Εάν το αίτημα γίνει “Rejected”, τότε ο Χρήστης – με τη βοήθεια του Hook “Navigate” – θα ανακατευθύνει στη σελίδα “Login”. Αυτός ο έλεγχος γίνεται σε όλες τις σελίδες, εκτός της “Login”.

3.3 Σελίδα «Login Χρήστη»

Όπως φαίνεται στο Σχήμα 3.8, η σελίδα «Login Χρήστη» αποτελείται από μία φόρμα, όπου ο εκάστοτε Χρήστης συμπληρώνει το Όνομα Χρήστη (“Name”) και το Κωδικό Χρήστη (“Password”). Κάνοντας κλικ, εν συνεχεία, στο κουμπί «Είσοδος» επιχειρεί να κάνει Login στην εφαρμογή μας. Στο Σχήμα 4.9, απεικονίζεται ο κώδικας για τη δημιουργία της εν λόγω σελίδας.



Σχήμα 3.8: Απεικόνιση Σελίδας «Login Χρήστη» σε Η/Υ.

```
return (
  <div className='login-page'>
    <h2 className='page-title-login'>timologio</h2>
    <form className='apodeixhForm box-login' onSubmit={onSubmit}>
      <div className='login-form-titlle'>
        <p className='login-form-titlle-top'>Τιμολόγιο</p>
        <p className='login-form-titlle-text'>Σύνδεση με προσωπικό λογαριασμό</p>
      </div>
      <div className='payment-type'>
        <div className='login-input'>
          <input type='text' className='create-user' id='name' name='name' value={name} onChange={onChange} placeholder='Όνομα Χρήστη' required
          />
        </div>
      </div>
      <div className='payment-type'>
        <div className='login-input'>
          <input type={showPass ? 'text' : 'password'} className='create-user' id='password'  name='password' value={password} onChange={onChange} placeholder='Κωδικός Χρήστη' required
          />
          <p className='show-pass-toggle' onClick={() => setShowPass((prevState) => !prevState)}>
            {showPass ? (
              <EyeClosed fill='#21293a' width='16px' height='16px' />
            ) : (
              <EyeOpen fill='#21293a' width='16px' height='16px' />
            )}
          </p>
        </div>
      </div>
      <div>
        <button className='mg-1-0 btn-submit width-100'>Είσοδος</button>
      </div>
    </form>
  </div>
)
```

Σχήμα 3.9: Δημιουργία Σελίδας «Login Χρήστη».

Όπως βλέπουμε στο Σχήμα 3.10, τα πεδία συμπλήρωσης Ονόματος και Κωδικού Χρήστη της παραπάνω φόρμας διαχειρίζονται από το Hook “useState()” και τη συνάρτηση “onChange()”. Κάθε φορά που ο Χρήστης εισάγει ή διαγράφει δεδομένα, καλείται η συνάρτηση “onChange()”, η οποία ανανεώνει τη κατάσταση των μεταβλητών.

```
const [formData, setFormData] = useState({
    name: '',
    password: '',
})

const onChange = (e) => {
    setFormData((prevState) => ({
        ...prevState,
        [e.target.name]: e.target.value,
    }))
}
```

Σχήμα 3.10: Δημιουργία φόρμας συμπλήρωσης στοιχείων Χρήστη και σύνταξη συνάρτησης “onChange()”.

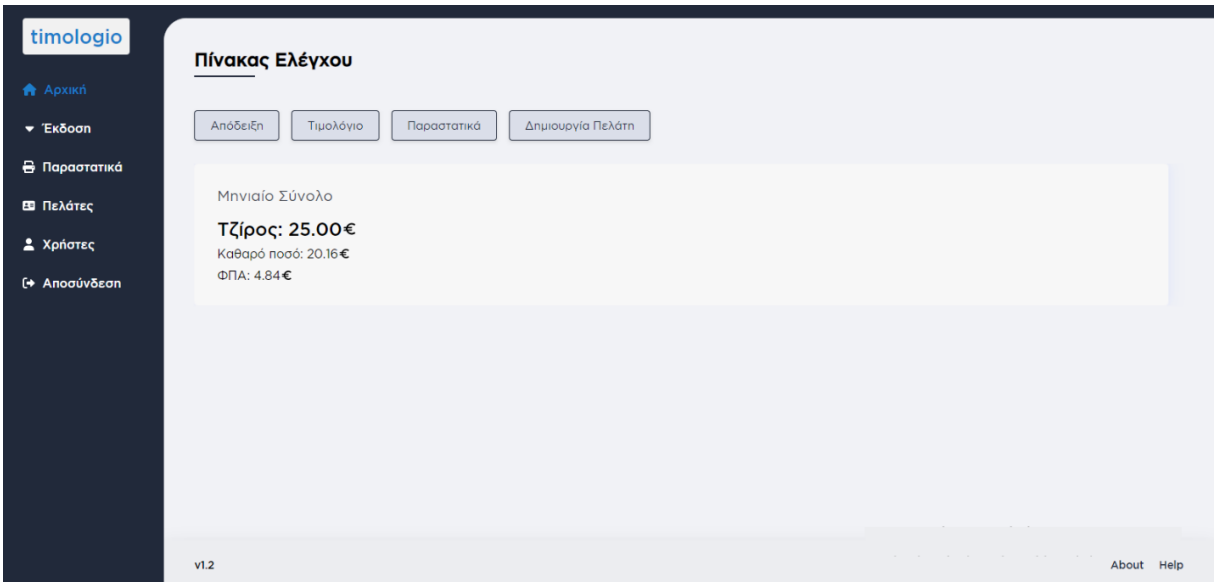
Όταν ο χρήστης κάνει κλικ στο κουμπί «Είσοδος», καλείται η συνάρτηση “onSubmit()”, η οποία – με τη βοήθεια του Hook “useDispatch()” – καλεί τη συνάρτηση “login()”, έχοντας σαν παράμετρο τα στοιχεία της φόρμας. Η συνάρτηση “login()” δημιουργεί ένα αίτημα τύπου “POST” στο Route “/api/users/login” του Back – End μας, έχοντας ως Body τα στοιχεία που πληκτρολογήσαμε. Εάν το αίτημα γίνει “Fulfilled”, τότε η παγκόσμια μεταβλητή “User”, που είναι τύπου “auth”, θα «γεμίσει» με τα στοιχεία του Χρήστη, μαζί και με το Token που δημιουργήθηκε. Επίσης, θα δημιουργηθεί μία νέα εγγραφή στο Localstorage του Browser, με όνομα “user”, που περιέχει τα ίδια στοιχεία. Στη συνέχεια, ο Χρήστης με τη χρήση του Hook “useNavigate()” θα ανακατευθύνει στην Αρχική Σελίδα. Εάν το αίτημα γίνει “Rejected”, τότε με τη βοήθεια του “React Tostify” εμφανίζεται σχετικό μήνυμα λάθους, όπως φαίνεται και στο Σχήμα 3.11.

```
const onSubmit = (e) => {
    e.preventDefault()

    const userData = {
        name,
        password,
    }

    dispatch(login(userData))
        .unwrap()
        .then(() => {
            //if login success navigate to home
            navigate('/')
        })
        //if login failed display message
        .catch(toast.error)
}
```

Σχήμα 3.11: Σύνταξη συνάρτησης “onSubmit()”.

3.4 Αρχική Σελίδα Χρήστη

Όπως βλέπουμε στο Σχήμα 3.12, με την είσοδο του εκάστοτε Χρήστη στην «Αρχική Σελίδα», εμφανίζονται στοιχεία σχετικά με τα παραστατικά που έχουν εκδοθεί τον τρέχοντα μήνα. Πιο συγκεκριμένα, εμφανίζεται το συνολικό ποσό των εκδοθέντων παραστατικών, δηλαδή η καθαρή αξία με προστιθέμενο το Φ.Π.Α., καθώς και ξεχωριστά το ποσό της καθαρής αξίας και του Φ.Π.Α..



Σχήμα 3.12: Απεικόνιση Σελίδας «Αρχική Σελίδα».

Με τη χρήση του Hook “useEffect()”, όταν γίνεται Render η Σελίδα «Αρχική Σελίδα» για πρώτη φορά, δημιουργούμε μία συνάρτηση τύπου “arrow”, όπου με τη χρήση του Hook “useDispatch()” καλείται η συνάρτηση “getMontlyInvoices()”, όπως φαίνεται στο Σχήμα 3.13. Εάν αυτή γίνει “Fulfilled”, επιστρέφει τα στοιχεία που είναι προς εμφάνιση και τα αποθηκεύει στο Redux σε μία global μεταβλητή “montlyInvoices” τύπου “invoice”. Στα περιεχόμενα της συγκεκριμένης μεταβλητής έχουμε πρόσβαση με τη χρήση του Hook “useSelector()”. Επίσης, υπάρχουν τέσσερα κουμπιά, τα οποία βοηθούν τον χρήστη να ανακατευθυνθεί ευκολότερα και συντομότερα στις Σελίδες που χρησιμοποιεί συχνότερα με τη βοήθεια του Element “Link” της Βιβλιοθήκης React – Router – DOM.

3.5 Δημιουργία Νέου Πελάτη

Ο Χρήστης, ενώ βρίσκεται στην «Αρχική Σελίδα», εάν κάνει κλικ στο κουμπί «Πελάτες» στα αριστερά της, θα εμφανιστεί μία λίστα, αντίστοιχη με αυτή του Σχήματος 3.15, με όλους τους καταχωρημένους – από αυτόν – πελάτες.

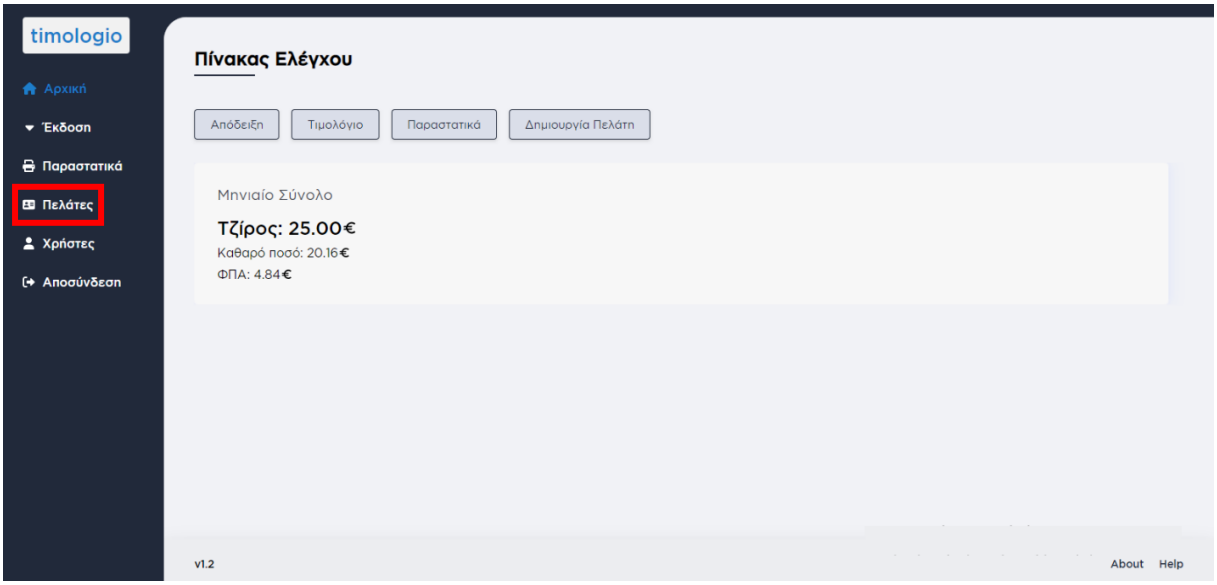
Ο Χρήστης, κάνοντας κλικ στο κουμπί «Δημιουργία Νέου Πελάτη» πάνω δεξιά, θα μεταφερθεί σε μία σελίδα, όπου θα μπορεί να δημιουργήσει έναν νέο πελάτη. Η Σελίδα «Δημιουργία Νέου Πελάτη» αποτελείται από ένα πεδίο αναζήτησης και ένα form. Ο χρήστης έχει δύο επιλογές:

- Μπορεί να συμπληρώσει μόνος του τα απαιτούμενα πεδία της φόρμας.

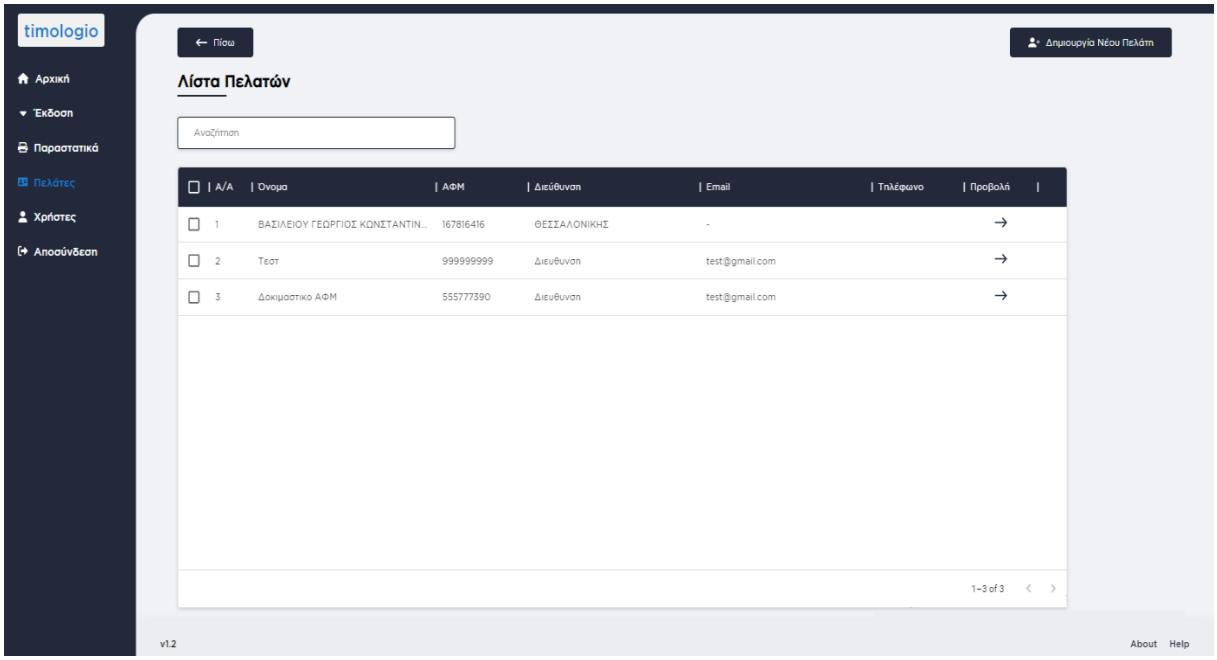
- Μπορεί να συμπληρώσει το Α.Φ.Μ. του πελάτη στο πεδίο «Αναζήτηση ΑΦΜ», να κάνει κλικ στο κουμπί «Αναζήτηση», να γίνει η αναζήτηση των στοιχείων του πελάτη και τελικά η αυτόματη συμπλήρωσή τους μέσω της εφορίας.

```jsx
import { useEffect } from 'react'
import Footer from '../components/Footer'
import { Link } from 'react-router-dom'
import { getMontlyInvoices } from '../features/invoice/invoiceSlice'
import { useSelector, useDispatch } from 'react-redux'
import MonthlyTotals from '../components/MonthlyTotals'

function Home() {
   const { montlyInvoices, isSuccessMontlyInvoice } = useSelector((state) => state.invoice)

   const dispatch = useDispatch()

   useEffect(() => {
      dispatch(getMontlyInvoices())
   }, [dispatch])

   return (
      <div className='page-content top-left-corner'>
         <div className='container'>
            <h2 className='page-title'>Πίνακας Ελέγχου</h2>
            <div className='usefullLinks-container'>
               <div className='usefullLink'>
                  <Link to='/apodeixh'>Απόδειξη</Link>
               </div>
               <div className='usefullLink'>
                  <Link to='/timologio'>Τιμολόγιο</Link>
               </div>
               <div className='usefullLink'>
                  <Link to='/invoices'>Παραστατικά</Link>
               </div>
               <div className='usefullLink'>
                  <Link to='/customers/new-customer'>Δημιουργία Πελάτη</Link>
               </div>
            </div>

            <MonthlyTotals montlyInvoices={montlyInvoices} isSuccessMontlyInvoice={isSuccessMontlyInvoice} />
         </div>

         <Footer />
      </div>
   )
}

export default Home
```

Σχήμα 3.13: Δημιουργία Σελίδας «Αρχική Σελίδα».

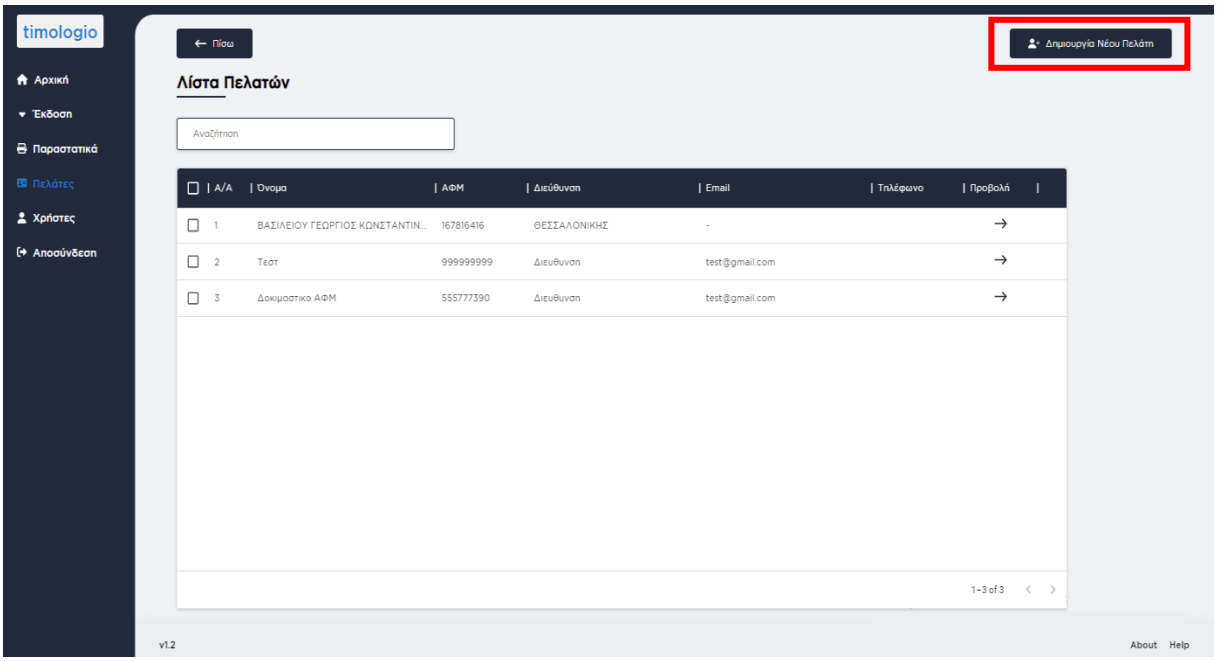


Σχήμα 3.14: Κλικ στο κουμπί «Πελάτες» για εμφάνιση της λίστας των καταχωρημένων πελατών.

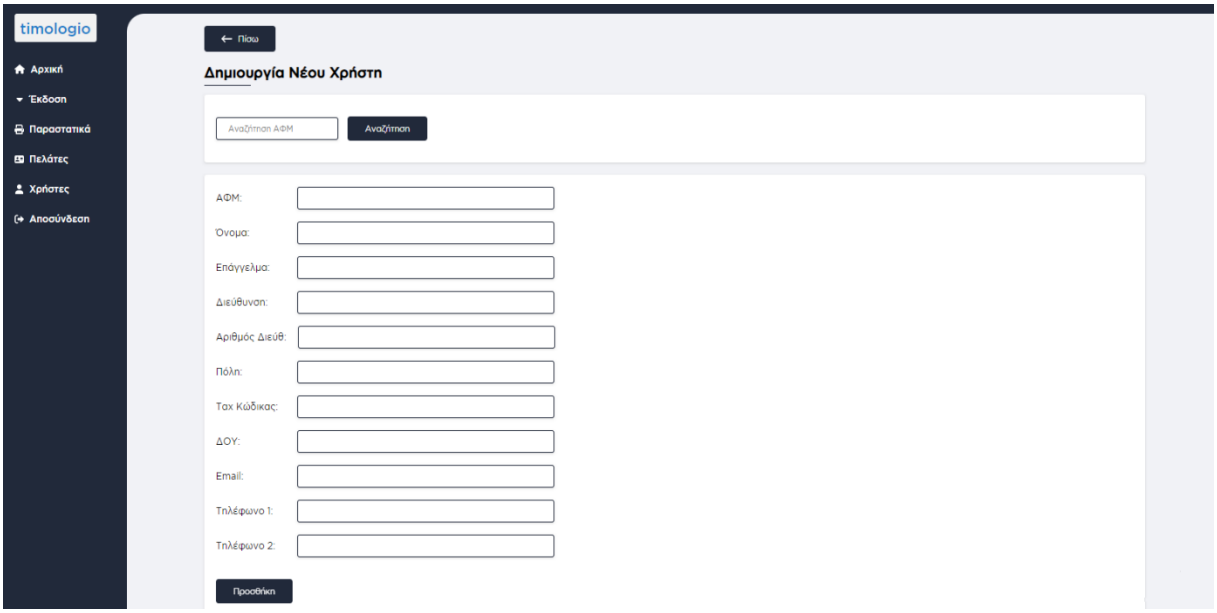


Σχήμα 3.15: Λίστα καταχωρημένων πελατών Χρήστη.

Τα πεδία του form διαχειρίζονται από το Hook “useState()” και τη συνάρτηση “onChange()”. Κάθε φορά που ο Χρήστης εισάγει ή διαγράφει δεδομένα, καλείται η συνάρτηση “onChange()”, η οποία ανανεώνει τη κατάσταση των μεταβλητών.



Σχήμα 3.16: Κλικ στο κουμπί «Δημιουργία Νέου Πελάτη» για τη δημιουργία νέου πελάτη.



Σχήμα 3.17: Σελίδα «Δημιουργία Νέου Πελάτη».

```
<div className='page-content top-left-corner'>
  <div className='container'>
    <BackButton url='/customers' />
    <h2 className='page-title'>Δημιουργία Νέου Χρήστη</h2>
    <div className='box'>
      <div className='new-customer-afm-search'>
        <form onSubmit={SearchCustomer}>
          <input  type='number'  className='create-user' id='reqAfm'  name='reqAfm' value={reqAfm || ''}  placeholder='Αναζήτηση ΑΦΜ' onChange={(e) => setReqAfm(e.target.value)}
          />
          <button className='mg-1-0 btn-submit mg-left-10'>
            {isLoadingAfm ? (
              <span className='small-spiner-flex'>
                Αναζήτηση <SmallSpinner />
              </span>
            ) : (
              'Αναζήτηση'
            )}
          </button>
        </form>
      </div>
    </div>
```

Σχήμα 3.18: Δημιουργία του κουμπιού «Δημιουργία Νέου Πελάτη» στη σελίδα «Λίστα Πελατών».

```
<form onSubmit={onSubmit}>
  <div className='box create-customer-form'>
    <label className='label-flex-input'>
      <span className='min-width-120'>ΑΦΜ:</span>
      <input
        type='text' className='create-user' id='afm' name='afm' value={afm || ''} onChange={onChange} required
      />
    </label>
    <label className='label-flex-input'>
      <span className='min-width-120'>Όνομα:</span>
      <input
        type='text' className='create-user' id='name' name='name' value={name || ''} onChange={onChange} required
      />
    </label>
    <label className='label-flex-input'>
      <span className='min-width-120'>Επάγγελμα:</span>
      <input
        type='text'  className='create-user' id='profession' name='profession' value={profession || ''}  onChange={onChange} required
      />
    </label>
    <label className='label-flex-input'>
      <span className='min-width-120'>Διεύθυνση:</span>
      <input
        type='text'  className='create-user' id='address' name='address'  value={address || ''} onChange={onChange} required
      />
    </label>
    <label className='label-flex-input'>
      <span className='min-width-120'>Αριθμός Διεύθ:</span>
      <input
        type='number' className='create-user'  id='addressNumber'  name='addressNumber' value={addressNumber || ''} onChange={onChange} required
      />
    </label>
    <label className='label-flex-input'>
      <span className='min-width-120'>Πόλη:</span>
      <input
        type='text' className='create-user' id='city' name='city' value={city || ''} onChange={onChange} required
      />
    </label>
    <label className='label-flex-input'>
      <span className='min-width-120'>Ταχ Κώδικας:</span>
      <input type='number' className='create-user' id='postalCode' name='postalCode' value={postalCode || ''} onChange={onChange} required
      />
    </label>
```

Σχήμα 3.19: Δημιουργία του form στη Σελίδα «Δημιουργία Νέου Πελάτη» (1).

```
            <label className='label-flex-input'>
              <span className='min-width-120'>ΔΟΥ:</span>
              <input type='text' className='create-user' id='doy'  name='doy' value={doy || ''} onChange={onChange} required
              />
            </label>
            <label className='label-flex-input'>
              <span className='min-width-120'>Email:</span>
              <input type='email' className='create-user'  id='email' name='email' value={email || ''}  onChange={onChange}
              />
            </label>
            <label className='label-flex-input'>
              <span className='min-width-120'>Τηλέφωνο 1:</span>
              <input type='number' className='create-user'  id='phone1'  name='phone1' value={phone1 || ''}  onChange={onChange}
              />
            </label>
            <label className='label-flex-input'>
              <span className='min-width-120'>Τηλέφωνο 2:</span>
              <input type='number' className='create-user' id='phone2' name='phone2' value={phone2 || ''} onChange={onChange}
              />
            </label>
            <div>
              <button className='mg-1-0 btn-submit'>Προσθήκη</button>
            </div>
          </div>
        </form>
      </div>
      <Footer />
```

Σχήμα 3.20: Δημιουργία του form στη Σελίδα «Δημιουργία Νέου Πελάτη» (2).

Εάν ο Χρήστης επιλέξει να κάνει αναζήτηση Α.Φ.Μ., συμπληρώνοντάς το στο πεδίο «Αναζήτηση ΑΦΜ», τότε κάνοντας κλικ στο κουμπί «Αναζήτηση» καλείται η συνάρτηση "SearchCustomer()" του Σχήματος 3.21, η οποία μετά τον έλεγχο για τον σωστό αριθμό ψηφίων – με τη χρήση του hook "useDispatch()" – καλεί την ασύγχρονη συνάρτηση "getInfo()" με παράμετρο το Α.Φ.Μ. που εισαγάγαμε. Η συνάρτηση "getInfo()" δημιουργεί ένα αίτημα "POST" στο Route "/api/afm", έχοντας ως Body το Α.Φ.Μ. του πελάτη. Εάν το αίτημα γίνει "Fulfilled", τότε τα στοιχεία του πελάτη αποθηκεύονται στη global σταθερά "info" τύπου "afm", στην οποία έχουμε πρόσβαση με τη χρήση του Hook "useSelector()". Έτσι, παίρνουμε τις κατάλληλες τιμές από την σταθερά "info" και τις τοποθετούμε στα αντίστοιχα πεδία του form.

```
// Get customer info from afm
const SearchCustomer = (e) => {
   e.preventDefault()

   if (reqAfm.length !== 9) {
      toast.error('Παρακαλώ εισάγετε 9 ψηφία')
      return
   }
   dispatch(getInfo({ reqAfm }))
      .unwrap()
      .then((info) => {
         // if success place info in input values
         const { rname, rproffesion, raddress, raddressNumber, rcity, rzip_code, rdoy } = info
         toast.success('Επιτυχής αναζήτηση πελάτη')
         setFormData({
            afm: reqAfm,
            name: rname,
            profession: rproffesion,
            address: raddress,
            addressNumber: raddressNumber,
            city: rcity,
            postalCode: rzip_code,
            doy: rdoy,
         })
      })
      // if error display message
      .catch(toast.error)
}
```

Σχήμα 3.21: Σύνταξη της συνάρτησης "SearchCustomer()".

Όταν τα πεδία συμπληρωθούν, ο Χρήστης πρέπει να αποθηκεύσει τον πελάτη στη βάση. Κάνοντας κλικ στο κουμπί «Προσθήκη», καλείται η συνάρτηση “onSubmit()”. Με τη χρήση του Hook “useDispatch()” καλείται η συνάρτηση “createCustomer()” του Σχήματος 3.22, έχοντας ως παράμετρο τα στοιχεία του form. Η συνάρτηση “createCustomer()” δημιουργεί ένα αίτημα “POST” στο Route “/api/customer”, έχοντας ως Body τα πεδία του form. Εάν το αίτημα γίνει “Fulfilled”, εμφανίζεται σχετικό μήνυμα επιτυχημένης εγγραφής πελάτη, τα στοιχεία του form γίνονται “Reset” και ο Χρήστης μεταφέρεται στη Σελίδα του πελάτη που μόλις δημιούργησε. Η μεταφορά γίνεται χρησιμοποιώντας το Α.Φ.Μ. του πελάτη.

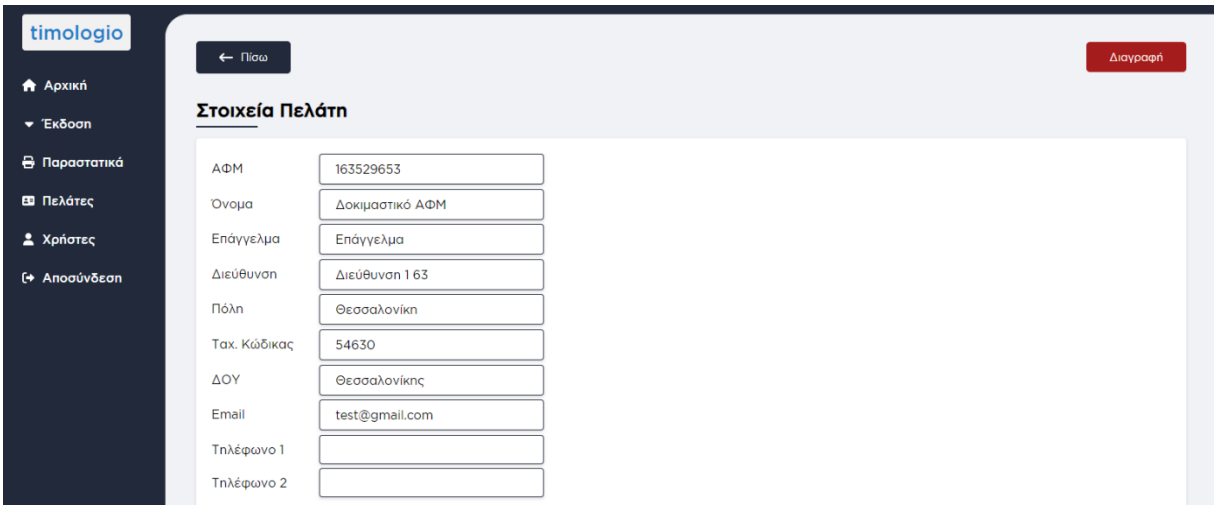
```
// Create new customer
const onSubmit = (e) => {
   e.preventDefault()

   dispatch(
      createCustomer({ afm, name, profession, address, addressNumber, city, postalCode, doy, email, phone1, phone2 })
   )
      .unwrap()
      .then(() => {
         toast.success('Επιτυχής καταχώρηση πελάτη')
         navigate(`/customers/${afm}`)
         // Reset form data
         setFormData({
            afm: '',
            name: '',
            profession: '',
            address: '',
            addressNumber: '',
            city: '',
            postalCode: '',
            doy: '',
            email: '',
            phone1: '',
            phone2: '',
         })
      })
      .catch(toast.error)
}
```

Σχήμα 3.22: Σύνταξη της συνάρτησης “createCostumer()”.

3.6 Προβολή Πελάτη

Στη Σελίδα αυτή εμφανίζονται τα στοιχεία του πελάτη που αποθηκεύσαμε. Στην εν λόγω Σελίδα υπάρχει ένα form, ώστε να μπορεί ο Χρήστης να αλλάξει μερικά από τα στοιχεία του πελάτη, όπως τη διεύθυνση e – mail, τα τηλέφωνα 1 και 2. Επίσης, μπορεί να διαγράψει ολοκληρωτικά έναν χρήστη, πατώντας το κουμπί «Διαγραφή» πάνω δεξιά, όπως φαίνεται στο Σχήμα 3.23.

Όταν ο Χρήστης μεταβεί στη σελίδα ενός πελάτη, καθώς φορτώνει η σελίδα, τρέχει το Hook “useEffect()”, δημιουργώντας μία συνάρτηση “getCustomer()” τύπου “arrow”, όπως φαίνεται στο Σχήμα 3.24. Η συνάρτηση “getCustomer()” καλείται με τη χρήση του Hook “useDispatch()”, έχοντας ως παράμετρο το Α.Φ.Μ. του πελάτη που πήραμε από το URL με τη χρήση του Hook “useParams()”. Η συνάρτηση “getCustomer()” δημιουργεί ένα αίτημα “GET” στο Route “/api/customers/customerId”, έχοντας ως Body το Α.Φ.Μ. του πελάτη. Εάν το αίτημα γίνει “Fulfilled”, τότε τα στοιχεία που επιστρέφονται, αποθηκεύονται στη global μεταβλητή “customer” τύπου “customers”. Από εκεί έχουμε πρόσβαση στα στοιχεία του πελάτη με τη χρήση του Hook “useSelector()” και έτσι τα εμφανίζουμε στη σελίδα. Εάν το αίτημα γίνει “Rejected”, εμφανίζεται σχετικό μήνυμα λάθους, όπως φαίνεται στο Σχήμα 3.25.



Σχήμα 3.23: Σελίδα προβολής στοιχείων νεοεισαχθέντος πελάτη.

```
// Get customer from global const
const { customer, isLoadingCustomer } = useSelector((state) => state.customers)

// Save AFM from URI
const { afm } = useParams()

// Call getCustomer to get all customers
useEffect(() => {
    dispatch(getCustomer(afm))
    .unwrap()
    .then(() => {})
    .catch(toast.error)
}, [afm])
```

Σχήμα 3.24: Σύνταξη συνάρτησης “getCostumer()”.

Το κουμπί «Διαγραφή» είναι ένα Component που δημιουργούμε και κάνουμε import στην αρχή της Σελίδας. Δέχεται ως αυθαίρετη είσοδο (Props) το id του πελάτη. Όταν ο Χρήστης κάνει κλικ στο κουμπί «Διαγραφή», καλείται η συνάρτηση “onDeleteCustomer()”, η οποία με τη χρήση του Hook “useDispatch()” καλεί τη συνάρτηση “deleteCustomer()”, έχοντας ως παράμετρο το id του πελάτη. Η συνάρτηση “deleteCustomer()” δημιουργεί ένα αίτημα “DELETE” στο Route “/api/customers/customerID”, έχοντας ως Body το id του πελάτη. Εάν το αίτημα γίνει “Fulfilled”, τότε ο πελάτης έχει διαγραφεί και ο Χρήστης θα μεταφερθεί στη σελίδα πελατών. Εάν το αίτημα γίνει “Rejected”, εμφανίζεται σχετικό μήνυμα λάθους, όπως βλέπουμε και στο Σχήμα 3.27.

Εάν ο Χρήστης θέλει να αλλάξει τα στοιχεία του πελάτη, κάνει κλικ στο κουμπί «Ενημέρωση». Τότε εμφανίζεται ένα “dialog”, το οποίο έχουμε κάνει import από τη Βιβλιοθήκη του Material UI, το οποίο απεικονίζεται στο Σχήμα 3.28. Το “dialog” περιέχει ένα form, ώστε ο Χρήστης να πληκτρολογήσει τις νέες επιθυμητές τιμές.

```
return (
  <div className='page-content top-left-corner'>
    <div className='container'>
      <div className='customers-top-btns'>
        <BackButton url='/customers' />
        <DeleteButton customerId={customerId} />
      </div>

      <h2 className='page-title'>Στοιχεία Πελάτη</h2>
      <div className='box'>
        <div className='customer-info'>
          <div className='min-width-120'>ΑΦΜ</div>
          <div className='customer-info-data'>{customer.afm}</div>
        </div>
        <div className='customer-info'>
          <div className='min-width-120'>Όνομα</div>
          <div className='customer-info-data'>{customer.name}</div>
        </div>
        <div className='customer-info'>
          <div className='min-width-120'>Επάγγελμα</div>
          <div className='customer-info-data'>{customer.profession}</div>
        </div>
        <div className='customer-info'>
          <div className='min-width-120'>Διεύθυνση</div>
          <div className='customer-info-data'>
            {customer.address} {customer.addressNumber}
          </div>
        </div>
        <div className='customer-info'>
          <div className='min-width-120'>Πόλη</div>
          <div className='customer-info-data'>{customer.city}</div>
        </div>
        <div className='customer-info'>
          <div className='min-width-120'>Ταχ. Κώδικας</div>
          <div className='customer-info-data'>{customer.postalCode}</div>
        </div>
        <div className='customer-info'>
          <div className='min-width-120'>ΔΟΥ</div>
          <div className='customer-info-data'>{customer.doy}</div>
        </div>
        <div className='customer-info'>
          <div className='min-width-120'>Email</div>
          <div className='customer-info-data'>{customer.email}</div>
        </div>
```

Σχήμα 3.25: Δημιουργία Σελίδας προβολής στοιχείων νεοεισαχθέντος πελάτη (1).

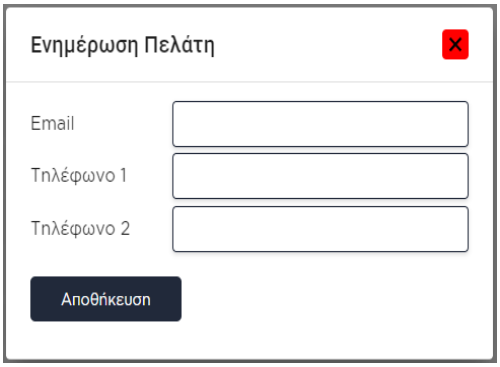
```
        <div className='customer-info'>
          <div className='min-width-120'>Τηλέφωνο 1</div>
          <div className='customer-info-data'>{customer.phone1}</div>
        </div>
        <div className='customer-info'>
          <div className='min-width-120'>Τηλέφωνο 2</div>
          <div className='customer-info-data'>{customer.phone2}</div>
        </div>
```

Σχήμα 3.26: Δημιουργία Σελίδας προβολής στοιχείων νεοεισαχθέντος πελάτη (2).

Συμπληρώνοντας ο Χρήστης τα νέα στοιχεία και κάνοντας κλικ στο κουμπί «Αποθήκευση», καλείται η συνάρτηση “onSubmit()”, η οποία δημιουργεί ένα νέο Object με τα νέα στοιχεία, και με τη χρήση του Hook “useDispatch()”, καλείται η συνάρτηση “updateCustomer()”, έχοντας ως παραμέτρους της το id του πελάτη και τα νέα στοιχεία. Η συνάρτηση “updateCustomer()” δημιουργεί ένα νέο αίτημα “POST” στο Route “/api/customers/customerID”, έχοντας ως Body τα νέα στοιχεία. Εάν το αίτημα γίνει “Fulfilled”, ο Χρήστης μεταφέρεται στη σελίδα πελατών με τη χρήση του Hook “useNavigate()”. Εάν το αίτημα γίνει “Rejected”, εμφανίζεται σχετικό μήνυμα λάθους, όπως φαίνεται στο Σχήμα 3.30.

```jsx
import React from 'react'
import { useDispatch } from 'react-redux'
import { useNavigate } from 'react-router-dom'
import { deleteCustomer } from '../features/customer/customerSlice'
import { toast } from 'react-toastify'

function DeleteButton({ customerId }) {
   const dispatch = useDispatch()
   const navigate = useNavigate()

   // delete Customer
   const onDeleteCustomer = () => {
      dispatch(deleteCustomer(customerId))
         .unwrap()
         .then(() => {
            toast.success('Επιτυχής διαγραφή πελάτη')
            navigate('/customers')
         })
         .catch(toast.error)
   }
   return (
      <button className='mg-1-0 btn-delete' onClick={onDeleteCustomer}>
         Διαγραφή
      </button>
   )
}

export default DeleteButton
```

Σχήμα 3.27: Σύνταξη συνάρτησης “DeleteButton()”.



Σχήμα 3.28: Form ενημέρωσης στοιχείων πελάτη.

```jsx
<Dialog open={openPopup} presentation>
   <div ref={popupRef}>
      <DialogTitle>
         <div className='dialog-header'>
            <span>Ενημέρωση Πελάτη</span>
            <GrFormClose size={26} style={mystyle} onClick={() => setOpenPopup(false)} />
         </div>
      </DialogTitle>
      <DialogContent dividers>
         <form onSubmit={onSubmit}>
            <div className='customer-info'>
               <div className='min-width-120'>Email</div>
               <div className='customer-info-data'>
                  <input type='email' className='create-user'  id='email' name='email' value={email} onChange={onChange}
                  />
               </div>
            </div>
            <div className='customer-info'>
               <div className='min-width-120'>Τηλέφωνο 1</div>
               <div className='customer-info-data'>
                  <input type='number' className='create-user' id='phone1' name='phone1' value={phone1} onChange={onChange}
                  />
               </div>
            </div>
            <div className='customer-info'>
               <div className='min-width-120'>Τηλέφωνο 2</div>
               <div className='customer-info-data'>
                  <input type='number' className='create-user' id='phone2' name='phone2' value={phone2}  onChange={onChange}
                  />
               </div>
            </div>
            <div>
               <button className='mg-1-0 btn-submit'>Αποθήκευση</button>
            </div>
         </form>
      </DialogContent>
   </div>
</Dialog>
```

Σχήμα 3.29: Δημιουργία form ενημέρωσης στοιχείων πελάτη.

```
// initialize hooks
const dispatch = useDispatch()
const navigate = useNavigate()

// update Customer
const onSubmit = (e) => {
   e.preventDefault()

   // Create new object with user data
   const userData = {
      email,
      phone1,
      phone2,
   }

   dispatch(updateCustomer({ Customer: customerId, Data: userData }))
      .unwrap()
      .then(() => {
         // if success
         // close pop up
         setOpenPopup(false)
         // display success message
         toast.success('Επιτυχής ενημέρωση πελάτη')
         // navigate to /customers
         navigate('/customers')
      })
      // if failed display error message
      .catch(toast.error)
}
```

Σχήμα 3.30: Σύνταξη συνάρτησης “updateCustomer()”.

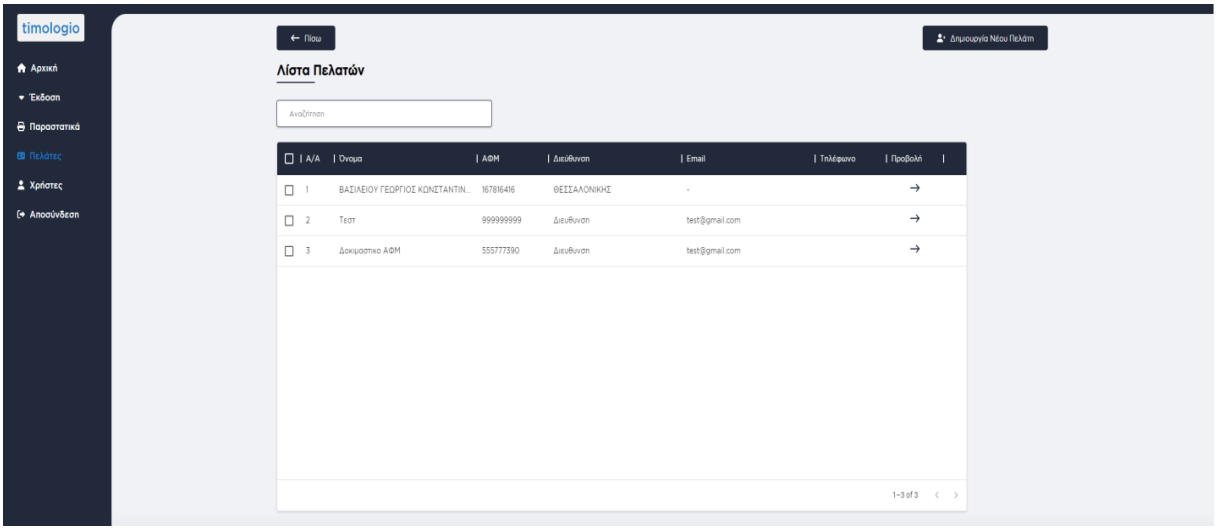
Κάνοντας κλικ στο κουμπί «Έκδοση Τιμολογίου», ο χρήστης μεταφέρεται στη Σελίδα δημιουργίας τιμολογίου, περνώντας ως query το Α.Φ.Μ. του πελάτη με τη χρήση.

3.7 Προβολή Λίστας Πελατών

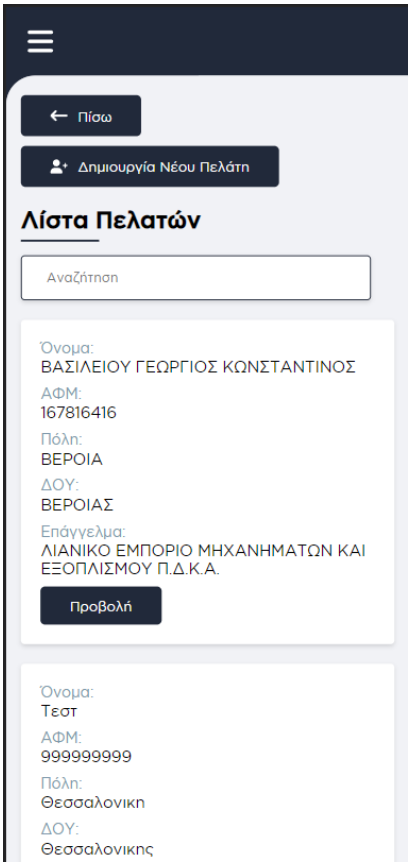
Στη συγκεκριμένη Σελίδα εμφανίζονται όσοι πελάτες έχουν καταχωρηθεί από τους εκάστοτε Χρήστες. Η εμφάνισή τους γίνεται υπό τη μορφή πίνακα σε οθόνες ηλεκτρονικού υπολογιστή και κάρτας για κάθε πελάτη ξεχωριστά σε οθόνες κινητών τηλεφώνων, όπως φαίνεται στα Σχήματα 3.31 και 3.32 αντίστοιχα. Πιο συγκεκριμένα, εάν το πλάτος της Σελίδας αυτής είναι μεγαλύτερο των 768 pixels, τότε οι πελάτες εμφανίζονται σε έναν πίνακα, τον οποίο κάνουμε import από τη Βιβλιοθήκη του Material UI. Εάν το πλάτος οθόνης είναι μικρότερο των 768 pixels, τότε εμφανίζεται ένα Component, το οποίο εμφανίζει μία «κάρτα» με ορισμένα στοιχεία του εκάστοτε πελάτη.

Υπάρχει, επίσης, το πεδίο «Αναζήτηση», όπου ο Χρήστης μπορεί να αναζητήσει ένα πελάτη βάσει Α.Φ.Μ. ή επωνυμίας. Το πεδίο αυτό διαχειρίζεται από το Hook “useState()”. Όταν η Σελίδα γίνεται Render, τρέχει το Hook “useEffect()” και δημιουργείται μία συνάρτηση τύπου “arrow”, η οποία αρχικά ελέγχει το μέγεθος (length) του “query”. Εάν το μέγεθος είναι μηδέν ή μεγαλύτερο του δύο, τότε με τη χρήση του Hook “useDispatch()” καλείται η συνάρτηση “getCustomers()”, η οποία έχει ως παράμετρο το query. Η συνάρτηση “getCustomers()” δημιουργεί ένα αίτημα “GET” στο Route “/api/customers”, έχοντας ως Body ένα Object που έχει τη τιμή του query. Εάν η τιμή του “query” είναι μηδέν, σημαίνει ότι ζητούμε την εμφάνιση όλων των πελατών, ενώ εάν περιέχει περισσότερους από δύο χαρακτήρες, σημαίνει ότι ζητούμε τους πελάτες με τα συγκεκριμένα κριτήρια. Εάν το αίτημα γίνει “Fulfilled”, τότε οι τιμές αποθηκεύονται στη global μεταβλητή “customers” τύπου “customers”.

Από εκεί έχουμε πρόσβαση στα στοιχεία του πελάτη με τη χρήση του Hook “useSelector()”, και έτσι τα εμφανίζουμε στη Σελίδα. Κάθε φορά που αλλάζει η τιμή του “query”, ξανακαλείται η συνάρτηση “useEffect()”, κάνοντας νέο αίτημα, ώστε να έχουμε τα σωστά δεδομένα. Εάν το αίτημα γίνει “Rejected”, εμφανίζεται σχετικό μήνυμα λάθους, όπως φαίνεται στο Σχήμα 3.34.



Σχήμα 3.31: Εμφάνιση Σελίδας «Λίστα Πελατών» σε οθόνη ενός ηλεκτρονικού υπολογιστή.



Σχήμα 3.32: Εμφάνιση Σελίδας «Λίστα Πελατών» σε οθόνη ενός κινητού τηλεφώνου.

```
return (
   <div className='page-content top-left-corner'>
      <div className='container'>
         <div className='customers-top-btns'>
            <BackButton url='/' />
            <Link to='/customers/new-customer' className='btn-back'>
               <CustomerPlus fill='#fff' width='18px' height='18px' /> Δημιουργία Νέου Πελάτη
            </Link>
         </div>

         <h2 className='page-title'>Λίστα Πελατών</h2>
         <input
            className='customer-search min-height-24'
            placeholder='Αναζήτηση'
            onChange={(e) => setQuery(e.target.value)}
         />
         <div className='clients-table'>
            {isLoadingCustomer ? (
               <MediumSpinner />
            ) : (
               <ClientsList
                  sx={{ display: { xs: 'none', sm: 'none', md: 'none' } }}
                  customers={customers}
                  className='list-10'
               />
            )}
         </div>
         <div className='invoices-mobile-list'>
            <MobileCustomersList customers={currentCustomers} />
         </div>
         <Stack spacing={2} className='pagination-div'>
            <Pagination
               count={Math.ceil(customers.length / perPage)}
               onChange={pageChange}
               variant='outlined'
               shape='rounded'
               className='pagination-list'
            />
         </Stack>
      </div>
      <Footer />
   </div>
)
```

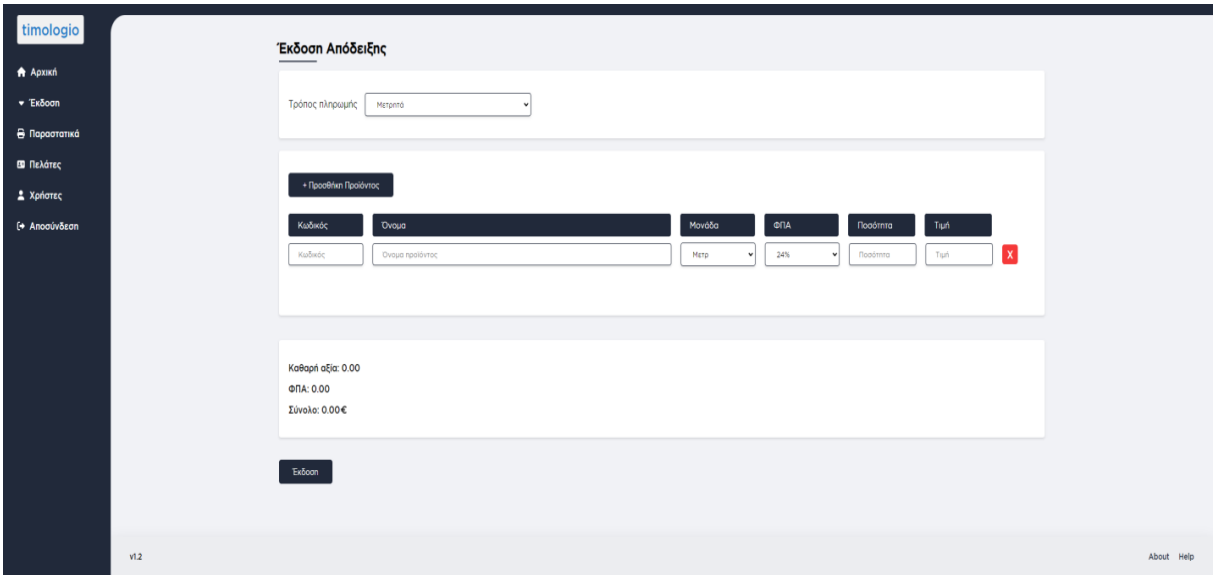
Σχήμα 3.33: Δημιουργία Σελίδας «Λίστα Πελατών».

```
const dispatch = useDispatch()

useEffect(() => {
   if (query.length === 0 || query.length > 2) {
      dispatch(getCustomers(query))
         .unwrap()
         .then(() => {})
         // if error display error message
         .catch(toast.error)
   }
}, [query])
```

Σχήμα 3.34: Σύνταξη Hook “useEffect()”.

3.8 Δημιουργία Απόδειξης Λιανικής Πώλησης

Στην εν λόγω Σελίδα ο Χρήστης μπορεί να εκδώσει νέο παραστατικό με ονομασία «Απόδειξη Λιανικής Πώλησης». Αφού επιλέξει τρόπο πληρωμής, θα πρέπει να κάνει κλικ στο κουμπί «Προσθήκη Προϊόντος». Αφού προσθέσει τα προϊόντα που επιθυμεί, συμπληρώνοντας όλα τα διαθέσιμα πεδία για το κάθε προϊόν, θα πρέπει να κάνει κλικ στο κουμπί «Έκδοση», ούτως ώστε να εκδοθεί το νέο παραστατικό, όπως φαίνεται στο Σχήμα 3.35.



Σχήμα 3.35: Σελίδα «Έκδοση Απόδειξης».

```
return (
  <div className='page-content top-left-corner'>
    <div className='container'>
      <h2 className='page-title'>Έκδοση Απόδειξης</h2>
      <form onSubmit={onSubmit} className='apodeixhForm'>
        <div className='payment-type box'>
          <label>
            Τρόπος πληρωμής
            <select defaultValue='3' onChange={changePaymentType} className='mg-1 ' id='tropos_plhrwmhs'>
              <option value='1'>Επαγ. Λογαριασμός Πληρωμών Ημεδαπής</option>
              <option value='2'>Επαγ. Λογαριασμός Πληρωμών Αλλοδαπής</option>
              <option value='3'>Μετρητά</option>
              <option value='4'>Επιταγή</option>
              <option value='5'>Επί Πιστώσει</option>
              <option value='6'>WebBanking</option>
              <option value='7'>POS / e-POS</option>
            </select>
          </label>
        </div>
      </form>
      <ProductInput onChange={getProducts} />
      <TotalPreview onChange={getTotalNetValue} products={products} />
      <div className='btn-submit-container'>
        <button type='submit' className='mg-1-0 btn-submit' onClick={onSubmit}>
          Έκδοση
        </button>
      </div>
    </div>
    <Footer />
  </div>
)
```

Σχήμα 3.36: Δημιουργία Σελίδας «Έκδοση Απόδειξης».

Η Σελίδα μας, η οποία αποτελεί στοιχείο React, ονομάζεται “Apodeixh.jsx” και αποτελείται από:

- Ένα form, το οποίο διαχειρίζεται τη κατάσταση του τρόπου πληρωμής με τη χρήση του Hook “useState()” και της συνάρτησης “changePaymentType()”, η οποία εικονίζεται στο Σχήμα 3.37.
- Ένα Component που διαχειρίζεται τη προσθήκη προϊόντων, ονομάζεται “ProductInput.jsx”, αποτελεί επίσης στοιχείο React, και τέλος,
- Ένα Component, το οποίο είναι στοιχείο React, είναι υπεύθυνο για την απεικόνιση συνόλων (συνολική Καθαρή Αξία, συνολικό Φ.Π.Α. και Σύνολο) και ονομάζεται “TotalPreview.jsx”.

```
// Change payment type
const changePaymentType = (e) => setTroposPlhrwmhs(e.target.value)
```

Σχήμα 3.37: Σύνταξη συνάρτησης “changePaymentType()”.

Στο Component για την εισαγωγή των προϊόντων ορίζουμε μία μεταβλητή με όνομα “products” με τη χρήση του Hook “useState()” και την ορίζουμε ως ένα κενό “array”. Όταν ο χρήστης κάνει κλικ στο κουμπί «Προσθήκη Προϊόντος», τότε καλείται η συνάρτηση “addProducts()”, η σύνταξη της οποίας απεικονίζεται στο Σχήμα 3.38. Αυτή ορίζει μία σταθερά τύπου object, η οποία περιέχει τις αρχικές τιμές για τα πεδία του κάθε προϊόντος και ορίζει μεταβλητές λαθών για κάθε τιμή. Αρχικά, όλες οι μεταβλητές λαθών είναι “null”. Τέλος, γίνεται έλεγχος για το εάν όλα τα πεδία του προϊόντος έχουν συμπληρωθεί σωστά. Ο εν λόγω έλεγχος γίνεται, καλώντας τη συνάρτηση “prevIsValid()”. Εάν δεν υπάρχουν λάθη, η συνάρτηση αυτή επιστρέφει “True” και προστίθεται νέα σειρά προϊόντος με κενά πεδία, ώστε να τα συμπληρώσει ο χρήστης.

```
// Add new product line
const addProducts = (e) => {
    e.preventDefault()

    const inputState = {
        product_code: '',
        product_name: '',
        product_count_in: 1,
        product_tax_percentage: 1,
        product_quantity: '',
        product_price: '',
        product_discount: 0,

        errors: {
            product_code: null,
            product_name: null,
            product_count_in: null,
            product_tax_percentage: null,
            product_quantity: null,
            product_price: null,
        },
    }

    if (prevIsValid()) {
        setProducts((prev) => [...prev, inputState])
    }
}
```

Σχήμα 3.38: Σύνταξη συνάρτησης “addProducts()”.

Η συνάρτηση "prevIsValid()" αρχικά ελέγχει εάν υπάρχουν προϊόντα, όπως βλέπουμε στο Σχήμα 3.39. Εάν δεν έχουν προστεθεί προϊόντα στη μεταβλητή "products", δηλαδή εάν το μέγεθός της (length) είναι μηδέν, τότε δεν υπάρχουν άλλα προϊόντα. Επομένως, δεν υπάρχουν λάθη, και έτσι, η συνάρτηση επιστρέφει "True". Εάν υπάρχουν προϊόντα στη μεταβλητή "products", τότε γίνεται έλεγχος για το εάν υπάρχει κάποιο πεδίο που δεν έχει συμπληρωθεί στο τελευταίο προϊόν. Εάν υπάρχει κάποιο κενό πεδίο, κάτω από αυτό εμφανίζεται σχετικό μήνυμα που ενημερώνει τον χρήστη πως δεν έχει συμπληρωθεί, και έτσι, δε γίνεται προσθήκη νέου προϊόντος. Εάν όλα τα πεδία είναι συμπληρωμένα, τότε η συνάρτηση επιστρέφει "True" και γίνεται προσθήκη νέου προϊόντος.

```
// Error hadling for products
const prevIsValid = () => {
   if (products.length === 0) {
      return true
   }

   const emptyField = products.some(
      (item) =>
         item.product_code === '' ||
         item.product_name === '' ||
         item.product_count_in === '' ||
         item.product_tax_percentage === '' ||
         item.product_quantity === '' ||
         item.product_price === ''
   )

   if (emptyField) {
      products.map((item, index) => {
         const allPrev = [...products]
         if (products[index].product_code === '') {
            allPrev[index].errors.product_code = 'Ο κωδικός προϊόντος είναι υποχρεωτικός'
         }
         if (products[index].product_name === '') {
            allPrev[index].errors.product_name = 'Το όνομα προϊόντος είναι υποχρεωτικό'
         }
         if (products[index].product_count_in === '') {
            allPrev[index].errors.product_count_in = 'Η μονάδα μέτρησης είναι υποχρεωτική'
         }
         if (products[index].product_tax_percentage === '') {
            allPrev[index].errors.product_tax_percentage = 'Το ΦΠΑ προϊόντος είναι υποχρεωτικό'
         }
         if (products[index].product_quantity === '') {
            allPrev[index].errors.product_quantity = 'Η ποσότητα είναι υποχρεωτική'
         }
         if (products[index].product_price === '') {
            allPrev[index].errors.product_price = 'Η τιμή είναι υποχρεωτική'
         }

         setProducts(allPrev)

         return
      })
   }

   return !emptyField
}
```

Σχήμα 3.39: Σύνταξη συνάρτησης "prevIsValid()".

Τέλος, επειδή θα χρειαστούμε τα προϊόντα στη σελίδα του Γονέα (Parent), δηλαδή στη σελίδα "Apodeixh.jsx", θα πρέπει να επιστρέψουμε τα δεδομένα μας, δηλαδή το array με τα προϊόντα, πίσω στο Component Γονέα. Αυτό επιτυγχάνεται με τη χρήση του Hook "useEffect()", όπως απεικονίζεται στο Σχήμα 3.40. Κάθε φορά που γίνεται μία αλλαγή στα προϊόντα, ενεργοποιείται η συνάρτηση "useEffect()" και, μέσω του props, περνάει τα προϊόντα στο Component Γονέα.

```
// Send the data back to parent component
useEffect(() => {
    props.onChange(products)
}, [products])
```

Σχήμα 3.40: Σύνταξη Hook “useEffect()”.

Για να μπορέσει το Component Γονέας (Parent) να δει τις τιμές που τού επιστρέφει το Component Παιδί (Child), δημιουργούμε στο Component Γονέα μία συνάρτηση που αποθηκεύει τις τιμές σε μία μεταβλητή που δημιουργήσαμε με τη χρήση του Hook “useState()”.

```
// Declear products
const [products, setProducts] = useState([])

// Get the inserted products from child component
const getProducts = (data) => {
    setProducts(data)
}
```

Σχήμα 3.41: Σύνταξη Hook “useState()”.

Ορίζουμε το Component “TotalPreview.jsx” στη Σελίδα του Γονέα “Apodeixh.jsx”, το οποίο δέχεται σαν αυθαίρετη είσοδο (props) την προαναφερθείσα μεταβλητή “products”. Το Component “TotalPreview.jsx” μπορεί να δεχθεί και τιμές εισόδου για φόρους. Ωστόσο, στην προκειμένη περίπτωση, δε τις έχουμε. Το συγκεκριμένο Component υπολογίζει τη συνολική καθαρή αξία, το συνολικό Φ.Π.Α. και το σύνολο βάσει των τιμών που δέχεται από το props. Στην αρχή, παίρνουμε τις τιμές που δεχόμαστε από το props. Από τη Σελίδα της Απόδειξης δε δέχεται κάποια τιμή για φόρους. Επομένως, η μεταβλητή “taxes” είναι “null”. Στη συνέχεια, όπως φαίνεται και στο Σχήμα 3.42, αρχικοποιούμε ορισμένες μεταβλητές που θα χρησιμοποιήσουμε παρακάτω. Χρησιμοποιούμε το Hook “useEffect()” για να κάνουμε τους απαραίτητους ελέγχους. Στα dependencies array του Hook “useEffect()” χρησιμοποιούμε τη σταθερά products, όπου είναι τα προϊόντα μας. Με τον τρόπο αυτό, κάθε φορά που αλλάζει η σταθερά, θα τρέχει το περιεχόμενο του Hook “useEffect()” και θα εμφανίζονται τα σωστά αποτελέσματα.

```
const [totalSunolo, setTotalSunolo] = useState(0)
const [totalNetValue, setTotalNetValue] = useState(0)
const [totalVatValue, setTotalVatValue] = useState(0)
const [totalWithheldAmount, setTotalWithheldAmount] = useState(0)
const [totalFeesAmount, setTotalFeesAmount] = useState(0)
const [totalStampDutyamount, setTotalStampDutyamount] = useState(0)
const [totalOtherTaxesAmount, setTotalOtherTaxesAmount] = useState(0)

const { products, taxes } = props

let timhFPA = 1.24
let product_net = 0
let product_vat = 0
let product_total = 0
let product_net_all = 0
let product_vat_all = 0
let product_total_all = 0

let totalParakratoumenoi = 0
let totalTelwn = 0
let totalXartoshmo = 0
let totalLoipwn = 0
```

Σχήμα 3.42: Ορισμός και αρχικοποίηση μεταβλητών.

Όπως βλέπουμε στο Σχήμα 3.43, μέσα στο Hook “useEffect()” αρχικά υπάρχει ένα for loop, το οποίο τρέχει για όλα μας τα προϊόντα, δηλαδή για το “products.length”. Μέσα στο for loop ελέγχουμε εάν έχουν συμπληρωθεί σωστά τα πεδία για τη ποσότητα και τη τιμή του προϊόντος. Εν συνεχεία, παίρνουμε τη τιμή Φ.Π.Α. του προϊόντος και υπολογίζουμε τη καθαρή αξία του, το Φ.Π.Α. του, τη συνολική τιμή του – σε περίπτωση που υπάρχουν περισσότερα από ένα τεμάχια – τη συνολική καθαρή αξία των προϊόντων, το συνολικό Φ.Π.Α. και το σύνολο. Όλοι οι υπολογισμοί στρογγυλοποιούνται σε δύο δεκαδικά ψηφία με τη χρήση της συνάρτησης “twoDecimals”, την οποία δημιουργήσαμε και κάναμε import από το αρχείο “format”.

```
useEffect(() => {
   for (let i = 0; i < products.length; i++) {
      if (products[i].product_quantity === '' || products[i].product_price === '') return

      timhFPA = fpaValue(products[i].product_tax_percentage)

      product_total = twoDecimals(products[i].product_quantity * products[i].product_price)
      product_total_all = Number(product_total_all) + Number(product_total)

      product_net = twoDecimals(product_total / timhFPA)
      product_net_all = Number(product_net_all) + Number(product_net)

      product_vat = twoDecimals(product_total - product_net)
      product_vat_all = Number(product_vat_all) + Number(product_vat)
   }

   if (taxes) {
      taxes?.map(({ tax_type, tax_category, tax_underlying_value, tax_amount }) =>
         tax_type === '1'
            ? (totalParakratoumenoi += Number(tax_amount))
            : tax_type === '2'
            ? (totalTelwn += Number(tax_amount))
            : tax_type === '3'
            ? (totalLoipwn += Number(tax_amount))
            : (totalXartoshmo += Number(tax_amount))
      )
   }
   setTotalWithheldAmount(Number(totalParakratoumenoi))
   setTotalFeesAmount(Number(totalTelwn))
   setTotalStampDutyamount(Number(totalXartoshmo))
   setTotalOtherTaxesAmount(Number(totalLoipwn))
   setTotalSunolo(
      Number(product_total_all) -
         Number(totalParakratoumenoi) +
         Number(totalTelwn) +
         Number(totalLoipwn) +
         Number(totalXartoshmo)
   )
   setTotalNetValue(product_net_all)
   setTotalVatValue(product_vat_all)
}, [products, taxes])
```

Σχήμα 3.43: Σύνταξη Hook “useEffect()” με for loop.

Αφού γίνουν οι συγκεκριμένοι υπολογισμοί και τελειώσει το for loop, γίνεται έλεγχος για το εάν υπάρχουν φόροι. Στην συγκεκριμένη περίπτωση είπαμε ότι δεν έχουμε. Έπειτα, όλες οι τιμές που υπολογίσαμε αποθηκεύονται σε σταθερές που δημιουργήσαμε με τη χρήση του Hook “useState()”. Στην περίπτωση της απόδειξης, επειδή δεν έχουμε φόρους, οι μεταβλητές που έχουν σχέση με φόρους είναι μηδέν.

Για να μπορέσουμε να χρησιμοποιήσουμε τις τιμές που υπολογίζουμε, χρησιμοποιούμε – όπως και στο Component για τα προϊόντα – το Hook “useEffect()”, βάζοντας όλες τις μεταβλητές στο “dependencies array” του “useEffect()”, έτσι ώστε όταν υπάρχει μία αλλαγή, να ξαναστέλνονται τα δεδομένα στο Component Γονέα. Μέσα στο Hook “useEffect()” δημιουργούμε μία συνάρτηση τύπου “arrow”, η οποία δημιουργεί ένα object που περιέχει όλες τις τιμές που θέλουμε να επιστρέψουμε, όπως φαίνεται στο Σχήμα 3.44. Με τη χρήση του “props”, στέλνουμε αυτό το object πίσω στο Component Γονέα. Στο Component Γονέα “Apodeixh.jsx” δημιουργούμε μία συνάρτηση που

αποθηκεύει όλες τις τιμές σε μία μεταβλητή που δηλώσαμε και την οποία διαχειριζόμαστε με τη χρήση του Hook “useState()”, όπως βλέπουμε στο Σχήμα 3.45.

```
// Send the data back to parent component
useEffect(() => {
    const totals = {
        totalNetValue,
        totalVatValue,
        totalSunolo,
        totalWithheldAmount,
        totalFeesAmount,
        totalStampDutyamount,
        totalOtherTaxesAmount,
    }
    props.onChange(totals)
}, [
    totalNetValue,
    totalVatValue,
    totalSunolo,
    totalWithheldAmount,
    totalFeesAmount,
    totalStampDutyamount,
    totalOtherTaxesAmount,
])
```

Σχήμα 3.44: Σύνταξη Hook “useEffect()”.

```
const [totals, setTotals] = useState({})

// Get the total values from child component
const getTotalNetValue = (data) => {
    setTotals(data)
}
```

Σχήμα 3.45: Σύνταξη του Hook “useState()”.

Η επικοινωνία με την Α.Α.Δ.Ε. γίνεται μέσω XML. Οπότε δημιουργούμε μία σταθερά, η οποία θα περιέχει το XML σε μορφή String. Στο XML θα υπάρχουν τα δεδομένα που δημιουργήσαμε. Για τη δημιουργία του XML ορίζουμε στην αρχή μερικές μεταβλητές, όπως βλέπουμε στο Σχήμα 3.46, καθώς παίρνουμε και τη τιμή του Αύξοντα Αριθμού (ΑΑ) για τις αποδείξεις από την παγκόσμια μεταβλητή “numbers” τύπου “aa” με τη χρήση του Hook “useSelector()”.

```
const { numbers } = useSelector((state) => state.aa)
let { aaApodeixh, aaTimologio, aaProsfora } = numbers

let today = new Date()
let dd = String(today.getDate()).padStart(2, '0')
let mm = String(today.getMonth() + 1).padStart(2, '0') //January is 0!
let yyyy = today.getFullYear()
let timhFPA = 1.24
let invoiceDetails
let totalProductValueRow
let totalProductVatRow

const current_date = yyyy + '-' + mm + '-' + dd
```

Σχήμα 3.46: Σύνταξη Hook “useSelector()”.

Όταν ο Χρήστης κάνει κλικ στο κουμπί «Έκδοση», καλείται η συνάρτηση “onSubmit()”, του Σχήματος 3.47. Σε αυτή, χωρίζουμε τη δημιουργία του XML σε τρία διαφορετικά μέρη:

1. Αρχικά, δημιουργούμε το κομμάτι που περιέχει τις πληροφορίες σχετικά με τα προϊόντα.
2. Στη συνέχεια, δημιουργούμε το κομμάτι που περιέχει τις πληροφορίες σχετικά με τα σύνολα (στη συγκεκριμένη περίπτωση τη συνολική καθαρή αξία, το συνολικό Φ.Π.Α. και το σύνολο).
3. Τέλος, δημιουργούμε το κομμάτι σχετικά με τις πληροφορίες του παραστατικού και του εκδότη του παραστατικού (Α.Φ.Μ., χώρα, ΑΑ κ.λπ.).

```
// Create and submit the XML to AADE
const onSubmit = async (e) => {
  e.preventDefault()

  const { totalNetValue, totalVatValue, totalSunolo } = totals

  if (!totalSunolo) return toast.error('Παρακαλώ προσθέστε προϊόντα')

  invoiceDetails = ''
  let line = 1

  products?.map(
    (item, index) =>
      (timhFPA = fpaValue(
        item.product_tax_percentage,
        (totalProductValueRow = twoDecimals((item.product_quantity * item.product_price) / timhFPA)),
        (totalProductVatRow = twoDecimals(item.product_quantity * item.product_price - totalProductValueRow)),
        (invoiceDetails =
          invoiceDetails +
          `<invoiceDetails> <lineNumber>${line}</lineNumber> <netValue>${twoDecimals(
            totalProductValueRow
          )}</netValue> <vatCategory>${item.product_tax_percentage}</vatCategory> <vatAmount>${twoDecimals(
            totalProductVatRow
          )}</vatAmount> <incomeClassification> <N1:classificationType>E3_561_003</N1:classificationType> <N1:classificationCategory>category1_1</N1:classificationCategory> <N1:amount>${twoDecimals(
            totalProductValueRow
          )}</N1:amount> </incomeClassification> </invoiceDetails>`),
        line++
      ))
  )
```

Σχήμα 3.47: Σύνταξη συνάρτησης “onSubmit()”.

Η συνάρτηση “onSubmit()” παίρνει τις τιμές “totalNetValue”, “totalVatValue”, “totalSunolo” από το object “totals”. Έπειτα, γίνεται ένας έλεγχος για την ορθότητα των δεδομένων και αρχικοποιούμε δύο μεταβλητές. Ορίζουμε ως κενή τη μεταβλητή “invoiceDetails”, ώστε κάθε φορά που καλούμε συνάρτηση “onSubmit()” να είμαστε σίγουροι πως είναι κενή και δε περιέχει τιμές από προηγούμενες κλήσεις. Εν συνεχεία, με τη χρήση της μεθόδου map, κάνουμε loop σε όλα τα προϊόντα, δημιουργούμε για κάθε προϊόν το κατάλληλο XML κομμάτι και το αποθηκεύουμε με τη μορφή String στη μεταβλητή “invoiceDetails”.
Αφού τελειώσει η μέθοδος “map”, δημιουργούμε το κομμάτι του XML που περιέχει τις συνολικές τιμές, τις οποίες αποθηκεύουμε με τη μορφή String στη σταθερά “invoiceSummary” του Σχήματος 3.48.

```
// Create the part of xml with the Summary values
const invoiceSummary = `<invoiceSummary> <totalNetValue>${twoDecimals(
  totalNetValue
)}</totalNetValue> <totalVatAmount>${twoDecimals(
  totalVatValue
)}</totalVatAmount> <totalWithheldAmount>0</totalWithheldAmount> <totalFeesAmount>0</totalFeesAmount> <totalStampDutyAmount>0</totalStampDutyAmount> <totalOtherTaxesAmount>0</totalOtherTaxesAmount>
  <totalDeductionsAmount>0</totalDeductionsAmount> <totalGrossValue>${twoDecimals(
  totalSunolo
)}</totalGrossValue> <incomeClassification> <N1:classificationType>E3_561_003</N1:classificationType> <N1:classificationCategory>category1_1</N1:classificationCategory> <N1:amount>${twoDecimals(
  totalNetValue
)}</N1:amount> </incomeClassification> </invoiceSummary> </invoice> </InvoicesDoc>`
```

Σχήμα 3.48: Ορισμός σταθεράς “invoiceSummary”.

Τέλος, δημιουργούμε το κομμάτι που περιέχει τις πληροφορίες σχετικά με το παραστατικό και το αποθηκεύουμε σε μορφή String στη σταθερά “xmls” του Σχήματος 3.49. Σε αυτή τη σταθερά προσθέτουμε και τα υπόλοιπα κομμάτια του XML. Με τη χρήση του Hook “useDispatch()” καλούμε τη συνάρτηση “postAadeInvoice()”, έχοντας ως παράμετρο τη μεταβλητή “xmls”. Η εν λόγω συνάρτηση δημιουργεί ένα αίτημα “POST” στο Route “/api/aadeinvoice”, έχοντας ως Body το XML που δημιουργήσαμε. Εάν το αίτημα γίνει “Fulfilled”, τότε η global μεταβλητή “aadeInfo” τύπου aade θα πάρει τις τιμές “mark”, “uid”, “statusCode” που επιστρέφει η Α.Α.Δ.Ε.. Εάν το αίτημα γίνει “Rejected”, τότε αποθηκεύεται το μήνυμα λάθους στη παγκόσμια μεταβλητή “isErrorAade” τύπου “aade” και εμφανίζεται το κατάλληλο μήνυμα λάθους.

```
// Unite all XML parts to create the final XML
let xmls = `<?xml version="1.0" encoding="utf-8"?> <InvoicesDoc xmlns="http://www.aade.gr/myDATA/invoice/v1.0" xmlns:N1="https://www.aade.gr/myDATA/incomeClassificaton/v1.0"
xmlns:N2="https://www.aade.gr/myDATA/expensesClassificaton/v1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.aade.gr/myDATA/invoice/v1.0 schema.xsd">
<invoice> <issuer> <vatNumber>167816416</vatNumber> <country>GR</country> <branch>1</branch> </issuer> <invoiceHeader> <series>3</series> <aa>${aaTimologio}</aa>
<issueDate>${current_date}</issueDate> <invoiceType>11.1</invoiceType> <currency>EUR</currency> </invoiceHeader> <paymentMethods>
<paymentMethodDetails> <type>${troposPlhrwmhs}</type> <amount>${twoDecimals(
  totalSunolo
)}</amount> </paymentMethodDetails> </paymentMethods>`

xmls = xmls + invoiceDetails + invoiceSummary

// Send XML to AADE
dispatch(postAadeInvoice({ xmls }))
```

Σχήμα 3.49: Κλήση της συνάρτησης “postAadeInvoice”.

Η αποθήκευση στη βάση δεδομένων μας γίνεται με τη χρήση του Hook “useEffect()”. Αρχικά, παίρνουμε τις τιμές “mark”, “uid”, “statusCode” και “isErrorAade” από το object “aadeInfo”, το οποίο γεμίζει με τις τιμές που στέλνει η Α.Α.Δ.Ε.. Περνώντας τις τιμές αυτές σαν “array dependencies” στο Hook “useEffect()”, αυτή καλείται κάθε φορά που αλλάζει τουλάχιστον μία από τις εν λόγω τιμές. Στην αρχή του “useEffect()” ελέγχεται εάν η τιμή της σταθεράς “statusCode” είναι ίση με “Fail” ή εάν η μεταβλητή “isErrorAade” είναι γεμάτη. Εάν ισχύει έστω μία από τις αυτές συνθήκες, αυτό σημαίνει ότι δεν έχει γίνει έκδοση παραστατικού. Επομένως, τα στοιχεία δε πρέπει να αποθηκευτούν. Εάν η τιμή της σταθεράς “statusCode” είναι ίση με “Success”, τότε δημιουργούμε ένα νέο object και περνάμε σε αυτό τις πληροφορίες προς αποθήκευση, όπως φαίνεται και στο Σχήμα 3.50.

Στη συνέχεια, με τη χρήση του Hook “useDispatch()” καλούμε τη συνάρτηση “createInvoice()” έχοντας ως παράμετρο το object με τις πληροφορίες. Η συνάρτηση “createInvoice()” δημιουργεί ένα αίτημα “POST” στο Route “/api/invoice”, περνώντας ως Body το object. Έπειτα, αυξάνουμε κατά ένα τον Αύξοντα Αριθμό και τον περνάμε σε ένα object. Με τη χρήση του Hook “useDispatch()” καλούμε τη συνάρτηση “updateAa” και περνάμε ως παράμετρο το object με τις νέες τιμές των Αυξόντων Αριθμών. Εάν το αίτημα που δημιουργεί η συνάρτηση “createInvoice()” γίνει “Fulfilled”, τότε γίνεται “True” η global μεταβλητή “isSuccessInvoice”. Εάν, όμως, το αίτημα γίνει “Rejected”, τότε γίνεται “True” η global μεταβλητή “isErrorInvoice”.

Τέλος, δημιουργούμε ακόμη ένα hook “useEffect()” , ώστε να ελέγχουμε εάν το παραστατικό αποθηκεύτηκε στη βάση μας. Περνάμε ως array dependencies τις σταθερές “isErrorInvoice” και “isSuccessInvoice”. Εάν η τιμή της σταθεράς “isErrorInvoice” είναι ίση με “True”, εμφανίζεται σχετικό μήνυμα λάθους. Εάν, όμως, η τιμή της σταθεράς “isSuccessInvoice” είναι ίση με “True”, τότε εμφανίζεται μήνυμα πως το παραστατικό δημιουργήθηκε επιτυχώς και ο Χρήστης μεταφέρεται στη σελίδα του παραστατικού με τη χρήση του Hook “useNavigate()”.

```
// Controls if disptach in AADE was successful or not displaying the proper message to end user
useEffect(() => {
   if (isErrorAade || statusCode === 'Fail') {
      toast.error('Το παραστατικό δεν στάλθηκε στην ΑΑΔΕ')
      return
   }
   // if dispatch to AADE was successful then invoices is saved to db
   if (statusCode === 'Success') {
      const { totalNetValue, totalVatValue, totalSunolo } = totals
      const aa = aaApodeixh
      const series = '3'
      const date = current_date
      const paymentType = 'Μετρητά'
      const totalPrice = totalSunolo
      const totalPriceNet = totalNetValue
      const totalPriceVat = totalVatValue
      const userId = user._id
      const userData = {
         type,
         uid,
         series,
         aa,
         date,
         mark,
         paymentType,
         products,
         totalPrice,
         totalPriceNet,
         totalPriceVat,
         userId,
      }

      dispatch(createInvoice(userData))
      // eslint-disable-next-line

      aaApodeixh++

      const aaData = { aaApodeixh, aaTimologio, aaProsfora }

      dispatch(updateAa(aaData))

      toast.success('Επιτυχής έκδοση παραστατικού')
   }
}, [statusCode, isErrorAade])
```

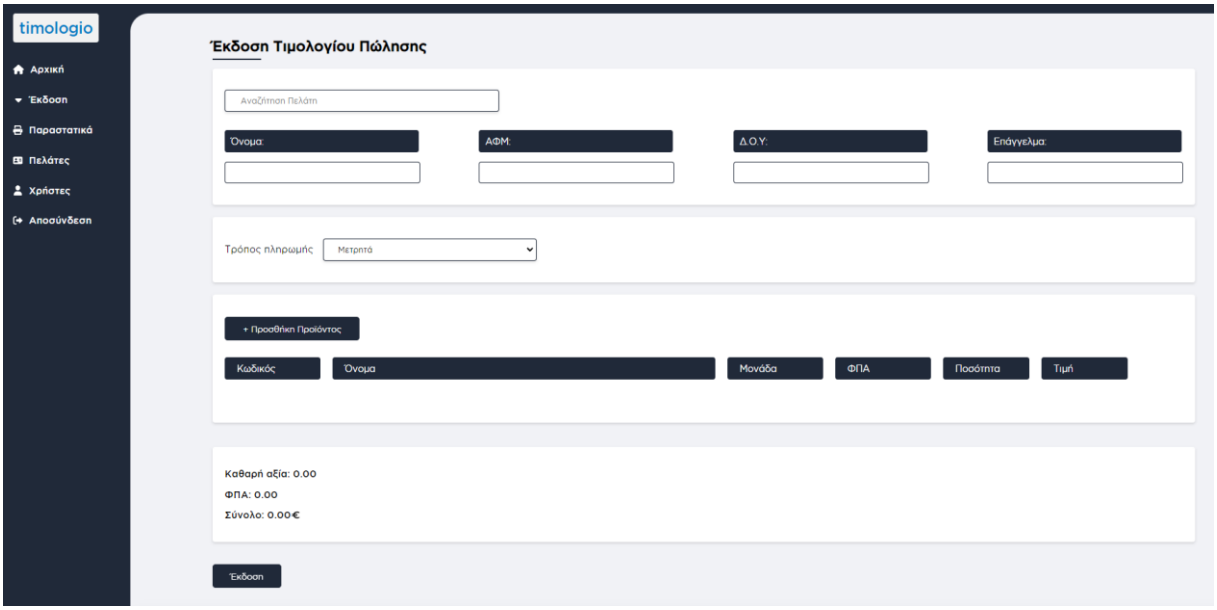
Σχήμα 3.50: Ορισμός του Hook “useEffect()”.

3.9 Δημιουργία Τιμολογίου Πώλησης

Στη Σελίδα που εικονίζεται στο Σχήμα 3.51, ο Χρήστης μπορεί να εκδώσει ένα νέο παραστατικό Τιμολογίου Πώλησης. Αρχικά, ο Χρήστης κάνει αναζήτηση ήδη καταχωρημένου πελάτη μέσω του Α.Φ.Μ. ή της Ονομασίας του πελάτη. Όταν πληκτρολογεί περισσότερους από δύο χαρακτήρες, όπως στο Σχήμα 3.52, τότε εμφανίζονται όλες οι δυνατές επιλογές. Κάνοντας κλικ πάνω σε κάποιον από τους πελάτες, προσθέτει τα στοιχεία του στο τιμολόγιο. Στη συνέχεια, με τον ίδιο τρόπο κάνει προσθήκη προϊόντων και όταν είναι έτοιμος, κάνει κλικ στο κουμπί «Έκδοση».

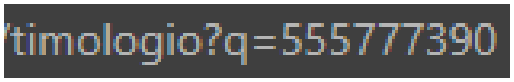
Στη συγκεκριμένη σελίδα μπορούμε να φτάσουμε και μέσω της σελίδας του εκάστοτε πελάτη, στην οποία και υπάρχει το κουμπί «Έκδοση Τιμολογίου». Σε αυτή τη περίπτωση, περνάμε στη διεύθυνση της σελίδας και ένα query με το Α.Φ.Μ. του εκάστοτε πελάτη, όπως αυτό του Σχήματος 4.53. Έτσι, η σελίδα μας γνωρίζει το Α.Φ.Μ. του πελάτη και μπορεί να γίνει η προσθήκη του στο τιμολόγιο.

Για την ανάπτυξη της Σελίδας χρησιμοποιήθηκαν τα ίδια components (“ProductInputs.jsx” και “TotalPreview.jsx”) με τη σελίδα της Απόδειξης, με τη προσθήκη του component για την αναζήτηση του Πελάτη (“CustomerSearch.jsx”).



Σχήμα 3.51: Σελίδα «Έκδοση Τιμολογίου Πώλησης».

Σχήμα 3.52: Αναζήτηση πελάτη βάσει Α.Φ.Μ..



Σχήμα 3.53: Query με το Α.Φ.Μ. πελάτη.

Στην αρχή της Σελίδας αυτής, με τη χρήση του Hook “useState()”, ορίζουμε μία μεταβλητή “query”, όπως φαίνεται στο Σχήμα 3.55. Για να μπορέσουμε να ελέγξουμε εάν υπάρχει κάποιο query στη διεύθυνση χρησιμοποιήσαμε για αρχή το Hook “useLocation()”. Με τη χρήση του συγκεκριμένου Hook καταφέραμε να πάρουμε μόνο το κομμάτι που μάς ενδιαφέρει από τη διεύθυνση. Στη συνέχεια, χρησιμοποιήσαμε το Hook “useEffect()” για να δούμε εάν το query μας είναι άδειο ή όχι. Εάν δεν είναι άδειο, παίρνουμε το Α.Φ.Μ. και το τοποθετούμε στην μεταβλητή “query” που δημιουργήσαμε πριν. Δημιουργούμε ακόμα μία συνάρτηση arrow με τη χρήση του Hook “useEffect()”, για να παρακολουθούμε τη κατάσταση της μεταβλητής “query”, η οποία γεμίζει εάν υπάρχει κάποιο Α.Φ.Μ. στη διεύθυνση της Σελίδας ή εάν ο Χρήστης πληκτρολογήσει πάνω από δύο χαρακτήρες. Κάθε φορά που αλλάζει η κατάστασή της, ενεργοποιείται το Hook “useEffect()”. Αρχικά, ελέγχει εάν το μέγεθος της μεταβλητής είναι μεγαλύτερο των δύο χαρακτήρων. Εάν η συνθήκη ισχύει, τότε με τη χρήση του Hook “useDispatch()” καλεί τη συνάρτηση “getCustomers()”, έχοντας ως παράμετρο το query. Η συνάρτηση “getCustomers()” δημιουργεί ένα αίτημα “GET” στο Route “/api/customer/?q=customerID”, έχοντας ως Body το “query”, όπως φαίνεται στο Σχήμα 3.56. Εάν το αίτημα γίνει “Fulfilled”, τότε η global μεταβλητή “customers” τύπου “customers” γεμίζει με τους

χρήστες που πληρούν τα κριτήρια. Στη συνέχεια, αλλάζει τη τιμή της σταθεράς “showSearch”, την οποία δηλώνουμε με τη βοήθεια του Hook “useState()”, όπως φαίνεται στο Σχήμα 3.57, ώστε να εμφανιστεί το πεδίο με τα αποτελέσματα της αναζήτησης. Στο πεδίο αυτό, με τη χρήση μίας συνάρτησης “map()” εμφανίζουμε όλα τα αποτελέσματα σε ξεχωριστή γραμμή.

```
return (
   <div className='page-content top-left-corner'>
      <div className='container'>
         <h2 className='page-title'>Έκδοση Τιμολογίου Πώλησης</h2>
         <CustomerSearch onChange={getCustomer} />
         <form onSubmit={onSubmit} className='apodeixhForm'>
            <div className='payment-type box'>
               <label>
                  Τρόπος πληρωμής
                  <select defaultValue='3' onChange={changePaymentType} className='mg-1 ' id='tropos_plhrwmhs'>
                     <option value='1'>Επαγ. Λογαριασμός Πληρωμών Ημεδαπής</option>
                     <option value='2'>Επαγ. Λογαριασμός Πληρωμών Αλλοδαπής</option>
                     <option value='3'>Μετρητά</option>
                     <option value='4'>Επιταγή</option>
                     <option value='5'>Επί Πιστώσει</option>
                     <option value='6'>WebBanking</option>
                     <option value='7'>POS / e-POS</option>
                  </select>
               </label>
            </div>
         </form>
         <ProductInput onChange={getProducts} />
         <TotalPreview onChange={getTotalNetValue} products={products} />
         <div className='btn-submit-container'>
            <button type='submit' className='mg-1-0 btn-submit' onClick={onSubmit}>
               Έκδοση
            </button>
         </div>
      </div>
      <Footer />
   </div>
)
```

Σχήμα 3.54: Κώδικας δημιουργίας της Σελίδας «Έκδοση Τιμολογίου Πώλησης».

```
const [query, setQuery] = useState('')

const location = useLocation()
const { search } = location


useEffect(() => {
   if (search) {
      const searchAfm = search.split('=')
      setQuery(searchAfm[1])
   }
}, [search])
```

Σχήμα 3.55: Ορισμός των Hooks “useEffect()” και “useLocation()”.

```
useEffect(() => {
    if (query.length > 2) {
        dispatch(getCustomers(query))
        setShowSearch(true)
        // eslint-disable-next-line
    } else {
        setShowSearch(false)
    }
}, [query])
```

Σχήμα 3.56: Ορισμός του Hook “useEffect()”.

```
{showSearch && (
    <div className='customer-search-results'>
        {customers?.map((customer) => (
            <p
                className='customer-search-select'
                key={customer._id}
                onClick={() =>
                    findSingleuser(
                        customer._id,
                        customer.name,
                        customer.afm,
                        customer.doy,
                        customer.profession,
                        customer.address,
                        customer.addressNumber,
                        customer.postalCode,
                        customer.city
                    )
                }
            >
                {customer.name} / {customer.afm}
            </p>
        ))}
    </div>
)}
```

Σχήμα 3.57: Κώδικας εμφάνισης των αποτελεσμάτων της αναζήτησης.

Όταν ο Χρήστης κάνει κλικ σε κάποιον από τους εμφανιζόμενους πελάτες – μετά την αναζήτηση – καλείται η συνάρτηση “findSingleuser()”, η οποία έχει ως παραμέτρους τα στοιχεία του συγκεκριμένου πελάτη. Η συνάρτηση αυτή, όπως βλέπουμε στο Σχήμα 3.58, παίρνει τα στοιχεία του πελάτη και τα αποθηκεύει σε μεταβλητές που δηλώνουμε με τη χρήση του Hook “useState()”.

Για να μπορέσουμε να χρησιμοποιήσουμε τις μεταβλητές κατά την δημιουργία του XML, θα πρέπει να τις επιστρέψουμε στον Γονέα (parent – Timologio.jsx). Για να το πετύχουμε αυτό, χρησιμοποιήσουμε το Hook “useEffect()”, το οποίο – με τη βοήθεια του “props” – επιστρέφει τις τιμές. Βάζουμε όλες τις τιμές στο “array dependencies” του “useEffect()”, όπως φαίνεται στο Σχήμα 3.59, ώστε κάθε φορά που γίνεται μία αλλαγή, να ενημερώνονται οι τιμές και στον Γονέα.

```
// Declare consts for Customers info
const [cnamePicked, setCnamePicked] = useState('')
const [cidPicked, setCidPicked] = useState('')
const [cafmPicked, setCafmPicked] = useState('')
const [cdoyPicked, setCdoyPicked] = useState('')
const [cprofessionPicked, setCprofessionPicked] = useState('')
const [caddressPicked, setCaddressPicked] = useState('')
const [caddressNumberPicked, setCaddressNumberPicked] = useState('')
const [cpostalPicked, setCpostalPicked] = useState('')
const [ccityPicked, setcityPicked] = useState('')

// Get the info into the consts
const findSingleuser = (id, name, afm, doy, profession, address, addressNumber, postalCode, city) => {
   setCnamePicked(name)
   setCafmPicked(afm)
   setCdoyPicked(doy)
   setCprofessionPicked(profession)
   setCidPicked(id)
   setCaddressPicked(address)
   setCaddressNumberPicked(addressNumber)
   setCpostalPicked(postalCode)
   setcityPicked(city)
   setQuery('')
}
```

Σχήμα 3.58: Ορισμός συνάρτησης “FindSingleuser()”.

```
useEffect(() => {
   props.onChange(customerInfo)
}, [
   cnamePicked,
   cidPicked,
   cafmPicked,
   cdoyPicked,
   cprofessionPicked,
   caddressPicked,
   caddressNumberPicked,
   cpostalPicked,
   ccityPicked,
])
```

Σχήμα 3.59: Εντοπισμός αλλαγής και ανανέωση στοιχείων πελάτη με χρήση του Hook “useEffect()”.

Για να μπορέσουμε να χρησιμοποιήσουμε τις τιμές στον Γονέα, δημιουργούμε με τη χρήση του Hook “useState()” μία μεταβλητή τύπου “object”. Στη συνέχεια, δημιουργούμε μία συνάρτηση τύπου “arrow”, η οποία τη γεμίζει με τα στοιχεία του πελάτη, όπως φαίνεται στο Σχήμα 3.60.

```
// Get the inserted customer from child component
const getCustomer = (data) => {
   setCustomer(data)
}
```

Σχήμα 3.60: Δημιουργία συνάρτησης τύπου “arrow”.

Οι υπόλοιπες διαδικασίες, παραμένουν ίδιες με αυτές για τη δημιουργία Απόδειξης Λιανικής Πώλησης, εκτός από δύο μέρη: Η Σελίδα μας παίρνει τα προϊόντα από το Component “ ProductInput.jsx ” και τις τελικές τιμές από το Component “TotalPreview.jsx”. Επίσης, η συγκεκριμένη Σελίδα δε περιλαμβάνει φόρους. Επομένως, η εν λόγω διαδικασία είναι ακριβώς η ίδια

με τη Σελίδα της δημιουργίας Απόδειξης Λιανικής Πώλησης. Στο Σχήμα 3.61, εικονίζονται ορισμένες μεταβλητές που προσαρμόστηκαν για τη δημιουργία Τιμολογίου Πώλησης.

```
let today = new Date()
let dd = String(today.getDate()).padStart(2, '0')
let mm = String(today.getMonth() + 1).padStart(2, '0') //January is 0!
let yyyy = today.getFullYear()
let timhFPA = 1.24
const type = 'Τιμολόγιο Πώλησης'
let invoiceDetails
let totalProductValueRow
let totalProductVatRow
```

Σχήμα 3.61: Μεταβλητές για τη δημιουργία Τιμολογίου Πώλησης.

Τέλος, υπάρχει ένα επιπλέον πεδίο στη δημιουργία του XML, το οποίο περιέχει τις πληροφορίες του πελάτη και εμφανίζεται κατά την δημιουργία της μεταβλητής “xmls”, όπως βλέπουμε στο Σχήμα 3.62.

```
let xmls = `<?xml version="1.0" encoding="utf-8"?> <InvoicesDoc xmlns="http://www.aade.gr/myDATA/invoice/v1.0"
xmlns:N1="https://www.aade.gr/myDATA/incomeClassificaton/v1.0"
xmlns:N2="https://www.aade.gr/myDATA/expensesClassificaton/v1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.aade.gr/myDATA/invoice/v1.0 schema.xsd">
<invoice> <issuer> <vatNumber>167816416</vatNumber> <country>GR</country> <branch>1</branch> </issuer>

<counterpart>
   <vatNumber>${Number(
   cafmPicked)}
   </vatNumber>
   <country>
      GR
   </country>
   <branch>1</branch>
   <address>
      <street>${caddressPicked}</street>
      <number>${Number(caddressNumberPicked)}</number>
      <postalCode>${Number(cpostalPicked)}</postalCode>
      <city>${ccityPicked}</city>
   </address>
</counterpart>
<invoiceHeader><series>4</series><aa>${Number(
   aaTimologio
)}</aa><issueDate>${current_date}</issueDate><invoiceType>1.1</invoiceType><currency>EUR</currency></invoiceHeader>
<paymentMethods> <paymentMethodDetails> <type>${troposPlhrwmhs}</type> <amount>${twoDecimals(
   totalSunolo
)}</amount> </paymentMethodDetails> </paymentMethods>`
```

Σχήμα 3.62: Εμφάνιση πεδίου στοιχείων πελάτη.

3.10 Δημιουργία Τιμολογίου Παροχής Υπηρεσιών

Και σε αυτή τη Σελίδα, η εμπειρία του Χρήστη παραμένει η ίδια, όπως και στις Σελίδες δημιουργίας Απόδειξης Λιανικής Πώλησης και Πώλησης. Συμπληρώνοντας τουλάχιστον δύο χαρακτήρες στην αναζήτηση πελάτη, εμφανίζονται όλοι οι πελάτες που πληρούν τα κριτήρια αναζήτησης. Στη συνέχεια, αφού επιλεχθεί ο επιθυμητός πελάτης, ο Χρήστης εισάγει τα προϊόντα που θέλει.

Στη συγκεκριμένη Σελίδα ο Χρήστης μπορεί να προσθέσει και φόρους, κάνοντας κλικ στο κουμπί «Προσθήκη Φόρου». Για να μπορέσει να προσθέσει κάποιο είδος φόρου, θα πρέπει προηγουμένως να έχει προσθέσει τουλάχιστον ένα προϊόν. Σε περίπτωση που ο Χρήστης επιλέξει το είδος του φόρου, το ποσό του οποίου μπορεί να προσθέσει ο ίδιος, π.χ. Παρακρατούμενος Φόρος – Λοιπές Παρακρατήσεις Φόρου, μπορεί να προκύψει αρνητικό υπόλοιπο και επομένως να εμφανίζεται σχετικό μήνυμα λάθους.

```
return (
  <div className='page-content top-left-corner'>
    <div className='container'>
      <h2 className='page-title'>Έκδοση Τιμολογίου Παροχής Υπηρεσιών</h2>
      <CustomerSearch onChange={getCustomer} />
      <form onSubmit={onSubmit} className='apodeixhForm'>
        <div className='payment-type box'>
          <label>
            Τρόπος πληρωμής
            <select defaultValue='3' onChange={changePaymentType} className='mg-1 ' id='tropos_plhrwmhs'>
              <option value='1'>Επαγ. Λογαριασμός Πληρωμών Ημεδαπής</option>
              <option value='2'>Επαγ. Λογαριασμός Πληρωμών Αλλοδαπής</option>
              <option value='3'>Μετρητά</option>
              <option value='4'>Επιταγή</option>
              <option value='5'>Επί Πιστώσει</option>
              <option value='6'>WebBanking</option>
              <option value='7'>POS / e-POS</option>
            </select>
          </label>
        </div>
      </form>
      <ProductInput onChange={getProducts} />
      <TaxIputs onChange={getTaxes} totalNetValueC={totals.totalNetValue} />
      <TotalPreview onChange={getTotalNetValue} products={products} taxes={taxes} />
      <div className='btn-submit-container'>
        <button type='submit' className='mg-1-0 btn-submit' onClick={onSubmit}>
          Έκδοση
        </button>
      </div>
    </div>
    <Footer />
  </div>
)
```

Σχήμα 3.63: Κώδικας δημιουργίας Σελίδας «Δημιουργία Τιμολογίου Παροχής Υπηρεσιών».

Από τη σελίδα “TimologioYphresiwn.jsx” περνάμε – μέσω του “props” – τη συνολική καθαρή αξία των προϊόντων. Στο κουμπί «Προσθήκη Φόρου» υπάρχει ένας έλεγχος για το εάν γίνει “Disable” ή όχι, όπως φαίνεται στο Σχήμα 3.64. Πιο συγκεκριμένα, ελέγχουμε εάν η συνολική καθαρή αξία είναι «άδεια». Εάν η συνθήκη ισχύει, αυτό σημαίνει ότι δεν έχουμε προσθέσει κάποιο προϊόν ακόμη και έτσι δε μπορούμε να προσθέσουμε και φόρους.

```
<button className='mg-tb-1 btn-submit' onClick={addTaxes} disabled={totalNetValueC ? false : true}>
  + Προσθήκη Φόρων
</button>
```

Σχήμα 3.64: Έλεγχος συνολικής καθαρής αξίας.

Με τη χρήση του Hook “useState()”, όπως φαίνεται στο Σχήμα 3.65, ορίζουμε μία σταθερά “taxes” ως ένα άδειο “array”, η οποία και θα αντιπροσωπεύει τους φόρους μας. Όταν ο Χρήστης κάνει κλικ στο κουμπί «Προσθήκη Φόρων», τότε καλείται η συνάρτηση “addTaxes()”. Αυτή ορίζει μία σταθερά τύπου “object”, η οποία περιέχει τις αρχικές τιμές για τα πεδία κάθε φόρου και ορίζει μεταβλητές λαθών για κάθε τιμή. Αρχικά, όλες οι μεταβλητές λαθών είναι “null”. Εν συνεχεία, υπάρχει ένας

έλεγχος για το εάν όλα τα πεδία του φόρου – που προστέθηκε προηγουμένως – έχουν συμπληρωθεί. Αυτός ο έλεγχος επιτυγχάνεται, καλώντας τη συνάρτηση “prevIsValid()”. Εάν δεν υπάρχουν λάθη, η συνάρτηση “prevIsValid()” επιστρέφει “True” και προστίθεται νέα σειρά φόρου, όπως φαίνεται στο Σχήμα 3.65.

```
const [taxes, setTaxes] = useState([])

// Add new tax line
const addTaxes = (e) => {
    e.preventDefault()

    const inputState = {
        tax_type: '1',
        tax_category: '1',
        tax_underlying_value: twoDecimals(totalNetValueC) || '',
        tax_amount: '',

        errors: {
            tax_type: null,
            tax_category: null,
            tax_underlying_value: null,
            tax_amount: null,
        },
    }

    if (prevIsValid()) {
        setTaxes((prev) => [...prev, inputState])
    }
}
```

Σχήμα 3.65: Σύνταξη συνάρτησης “addTaxes()”.

Η συνάρτηση “prevIsValid()” αρχικά ελέγχει εάν υπάρχουν φόροι. Εάν δεν έχουν προστεθεί φόροι στη σταθερά “products”, εάν δηλαδή το μέγεθός της (length) είναι μηδέν, τότε δεν υπάρχουν άλλοι φόροι. Επομένως, δεν υπάρχουν λάθη και η συνάρτηση επιστρέφει “True”. Εάν, όμως, υπάρχουν φόροι στη σταθερά “products”, τότε γίνεται έλεγχος για το εάν υπάρχει κάποιο πεδίο που δεν έχει συμπληρωθεί στον τελευταίο φόρο. Εάν υπάρξει κάποιο κενό πεδίο, εμφανίζεται κάτω από αυτό ένα σχετικό μήνυμα, το οποίο ενημερώνει τον Χρήστη πως δεν έχει συμπληρωθεί, και έτσι, δε γίνεται προσθήκη νέου φόρου. Εάν όλα τα πεδία είναι συμπληρωμένα, τότε η συνάρτηση επιστρέφει “True” και γίνεται προσθήκη νέου προϊόντος, όπως φαίνεται στο Σχήμα 3.66.

```
// Error hadling for taxes
const prevIsValid = () => {
    if (taxes.length === 0) {
        return true
    }

    const emptyField = taxes.some(
        (item) => item.tax_type === '' || item.tax_category === '' || item.tax_amount === ''
        )

        if (emptyField) {
            taxes.map((item, index) => {
                const allPrev = [...taxes]
            if (taxes[index].tax_type === '') {
                allPrev[index].errors.tax_type = 'Το είδος φόρου είναι υποχρεωτικό'
            }
            if (taxes[index].tax_category === '') {
                allPrev[index].errors.tax_category = 'Η κατηγορία φόρου είναι υποχρεωτική υποχρεωτικό'
            }
            if (taxes[index].tax_amount === '') {
                allPrev[index].errors.tax_amount = 'Το Ποσό Φόρου είναι υποχρεωτικό'
            }

            setTaxes(allPrev)

            return
        })
    }

    return !emptyField
}
```

Σχήμα 3.66: Σύνταξη συνάρτησης “prevIsValid()”.

Εάν ο χρήστης θέλει να αφαιρέσει μία γραμμή φόρου που έχει προσθέσει, μπορεί να κάνει κλικ στο κόκκινο “X”. Καλείται η συνάρτηση “handleRemoveProduct()”, η οποία έχει ως παράμετρο το “index” του φόρου στη σταθερά “taxes” τύπου “array”. Η συνάρτηση αυτή αφαιρεί το object με το συγκεκριμένο “index”, όπως φαίνεται στο Σχήμα 3.67.

```
// Remove items from taxes
const handleRemoveProduct = (e, index) => {
   e.preventDefault()

   setTaxes((prev) => prev.filter((item) => item !== prev[index]))
}
```

Σχήμα 3.67: Σύνταξη συνάρτησης “handleRemoveProduct()”.

Η αλλαγή που κάνει ο χρήστης στις τιμές των φόρων διαχειρίζονται από τη συνάρτηση “onChangeTax()”, η οποία έχει ως παράμετρο το “index” του φόρου στη σταθερά “taxes” τύπου “array”, όπως βλέπουμε στο Σχήμα 3.68. Στη συνέχεια, κάνει “map” της τιμές που έχει η σταθερά “taxes”. Εάν το “index” ταιριάζει με το “index” του object που ελέγχεται, τότε προσθέτει τη τιμή που πληκτρολογήσαμε στο σωστό πεδίο. Για να βρει το σωστό πεδίο, χρησιμοποιεί το “name” που έχουμε ορίσει σε κάθε “input”.

```
const onChangeTax = (index, event) => {
   event.preventDefault()
   event.persist()

   setTaxes((prev) => {
      return prev.map((item, i) => {
         if (i !== index) {
            return item
         }

         return {
            ...item,
            [event.target.name]: event.target.value,
            errors: {
               ...item.errors,
               [event.target.name]:
                  event.target.value.length > 0 ? null : [event.target.value.name] + 'είναι υποχρεωτικό',
            },
         }
      })
   })
}
```

Σχήμα 3.68: Σύνταξη συνάρτησης “onChangeTax()”.

Καθώς μερικοί φόροι έχουν συγκεκριμένο ποσοστό φόρου, με τη χρήση του Hook “useEffect()” δημιουργούμε μία συνάρτηση τύπου “arrow”, η οποία ελέγχει – με τη χρήση της συνάρτησης “map” – τις τιμές είδους φόρου και κατηγορίας φόρου και αναθέτει τη κατάλληλη τιμή στο ποσό φόρου, όπως βλέπουμε στα Σχήματα 3.69 και 3.70.

Στη συγκεκριμένη Σελίδα ορίζουμε και φόρους. Επομένως, στο Component “TotalPreview.jsx” υπολογίζουμε το συνολικό ποσό βάσει και αυτών. Με τη βοήθεια του Hook “useEffect()”, γίνονται οι υπολογισμοί. Συγκεκριμένα, στο statement if ελέγχεται εάν υπάρχουν φόροι. Εφόσον ο χρήστης έχει εισάγει κάποιο φόρο, με τη βοήθεια της συνάρτησης “map()”, κάνουμε “loop” στο object με τους

φόρους. Εκεί υπολογίζουμε το συνολικό ποσό για κάθε κατηγορία φόρου ξεχωριστά. Βάσει των τιμών των φόρων υπολογίζεται τελικά και το συνολικό ποσό, όπως φαίνεται στο Σχήμα 3.71.

```
useEffect(
    (e) => {
        taxes.map((item, index) =>
            item.tax_type === '1'
                ? item.tax_category === '1' || item.tax_category === '10' || item.tax_category === '12'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.15))
                    : item.tax_category === '2' || item.tax_category === '3' || item.tax_category === '12'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.2))
                    : item.tax_category === '9' || item.tax_category === '13'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.1))
                    : item.tax_category === '7'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.08))
                    : item.tax_category === '18'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.05))
                    : item.tax_category === '6'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.04))
                    : item.tax_category === '4'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.03))
                    : ''
                : item.tax_type === '2'
                ? item.tax_category === '6' || item.tax_category === '13' || item.tax_category === '14'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.12))
                    : item.tax_category === '1' || item.tax_category === '5'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.12))
                    : item.tax_category === '2'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.15))
                    : item.tax_category === '3'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.18))
                    : item.tax_category === '7'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.05))
                    : item.tax_category === '9'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.04))
                    : item.tax_category === '15'
                    ? (item.tax_amount = twoDecimals(totalNetValueC))
```

Σχήμα 3.69: Σύνταξη Hook “useEffect()” με τη βοήθεια της συνάρτησης “map” (1).

```
                : item.tax_type === '3'
                ? item.tax_category === '1' || item.tax_category === '4'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.15))
                    : item.tax_category === '12' || item.tax_category === '13'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.1))
                    : item.tax_category === '2' || item.tax_category === '11'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.05))
                    : item.tax_category === '14'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.8))
                    : item.tax_category === '15'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.2))
                    : item.tax_category === '3'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.04))
                    : item.tax_category === '5'
                    ? (item.tax_amount = twoDecimals(totalNetValueC))
                    : ''
                : item.tax_type === '4'
                ? item.tax_category === '1'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.012))
                    : item.tax_category === '2'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.024))
                    : item.tax_category === '3'
                    ? (item.tax_amount = twoDecimals(totalNetValueC * 0.036))
                    : ''
                : ''
        )
    },
    [taxes]
)
```

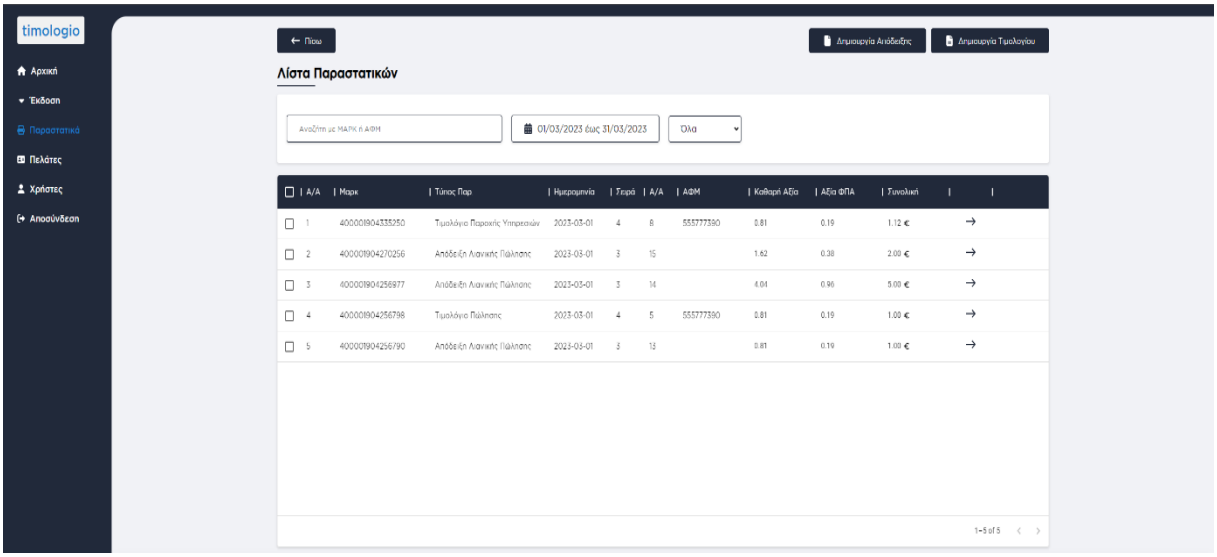
Σχήμα 3.70: Σύνταξη Hook “useEffect()” με τη βοήθεια της συνάρτησης “map” (2).

```
useEffect(() => {
    for (let i = 0; i < products.length; i++) {
        if (products[i].product_quantity === '' || products[i].product_price === '') return

        timhFPA = fpaValue(products[i].product_tax_percentage)

        product_total = twoDecimals(products[i].product_quantity * products[i].product_price)
        product_total_all = Number(product_total_all) + Number(product_total)

        product_net = twoDecimals(product_total / timhFPA)
        product_net_all = Number(product_net_all) + Number(product_net)

        product_vat = twoDecimals(product_total - product_net)
        product_vat_all = Number(product_vat_all) + Number(product_vat)
    }

    if (taxes) {
        taxes?.map(({ tax_type, tax_category, tax_underlying_value, tax_amount }) =>
            tax_type === '1'
                ? (totalParakratoumenoi += Number(tax_amount))
                : tax_type === '2'
                ? (totalTelwn += Number(tax_amount))
                : tax_type === '3'
                ? (totalLoipwn += Number(tax_amount))
                : (totalXartoshmo += Number(tax_amount))
        )
    }
    setTotalWithheldAmount(Number(totalParakratoumenoi))
    setTotalFeesAmount(Number(totalTelwn))
    setTotalStampDutyamount(Number(totalXartoshmo))
    setTotalOtherTaxesAmount(Number(totalLoipwn))
    setTotalSunolo(
        Number(product_total_all) -
            Number(totalParakratoumenoi) +
            Number(totalTelwn) +
            Number(totalLoipwn) +
            Number(totalXartoshmo)
    )
    setTotalNetValue(product_net_all)
    setTotalVatValue(product_vat_all)
}, [products, taxes])
```

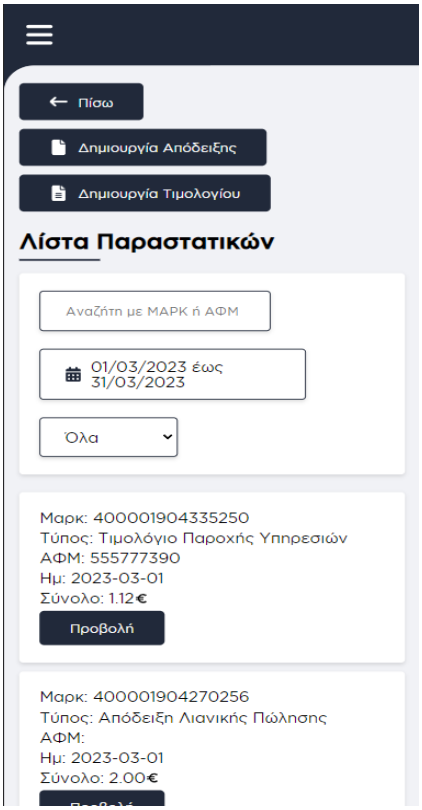
Σχήμα 3.71: Έλεγχος ύπαρξης φόρων και υπολογισμός των τιμών τους.

3.11 Σελίδα Παραστατικών

Στη συγκεκριμένη Σελίδα, ο Χρήστης βλέπει μία λίστα με τα παραστατικά που πληρούν τα κριτήρια αναζήτησης. Όταν γίνεται Render για πρώτη φορά, εμφανίζονται τα τιμολόγια του τελευταίου μήνα. Ο χρήστης μπορεί να αναζητήσει παραστατικά βάσει του Μοναδικού Αριθμού Καταχώρησης (Μ.ΑΡ.Κ.) ή του Α.Φ.Μ. του πελάτη εάν πρόκειται για τα τιμολόγια πώλησης και τιμολόγια παροχής υπηρεσιών, της αρχικής και τελικής ημερομηνίας και του τύπου παραστατικού. Η αναζήτηση μπορεί να γίνει μεμονωμένα για κάθε κριτήριο αναζήτησης ή με τα δύο ή και τα τρία κριτήρια μαζί. Παραδείγματος χάρη, ένας Χρήστης μπορεί να αναζητήσει παραστατικά τύπου «Τιμολόγιο Πώλησης» για τον πελάτη με Α.Φ.Μ. 555777390, τα οποία εκδόθηκαν εντός του χρονικού διαστήματος 1 Μαρτίου 2023 – 31 Μαρτίου 2023. Τα αναζητηθέντα παραστατικά εμφανίζονται σε έναν πίνακα και – κάνοντας κλικ επάνω στο πεδίο Μ.ΑΡ.Κ. ή στο βελάκι – ο Χρήστης μεταφέρεται στη σελίδα του εκάστοτε παραστατικού. Η εμφάνιση των παραστατικών γίνεται υπό τη μορφή πίνακα σε οθόνες ηλεκτρονικών υπολογιστών και υπό τη μορφή κάρτας για κάθε παραστατικό ξεχωριστά σε οθόνες κινητών τηλεφώνων, όπως φαίνεται στα Σήματα 3.72 και 3.73 αντίστοιχα.



Σχήμα 3.72: Εμφάνιση της λίστας παραστατικών υπό μορφή πίνακα σε οθόνες ηλεκτρονικών υπολογιστών.



Σχήμα 3.73: Εμφάνιση της λίστας παραστατικών υπό μορφή πίνακα σε οθόνες κινητών τηλεφώνων.

Η διαχείριση των μεταβλητών για την αναζήτηση παραστατικών γίνεται με την χρήση του Hook “useState()”. Όταν γίνεται Render η σελίδα, με τη χρήση του Hook “useEffect()”, δημιουργούμε μία συνάρτηση τύπου “arrow”, η οποία δημιουργεί μία μεταβλητή που περιέχει τα στοιχεία αναζήτησης. Στη συνέχεια, με τη χρήση του Hook “useDispatch()”, καλούμε τη συνάρτηση “getInvoices()”, η οποία έχει ως παράμετρο τη μεταβλητή που δημιουργήσαμε, όπως φαίνεται στο Σχήμα 3.75. Η

συνάρτηση “getInvoices()” δημιουργεί ένα νέο αίτημα “GET” στο Route “/api/invoices/ ?q=query/startData/endData/invoiceType”. Στο Hook “useEffect()” έχουμε ως “array dependencies” τα στοιχεία αναζήτησης. Επομένως, κάθε φορά που αλλάζει κάποιο στοιχείο, γίνεται εκ νέου αναζήτηση.

```
<div className='page-content top-left-corner'>
   <div className='container'>
      <div className='customers-top-btns'>
         <BackButton url='/' />
         <div className='create-invoices'>
            <Link to='/apodeixh' className='btn-back'>
               <Apodeixh fill='#fff' width='18px' height='18px' /> Δημιουργία Απόδειξης
            </Link>
            <Link to='/timologio' className='btn-back'>
               <Timologio fill='#fff' width='18px' height='18px' /> Δημιουργία Τιμολογίου
            </Link>
         </div>
      </div>
      <h2 className='page-title'>Λίστα Παραστατικών</h2>
      <div className='box mg-top-15 flex-column-w100'>
         <div className='flex-center'>
            <div>
               <input className='customer-search min-height-24' placeholder='Αναζήτη με ΜΑΡΚ ή ΑΦΜ' onChange={(e) => setQuery(e.target.value)}
               />
            </div>
            <div className='pos-relative calendar-input min-height-24' ref={detailedInvoicesRef}>
               <Calendar fill='#21293a' width='18px' height='18px' />
               <span
                  onClick={() => {
                     setOpenDate(!openDate)
                  }}
                  className='display-date-range'
               >{`${format(date[0].startDate, 'dd/MM/yyyy')} έως ${format(date[0].endDate, 'dd/MM/yyyy')}`}</span>
               {openDate && (
                  <DateRange
                     editableDateInputs={true} onChange={(item) => setDate([item.selection])} moveRangeOnFirstSelection={false} ranges={date} className='date' locale={rdrLocales.el}
                  />
               )}
            </div>
            <div>
               <select defaultValue='all' onChange={invoiceType} className='min-height-24 height-45' id='tropos_plhrwmhs'
               >
                  <option value=''>Όλα</option>
                  <option value='Απόδειξη Λιανικής Πώλησης'>Αποδείξεις</option>
                  <option value='Τιμολόγιο Πώλησης'>Τιμολόγια</option>
                  <option value='Τιμολόγιο Παροχής Υπηρεσιών'>Τιμολόγια</option>
               </select>
            </div>
         </div>
      </div>
   </div>
```

Σχήμα 3.74: Κώδικας δημιουργίας της Σελίδας «Λίστα Παραστατικών».

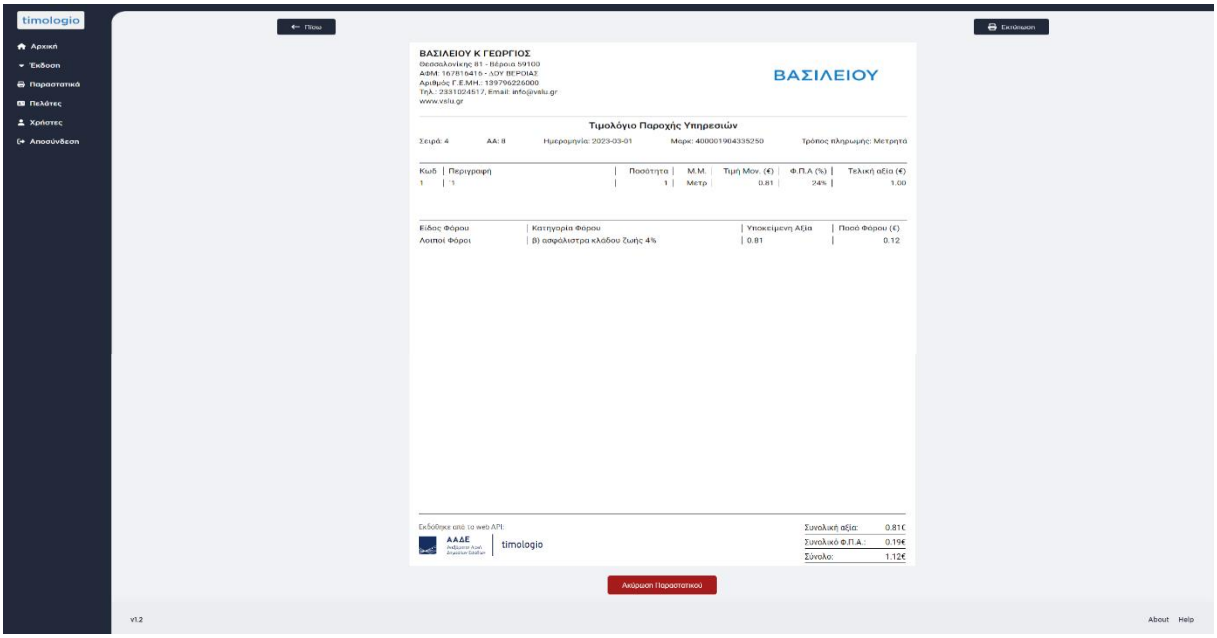
```
useEffect(() => {
   const send = `${query}/${format(date[0].startDate, 'yyyy-MM-dd')}/${format(
      date[0].endDate,
      'yyyy-MM-dd'
   )}/${displayInvoiceType}`

   dispatch(getInvoices(send))
   // eslint-disable-next-line
}, [query, date[0].startDate, date[0].endDate, displayInvoiceType])
```

Σχήμα 3.75:Σύνταξη Hook “useEffect()”.

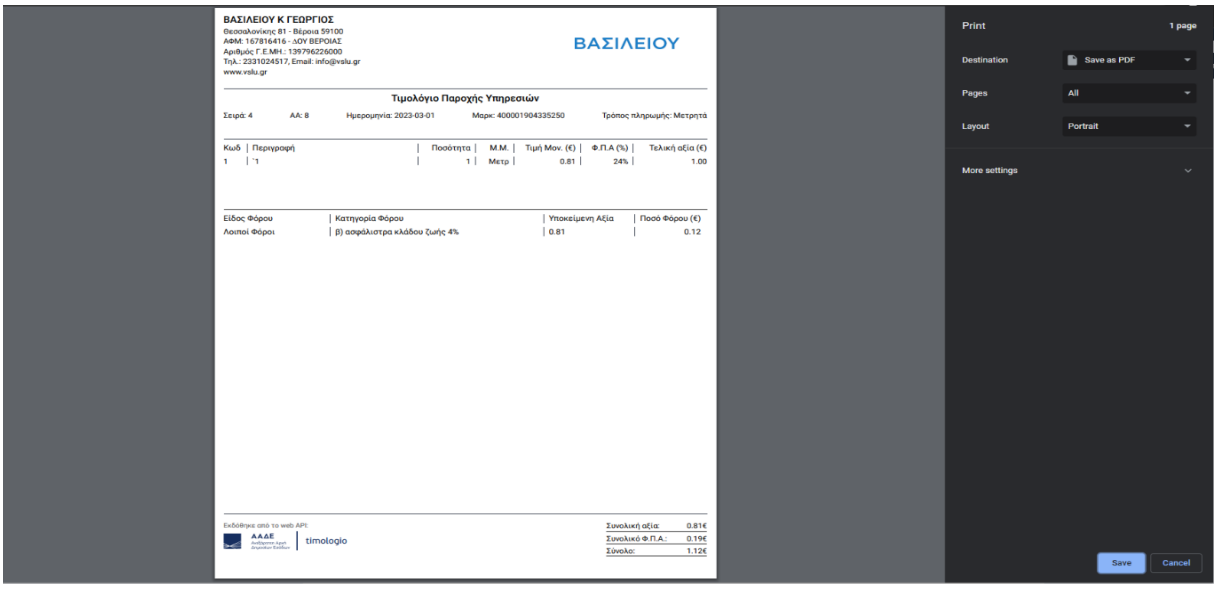
3.12 Σελίδα Παραστατικού

Σε αυτή τη Σελίδα, ο Χρήστης μπορεί να δει ένα παραστατικό με όλα τα στοιχεία του, όπως φαίνεται στο Σχήμα 3.76. Επιπροσθέτως, κάνοντας κλικ στο κουμπί «Εκτύπωση», μπορεί να εκτυπώσει ή να αποθηκεύσει ως αρχείο PDF το εκάστοτε παραστατικό, όπως βλέπουμε στο Σχήμα 3.77. Επίσης, κάνοντας κλικ στο κουμπί «Ακύρωση Παραστατικού», εμφανίζεται ένα μήνυμα επιβεβαίωσης διαγραφής, η οποία εάν γίνει αποδεκτή, ακυρώνεται το παραστατικό.

Όταν ο χρήστης κάνει κλικ στο κουμπί «Διαγραφή» και επιβεβαιώνει πως θέλει να διαγράψει το παραστατικό, καλείται η συνάρτηση “handleDelete()”, η οποία με τη χρήση του hook “useDispatch()” καλεί την συνάρτηση “postAadeDeletion(), έχοντας σαν παράμετρο τον Μοναδικό Αριθμό Καταχώρησης (M.AP.K.) του παραστατικού. Η συνάρτηση postAadeDeletion δημιουργεί ένα αίτημα POST στο route /api/invoice/cancelInvoice έχοντας σαν body το M.AP.K.. Εάν το αίτημα γίνει “fulfilled”, τότε εμφανίζεται σχετικό μήνυμα επιτυχημένης διαγραφής και – με τη βοήθεια του hook “useDispatch()” – καλείται η συνάρτηση “updateInvoice()”, έχοντας ως παράμετρο το “id” του παραστατικού. Η συνάρτηση “updateInvoice()” δημιουργεί ένα νέο αίτημα “UPDATE” στο route /api/invoice, έχοντας ως body το “id” του τιμολογίου. Αυτή η συνάρτηση αλλάζει τη κατάσταση του τιμολογίου στη βάση μας και πλέον το τιμολόγιο φαίνεται ως ακυρωμένο.



Σχήμα 3.76: Σελίδα προβολής παραστατικού.



Σχήμα 3.77: Σελίδα αποθήκευσης παραστατικού υπό τη μορφή αρχείου PDF.

```
<div className='invoice-box invoice-template' ref={componentRef}>
   <div className='invoice-max-width'>
      <div className='logo-and-vslu-info'>
         <div className='vslu-info'>
            <p style={{ fontWeight: 500 }}>ΒΑΣΙΛΕΙΟΥ Κ ΓΕΩΡΓΙΟΣ</p>
            <p>Θεσσαλονίκης 81 - Βέροια 59100</p>
            <p>ΑΦΜ: 167816416 - ΔΟΥ ΒΕΡΟΙΑΣ</p>
            <p>Αριθμός Γ.Ε.ΜΗ.: 139796226000</p>
            <p>Τηλ.: 2331024517, Email: info@vslu.gr</p>
            <p>www.vslu.gr</p>
         </div>
         <div className='logo-container'>
            <Logo width='85%' />
         </div>
      </div>
      <div className='invoice-info-container'>
         <div className='invoice-type' style={{ fontWeight: 500 }}>
            {type}
         </div>
         <div className='invoice-general-info'>
            <p>Σειρά: {series}</p>
            <p>ΑΑ: {aa}</p>
            <p>Ημερομηνία: {date}</p>
            <p>Μαρκ: {mark}</p>
            <p>Τρόπος πληρωμής: {paymentType}</p>
         </div>
      </div>
      {invoice.type === 'Τιμολόγιο Πώλησης' && (
         <div className='invoice-info-container'>
            <div className='invoice-type' style={{ fontWeight: 500 }}>
               Στοιχεία Πελάτη
            </div>
            <div className='invoice-general-info'>
               <p>Α.Φ.Μ: {invoice.customer?.afm}</p>
               <p>Επωνυμία: {invoice.customer?.name}</p>
               <p>Επάγγελμα: {invoice.customer?.profession}</p>
            </div>
         </div>
      )}
```

Σχήμα 3.78: Κώδικας για τη προβολή παραστατικού (1).

```
   <div className='products-container'>
      <div className='invoice-product-titles'>
         <div className='invoice-product-code'>Κωδ</div>
         <div className='invoice-product-name'>Περιγραφή</div>
         <div className='invoice-product-qty'>Ποσότητα</div>
         <div className='invoice-product-mm'>Μ.Μ.</div>
         <div className='invoice-product-net-price'>Τιμή Μον. (€)</div>
         <div className='invoice-product-fpa-perc'>Φ.Π.Α (%)</div>
         <div className='invoice-product-final-price'>Τελική αξία (€)</div>
      </div>
      {invoice.products?.map((product) => (
         <SingleInvoiceItem key={product.product_code} product={product} isLoading={isLoadingInvoice} />
      ))}
   </div>
   {taxes?.length > 0 && (
      <>
         <div className='taxes-invoice taxes-outline-top'>
            <div className='tax-type-dis'>Είδος Φόρου</div>
            <div className='tax-category-dis'>Κατηγορία Φόρου</div>
            <div className='tax-underline-dis'>Υποκείμενη Αξία</div>
            <div className='tax-amount-dis'>Ποσό Φόρου (€)</div>
         </div>
         {taxes?.map((tax, index) => (
            <div className='taxes-invoice taxes-m-top-10' key={index}>
               <div className='tax-type-dis'>
                  {tax.tax_type === 1
                     ? 'Παρακρατούμενος Φόρος'
                     : tax.tax_type === 2
                     ? 'Τέλη'
                     : tax.tax_type === 3
                     ? 'Λοιποί Φόροι'
                     : 'Χαρτόσημο'}
               </div>
               <div className='tax-category-dis'>
                  {taxCategoryString(tax.tax_type, tax.tax_category)}
               </div>
               <div className='tax-underline-dis'>{twoDecimals(tax.tax_underlying_value)}</div>
               <div className='tax-amount-dis'>{twoDecimals(tax.tax_amount)}</div>
            </div>
         ))}
      </>
   )}
</div>
```

Σχήμα 3.79: Κώδικας για τη προβολή παραστατικού (2).

```
      <div className='invoice-totals-container'>
        {/* <TaxAnalysis products={invoice.products} /> css 1303 && 1318 space-between */}
        <div className='aade-logo'>
          <div className='top-aade'>Εκδόθηκε από το web API:</div>
          <div className='bottom-aade'>
            <div className='logo-text'>
              {/* <AadeLogo width={50} /> */}
              <img src={AadeLogo} alt='' width={140} height={60} />
            </div>
            <div className='timologio-text'>timologio</div>
          </div>
        </div>
        <div className='invoice-totals-border'>
          <p>
            Συνολική αξία:
            <span className='invoice-totals'>{twoDecimals(totalPriceNet)}€</span>
          </p>
          <p>
            Συνολικό Φ.Π.Α.:
            <span className='invoice-totals'>{twoDecimals(totalPriceVat)}€</span>
          </p>
          <p>
            Σύνολο:
            <span className='invoice-totals'>{twoDecimals(totalPrice)}€ </span>
          </p>
        </div>
      </div>
      {isVoid && <div className='isVoid'>ΑΚΥΡΟ</div>}
    </div>
    <div className='delete-invoice-btn' onClick={handleClickOpen}>
      Ακύρωση Παραστατικού
    </div>
</div>
```

Σχήμα 3.80: Κώδικας για τη προβολή παραστατικού (3).

Στην αρχή της Σελίδας, παίρνουμε το “id” του παραστατικού από τη διεύθυνση της σελίδας με τη χρήση του Hook “useParams()”, όπως βλέπουμε στο Σχήμα 3.81. Με τη χρήση του Hook “useEffect()”, δημιουργούμε μία συνάρτηση τύπου “arrow”. Στη συνάρτηση αυτή, με τη χρήση του Hook “useDispatch()”, καλούμε τη συνάρτηση “getInvoice()”, έχοντας ως παράμετρο το “id” του παραστατικού. Η συνάρτηση “getInvoice()” δημιουργεί ένα νέο αίτημα post τύπου “GET” στο Route “/api/invoice/invoiceId”. Εάν το αίτημα γίνει “Fulfilled”, τότε η global μεταβλητή “invoice” τύπου “invoice” γεμίζει με τα στοιχεία του τιμολογίου. Στη μεταβλητή “invoice” έχουμε πρόσβαση με τη χρήση του Hook “useSelector()”. Εάν το αίτημα γίνει “Rejected”, εμφανίζεται σχετικό μήνυμα λάθους και η μεταβλητή “isErrorInvoice” τύπου “invoice” γίνεται “True”. Εάν η μεταβλητή “isErrorInvoice” είναι “True”, τότε στη σελίδα εμφανίζεται μήνυμα λάθους πως το παραστατικό δεν υπάρχει.

```
const { invoiceId } = useParams()
const dispatch = useDispatch()

const { invoice, isLoadingInvoice, isErrorInvoice } = useSelector((state) => state.invoice)

const { type, series, aa, date, mark, paymentType, totalPriceNet, totalPriceVat, totalPrice, isVoid, _id, taxes } =
  invoice

const componentRef = useRef()

useEffect(() => {
  // Get invoice from db
  dispatch(getInvoice(invoiceId))
    .unwrap()
    .then(() => {})
    // if error display error message
    .catch(toast.error)
}, [invoiceId, dispatch])
```

Σχήμα 3.81: Εντοπισμός μη υπάρχοντος παραστατικού.

Τέλος, για τη δημιουργία του παραστατικού σε μορφή αρχείου PDF, χρησιμοποιείται η Βιβλιοθήκη React – to – Print. Για να χρησιμοποιούμε τη συγκεκριμένη Βιβλιοθήκη, κάνουμε import ένα της Element. Σε αυτό ορίζουμε τον τίτλο που θα έχει το αρχείο PDF που θα δημιουργηθεί, ένα κουμπί που το κάνει Trigger και ένα ref που τού υποδεικνύει τι να τυπώσει.

```
<ReactToPrint
   documentTitle='invoice'
   trigger={() => (
      <div>
         <button className='btn-back'>
            <Parastatika fill='#fff' width='18px' height='18px' /> Εκτύπωση
         </button>
      </div>
   )}
   content={() => componentRef.current}
/>
```

Σχήμα 3.82: Κώδικας δημιουργίας αρχείο PDF παραστατικού.

Κεφάλαιο 4ο: Υλοποίηση Back – End Μέρους Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης

Έχοντας παρουσιάσει και τις διαθέσιμες μεθόδους και τεχνολογίες που χρησιμοποιούνται για την ανάπτυξη του Back – End μέρους μίας Διαδικτυακής Εφαρμογής, στο κεφάλαιο αυτό θα επιλέξουμε τη καταλληλότερη τεχνολογία για την υλοποίηση του Back – End μέρους της Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης. Στο παρόν, λοιπόν, κεφάλαιο θα δούμε βήμα προς βήμα τη διαδικασία ανάπτυξης του Back – End μέρους της Διαδικτυακής Εφαρμογής μας για την Ηλεκτρονική Τιμολόγηση.

4.1 Επιλογή Τεχνολογίας Ανάπτυξης Back – End

Όπως είδαμε στο Κεφάλαιο 2, οι NodeJS, Java και Python είναι οι τεχνολογίες που χρησιμοποιούνται ευρέως για την ανάπτυξη του Back – End μέρους των εκάστοτε Διαδικτυακών Εφαρμογών.

Για την επιλογή της καταλληλότερης από τις τρεις προς ανάπτυξη του Back – End μέρους της Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης, θα πρέπει να λάβουμε υπόψη τις εξής προϋποθέσεις:

- Να διευκολύνει τον χειρισμό πολλαπλών αιτημάτων, και επομένως πολλαπλών Χρηστών ταυτόχρονα.
- Να επεξεργάζεται γρήγορα και αποτελεσματικά τα αιτήματα και τις απαντήσεις από και προς τους Χρήστες.
- Να υποστηρίζει τη δημιουργία REST API για τη δημιουργία αποτελεσματικής και απρόσκοπτης επικοινωνίας του Front – End μέρους με το Back – End μέρος της Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης, αλλά και τη Πλατφόρμα myData της Α.Α.Δ.Ε..
- Να διαθέτει μεγάλη και ενεργή κοινότητα προγραμματιστών που το αναπτύσσουν και εμπλουτίζουν περαιτέρω διαρκώς, η οποία να προσφέρει συνεχώς νέα εργαλεία και επεκτάσεις.

Λαμβάνοντάς τες υπόψη, επιλέξαμε το Node της JavaScript, καθώς:

- Διευκολύνει τον χειρισμό πολλαπλών αιτημάτων πολλαπλών Χρηστών ταυτόχρονα.
- Επιτρέπει την επαναχρησιμοποίηση κώδικα.
- Δεν αποκλείει κανένα αίτημα όταν ανταποκρίνεται σε άλλο.
- Επεξεργάζεται γρήγορα και αποτελεσματικά τις ροές δεδομένων.
- Υποστηρίζει τη δημιουργία REST APIs για τη δημιουργία αποτελεσματικής και απρόσκοπτης επικοινωνίας του Front – End μέρους με το Back – End μέρος μίας Διαδικτυακής Εφαρμογής.
- Διαθέτει μία αρκετά μεγάλη και ενεργή παγκόσμια κοινότητα προγραμματιστών που το αναπτύσσουν διαρκώς.

4.2 Δημιουργία Back – End Μέρους Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης

Για τη δημιουργία του Front – End της Διαδικτυακής Εφαρμογής χρησιμοποιείται η Βιβλιοθήκη React της JavaScript. Έτσι, επιτρέπεται στους χρήστες να καταχωρούν, να ανανεώνουν και να επεξεργάζονται στοιχεία, καθώς και να εκδίδουν σημαντικά – για την επιχείρησή τους – παραστατικά. Οι σελίδες που δημιουργούνται και χρησιμοποιούνται είναι οι εξής παρακάτω:

4.2.1 Πακέτα

Για την ανάπτυξη του Back – End μέρους της Διαδικτυακής Εφαρμογής μας χρησιμοποιείται η πλατφόρμα ανάπτυξης λογισμικού Node της JavaScript και για τη διαχείριση των πακέτων τον NPM (Node Package Manager). Αρχικά, με την εντολή "npm install" δημιουργούμε ένα αρχείο "package.json", το οποίο θα περιέχει τα μεταδεδομένα και τις εξαρτήσεις (dependencies). Τα πακέτα που χρησιμοποιούνται στη πτυχιακή αυτή εργασία, είναι τα εικονιζόμενα στο Σχήμα 4.1.

```
"dependencies": {
    "axios": "^0.27.2",
    "bcryptjs": "^2.4.3",
    "colors": "^1.4.0",
    "concurrently": "^7.1.0",
    "deploy": "^1.0.3",
    "dotenv": "^16.0.0",
    "express": "^4.18.1",
    "express-async-handler": "^1.2.0",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^6.3.1",
    "react-scripts": "^5.0.1",
    "setup": "^0.0.3",
    "xml-js": "^1.6.11"
},
"devDependencies": {
    "nodemon": "^2.0.19"
}
```

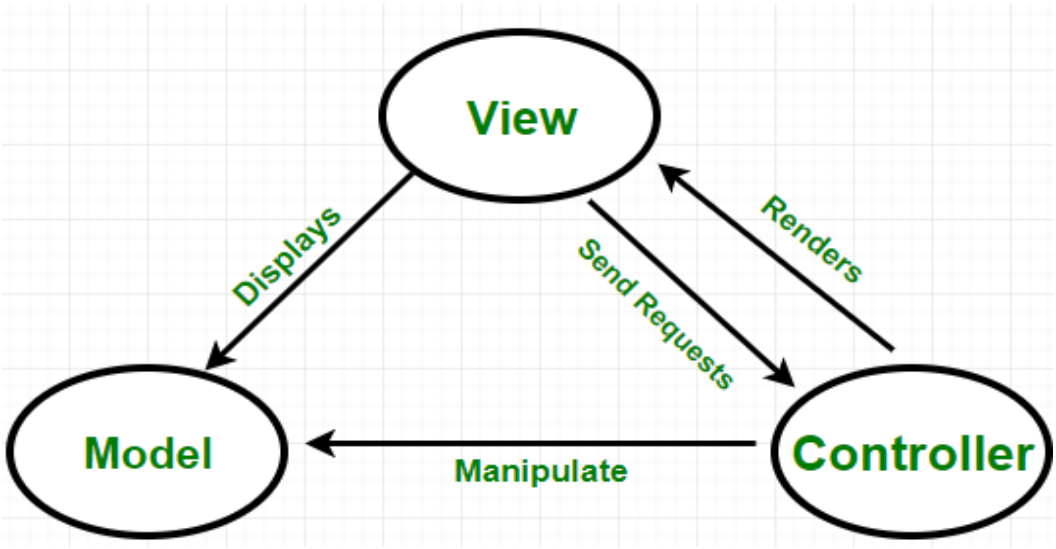
Σχήμα 4.1: Χρησιμοποιηθέντα πακέτα.

- **axios**: Είναι μία βιβλιοθήκη HTTP (Πρωτόκολλο Μεταφοράς Υπερκειμένου – HyperText Transfer Protocol), που μάς επιτρέπει να υποβάλλουμε αιτήματα είτε σε δικό μας είτε σε διακομιστή τρίτου για την ανάκτηση δεδομένων. Προσφέρει υποβολή αιτημάτων, όπως GET, POST, PUT/PATCH και DELETE.
- **bcrypt.js**: Είναι μία δημοφιλής βιβλιοθήκη της JavaScript για τον κατακερματισμό και την επαλήθευση κωδικών πρόσβασης, η οποία έχει σχεδιαστεί να δυσκολεύει τυχόν εισβολείς να «σπάσουν» τους κωδικούς πρόσβασης.
- **concurrently**: Είναι ένα πακέτο, το οποίο μάς επιτρέπει να εκτελούμε πολλαπλές εντολές ταυτόχρονα στο ίδιο τερματικό, γεγονός πού μάς βοηθάει να «τρέχουμε» τα Front – End και Back – End μέρη της εφαρμογής μας, χρησιμοποιώντας μία μόνο εντολή.

- **dotenv**: Πρόκειται για μία λειτουργική μονάδα μηδενικής εξάρτησης, η οποία μάς επιτρέπει να αποθηκεύουμε μεταβλητές περιβάλλοντος από ένα αρχείο .env, τις οποίες χρησιμοποιούμε για να αποθηκεύουμε ευαίσθητα δεδομένα, όπως κωδικούς πρόσβασης.
- **express**: Πρόκειται για ένα πλαίσιο διαδικτυακής εφαρμογής Back – End, το οποίο χρησιμοποιείται για τη δημιουργία RESTful API με τη πλατφόρμα Node της Javascript.
- **express – asynch – handler**: Είναι ένα ενδιάμεσο λογισμικό (middleware) προς τον χειρισμό εξαιρέσεων μέσα σε ασύγχρονες διαδρομές και τη διαβίβασή τους στους χειριστές σφαλμάτων express.
- **jsonwebtoken**: Πρόκειται για τη βιβλιοθήκη JSON Web Token (JWT), η οποία χρησιμοποιείται για τη δημιουργία δεδομένων με προαιρετική υπογραφή ή/και προαιρετική κρυπτογράφηση. Έτσι, όταν ένας χρήστης κάνει "login" στην εφαρμογή μας, δημιουργούμε ένα "token" – δηλαδή ένα "String", το οποίο περιέχει πληροφορίες σχετικά με τον χρήστη – με τη χρήση της βιβλιοθήκης αυτής, για λόγους ασφαλείας. Δημιουργούμε ένα ιδιωτικό κλειδί, με το οποίο κρυπτογραφείται το δημιουργηθέν "token", ώστε να μη μπορεί οποιοσδήποτε να αποκτήσει πρόσβαση στην εφαρμογή μας χωρίς να πρέπει ή ακόμη και να υποκλέψει πληροφορίες.
- **xml – js**: Είναι μία βιβλιοθήκη που μετατρέπει δεδομένα από μορφή XML σε μορφή json.
- **nodemon**: Πρόκειται για ένα εργαλείο, το οποίο Το nodemon είναι ένα εργαλείο, το οποίο παρακολουθεί τον κατάλογο του έργου και επανεκκινεί αυτομάτως την εφαρμογή κόμβου (Node.js) όταν εντοπίζει τυχόν αλλαγές.

4.2.2 Δομή Back – End

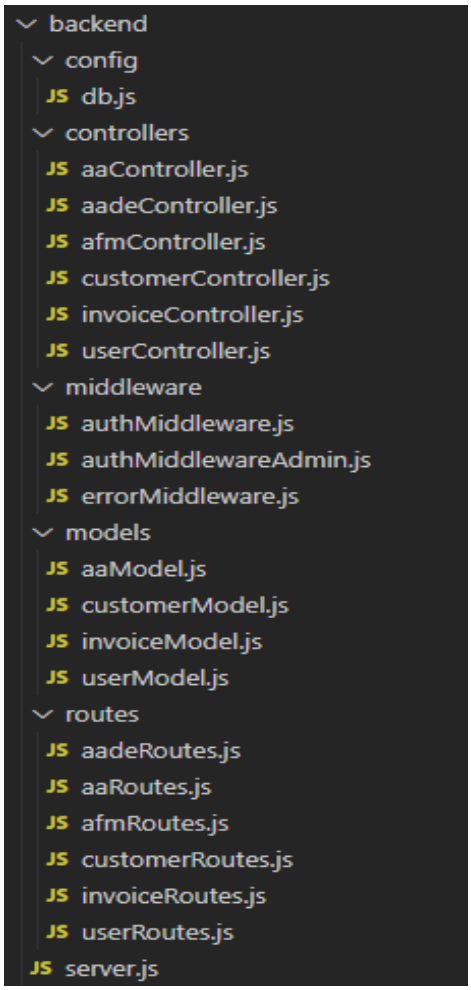
Αφού εγκαταστήσουμε όλα τα απαραίτητα πακέτα που θα χρησιμοποιήσουμε, δημιουργούμε τη δομή της Εφαρμογής μας. Το Back – End μέρος της Διαδικτυακής Εφαρμογής Ηλεκτρονικής Τιμολόγησης δημιουργήθηκε ακολουθώντας την Αρχιτεκτονική MVC (Model – View – Controller), η οποία εικονίζεται στο Σχήμα 4.2.



Σχήμα 4.2: Αρχιτεκτονική MVC.

Έτσι, όπως μπορούμε να δούμε στο Σχήμα 4.3, έχουμε δημιουργήσει:

- Έναν φάκελο “models”, ο οποίος περιέχει όλα τα αρχεία με τα οποία ορίζουμε τη δομή (δηλαδή τα πεδία) των “collection” της βάσης μας.
- Έναν φάκελο “routes”, στον οποίο ορίζουμε τα “endpoint” της εφαρμογής μας.
- Έναν φάκελο “controllers”, στον οποίο ορίζουμε τις συναρτήσεις που ανταποκρίνονται σε κάθε “route”.
- Έναν φάκελο “middleware”, όπου δημιουργούμε όλες τις ενδιάμεσες συναρτήσεις.
- Έναν φάκελο “config”, ο οποίος περιέχει το αρχείο “db.js”, που είναι υπεύθυνο για τη σύνδεση με τη βάση μας.
- Και το αρχείο “server.js”, το οποίο είναι υπεύθυνο για την εκκίνηση του Διακομιστή (Server), τη χρήση Ενδιάμεσου Λογισμικού (middleware) και των διαδρομών (Routes).



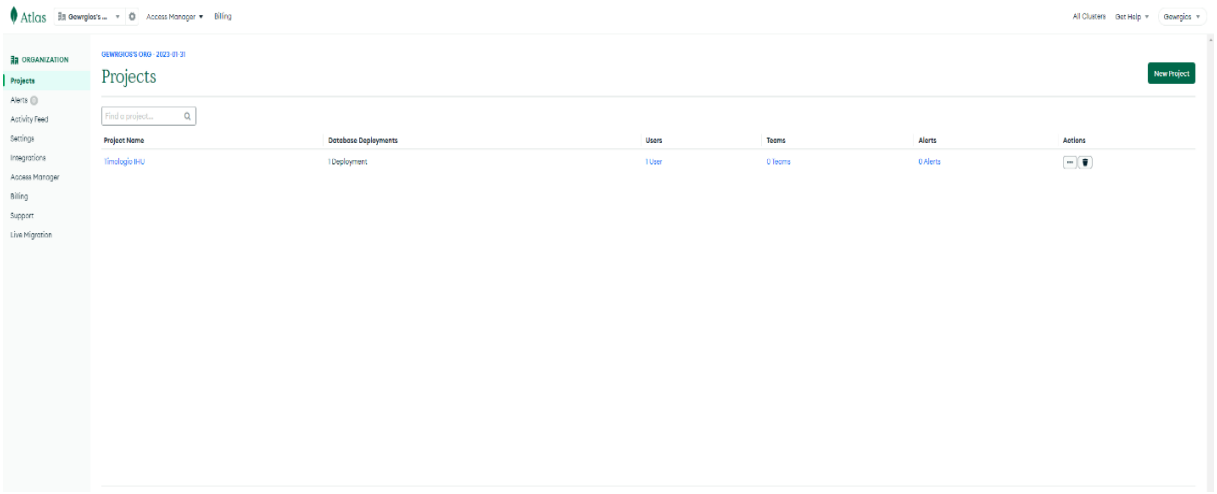
Σχήμα 4.3: Δομή Back – End.

Εφόσον ολοκληρωθεί η εγκατάσταση όλων των πακέτων, δημιουργούμε το αρχείο “server.js” στον φάκελο “backend”. Για να μπορέσουμε να χρησιμοποιήσουμε το “express” ως Server, το κάνουμε import εκτελώντας την εντολή “const express = require('express')”. Εν συνέχεια, δηλώνουμε την “PORT” που θα «τρέχει» ο Server μας. Λαμβάνοντας υπόψη ότι η Βιβλιοθήκη React δε «τρέχει» στην “PORT” 3000, διαλέγουμε μία διαφορετική. Δημιουργούμε, λοιπόν, μία νέα μεταβλητή με

όνομα “PORT” και τής δίνουμε τη τιμή που ορίσαμε στο “.env” αρχείο μας. Εάν δεν έχουμε δηλώσει κάποια τιμή, τότε αυτή γίνεται 5000 με την εντολή “const PORT = process.env.PORT || 5000”. Έπειτα, εγκαθιδρύουμε (set up) το “express” με την εντολή “const app = express()”. Τέλος, για να ξεκινήσει ο Server μας, καλούμε την εντολή “app.listen(PORT, () => console.log(`server started on port ${PORT}`))” στο object έχουμε δημιουργήσει.

4.2.3 Σύνδεση σε Βάση Δεδομένων

Ως βάση δεδομένων χρησιμοποιήθηκαν οι MongoDB, mongoose και MongoDB Atlas. Για να δημιουργήσουμε μία βάση δεδομένων MongoDB, πρέπει να επισκεφθούμε τη σελίδα “https://www.mongodb.com/”. Σε αυτήν, αφού συνδεθούμε με τα στοιχεία μας, θα χρειαστεί να δημιουργήσουμε ένα “Cluster”. Αφήνουμε τις default επιλογές του free tier και αλλάζουμε το όνομα του “Cluster” σε “TimologioIHU”. Στη συνέχεια, δημιουργούμε ένα νέο project με όνομα “Timologio IHU”, όπως φαίνεται στο Σχήμα 4.4.

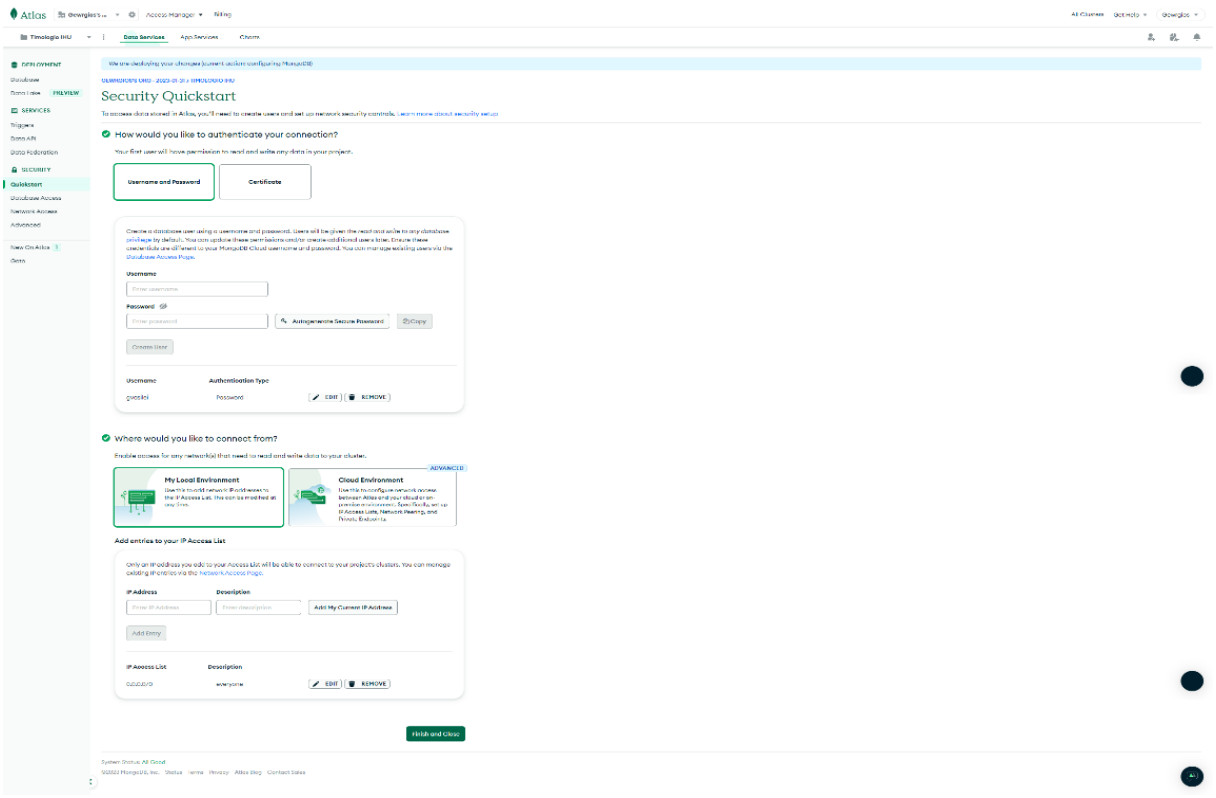


Σχήμα 4.4: Δημιουργία project Βάσης Δεδομένων.

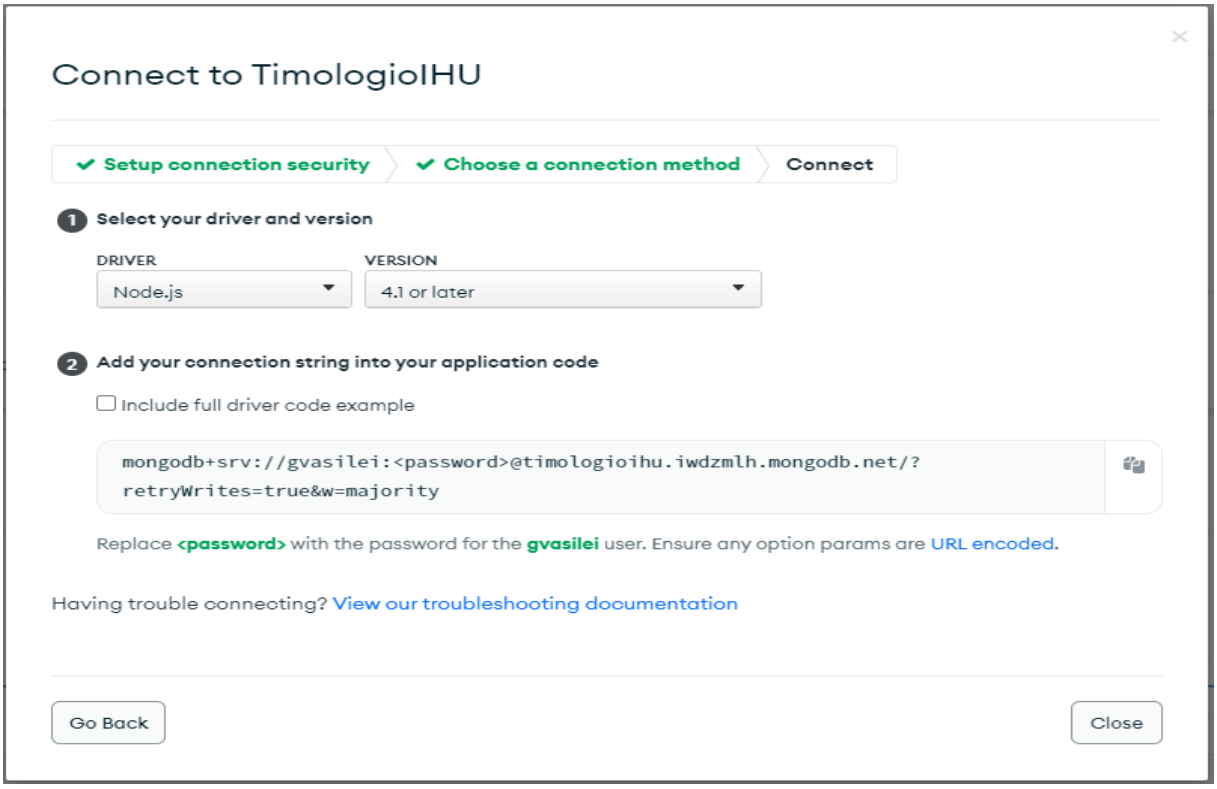
Για να μπορέσουμε να συνδεθούμε στη Βάση μας από την Εφαρμογή μας, θα πρέπει από το “Menu”, στο πεδίο “Security” να επιλέξουμε το “QuickStart”. Για το “Authendication” επιλέγουμε “Username” και “Password”. Δημιουργούμε έναν χρήστη με τα στοιχεία που επιθυμούμε και προσθέτουμε την IP “0.0.0.0”, ούτως ώστε να έχουμε πρόσβαση στη βάση μας από οποιοδήποτε υπολογιστή.

Για τη σύνδεση με τη Βάση μας, το MongoDB μάς παρέχει μία διεύθυνση. Από τον πίνακα ελέγχουμε, επιλέγουμε “Connect” και στη συνέχεια “Connect your application”. Αντικαθιστούμε το “password” με τον κωδικό μας και αποθηκεύουμε την τιμή σε μία μεταβλητή “.env”, ώστε να προφυλάξουμε τον κωδικό πρόσβασής μας.

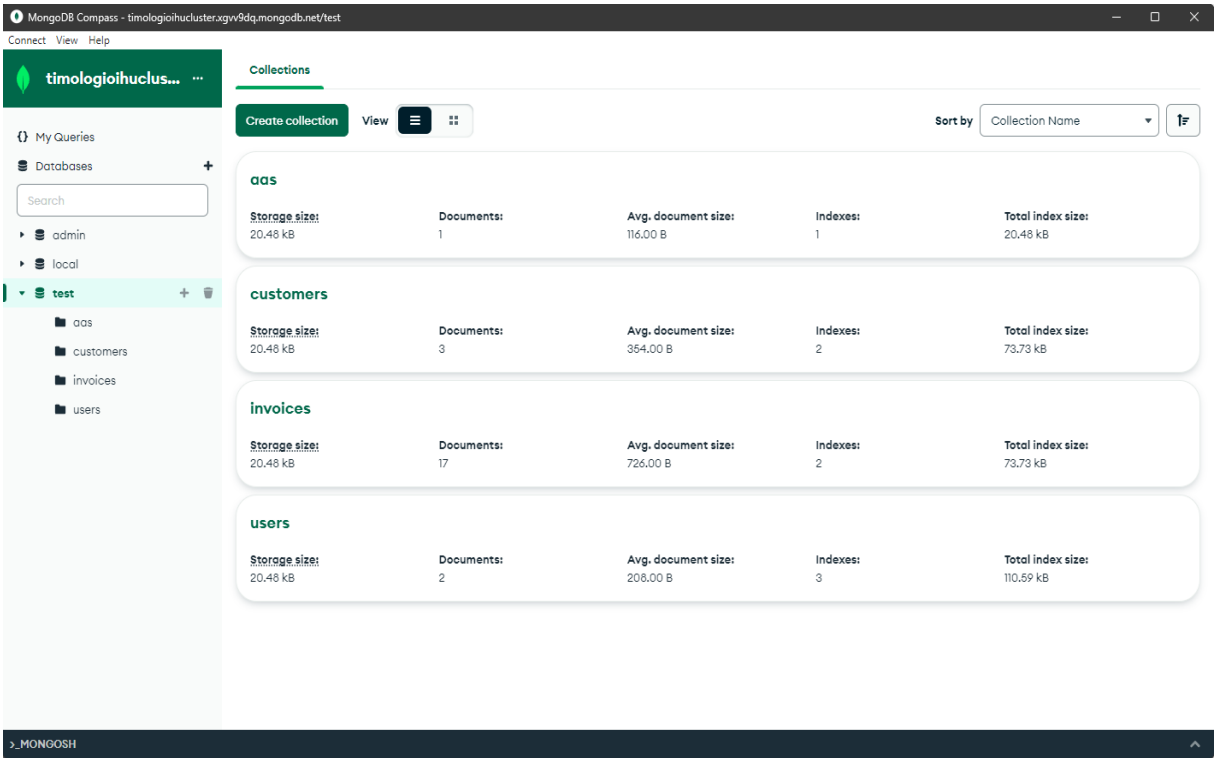
Όπως βλέπουμε στο Σχήμα 4.7, μπορούμε να κάνουμε το ίδιο και για τη σύνδεση με το MongoDB atlas, το οποίο μάς επιτρέπει να έχουμε πρόσβαση στη βάση και να τη διαχειριζόμαστε απλά και εύκολα.



Σχήμα 4.5: Δημιουργία χρήστη και πρόσθεση διεύθυνσης IP.



Σχήμα 4.6: Σύνδεση με τη Βάση Δεδομένων και εισαγωγή password.



Σχήμα 4.7: Διαχείριση δημιουργηθείσας Βάσης Δεδομένων.

Η σύνδεση της Εφαρμογής μας με τη Βάση Δεδομένων γίνεται στο αρχείο “./config/db.js”.

```
const connectDB = require('./config/db')

// connect to database
connectDB()
```

Σχήμα 4.8: Σύνδεση Εφαρμογής και Βάσης Δεδομένων.

Το MongoDB προσφέρει Cloud Database. Έτσι, με τη χρήση του πακέτου “mongoose”, καλούμε τη συνάρτηση σύνδεσης, χρησιμοποιώντας τη διεύθυνση που αποθηκεύσαμε νωρίτερα. Για να καλέσουμε τη συνάρτησή μας όταν τρέχει ο Server μας, τη κάνουμε import στο αρχείο “server.js” και έπειτα τη καλούμε.

```
const mongoose = require('mongoose')

const connectDB = async () => {
    try {
        const conn = await mongoose.connect(process.env.MONGO_URI)
        console.log(`Mongo db connected: ${conn.connection.host}.cyan.underline`.cyan.underline)
    } catch (error) {
        console.log(`Error: ${error.message}`.red.underline.bold)
        process.exit(1)
    }
}

module.exports = connectDB
```

Σχήμα 4.9: Κλήση της συνάρτησης σύνδεσης.

4.2.4 Models

Εφόσον επιτευχθεί η σύνδεση με τη Βάση Δεδομένων, δημιουργούμε τις συλλογές (models) που χρειαζόμαστε. Για κάθε συλλογή δημιουργούμε και ένα μοντέλο (model). Πιο συγκεκριμένα, στον φάκελο "models", δημιουργούμε τέσσερα μοντέλα:

1. Το μοντέλο "aaModel για τους Αύξοντες Αριθμούς (ΑΑ) στα παραστατικά.
2. Το μοντέλο "customerModel" για την αποθήκευση των στοιχείων των πελατών.
3. Το μοντέλο "invoiceModel" για την αποθήκευση των παραστατικών.
4. Το μοντέλο "userModel" για την αποθήκευση των δεδομένων των χρηστών.

Η δημιουργία του εκάστοτε "model" γίνεται με την χρήση του πακέτου "mongoose". Αρχικά, δημιουργούμε μία μεταβλητή που περιέχει το "Schema", όπως βλέπουμε στο Σχήμα 4.10. Κάθε "Schema" αντιστοιχίζεται σε μία συλλογή MongoDB και ορίζει το σχήμα των εγγράφων μέσα σε αυτή τη συλλογή. Κάθε κλειδί στον κώδικα του ενός "Schema" ορίζει μία ιδιότητα στα έγγραφά μας, η οποία θα μεταδοθεί στο σχετικό τύπο "Schema". Προκειμένου να χρησιμοποιήσουμε ένα "Schema", πρέπει να το μετατρέψουμε σε ένα μοντέλο, με το οποίο και μπορούμε να εργαστούμε. Για να το κάνουμε αυτό, το περνάμε στη συνάρτηση "mongoose.model()" ως εξής: "mongoose.model(modelName, Schema)".

```
const mongoose = require('mongoose')

const userSchema = mongoose.Schema(
   {
      name: {
         type: String,
         required: [true, 'Please add a name'],
         unique: true,
      },
      email: {
         type: String,
         required: [true, 'Please add an email'],
         unique: true,
      },
      password: {
         type: String,
         required: [true, 'Please add a password'],
      },
      isAdmin: {
         type: Boolean,
         required: true,
         default: false,
      },
   },
   {
      timestamps: true,
   }
)

module.exports = mongoose.model('User', userSchema)
```

Σχήμα 4.10: Δημιουργία models.

4.2.5 Διαχείριση Σφαλμάτων

Προτού ξεκινήσουμε να δηλώσουμε κάποιο route ή κάποιον controller, δημιουργούμε μερικά middleware.

Επειδή το πακέτο “express” από default επιστρέφει μία σελίδα HTML σε τυχόν λάθος “requests”, δημιουργούμε ένα νέο middleware, το οποίο θα επιστρέφει το σφάλμα (error) υπό τη μορφή “json”. Με τον τρόπο αυτό, θα μπορούμε να κατανοούμε και να διαχειριζόμαστε το σφάλμα πολύ ευκολότερα.

Πιο συγκεκριμένα, στον φάκελο “middleware” δημιουργούμε το αρχείο “errorMiddleware.js”. Δημιουργούμε μία συνάρτηση “errorHandler()” τύπου “arrow”, η οποία δέχεται ως παραμέτρους τις μεταβλητές “err”, “req”, “res” και “next”. Στη συνέχεια, δηλώνουμε μία μεταβλητή “statusCode”, η οποία λαμβάνει τον κωδικό του λάθους μέσω της μεταβλητής “res”, στην οποία ορίζουμε εμείς πού θα χρειαστεί να «πιάσει» (catch) ένα σφάλμα (error), εάν υπάρχει κάποιο. Ειδάλλως, ορίζουμε εμείς τη τιμή της σε “500”, όπως βλέπουμε στο Σχήμα 5.11. Έπειτα, επιστρέφουμε ένα “Promise”, το οποίο έχει ως “statusCode” τη τιμή που ορίσαμε και ως μήνυμα το μήνυμα λάθους που παίρνουμε από την μεταβλητή “err”. Τέλος, κάνουμε “export” τη δημιουργηθείσα συνάρτηση.

```
const errorHandler = (err, req, res, next) => {
    const statusCode = res.statusCode ? res.statusCode : 500
    res.status(statusCode)
    res.json({
        message: err.message,
    })
}

module.exports = { errorHandler }
```

Σχήμα 4.11: Σύνταξη συνάρτησης “errorHandler()”.

Για να μπορούμε να χρησιμοποιούμε το “middleware” που δημιουργούμε, το κάνουμε import στο αρχείο “server.js” και με τη βοήθεια του “app.use” το καλούμε.

```
const { errorHandler } = require('./middleware/errorMiddleware')

app.use(errorHandler)
```

Σχήμα 4.12: Εισαγωγή του ”middleware” στο αρχείο “server.js”.

4.2.6 Authentication & Authorization

Για την ασφαλή χρήση της Εφαρμογής μας από τους Χρήστες, χρησιμοποιήσαμε JSON Web Tokens (JWT). Με τη χρήση των Tokens, μπορούμε να προσδιορίσουμε εάν ένας Χρήστης έχει κάνει “login” και εάν το “Token” του είναι έγκυρο – εάν δηλαδή έχει λήξει ή όχι.

Για τον έλεγχο των JSON Web Tokens, δημιουργούμε ένα ακόμη "middleware", το οποίο θα ελέγχει την εγκυρότητά τους. Έτσι, στον φάκελο "middlware" δημιουργούμε ένα νέο αρχείο "authMiddleware.js". Για να μπορούμε από το Front – End να έχουμε πρόσβαση σε "routes" που είναι μόνο για χρήστες που έχουν κάνει "login" στην εφαρμογή μας, πρέπει να στέλνουμε και το token που δημιουργείται όταν γίνεται "login". Το Token θα στέλνεται ως "header" με όνομα "authorization" και η τιμή του θα έχει την μορφή "Bearer tokenValue".

Δημιουργούμε μία ασύγχρονη συνάρτηση "protect()", η οποία δέχεται ως παραμέτρους τις μεταβλητές "req", "res" και "next". Αυτή η συνάρτηση, αρχικά, ορίζει μία κενή μεταβλητή. Στη συνέχεια, με τη χρήση if statement, ελέγχουμε εάν στα "headers" του "request" υπάρχει "header" με όνομα "authorization" και εάν η τιμή του ξεκινά με τη λέξη "Bearer". Εάν η συνθήκη ισχύει, τότε δημιουργούμε μία δομή try – catch, ώστε να ελέγξουμε το Token. Μέσα στην "try" παίρνουμε το token, το χωρίζουμε με βάση το κενό (' ') και παίρνουμε το δεύτερο μέρος που περιέχει την τιμή του και την τοποθετούμε στη κενή μεταβλητή που δημιουργούμε προηγουμένως. Έπειτα, χρησιμοποιούμε τη συνάρτηση "verify" της Βιβλιοθήκης jwt, η οποία δέχεται ως παραμέτρους το Token και το μοναδικό κλειδί που χρησιμοποιούμε για τη δημιουργία των Token και είναι αποθηκευμένο στο αρχείο ".env". Η συνάρτηση αυτή επιστρέφει ένα object, το οποίο περιέχει πληροφορίες σχετικά με το Token, και πιο συγκεκριμένα, περιέχει το "id" του χρήστη, το πότε δημιουργήθηκε το Token και το πότε λήγει.

Μετέπειτα, συγκρίνουμε την ημερομηνία λήξης του Token με την εκάστοτε σημερινή ημερομηνία, για να διαπιστώσουμε εάν έχει λήξει. Σε περίπτωση που έχει λήξει, τότε επιστρέφεται ένα σφάλμα με το κατάλληλο "error". Χρησιμοποιώντας το "id" που έχει το Token, ψάχνουμε τον χρήστη στην βάση μας. Εάν δε βρούμε κάποιον χρήστη, επιστρέφουμε ένα σφάλμα με το κατάλληλο "error". Εάν, όμως, βρούμε τον χρήστη, τότε ενεργοποιείται το "next()". Εάν στο "try" υπάρξει κάποιο πρόβλημα, τότε επιστρέφουμε ένα σφάλμα με το κατάλληλο "error". Επίσης, εάν δεν υπάρχει κάποιο Token στα "headers", επιστρέφουμε ένα σφάλμα με το κατάλληλο "error". Τέλος, κάνουμε export τη συνάρτηση, όπως φαίνεται στο Σχήμα 4.13.

4.2.7 Routes & Controllers

Για κάθε πόρο που θα χρειαστούμε, δημιουργούμε ξεχωριστές διαδρομές (Routes), ώστε να διαχειρίζεται τις μεθόδους "GET", "PUT", "DELETE", "UPDATE" που θα χρησιμοποιήσουμε. Συνολικά έχουμε τις εξής έξι λειτουργίες:

1. Επικοινωνία με την Α.Α.Δ.Ε. (aadeRoutes.js),
2. Επικοινωνία με την εφορία για τα στοιχεία πελάτη (afmRoutes.js)
3. Αύξοντες αριθμοί (aaRoutes.js),
4. Διαχείριση πελατών (customerRoutes.js),
5. Διαχείριση παραστατικών (invoiceRoutes.js),
6. Διαχείριση χρηστών (userRoutes.js).

Στο Σχήμα 4.14, βλέπουμε ένα παράδειγμα δημιουργίας ενός Route.

```
const jwt = require('jsonwebtoken')
const asyncHandler = require('express-async-handler')
const User = require('../models/userModel')

const protect = asyncHandler(async (req, res, next) => {
    let token

    if (req.headers.authorization && req.headers.authorization.startsWith('Bearer')) {
        try {
            // Get token from header
            token = req.headers.authorization.split(' ')[1]

            // Verify token
            const decoded = jwt.verify(token, process.env.JWT_SECRET)

            if (Date.now() >= decoded.exp * 1000) {
                res.status(401)
                throw new Error('Not authirised')
            }

            //Get the user from token
            req.user = await User.findById(decoded.id).select('-password')

            if (!req.user) {
                res.status(401)
                throw new Error('Not authirised')
            }

            next()
        } catch (error) {
            //console.log(error)
            res.status(401)
            throw new Error('Not Authorized')
        }
    }

    if (!token) {
        //console.log(error)
        res.status(401)
        throw new Error('Not Authorized')
    }
})

module.exports = { protect }
```

Σχήμα 4.13: Σύνταξη συνάρτησης “protect()”.

```
const express = require('express')
const router = express.Router()
const {
    getInvoice,
    createInvoice,
    getAllInvoices,
    getLastMonthInvoices,
    updateInvoice,
} = require('../controllers/invoiceController')

const { protect } = require('../middleware/authMiddleware')

router.route('/').post(protect, createInvoice).get(protect, getAllInvoices).get(protect, getLastMonthInvoices)

router.route('/monthly').get(protect, getLastMonthInvoices)

router.route('/:id').get(protect, getInvoice).put(updateInvoice)

module.exports = router
```

Σχήμα 4.14: Παράδειγμα δήλωσης ενός Route.

Μπορούμε να δηλώσουμε ένα Route (διαδρομή) με τη βοήθεια του “express.Router”. Στην αρχή, δηλώνουμε το “path”, όπου θα τρέχει το συγκεκριμένο “request”, μετά δηλώνουμε τον τύπο του “request” που θέλουμε (“GET”, “POST”, “PUT” ή “DELETE”) και, τέλος, δηλώνουμε τις συναρτήσεις που θέλουμε να εκτελούνται. Μπορούμε να τρέξουμε παραπάνω από μία συναρτήσεις

όπως βλέπουμε και στο εν λόγω παράδειγμα. Με τη χρήση της συνάρτησης "protect()" προστατεύουμε το "request" από τυχόν χρήστες που δεν είναι συνδεμένοι. Τέλος, κάνουμε "export" το route, όπως φαίνεται στο Σχήμα 4.14.

Όπως μπορούμε να δούμε στο Σχήμα 4.15, για να μπορέσουμε να χρησιμοποιήσουμε τα Routes που δημιουργούμε στο αρχείο "server.js" με τη χρήση του "app.use", ορίζουμε σε ποιο "path" «ακούει» το κάθε Route.

```
// Routes
app.use('/api/users', require('./routes/userRoutes'))
app.use('/api/invoice', require('./routes/invoiceRoutes'))
app.use('/api/customer', require('./routes/customerRoutes'))
app.use('/api/aa', require('./routes/aaRoutes'))
app.use('/api/afm', require('./routes/afmRoutes'))
app.use('/api/aadeinvoice', require('./routes/aadeRoutes'))
```

Σχήμα 4.15: Ορισμός "path" για κάθε Route.

Επομένως, για να δημιουργήσουμε ένα νέο "invoice", θα πρέπει να δημιουργήσουμε ένα "POST" request στο Path "/api/invoice". Και για να πάρουμε τα μηνιαία παραστατικά, πρέπει να δημιουργήσουμε ένα "GET" request στο Path "/api/invoice/monthly".

Στην Εφαρμογή μας δημιουργούμε και χρησιμοποιούμε έξι (6) controllers:

1. **AADE postInvoice Controller**
   Είναι η μέθοδος αποστολής των πληροφοριών του παραστατικού του οποίου θέλουμε να γίνει έκδοση και απεικονίζεται στο Σχήμα 4.16. Στην αρχή, παίρνουμε το "xml" από το Body του "request" μας. Στη συνέχεια, δημιουργούμε τα "headers" που περιέχουν τα κατάλληλα πεδία, το "user" και τον κωδικό χρήστη. Με τη χρήση μίας try – catch, δημιουργούμε ένα νέο "POST" request στη κατάλληλη διεύθυνση της Α.Α.Δ.Ε., έχοντας ως παραμέτρους το "xml" και τα "headers" που δημιουργήσαμε. Έπειτα λαμβάνουμε την απάντηση της κλήσης μας. Καθώς η ληφθείσα απάντηση είναι σε μορφή "xml", την μετατρέπουμε σε μορφή "json" και αποθηκεύουμε τις πληροφορίες "Mark", "id" και "statusCode" που χρειαζόμαστε σε μεταβλητές. Ελέγχουμε εάν η τιμή της μεταβλητής είναι ίση με "Success". Εάν ναι, δημιουργούμε ένα object και το επιστρέφουμε. Εάν υπάρξει κάποιο λάθος, τότε επιστρέφουμε κατάλληλο μήνυμα λάθους.

2. **AADE postCancelInvoice Controller**
   Είναι η μέθοδος ακύρωσης ενός παραστατικού με βάση του Μοναδικού Αριθμού Καταχώρισής (Μ.ΑΡ.Κ.) του και εικονίζεται στο Σχήμα 4.17. Η ακύρωση γίνεται με την αποστολή της μεταβλητής "Mark" στη κατάλληλη διεύθυνση. Στην αρχή, παίρνουμε το ΜΑΡΚ του παραστατικού που θέλουμε να ακυρώσουμε από το body του "request μας". Στη συνέχεια, δημιουργούμε τα "headers" που περιέχουν τα κατάλληλα πεδία, δηλαδή το "user" και τον κωδικό χρήστη. Με τη χρήση μίας try – catch, δημιουργούμε ένα νέο "POST" request στη κατάλληλη διεύθυνση της Α.Α.Δ.Ε., έχοντας ως παραμέτρους τη μεταβλητή "Mark" που πήραμε από το Body και τα "headers" που δημιουργήσαμε. Έπειτα, παίρνουμε την απάντηση της κλήσης μας, την οποία μετατρέπουμε από "xml" σε "json", και αποθηκεύουμε το

"statusCode" της απάντησης. Εάν αυτό είναι ίσο με "Success", τότε το επιστρέφουμε. Εάν υπάρξει κάποιο λάθος, τότε επιστρέφουμε κατάλληλο μήνυμα λάθους.

```
// @desc returns invoice info
// @route GET /api/aadeinvoice
// @access Private
const postInvoice = asyncHandler(async (req, res) => {
    const { xmls } = req.body

    const config = {
        headers: {
            'aade-user-id': `${process.env.AADE_USER}`,
            'ocp-apim-subscription-key': `${process.env.AADE_PASS}`,
            rejectUnauthorized: 'false',
        },
    }

    const ssl = {
        strictSSL: false,
    }
    process.env['NODE_TLS_REJECT_UNAUTHORIZED'] = '0'
    try {
        const response = await axios
            .post('https://mydataapidev.aade.gr/SendInvoices', xmls, config, ssl)
            .then((response) => {
                const responseObject = convert.xml2js(response.data, { compact: true })
                //res.status(200).json(responseObject)
                const intrestingStuff = responseObject['ResponseDoc']['response']
                const mark = intrestingStuff['invoiceMark']['_text']
                const uid = intrestingStuff['invoiceUid']['_text']
                const statusCode = intrestingStuff['statusCode']['_text']

                if (statusCode === 'Success') {
                    const info = { mark, uid, statusCode }

                    res.status(200).json(info)
                } else {
                    res.status(404).json('Το τιμολόγιο δεν στάλθηκε, δοκιμάστε ξανά')
                }
            })
    } catch (err) {
        res.status(400).send(err.message)
    }
})
```

Σχήμα 4.16: Σύνταξη "postInvoice()".

```
// @desc returns invoice info
// @route GET /api/aadeinvoice
// @access Private
const postCancelInvoice = asyncHandler(async (req, res) => {
    const { mark } = req.body

    const config = {
        headers: {
            'aade-user-id': `${process.env.AADE_USER}`,
            'ocp-apim-subscription-key': `${process.env.AADE_PASS}`,
        },
    }
    process.env['NODE_TLS_REJECT_UNAUTHORIZED'] = '0'

    try {
        const response = await axios
            .post(`https://mydataapidev.aade.gr/CancelInvoice?mark=${mark}`, null, config)
            .then((response) => {
                const responseObject = convert.xml2js(response.data, { compact: true })
                const intrestingStuff = responseObject['ResponseDoc']['response']
                const statusCode = intrestingStuff['statusCode']['_text']

                if (statusCode === 'Success') {
                    res.status(200).json(statusCode)
                } else {
                    res.status(404).json('Το τιμολόγιο δεν διαγράφηκε, δοκιμάστε ξανά')
                }
            })
    } catch (err) {
        res.status(400).send(err.message)
    }
})
```

Σχήμα 4.17: Σύνταξη "postCancelInvoice()".

3. **Invoice createInvoice Controller**

Είναι η μέθοδος αποθήκευσης των πληροφοριών ενός παραστατικού στη βάση μας και φαίνεται στο Σχήμα 4.18. Στην αρχή, παίρνουμε τα πεδία που θέλουμε να αποθηκεύσουμε από το Body του "request" μας. Στη συνέχεια, βάσει του "id" του χρήστη που κάνει τη κλήση – το id το παίρνουμε από το Body – βρίσκουμε τον χρήστη από τη βάση μας. Το ίδιο κάνουμε και για τον πελάτη για τον οποίο θέλουμε να εκδοθεί το παραστατικό. Σε περίπτωση που το παραστατικό είναι απόδειξη λιανικής πώλησης, τότε ο χρήστης είναι κενός. Τέλος αποθηκεύουμε τις πληροφορίες στη βάση μας και στέλνουμε κατάλληλο μήνυμα επιτυχής δημιουργίας.

```
// @route POST /api/invoice
// @access Private
const createInvoice = asyncHandler(async (req, res) => {
   const {type,uid,series,aa,date,mark,paymentType,customerId,customerAFM,customerProfession,customerName,totalPrice,
      totalPriceNet,totalPriceVat,products,taxes,userId,
   } = req.body
   const createdBy = await User.findById(userId)

   const customer = await Customer.findById(customerId)

   const invoice = await Invoice.create({type,uid,series,aa,date,mark,paymentType,customer,customerAFM,customerProfession,
      customerName, products, taxes, totalPrice, totalPriceNet, totalPriceVat, createdBy,
   })

   res.status(201).json(invoice)
})
```

Σχήμα 4.18: Σύνταξη "createInvoice()".

4. **Invoice getAllInvoice Controller**

Είναι η μέθοδος που επιστρέφει τα παραστατικά βάσει των κριτηρίων αναζήτησης που διαλέγει ο χρήστης, όπως φαίνεται στο Σχήμα 4.19. Τα κριτήρια είναι ημερομηνίες από έως, το Μ.ΑΡ.Κ., ο Α/Α, το Α.Φ.Μ. του πελάτη και ο τύπος του παραστατικού. Στην αρχή, παίρνουμε τα κριτήρια αναζήτησης που έχουμε περάσει ως παραμέτρους στη διεύθυνση του "request" μας ως ένα "query". Αφού πάρουμε το "query", το χωρίζουμε με βάση το "/" (slash) και αποθηκεύουμε τις πληροφορίες σε μεταβλητές. Πιο συγκεκριμένα, στη μεταβλητή "markAA" αποθηκεύουμε τους χαρακτήρες που έχει πληκτρολογήσει ο χρήστης στο πεδίο αναζήτησης του Front – End μέρους. Στις μεταβλητές "dateGTE" και "dateLT" αποθηκεύουμε τις ημερομηνίες από και έως αντίστοιχα, και τέλος στη μεταβλητή "typeInv" αποθηκεύουμε τον τύπο του παραστατικού. Στη συνέχεια, βρίσκουμε τα παραστατικά που πληρούν τα κριτήρια αναζήτησης και τα αποθηκεύουμε σε μία μεταβλητή. Για τα κριτήρια αναζήτησης δημιουργούμε μία συνθήκη and, η οποία αποτελείται από δύο κριτήρια:

- Το πρώτο κριτήριο ελέγχει εάν η μεταβλητή "markAA" ταιριάζει με κάποια τιμή από Μ.ΑΡ.Κ., Α/Α ή Α.Φ.Μ. πελάτη. Με τη χρήση του "$regex" σιγουρεύουμε πως εάν π.χ. δε γίνει αντιστοίχιση αναζήτησης για τις τιμές του Μ.ΑΡ.Κ., αυτή η αναζήτηση δεν επηρεάζει το αποτέλεσμα της γενικής αναζήτησης.
- Το δεύτερο κριτήριο είναι για την ημερομηνία δημιουργίας του παραστατικού. Είναι ένα εύρος που ελέγχει ποια τιμολόγια δημιουργήθηκαν από μία ημερομηνία έως μία άλλη.

Τέλος, ελέγχουμε τη τιμή της μεταβλητής "typeInv", ώστε να δούμε τον τύπο του παραστατικού. Εάν η μεταβλητή είναι κενή, τότε επιστρέφουμε όλα τα παραστατικά

ανεξαρτήτως τύπου. Με την επιλογή “sort” επιλέγουμε να μάς επιστραφούν τα παραστατικά από το πιο πρόσφατο στο πιο παλιό. Εάν δεν βρεθούν παραστατικά, επιστρέφουμε κατάλληλο μήνυμα λάθους. Τέλος, επιστρέφουμε ένα αντικείμενο που περιέχει τα παραστατικά.

```
// @desc get all invoices
// @route GET /api/invoice/all
// @access Private
const getAllInvoices = asyncHandler(async (req, res) => {
    const { q } = req.query
    const words = q.split('/')
    const markAA = words[0]
    const dateGTE = words[1]
    const dateLT = words[2]
    const typeInv = words[3]

    const invoices = await Invoice.find({
        $and: [
            {
                $or: [
                    { mark: { $regex: markAA || '' } },
                    { aa: { $regex: markAA || '' } },
                    { customerAFM: { $regex: markAA || '' } },
                ],
            },
            {
                createdAt: { $gte: dateGTE, $lt: dateLT },
            },
            {
                type: typeInv || [
                    'Τιμολόγιο Πώλησης',
                    'Τιμολόγιο Παροχής Υπηρεσιών',
                    'Απόδειξη Λιανικής Πώλησης',
                    'Προσφορά',
                ],
            },
        ],
    }).sort({
        createdAt: -1,
    })

    if (!invoices) {
        res.status(404)
        throw new Error('There are no invoices')
    }

    res.status(200).json(invoices)
})
```

Σχήμα 4.19: Σύνταξη “getAllInvoice()”.

5. Invoice getInvoice Controller

Είναι η συνάρτηση που επιστρέφει ένα συγκεκριμένο παραστατικό με βάση βάσει του M.AP.K. του, όπως φαίνεται στο Σχήμα 4.20. Αρχικά, παίρνουμε το παραστατικό που θέλουμε από τη βάση σύμφωνα με το M.AP.K. που πήραμε από το Body του “request”. Στη συνέχεια, παίρνουμε το “id” του πελάτη από τα στοιχεία του παραστατικού και τα στοιχεία από τη βάση δεδομένων, και τα αποθηκεύουμε στα στοιχεία του παραστατικού. Τέλος, επιστρέφουμε ένα αντικείμενο με τα στοιχεία του παραστατικού.

```
// @desc get invoice
// @route GET /api/invoice/:id
// @access Private
const getInvoice = asyncHandler(async (req, res) => {
    const invoice = await Invoice.findOne({ mark: req.params.id })
    const { customer } = invoice
    const client = await Customer.findById(customer)
    invoice['customer'] = client

    if (!invoice) {
        res.status(401)
        throw new Error('Invoice not found')
    }

    res.status(200).json(invoice)
})
```

Σχήμα 4.20: Σύνταξη “getInvoice()”.

6. AFM postAfm Controller

Είναι η συνάρτηση αναζήτησης των στοιχείων πελάτη βάσει του Α.Φ.Μ. του και απεικονίζεται στο Σχήμα 4.21. Αρχικά, παίρνουμε το Α.Φ.Μ. του πελάτη από το Body της κλήσης και δημιουργούμε το κατάλληλο "xml" σε μορφή "String". Στη συνέχεια, με τη χρήση μίας try – catch, δημιουργούμε ένα νέο "POST" request στη κατάλληλη διεύθυνση, έχοντας σαν "headers" τη μεταβλητή που περιέχει το "xml" και τα απαραίτητα "headers", ώστε να είναι έγκυρο το "request" μας. Λαμβάνοντας την απάντηση, στην αρχή παίρνουμε τα επαγγέλματα του πελάτη. Έπειτα, ελέγχουμε τον αριθμό τους. Εάν ο πελάτης δηλώνει μόνο ένα επάγγελμα, τότε ισχύει η συνθήκη if που δημιουργήσαμε. Έτσι, αποθηκεύουμε το επάγγελμά του μαζί με τις υπόλοιπες πληροφορίες που χρειαζόμαστε (Όνομα, Επάγγελμα, Διεύθυνση, Αριθμός Διεύθυνσης, Πόλη και Ταχυδρομικός Κώδικας). Δημιουργούμε ένα αντικείμενο που περιέχει αυτές τις πληροφορίες και το επιστρέφουμε. Εάν ο πελάτης δηλώνει πάνω από ένα επαγγέλματα, τότε αποθηκεύουμε σε μία μεταβλητή μόνο το κύριο επάγγελμά του και μαζί με τις υπόλοιπες πληροφορίες που χρειαζόμαστε (Όνομα, Επάγγελμα, Διεύθυνση, Αριθμός Διεύθυνσης, Πόλη και Ταχυδρομικός Κώδικας). Δημιουργούμε ένα αντικείμενο που περιέχει τις πληροφορίες αυτές και το επιστρέφουμε. Εάν υπάρξει κάποιο λάθος, επιστρέφουμε σχετικό μήνυμα λάθους.

```
// @desc returns info from afm request
// @route GET /api/requestAfm
// @access Private
const postAfm = asyncHandler(async (req, res) => {
   const { reqAfm } = req.body

   const data = `<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" xmlns:ns1="http://docs.oasis-ope
   //try {
   const response = await axios
      .post('https://www1.gsis.gr/wsaade/RgWsPublic2/RgWsPublic2', data, {
         headers: { 'Content-Type': 'application/soap+xml', 'Access-Control-Allow-Origin': '*' },
      })
      .then((response) => {
         const responseObject = convert.xml2js(response.data, { compact: true })

         const intrestingStuff =
            responseObject['env:Envelope']['env:Body']['srvc:rgWsPublic2AfmMethodResponse']['srvc:result'][
               'rg_ws_public2_result_rtType'
            ]
         const basicRec = intrestingStuff['basic_rec']
         const proffesionsLenght = intrestingStuff['firm_act_tab']['item'].length

  if (proffesionsLenght > 1) {
      const firmActTab = intrestingStuff['firm_act_tab']['item'][0]

      const rname = basicRec['onomasia']['_text']
      const rproffesion = firmActTab['firm_act_descr']['_text']
      const raddress = basicRec['postal_address']['_text']
      const raddressNumber = basicRec['postal_address_no']['_text'].trim()
      const rcity = basicRec['postal_area_description']['_text']
      const rzip_code = basicRec['postal_zip_code']['_text']
      const rdoy = basicRec['doy_descr']['_text']
      const try2 = '2'
      const rinfo = {
         rname,rproffesion,raddress,raddressNumber,rcity,rzip_code,rdoy,
      }

      res.status(200).json(rinfo)
```

Σχήμα 4.21: Σύνταξη "postAfm()" (1).

```
} else {
    const firmActTab = intrestingStuff['firm_act_tab']['item']

    const rname = basicRec['onomasia']['_text']
    const rproffesion = firmActTab['firm_act_descr']['_text']
    const raddress = basicRec['postal_address']['_text']
    const raddressNumber = basicRec['postal_address_no']['_text'].trim()
    const rcity = basicRec['postal_area_description']['_text']
    const rzip_code = basicRec['postal_zip_code']['_text']
    const rdoy = basicRec['doy_descr']['_text']

    const rinfo = {
        rname, rproffesion, raddress, raddressNumber, rcity, rzip_code, rdoy,
    }

    res.status(200).json(rinfo)
}
```

Σχήμα 4.22: Σύνταξη “postAfm()” (2).

4.3 Ηλεκτρονική Πλατφόρμα myData

Η Ανεξάρτητη Αρχή Δημοσίων Εσόδων (Α.Α.Δ.Ε.) παρέχει μία Διεπαφή REST API, η οποία ονομάζεται myData (my Digital Accounting and Tax Application) και δίνει τη δυνατότητα στις επιχειρήσεις που διαθέτουν μηχανογραφημένα λογιστήρια ή χρησιμοποιούν κάποιο πληροφοριακό σύστημα να συνδέονται με αυτήν για την ανταλλαγή δεδομένων [30].

4.3.1 Προσφερόμενες Λειτουργίες & Μέθοδοι

Οι λειτουργίες που προσφέρει η εν λόγω Διεπαφή στις επιχειρήσεις είναι:

- Η αποστολή δεδομένων για παραστατικά που εκδίδουν.
- Η αποστολή χαρακτηρισμών εσόδων που αφορούν παραστατικά που εκδίδουν.
- Η λήψη δεδομένων όσων παραστατικών έχουν εκδοθεί για αυτές και έχουν διαβιβαστεί από τους αντίστοιχους εκδότες στην Α.Α.Δ.Ε..

Η αποστολή δεδομένων χαρακτηρισμών εξόδων στην Α.Α.Δ.Ε. [30].

Βάσει των λειτουργιών αυτών, παρέχει τις παρακάτω μεθόδους:

- Τη **/SendInvoices**, η οποία αποτελεί διαδικασία υποβολής ενός ή περισσότερων παραστατικών, συμπεριλαμβανομένων και των όποιων διορθωμένων – τροποποιητικών.
- Τη **/RequestDocs**, η οποία αποτελεί διαδικασία λήψης ενός ή περισσότερων παραστατικών, χαρακτηρισμών, ή ακυρώσεων παραστατικών που έχουν υποβάλλει άλλες επιχειρήσεις.
- Τη **/RequestTransmittedDocs**, η οποία αποτελεί διαδικασία λήψης ενός ή περισσότερων παραστατικών, χαρακτηρισμών ή ακυρώσεων παραστατικών που έχει υποβάλλει μία επιχείρηση.
- Τη **/SendIncomeClassification**, η οποία αποτελεί διαδικασία υποβολής χαρακτηρισμών εσόδων, ενός ή περισσότερων, που θα αντιστοιχούν σε ήδη υποβεβλημένα παραστατικά.

- Τη **/SendExpensesClassification**, η οποία αποτελεί διαδικασία υποβολής χαρακτηρισμών εξόδων, ενός ή περισσότερων, που θα αντιστοιχούν σε ήδη υποβεβλημένα παραστατικά.
- Τη **/CancelInvoices**, η οποία αποτελεί διαδικασία ακύρωσης παραστατικού, χωρίς τη ταυτόχρονη υποβολή νέου [30].

4.3.2 Τεχνολογικές Απαιτήσεις

Για την υλοποίηση επικοινωνίας ενός λογισμικού συστήματος με τη Διεπαφή της Α.Α.Δ.Ε., χρησιμοποιούνται οι εξής τεχνολογίες:

- Secure HTTP,
- Webservice,
- REST API, και
- XML [30].

Η Διεπαφή μπορεί να χρησιμοποιηθεί από λογισμικά συστήματα που μπορούν να υλοποιούν HTTPS αιτήματα, μέσω των οποίων να στέλνουν και αυτοματοποιημένα τις πληροφορίες για τη ταυτοποίηση του Χρήστη, και να δημιουργούν έγγραφα XML συμβατά με το σχήμα που περιγράφεται παρακάτω [30].

4.3.2.1 Εγγραφή Χρήστη & Ταυτοποίηση

Για χρήση των λειτουργιών της Διεπαφής απαιτεί διαδικασία ταυτοποίησης του Χρήστη (authentication). Η ταυτοποίηση πραγματοποιείται μέσω αποστολής – σε κάθε αίτημα – ενός Ονόματος Χρήστη, καθώς και ενός Subscription Key στην ενότητα Headers, που θα δούμε παρακάτω. Το Subscription Key είναι ένα String, μοναδικό ανά Χρήστη και κοινό σε όλες τις λειτουργίες της Διεπαφής. Για να αποκτήσει ένας Χρήστης τα εν λόγω διαπιστευτήρια (credentials), θα πρέπει να δημιουργήσει έναν λογαριασμό στο μητρώο των διεπαφών μέσω μίας ειδικής διαδικασίας εγγραφής, η οποία προσφέρεται από την ηλεκτρονική πλατφόρμα myDATA και ονομάζεται «Φόρμα εγγραφής στο myData REST API», όπως φαίνεται στο Σχήμα 4.23 [30].

Όπως φαίνεται και στο Σχήμα 4.24, συμπληρώνοντας Όνομα Χρήστη, Κωδικό και Ε – mail, δημιουργείται ο Χρήστης στο σχετικό μητρώο του REST API, και τού παρέχεται ειδικό Subscription Key, το οποίο θα χρησιμοποιεί για τη ταυτοποίησή του κατά τα αιτήματα των υπηρεσιών της Διεπαφής. Το Subscription Key είναι η τιμή της πρώτης στήλης «Κωδικός API». Μετά και την εγγραφή, ο Χρήστης θα μπορεί να συνδεθεί στη Διεπαφή με τα στοιχεία του λογαριασμού του [30].

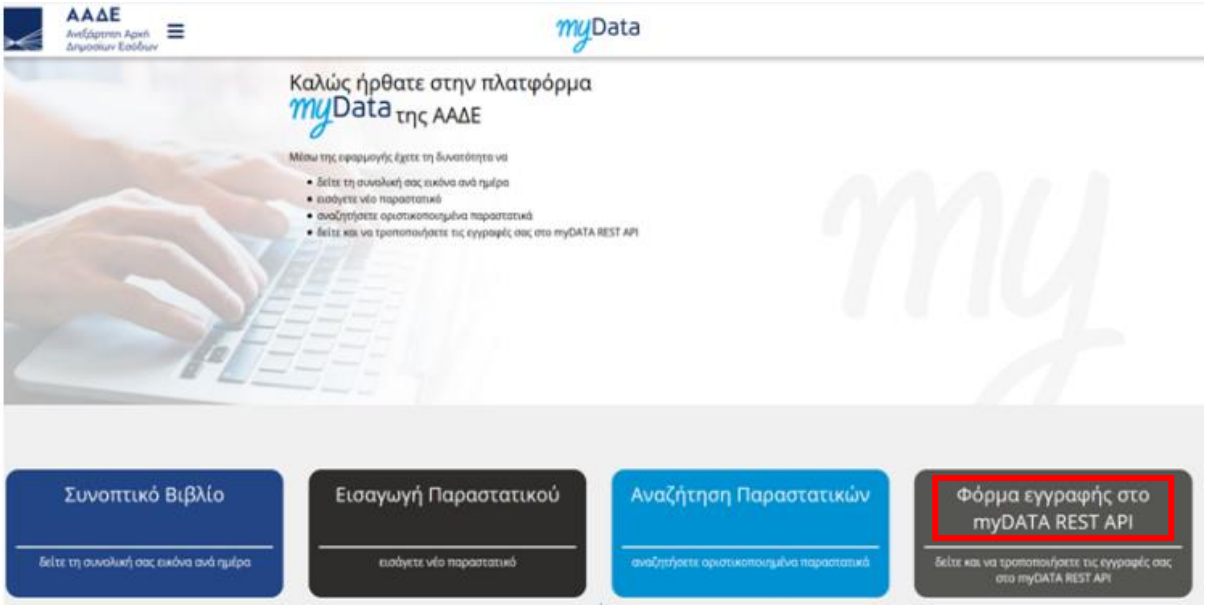
4.3.2.2 HTTPS Αιτήματα

Η χρήση κάθε λειτουργίας της Διεπαφής της Α.Α.Δ.Ε. πραγματοποιείται μέσω της αποστολής ενός HTTPS αιτήματος στον αντίστοιχο σύνδεσμο URL. Το αίτημα πρέπει να εμπεριέχει:

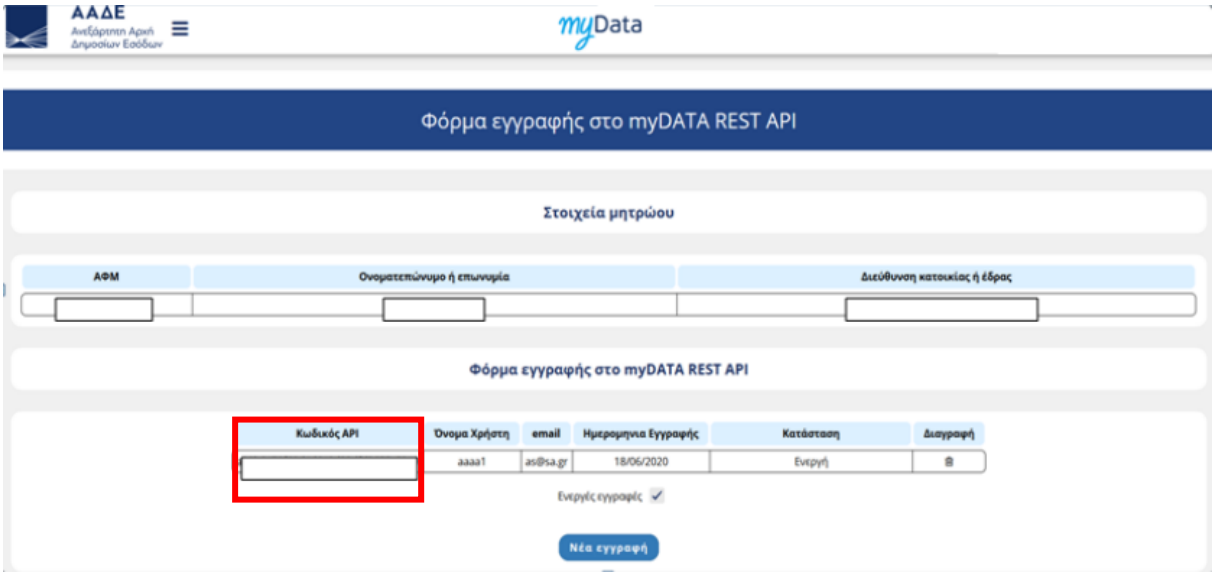
- τη κατάλληλη κεφαλίδα (header), η οποία θα περιέχει πληροφορίες απαραίτητες για τη ταυτοποίηση του Χρήστη, και

- ένα σώμα (body) σε XML μορφή, η δομή του οποίου θα εξαρτάται από τη λειτουργία που καλείται.

Για κάθε αίτημα, ο Χρήστης θα λαμβάνει μία απάντηση με πληροφορίες για την έκβαση του αιτήματός του σε XML μορφή.



Σχήμα 4.23: Επιλογή προς εγγραφή ενός Χρήστη στην ηλεκτρονική πλατφόρμα myData [30].



Σχήμα 4.24: Εγγραφή Χρήστη στο myData [30].

Κεφαλίδες (Headers)

Κάθε HTTPS αίτημα πρέπει να περιέχει – υπό τη μορφή ζευγαριών – τιμών – τις κεφαλίδες (headers) που φαίνονται στο Σχήμα 4.25, τα οποία είναι απαραίτητα για τη ταυτοποίηση του Χρήστη.

KEY	Data Type	VALUE	DESCRIPTION
aade-user-id	String	{Όνομα Χρήστη}	Το όνομα χρήστη του λογαριασμού
ocp-apim-subscription-key	String	{Subscription Key}	Το subscription key του χρήστη

Σχήμα 4.25: Χρήση των διαπιστευτηρίων του Χρήστη ως κεφαλίδες των HTTP αιτημάτων προς ταυτοποίηση [30].

Σε περίπτωση εισαγωγής λανθασμένων στοιχείων, ο Χρήστης θα λάβει σχετικό μήνυμα σφάλματος. Μέσα από τη ταυτοποίηση του Χρήστη μέσω των headers, η Διεπαφή θα αποκτά αυτομάτως πρόσβαση και στο Α.Φ.Μ. που είχε δηλώσει κατά την εγγραφή του, ούτως ώστε να μην είναι απαραίτητη η εισαγωγή του εν λόγω στοιχείου ξανά και ξανά σε κάθε αίτημα λειτουργίας [30].

Σώμα (Body)

Στις λειτουργίες και μεθόδους υποβολής, όπου το αίτημα είναι τύπου POST, ο Χρήστης μπορεί να στείλει ένα ή πολλά αντικείμενα, ενσωματώνοντάς τα στο σώμα του αιτήματος σε ειδική μορφή XML (παραστατικά/λογιστικές εγγραφές ή χαρακτηρισμούς). Έτσι, η απάντηση μπορεί να περιέχει – για κάθε παραστατικό – ένα ή περισσότερα μηνύματα σφάλματος ή ένα μήνυμα πετυχημένης υποβολής. Σε περίπτωση που ένα αντικείμενο υποβληθεί πάνω από μία φορά – έχοντας τα ίδια αναγνωριστικά στοιχεία με το προηγουμένως αποσταλμένο – διατηρείται το τελευταίο στη Βάση Δεδομένων των Ηλεκτρονικών Βιβλίων ως έγκυρο, και αυτομάτως το προηγούμενο ακυρώνεται [30].

Στις λειτουργίες και μεθόδους λήψης ή απλής ακύρωσης παραστατικού, όπου το αίτημα είναι τύπου GET, ο Χρήστης θα αποστέλλει ως παραμέτρους του αιτήματος τους μοναδικούς αριθμούς των παραστατικών που τον ενδιαφέρουν [30].

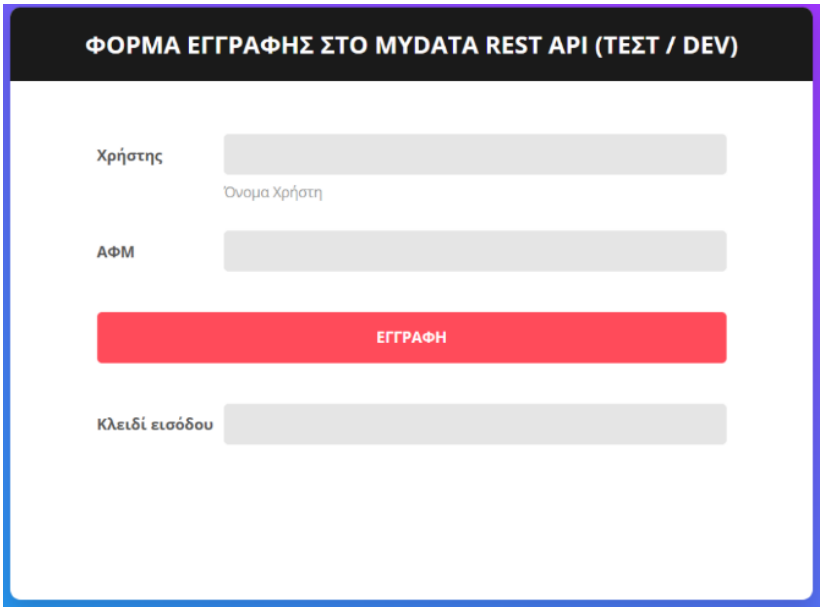
4.4 Χρήση του Δοκιμαστικού Περιβάλλοντος του myData

Για να μπορέσουμε να χρησιμοποιήσουμε το REST API του myData (my Digital Accounting and Tax Application), αρχικά θα πρέπει να δημιουργήσουμε έναν λογαριασμό στο δοκιμαστικό περιβάλλον που μάς παρέχεται. Η φόρμα εγγραφής στο δοκιμαστικό αυτό περιβάλλον, η οποία απεικονίζεται στο Σχήμα 4.26, βρίσκεται στην ηλεκτρονική διεύθυνση https://mydata-dev-register.azurewebsites.net/.

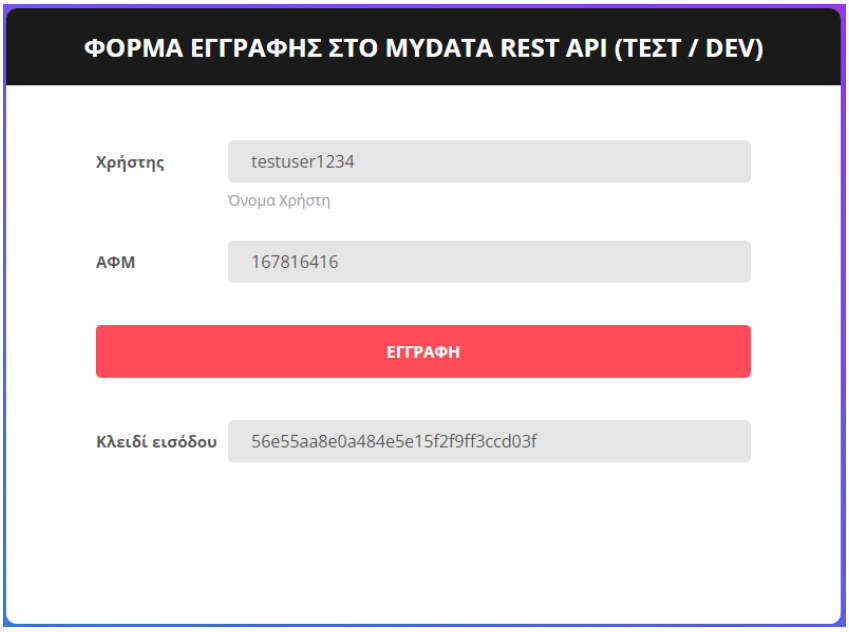
Εφόσον, λοιπόν, συμπληρώσουμε ένα Όνομα Χρήστη και το Α.Φ.Μ. μας, όπως στο Σχήμα 4.27, δημιουργείται ένα κλειδί εισόδου.

Για την λειτουργία της δικής μας Εφαρμογή Τιμολόγησης, θα χρησιμοποιήσουμε τις εξής – παρεχόμενες από το REST API της Α.Α.Δ.Ε. – μεθόδους:

- Τη **/SendInvoices** για τη δημιουργία Απόδειξης Λιανικής Πώλησης, Τιμολογίου Πώλησης και Τιμολογίου Παροχής Υπηρεσιών, και
- Τη **/CancelInvoice** για την ακύρωση παραστατικών [30].



Σχήμα 4.26: Φόρμα εγγραφής στο δοκιμαστικό περιβάλλον του myData [30].



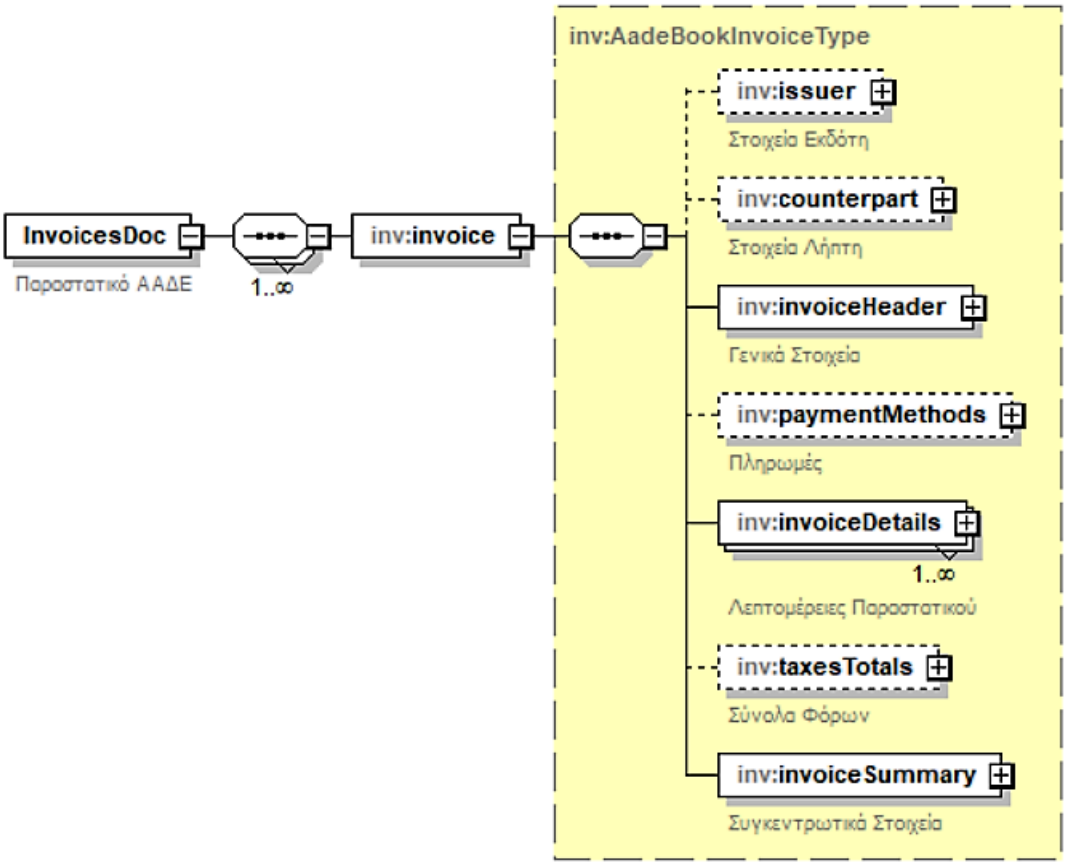
Σχήμα 4.27: Πραγματοποίηση εγγραφής στο δοκιμαστικό περιβάλλον του myData.

Μέθοδος /SendInvoices

Το HTTP αίτημα αυτής της μεθόδου είναι της τύπου POST. Στη κεφαλίδα (header) του αιτήματος περνάμε τα στοιχεία που δημιουργήσαμε με την εγγραφή μας στο δοκιμαστικό περιβάλλον.

Στο σώμα (body) του αιτήματος, το οποίο είναι σε μορφή XML, περνάμε τα στοιχεία του τιμολογίου. Το σώμα περιέχει το στοιχείο (element) InvoicesDoc, το οποίο περιλαμβάνει τα στοιχεία του παραστατικού, και περιέχει ένα ή περισσότερα invoice. Κάθε invoice συμβολίζει ένα ξεχωριστό

παραστατικό και μέσα σε αυτό περιέχονται όλες οι πληροφορίες του παραστατικού. Στο Σχήμα 4.28 φαίνεται η δομή του [30].



Σχήμα 4.28: Δομή του περιεχομένου του παραστατικού [30].

Πεδίο	**Περιγραφή**
issuer	Εκδότης Παραστατικού
counterpart	Λήπτης Παραστατικού
paymentMethods	Τρόποι Πληρωμής
invoiceHeader	Επικεφαλίδα Παραστατικού
invoiceDetails	Γραμμές Παραστατικού
invoiceSummary	Περίληψη Παραστατικού

Σχήμα 4.29: Πεδία παραστατικού [30].

Αναλυτικότερα, έχουμε τα πεδία:

- **invoiceHeader**: Περιέχει τη Σειρά Παραστατικού (series), τον Α/Α (aa), την Ημερομηνία (issueDate), τον Τύπο Παραστατικού (invoiceType) (Τιμή 1.1 για Τιμολόγιο Πώλησης, 2.1

για Τιμολόγιο Παροχής Υπηρεσιών και 11.1 για Απόδειξη Λιανικής Πώλησης) και το Νόμισμα (currency).

Αντικριζόμενα Παραστατικά Εκδότη ημεδαπής / αλλοδαπής	**Κωδικός**	**Περιγραφή**
Τιμολόγιο Πώλησης		
	1.1	Τιμολόγιο Πώλησης
Τιμολόγιο Παροχής Υπηρεσιών		
	2.1	Τιμολόγιο Παροχής
Μη Αντικριζόμενα Παραστατικά Εκδότη ημεδαπής / αλλοδαπής		
Παραστατικά Λιανικής		
	11.1	ΑΛΠ

Σχήμα 4.30: Χρησιμοποιούμενα είδη παραστατικών της Α.Α.Δ.Ε. [30].

- **issuer**: Περιέχει τις πληροφορίες του Εκδότη του εκάστοτε παραστατικού. Πιο συγκεκριμένα, περιέχει το Α.Φ.Μ. του (vatNumber), τη Χώρα (country) και τον Αριθμό Εγκατάστασης (branch).
- **counterpart**: Πρόκειται για τον Λήπτη του εκάστοτε παραστατικού. Του ατόμου, δηλαδή, στο όνομα του οποίου εκδίδεται το παραστατικό. Τα στοιχεία που στέλνουμε είναι το Α.Φ.Μ. του (vatNumber), η Χώρα (country), ο Αριθμός Εγκατάστασης (branch) και η Διεύθυνσή του (address), η οποία αποτελείται από την Οδό (street), τον Αριθμό (number), τον Ταχυδρομικό Κώδικα (postalCode) και τη Πόλη (city).

  Στη περίπτωση των Αποδείξεων Λιανικής Πώλησης, καθώς δεν επιλέγουμε κάποιον πελάτη, δε συμπεριλαμβάνουμε το πεδίο αυτό.

- **paymentMethods**: Περιέχει τον τρόπο πληρωμής. Πιο συγκεκριμένα, περιέχει τον Τύπο Πληρωμής (type) και το Ποσό Πληρωμής (amount).

Πεδίο	**Τύπος**	**Περιγραφή**	**Αποδεκτές τιμές**
type	xs:int	Τύπος Πληρωμής	Ελάχιστη τιμή = 1 Μέγιστη τιμή = 9
amount	xs:decimal	Ποσό Πληρωμής	Ελάχιστη τιμή = 0 Δεκαδικά ψηφία = 2

Σχήμα 4.31: Δομή πεδίου paymentMethods [30].

- **invoiceDetails**: Περιέχει τις πληροφορίες σχετικά με το προϊόν του εκάστοτε παραστατικού. Κάθε παραστατικό μπορεί να περιλαμβάνει περισσότερα από ένα προϊόντα. Το πεδίο αυτό περιέχει τον Α/Α (lineNumber) των προϊόντων, τη καθαρή αξία (netValue), τη κατηγορία Φ.Π.Α. στην οποία ανήκουν (vatCategory), το σύνολο του Φ.Π.Α. (vatAmount) και τους

Χαρακτηρισμούς Εσόδων (incomeClassification). Ο χαρακτηρισμός εσόδων περιέχει τα πεδία classificationType και classificationCategory, που είναι ο κωδικός χαρακτηρισμών και η κατηγορία χαρακτηρισμού αντίστοιχα και τη καθαρή αξία [30].

- **taxesTotals**: Περιέχει τους φόρους του εκάστοτε παραστατικού. Κάθε φόρος περιέχεται στο taxes, το οποίο διαθέτει τα πεδία Τύπος Φόρου (taxType), Κατηγορία Φόρου (taxCategory), Καθαρή Αξία (underlyingValue) και Αξία του Φ.Π.Α. (taxAmount).
- **invoiceSummary**: Περιέχει όλα τα σύνολα του παραστατικού και συγκεκριμένα τη Συνολική Καθαρή Αξία (totalNetValue), το Συνολικό Φ.Π.Α. (totalVatAmount), το Σύνολο Παρακρατήσεων Φόρων (totalWithheldValue), το Σύνολο Τελών (totalFeesAmount), το Σύνολο Χαρτοσήμου (totalStampDutyAmount), το Σύνολο Λοιπών Φόρων (totalOtherTaxesAmount), το Σύνολο Κρατήσεων (totalDeductionsAmount), τη Συνολική Αξία (totalGrossValue) και τους Χαρακτηρισμούς Εσόδων (incomeClassification).

Βάσει των παραπάνω, το σώμα του αιτήματος έχει τη μορφή που απεικονίζεται στα Σχήματα 4.32 και 4.33.

```
<?xml version="1.0" encoding="utf-8"?>
<InvoicesDoc
    xmlns="http://www.aade.gr/myDATA/invoice/v1.0"
    xmlns:N1="https://www.aade.gr/myDATA/incomeClassificaton/v1.0"
    xmlns:N2="https://www.aade.gr/myDATA/expensesClassificaton/v1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.aade.gr/myDATA/invoice/v1.0 schema.xsd">
    <invoice>
        <issuer>
            <vatNumber>167816416</vatNumber>
            <country>GR</country>
            <branch>1</branch>
        </issuer>
        <counterpart>
            <vatNumber>988047780</vatNumber>
            <country>GR</country>
            <branch>0</branch>
            <address>
                <street>Δοκιμη 1</street>
                <number>10</number>
                <postalCode>54645</postalCode>
                <city>Θεσσαλονικη</city>
            </address>
        </counterpart>
        <invoiceHeader>
            <series>4</series>
            <aa>7</aa>
            <issueDate>2023-07-13</issueDate>
            <invoiceType>1.1</invoiceType>
            <currency>EUR</currency>
        </invoiceHeader>
        <paymentMethods>
            <paymentMethodDetails>
                <type>3</type>
                <amount>0.88</amount>
            </paymentMethodDetails>
        </paymentMethods>
        <invoiceDetails>
            <lineNumber>1</lineNumber>
            <netValue>0.81</netValue>
            <vatCategory>1</vatCategory>
            <vatAmount>0.19</vatAmount>
            <incomeClassification>
                <N1:classificationType>E3_561_001</N1:classificationType>
                <N1:classificationCategory>category1_3</N1:classificationCategory>
                <N1:amount>0.81</N1:amount>
            </incomeClassification>
        </invoiceDetails>
```

Σχήμα 4.32: Σώμα του αιτήματος της μεθόδου /SendInvoice (1).

```
        <taxesTotals>
            <taxes>
                <taxType>1</taxType>
                <taxCategory>1</taxCategory>
                <underlyingValue>0.81</underlyingValue>
                <taxAmount>0.12</taxAmount>
            </taxes>
        </taxesTotals>
        <invoiceSummary>
            <totalNetValue>0.81</totalNetValue>
            <totalVatAmount>0.19</totalVatAmount>
            <totalWithheldAmount>0.12</totalWithheldAmount>
            <totalFeesAmount>0.00</totalFeesAmount>
            <totalStampDutyAmount>0.00</totalStampDutyAmount>
            <totalOtherTaxesAmount>0.00</totalOtherTaxesAmount>
            <totalDeductionsAmount>0</totalDeductionsAmount>
            <totalGrossValue>0.88</totalGrossValue>
            <incomeClassification>
                <N1:classificationType>E3_561_001</N1:classificationType>
                <N1:classificationCategory>category1_3</N1:classificationCategory>
                <N1:amount>0.81</N1:amount>
            </incomeClassification>
        </invoiceSummary>
    </invoice>
</InvoicesDoc>
```

Σχήμα 4.33: Σώμα του αιτήματος της μεθόδου /SendInvoice (2).

Μέθοδος /CancelInvoice

Η μέθοδος αυτή χρησιμοποιείται για την ακύρωση παραστατικού χωρίς την επαναϋποβολή καινούριου. Ο χρήστης τη καλεί υποβάλλοντας ως παράμετρο μόνο το mark του παραστατικού, δηλαδή τον Μοναδικό Αριθμό Καταχώρησης, το οποίο και θέλει να ακυρώσει. Δεν απαιτείται αποστολή xml body [30].

Κεφάλαιο 5ο: Συμπεράσματα και Πρόταση Βελτίωσης

Η δημιουργία μίας διαδικτυακής εφαρμογής, χρησιμοποιώντας τη Βιβλιοθήκη React και τη Πλατφόρμα Node της Javascript, ενώ ίσως φαίνεται δύσκολη και ιδιαιτέρως περίπλοκη, μπορεί να οδηγήσει σε ένα εξαιρετικά ικανοποιητικό αποτέλεσμα. Η ανάπτυξη των Front – End και Back – End μερών της απαιτεί ιδιαίτερη προσοχή για την ορθή και επιθυμητή τελικά λειτουργία της. Από τη στιγμή που και τα δύο προαναφερθέντα εργαλεία βασίζονται στην ίδια γλώσσα προγραμματισμού, περιορίζεται η περιπλοκότητα δημιουργίας της εφαρμογής μας.

Η Βιβλιοθήκη React, προσφέροντας πληθώρα στοιχείων, καθώς και τη δυνατότητα χρήσης επαναχρησιμοποιούμενων στοιχείων, μάς διευκολύνει στη μαζική διαχείριση και εύκολή τροποποίησή τους. Η Πλατφόρμα Node, λαμβάνοντας όλα τα δεδομένα της εφαρμογής μας, καταφέρνει γρήγορα και τα λαμβάνει, να τα διαχειρίζεται, να τα επεξεργάζεται και να τα ανανεώνει.

Θέτοντας από την αρχή τους επιθυμητούς στόχους και τις επιθυμητές λειτουργίες που θέλουμε να έχει μία διαδικτυακή εφαρμογή που θα δημιουργήσουμε, μπορούμε να επιτύχουμε το μέγιστο δυνατό αποτέλεσμα, με σεβασμό στον τελικό χρήστη και στην ευκολία του, την ευχάριστη περιήγησή του και διευκόλυνση της καθημερινότητάς του.

Μία βασική και ουσιώδης βελτίωση που προτείνονται για τη διαδικτυακή εφαρμογή ηλεκτρονικής τιμολόγησης που έχουμε δημιουργήσει, είναι η περαιτέρω ανάπτυξή της για «εγκαθίδρυση» χρήσης της από επιχειρήσεις. Να χρησιμοποιείται, δηλαδή, από περισσότερους και ξεχωριστούς εργαζόμενους – χρήστες μίας επιχείρησης για τη δημιουργία και διαχείριση προσωπικών πελατολογίων και την έκδοση βαρυσήμαντων – για την εύρυθμη λειτουργία της επιχείρησης – παραστατικών.

Επίσης, για τη χρήση της εφαρμογής μας και από ανεξάρτητους και ελεύθερους επαγγελματίες, προτείνεται η δημιουργία και ύπαρξη καταβολής μίας μηνιαίας ή ετήσιας συνδρομής. Να έχει, δηλαδή, τη δυνατότητα ένας επαγγελματίας να δημιουργήσει έναν δικό του μεμονωμένο και προσωπικό λογαριασμό, ώστε να διαχειρίζεται ο ίδιος τις όποιες διαδικασίες συμβάλλουν στην εργασία του.

Βιβλιογραφία

Βιβλία

[32] ΙΝΣΤΙΤΟΥΤΟ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΕΚΔΟΣΕΩΝ «ΔΙΟΦΑΝΤΟΣ», *Εφαρμογές Πληροφορικής*.

Internet Site

[1] Prosvasis Go, «Τι είναι η Ηλεκτρονική Τιμολόγηση». [Διαδικτυακό]. Διαθέσιμο: https://go.prosvasis.com/%CF%84%CE%B9-%CE%B5%CE%AF%CE%BD%CE%B1%CE%B9-%CE%B7%CE%B7%CE%BB%CE%B5%CE%BA%CF%84%CF%81%CE%BF%CE%BD%CE%B9%CE%BA%CE%AE%CF%84%CE%B9%CE%BC%CE%BF%CE%BB%CF%8C%CE%B3%CE%B7%CF%83%CE%B7/.

[2] Η ΚΑΘΗΜΕΡΙΝΗ, «Κίνητρα για ηλεκτρονική τιμολόγηση», 2023. [Διαδικτυακό]. Διαθέσιμο: https://www.kathimerini.gr/economy/562328809/kinitra-gia-ilektroniki-timologisi/.

[3] Upwork, "What Is Web Application Development and How Do I Get Started?", 2021. [Online]. Available: https://www.upwork.com/resources/what-is-web-application-development.

[4] Computer Science, "Front-End vs. Back-End: What's the Difference?", 2023. [Online]. Available: https://www.computerscience.org/bootcamps/resources/frontend-vs-backend/.

[6] JavaTPoint, "Database". [Online]. Available: https://www.javatpoint.com/what-is-database.

[7] Βικιπαίδεια, «Βάση δεδομένων», 2022. [Διαδικτυακό]. Διαθέσιμο: https://el.wikipedia.org/wiki/%CE%92%CE%AC%CF%83%CE%B7_%CE%B4%CE%B5%CE%B4%CE%BF%CE%BC%CE%AD%CE%BD%CF%89%CE%BD.

[8] AppMaster, "SQL vs NoSQL Databases: What's the Difference?", 2022. [Online]. Available: https://appmaster.io/blog/sql-vs-nosql-databases.

[9] IP.GR, «Τι είναι API – Application Programming Interface». [Διαδικτυακό]. Διαθέσιμο: https://www.ip.gr/el/dictionary/378API___Application_Programming_Interface?gclid=CjwKCAjw36GjBhAkEiwAKwIWySBzaZ3ZyB2uwb37Y9VvGt_CVarR8bu8TjHH9OZrCFsRSMkaxvrRoCD94QAvD_BwE.

[10] OneLogin, "Authentication vs. Authorization". [Online]. Available: https://www.onelogin.com/learn/authentication-vsauthorization#:~:text=Authentication%20and%20authorization%20are%20two,authorization%20determines%20their%20access%20rights.

[11] ClickIT, "Web Application Architecture: The Latest Guide 2022". [Online]. Available: https://www.clickittech.com/devops/web-application-architecture/amp/.

[12] Digiteum, "Modern Web App Architectures: How to Choose the Right One for Your Project", 2023. [Online]. Available: https://www.digiteum.com/web-application-architecture/.

[13] Big Blue Data Academy, "Τι Είναι το API και Πώς Λειτουργεί; (Οδηγός 2023)", 2022. [Διαδικτυακό]. Διαθέσιμο: https://bigblue.academy/gr/ti-einai-to-api.

[14] Angular, "What is Angular?". [Online]. Available: https://angular.io/guide/what-is-angular.

[15] After Academy, "MVC Architecture in Web Applications", 2020. [Online]. Available: https://afteracademy.com/blog/mvc-architecture-in-web-applications/.

[16] CIPHER TRICK, "Core Features of AngularJS", 2023. [Online]. Available: https://ciphertrick.com/core-features-of-angularjs/.

[17] Wikimedia Commons, "File:AngularJS logo.svg", 2023. [Online]. Available: https://commons.wikimedia.org/wiki/File:AngularJS_logo.svg.

[18] Tutorials Teacher, "What is AngularJS?". [Online]. Available: https://www.tutorialsteacher.com/angularjs/what-is-angularjs.

[19] GeeksforGeeks, "AngularJS Tutorial", 2023. [Online]. Available: https://www.geeksforgeeks.org/angularjs/.

[20] Wikipedia, "React (software)", 2023. [Online]. Available: https://en.wikipedia.org/wiki/React_%28software%29.

[21] GeeksforGeeks, "ReactJS Tutorials", 2023. [Online]. Available: https://www.geeksforgeeks.org/reactjs-tutorials/.

[22] GeeksforGeeks, "ReactJS Introduction", 2023. [Online]. Available: https://www.geeksforgeeks.org/reactjs-introduction/?ref=lbp.

[23] GeeksforGeeks, "What are the features of ReactJS?", 2023. [Online]. Available: https://www.geeksforgeeks.org/what-are-the-features-of-reactjs/.

[24] JavaTPoint, "Pros and Cons of ReactJS". [Online]. Available: https://www.javatpoint.com/pros-and-cons-of-react.

[25] JavaTPoint, "Vue.js Tutorial". [Online]. Available: https://www.javatpoint.com/vue-js.

[26] Rootstack, "5 global companies are using Vue.js in 2023 and have boosted its performance". [Online]. Available: https://rootstack.com/en/blog/5-global-companies-are-using-vuejs-2023-and-have-boosted-its-performance.

[27] Wikipedia, "Vue.js", 2023. [Online]. Available: https://en.wikipedia.org/wiki/Vue.js.

[28] SPACE TECHNOLOGIES, "What is Vue.js? Vue.js Advantages And Disadvantages", [Online]. 2023. Available: https://www.spaceo.ca/blog/vue-js-pros-and-cons/.

[29] GeeksforGeeks, "Top 10 Backend Technologies You Must Know", 2023. [Online]. Available: https://www.geeksforgeeks.org/backend-technologies/.

[30] ΑΑΔΕ, «myData – Ηλεκτρονικά Βιβλία ΑΑΔΕ – Τεχνική περιγραφή διεπαφής REST API για διαβίβαση & λήψη δεδομένων», 2020. [Διαδικτυακό]. Διαθέσιμο: https://www.aade.gr/sites/default/files/2020-02/myDATA%20API%20Documentation%20v0.6.pdf.

[31] GeeksforGeeks, "Why Use Node.js For Backend Development?", 2023. [Online]. Available: https://www.geeksforgeeks.org/why-to-use-node-js-for-backend-development/.

[33] W3Schools, "HTML Introduction". [Online]. Available: https://www.w3schools.com/html/html_intro.asp.

[34] GeeksforGeeks, "Advantages and Disadvantages of HTML", 2022. [Online]. Available: https://www.geeksforgeeks.org/advantages-and-disadvanatges-of-html/.

[35] Wikipedia, “CSS”, 2023. [Online]. Available: https://en.wikipedia.org/wiki/CSS.

[36] ΚΑΛΛΙΠΟΣ, «ΕΚΠΑΙΔΕΥΤΙΚΑ ΠΕΡΙΒΑΛΛΟΝΤΑ ΔΙΑΔΙΚΤΥΟΥ», Τσιάτσος Θρασύβουλος Κωνσταντίνος, 2015. [Διαδικτυακό]. Διαθέσιμο: https://repository.kallipos.gr/handle/11419/3200.

[37] Tutorials Point, “What is CSS?”. [Online]. Available: https://www.tutorialspoint.com/css/what_is_css.htm.

[38] W3Schools, “CSS Website Layout”. [Online]. Available: https://www.w3schools.com/css/css_website_layout.asp.

[39] Big Blue DATA ACADEMY, «Τι Είναι η Javascript και Πού Χρησιμοποιείται;», Απρίλιος 2023. [Διαδικτυακό]. Διαθέσιμο: https://bigblue.academy/gr/javascript.

[40] IBM, “What is an API?”. [Online]. Available: https://www.ibm.com/topics/api.

[41] Contentful, “What is an API? How APIs work, simply explained”. [Online]. Available: https://www.contentful.com/api/.

[42] Βικιπαίδεια, “Nodejs”, 2020. [Διαδικτυακό]. Διαθέσιμο: https://el.wikipedia.org/wiki/Nodejs.

[43] GeeksforGeeks, “Why to Use Node.js For Backend Development?”, 2023. [Online]. Available: https://www.geeksforgeeks.org/why-to-use-node-js-for-backend-development/.

[44] HubSpot, “An Intro to Java for Back-End Programming”, 2023. https://blog.hubspot.com/website/javascript-for-backend.

[45] Squareboat Solutions, “Advantages and Disadvantages of Python – Make a Favorable Decision”, 2023. [Online]. Available: https://squareboat.com/blog/advantages-and-disadvantages-of-python.

[46] Astera Software, “Understanding the Basics of REST APIs”, June 2023. [Online]. Available: https://www.astera.com/type/blog/rest-api-definition/#How-Does-a-REST-API-work?.

[47] JavaTPoint, “Introduction of SOAP and REST Web Services”. [Online]. Available: https://www.javatpoint.com/soap-and-rest-web-services.

[48] GeeksforGeeks, “Remote Procedure Call (RPC) in Operating System”, April 2023. [Online]. Available: https://www.geeksforgeeks.org/remote-procedure-call-rpc-in-operating-system/.

Paper

[5] Rishi Vyas, “Comparative Analysis on Front-End Frameworks for Web Applications,”, July 2022.