Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης
Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Χρήση σύγχρονων δομών ανάπτυξης διαδικτυακών εφαρμογών για την υλοποίηση πλατφόρμας κοινωνικής δικτύωσης

Εκπονητές Φοιτητές:
Θεοδωρακόπουλος Ιωάννης – Αρ. Μητρώου: 3566
Τζιούτζικ Άρνολντ Ιωσήφ – Αρ. Μητρώου: 3573

Επιβλέπων Καθηγητής:
Παπαδάκης Νικόλαος

Ηράκλειο
2019

Αφιερώνω την παρούσα πτυχιακή εργασία στην μνήμη της γιαγιάς μου, Jadwiga Sala, και της θείας μου, Katarzyna Dziudzik, τις οποίες αποχωρίστηκα πρόωρα χωρίς να προλάβω να μοιραστώ μαζί τους την χαρά για το επίτευγμά αυτό. Τους είμαι παντοτινά ευγνώμων που μου έμαθαν με τον τρόπο τους πως οι πιο σκοτεινές νύχτες φέρνουν τις λαμπρότερες αυγές και πως πάντα αξίζει να μάχεσαι για τον εαυτό σου.

Arnold Iosif Dziudzik –

Αφιερώνω την παρούσα πτυχιακή εργασία στη μνήμη της γιαγιάς μου Παναγιώτας και του παππού μου Ηλία. Τους ευχαριστώ θερμά για την αγάπη και τη στήριξη που μου πρόσφεραν ανιδιοτελώς και θα τους είμαι για πάντα ευγνώμων.

Ιωάννης Θεοδωρακόπουλος –

Ευχαριστίες

Αρχικά, θα θέλαμε να ευχαριστήσουμε εγκάρδια τον Επιβλέπων Καθηγητή μας, Παπαδάκη Νικόλαο, για την εμπιστοσύνη που μας έδειξε αναθέτοντάς μας αυτήν την πτυχιακή εργασία, επιτρέποντάς μας συγχρόνως να ασχοληθούμε με ένα θέμα που είναι τόσο κοντινό στις φιλοδοξίες και τα όνειρά μας.

Επίσης, ευχαριστούμε το Ακαδημαϊκό, Διδακτικό και Εργαστηριακό Προσωπικό του Τμήματός μας, για όλες τις γνώσεις και την καθοδήγηση που μας προσέφεραν κατά την διάρκεια των σπουδών μας. Επιπλέον, ευχαριστούμε όλους του συναδέλφους που βάδισαν μαζί μας αυτό το μονοπάτι, για την υποστήριξη και την ευγένεια που μας προσέφεραν· εν μέρει, μας ενέπνευσαν για την ανάπτυξη της ιδέας πίσω από την παρούσα πτυχιακή εργασία.

Τέλος, αλλά εξίσου σημαντικά, ευχαριστούμε τις οικογένειές μας – τους γονείς, τα αδέρφια, τις συντρόφους και τους κοντινούς φίλους μας για την αδιαμφισβήτητη και ακατάπαυστη στήριξη, υπομονή, εμψύχωση και πίστη τους σε εμάς καθ’ όλη την διάρκεια του άθλου μας.

Abstract

Usage of modern web application development frameworks for the implementation of a social media platform.

This undergraduate thesis analyses the implementation of a web-based social media platform with the use of modern, full-stack web application development frameworks (React.js, Node.js, Express, PostgreSQL). This platform is aimed at specific users, namely students of tertiary educational institutions, allowing them to interconnect for the purpose of group studying, according to the courses the have signed up for at any given time, as well as for the purpose of finding other individuals to help them in better understanding the course's material, based on their success in that endeavor at the given subject.

Η παρούσα πτυχιακή εργασία αναλύει την υλοποίηση μια διαδικτυακής πλατφόρμας κοινωνικής δικτύωσης με την χρήση σύγχρονων δομών ανάπτυξης διαδικτυακών εφαρμογών που σε εύρος Full Stack (React.js, Node.js, Express.js, PostgreSQL). Η πλατφόρμα αυτή θα αποσκοπεί σε στοχευμένους χρήστες, ήτοι φοιτητές εντός του πλαισίου της Τριτοβάθμιας Εκπαίδευσης, και θα εξυπηρετεί την διασύνδεσή τους για τη κοινή μελέτη μαθημάτων που παρακολουθούν και την εύρεση ατόμων πρόθυμων να βοηθήσουν στην επίλυση αποριών, με βάση το ιστορικό επιτυχίας τους στο μάθημα υπό παρακολούθηση.

Πίνακας περιεχομένων

Λίστα Πινάκων

Πίνακας Εικόνων

1. Εισαγωγή

Η συγγραφή της πτυχιακής εργασίας αποτελεί μια έξοχη ευκαιρία για την συνάθροιση όλων των γνώσεων που έχει αποκτήσει ένας/μια προπτυχιακός/η φοιτητής/ρια για την διεκπεραίωση ενός έργου, το οποίο επιτρέπει την ανάπτυξη ικανοτήτων και επίγνωσης που θα παραμείνουν χρήσιμες στη μελλοντική σταδιοδρομία του/της, ανεξαρτήτως του πεδίου της ειδικότητάς του/της, όπως και την ικανοποίηση της πνευματικής περιέργειάς του/της. Αυτή η σχολαστική, πολυεπίπεδη διαδικασία, περιλαμβάνει την διαχείριση του προαναφερθέντος έργου, την επίλυση προβλημάτων που προκύπτουν στην πορεία της υλοποίησής του, την ομαδική συνεργασία και την ορθή επικοινωνία ιδεών, καθώς και την ανάπτυξη σχέσεων με τον/την επιβλέπων/ουσα καθηγητή/ρια και μια πρώτη επαφή με την ερευνητική διαδικασία. Η ολοκλήρωσή μιας τέτοιας εργασίας είναι μια ανταποδοτική εμπειρία που επιδεικνύει τον/την φοιτητή/φοιτήτρια ως έναν άνθρωπο ικανό στην έρευνα, την συγγραφή, την αναλυτική και κριτική σκέψη και, ως πολύ σημαντικό στον μοντέρνο χώρο της Πληροφορικής, την συνεχή μάθηση και βελτίωση πρακτικών, καθώς και στην καταβολή του απαιτούμενου σχεδιασμού, προσήλωσης και σκληρής δουλειάς.

1.1 Αντικείμενο Εργασίας

Το αντικείμενο αυτής της πτυχιακής εργασίας είναι η ανάπτυξη μιας πλατφόρμας κοινωνικής δικτύωσης, με την χρήση των πλέον μοντέρνων δομών ανάπτυξης. Ως το κεντρικό σημείο του αντικειμένου της εργασίας, τα μέσα κοινωνικής δικτύωσης αποτελούν διαδραστικές τεχνολογίες, οι οποίες, με την μεσολάβηση υπολογιστών διευκολύνουν τη δημιουργία και διαμοιρασμό ιδεών, ενδιαφερόντων σταδιοδρομίας και άλλων μορφών έκφρασης μέσω εικονικών κοινοτήτων και δικτύων. [1] Η ποικιλία των αυτόνομων και ενσωματωμένων υπηρεσιών κοινωνικών δικτύων που διατίθενται σήμερα εισάγουν προκλήσεις στον ακριβή ορισμό τους. Ωστόσο, υπάρχουν ορισμένα κοινά χαρακτηριστικά, όπως το γεγονός πως τα μέσα κοινωνικής δικτύωσης είναι διαδραστικές διαδικτυακές εφαρμογές, το περιεχόμενό τους παράγεται από τους χρήστες - όπως μέσω αναρτήσεων (posts) και σχολίων (comments) – οι χρήστες δημιουργούν προσωπικά προφίλ για την αναγνώρισή τους, και αναπτύσσονται μεταξύ χρηστών ή ομάδων χρηστών κοινωνικά δίκτυα. [2] [3] [4] Έρευνες δείχνουν πως ένας μέσος χρήστης του Διαδικτύου διαθέτει το 22% του χρόνου του σε αυτό στην χρήση μέσων κοινωνικής δικτύωσης. [5] Κατά κόρον, τα μέσα κοινωνικής δικτύωσης χρησιμοποιούνται από τους χρήστες τους για την αρχειοθέτηση αναμνήσεων και εμπειριών, την μάθηση και την εξερεύνηση πάνω σε διάφορα αντικείμενα, την διαφήμιση, την δημιουργία φιλιών, καθώς και την ανάπτυξη ιδεών μέσω της δημιουργίας blogs, podcasts και βίντεο. [6] Οι δικτυωμένοι χρήστες συχνά δημιουργούν, επεξεργάζονται και διαχειρίζονται περιεχόμενο σε συνεργασία με άλλα άτομα στο Διαδίκτυο, γεγονός που υποδεικνύει μια συρροή των πλατφόρμων κοινωνικής δικτύωσης με τις πλατφόρμες συνεργασίας και παραγωγικότητας, μια πτυχή με την οποία πειραματίζεται και αυτή η πτυχιακή εργασία. Σύμφωνα με το διαδικτυακή πύλη στατιστικών Statista, το 2019, μερικά από τα πλέον δημοφιλή μέσα κοινωνικής δικτύωσης είναι η υπηρεσία κοινωνικής δικτύωσης - **Facebook**, η πλατφόρμα διαμοιρασμού βίντεο - **YouTube**, η υπηρεσία κοινωνικής δικτύωσης μέσω διαμοιρασμού εικόνων και βίντεο - **Instagram**, η υπηρεσία microblogging - **Twitter**, η ιστοσελίδα συσσωμάτωσης κοινωνικών νέων, διατίμησης διαδικτυακού περιεχομένου - **Reddit** και ο ιστοχώρος επαγγελματικής δικτύωσης, με πυρήνα την διευκόλυνση της πρόσληψης εργαζομένων και εύρεσης εργασίας - **LinkedIn**. [7]

Εξίσου εστιακό σημείο της παρούσας πτυχιακής εργασίας είναι η ανάπτυξη εφαρμογών διαδικτύου (web development). Ως **web development**, χαρακτηρίζεται το έργο που συνδέεται

με την ανάπτυξη ιστοσελίδων στο Διαδίκτυο, οι οποίες κυμαίνονται από απλές, στατικές σελίδες κειμένου μέχρι πολυσύνθετες διαδικτυακές εφαρμογές (web apps). Η μοντέρνες διαδικτυακές εφαρμογές συχνά απαρτίζονται από τρία ή περισσότερα επίπεδα:

1. το **front-end**, ήτοι το περιεχόμενο που αποδίδει ο φυλλομετρητής, το οποίο μπορεί να είναι στατικό ή δυναμικά παραγόμενο, και αφορά τεχνολογίες όπως η γλώσσα σήμανσης - **HTML**, η γλώσσα φύλλων ύψους – **CSS**, η διερμηνευμένη γλώσσα προγραμματισμού – **JavaScript** και μια από τις βιβλιοθήκες της JavaScript – **React.js**.
2. ένα μεσαίο εξυπηρετητή εφαρμογών για την παραγωγή και επεξεργασία δυναμικού περιεχομένου που αφορά τεχνολογίες όπως η πλατφόρμα ανάπτυξης λογισμικού - **Node.js**.
3. το **back-end**, που περιλαμβάνει την βάση δεδομένων και το σύστημα διαχείρισής της (DBMS). Ένα παράδειγμα σχεσιακού συστήματος διαχείρισης μιας βάσης δεδομένων είναι η **PostgreSQL**, γνωστή και ως **Postgres**.

Πολύ συχνά, τα επίπεδα 2 και 3 ορίζονται μαζί ως back-end, καθώς αποτελούν την server-side πλευρά της αναπτυσσόμενης εφαρμογής, σε αντίθεση με το front-end που αφορά την client-side πλευρά της. Ο προγραμματιστής που ενασχολείται με το αντικείμενο της ανάπτυξης εφαρμογών διαδικτύου μπορεί να ειδικεύεται σε ένα ή περισσότερα από αυτά τα επίπεδα και, συχνά, σημαντικός παράγοντας για τον ορισμό της έκτασης του έργου του είναι το μέγεθος της εφαρμογής: στην σύγχρονη βιομηχανία, όσο περισσότερα είναι τα χαρακτηριστικά και οι απαιτήσεις μια εφαρμογής, τόσο περισσότεροι και εξειδικευμένοι είναι οι προγραμματιστές που συμβάλουν στην δημιουργία και συντήρησή της.

1.2 Περίληψη Εργασίας

Ο σκοπός της παρούσας πτυχιακής εργασίας είναι, η διεξοδική μελέτη των σύγχρονων δομών ανάπτυξης διαδικτυακών εφαρμογών και ροών εργασίας που επικρατούν στον μοντέρνο τεχνολογικό κλάδο, κυρίως όσον αφορά τις πρακτικές και φιλοσοφίες που κυριαρχούν στην πρώτη γραμμή της προόδου του, με σκοπό τον σχεδιασμό και την υλοποίηση μιας πλατφόρμας κοινωνικής δικτύωσης. Η πλατφόρμα αυτή αποσκοπεί σε στοχευμένους χρήστες, ήτοι φοιτητές εντός του πλαισίου της Τριτοβάθμιας Εκπαίδευσης, προσφέροντάς τους την δυνατότητα αλληλεπίδρασης σε ένα περιβάλλον αφιερωμένο στην διευκόλυνση της δημιουργίας και εύρεσης ομάδων για κοινή μελέτη και διεκπεραίωση ομαδικών εργασιών, εντός μιας κοινότητας που χαρακτηρίζεται από την κοινή προσπάθεια για την επιτυχή ολοκλήρωση των σπουδών τους.

Η εφαρμογή επιτρέπει στους ενδιαφερόμενους χρήστες να εγγραφούν στην υπηρεσίας αυτή (signup), με την χρήση βασικών προσωπικών στοιχείων, έμφυτο χαρακτηριστικό της κοινωνικής δικτύωσης, και, στη συνέχεια, την εγγραφή (subscription) σε μαθήματα που παρακολουθούν επί του παρόντος. Εντός της σελίδας του κάθε μαθήματος, μπορούν να εισάγουν ή να συμμετάσχουν σε προ υπάρχουσες συζητήσεις στα πλαίσια του εν λόγω μαθήματος μαζί με άλλους χρήστες εγγεγραμμένους σε αυτό. Επίσης, οι χρήστες μπορούν να δημιουργήσουν δικές τους ομάδες, μαζί με άλλους χρήστες, είτε με σκοπό την οργάνωση ομαδικής μελέτης πάνω στην ύλη του μαθήματος, είτε για την εκπόνηση τυχουσών ομαδικών εργασιών.

Οι χρήστες έχουν πρόσβαση σε ένα feed, το οποίο θα εμφανίζει posts από τους χώρους συζητήσεων στα μαθήματα και στις ομάδες στις οποίες συμμετέχουν, οπότε οι χρήστες διαθέτουν όλη την επιμέλεια για το περιεχόμενο το οποίο τους παρουσιάζεται, εξυπηρετώντας έτσι το κίνητρο παροχής ενός περιβάλλοντος όπου οι χρήστες μπορούν να εστιάσουν την προσοχή τους, δίχως να αποσπώνται από άσχετο και μη θεμιτό περιεχόμενο.

Επίσης, οι χρήστες που έχουν την φυσική κλίση προς υποστήριξη των συναδέλφων τους, μπορούν να εκφράζουν το ενδιαφέρον τους για την παροχή βοήθειας μέσω της λύσης απορίων.

1.3 Κίνητρα και Στόχοι Εργασίας

Το βασικό κίνητρο αυτής της πτυχιακής εργασίες είναι η μελέτη και η αναπαράσταση της διαδικασίας ανάπτυξης μια μοντέρνας διαδικτυακής εφαρμογής, ακολουθώντας σύγχρονες μεθοδολογίες και χρησιμοποιώντας τα καταλληλότερα από τα δημοφιλέστερα σήμερα εργαλεία, βιβλιοθήκες και δομές ανάπτυξης.

Η παραπάνω συλλογή, όμως, δεν θα αποσκοπούσε πουθενά, αν δεν υπήρχε κάποια συγκεκριμένη ιδέα προς υλοποίηση, αλλά και πρόβλημα προς επίλυση. Ως Μηχανικοί, σκοπός μας είναι να σχεδιάζουμε, να αναλύουμε, να χτίζουμε και να δοκιμάζουμε περίπλοκα συστήματα που εξυπηρετούν τις πρακτικές ανάγκες των συνανθρώπων και τις κοινότητάς μας και να επιλύουμε προβλήματα για την προώθηση της τεχνικής γνώσης και ικανότητας του τομέα ενασχόλησής μας αλλά και για την βελτίωση της ανθρώπινης ευημερίας.

Από προσωπική εμπειρία, έχει παρατηρηθεί πως πολλοί συνάδελφοι στο Τμήμα μας αντιμετωπίζουν μια σειρά από δυσκολίες στην επιτυχή ολοκλήρωση των σπουδών τους, ειδικά στην κάλυψη των απαιτήσεων για αρκετά από τα μαθήματα του προγράμματος σπουδών. Αν και η χαμηλή απόδοση μπορεί να αποδοθεί σε μια πληθώρα αιτιών, όπως ιατρικοί ή ψυχολογικοί λόγοι, το οικογενειακό περιβάλλον, η κατάσταση του εκπαιδευτικού συστήματος, η πίεση και το άγχος, η έλλειψη κινήτρου για την επιτυχία, ακόμα και η στάση του φοιτητή/τριας προς τις σπουδές [8], εμπειρικά, έχει παρατηρηθεί η έλλειψη της αίσθησης κοινότητας μεταξύ των φοιτητών του Τμήματος, δηλαδή μιας κοινής ακαδημαϊκής κουλτούρας, ενός κοινού συστήματος αξιών και συμπεριφορών που χαρακτηρίζουν την στάση της πλειοψηφίας των μελών του Ιδρύματος. [9]

Μια γιγάντια τάση της εποχής μας είναι η εξάπλωση της χρήσης των μέσων κοινωνικής δικτύωσης. Εν έτει 2019, ο αριθμός των ανθρώπων που χρησιμοποιούν τακτικά κάποια πλατφόρμα κοινωνικής δικτύωσης βρίσκεται στο εύρος των 2,89 δισεκατομμυρίων, με στατιστικές προβολές να δείχνουν πως μέχρι το 2021, ο αριθμός αυτός θα ξεπεράσει τα 3 δισεκατομμύρια. [10] Αυτή η διείσδυση των κοινωνικών δικτύων στον ανθρώπινο πολιτισμό φαίνεται πως θα συνεχίσει, με τον μέσο χρήστη του Διαδικτύου να αναλώνει τον χρόνου του σε αυτό κυρίως στην χρήση αυτών, με το περιβάλλον στο οποίο πλοηγούνται να είναι περιορισμένο στις πλέον πιο δημοφιλείς εφαρμογές. [11]

Γι’ αυτόν τον λόγο, λήφθηκε η απόφαση την ανάπτυξης μιας πλατφόρμας κοινωνικής δικτύωσης που αποσκοπεί σε ένα σενάριο χρήσης, όπου οι φοιτητές στην Τριτοβάθμια εκπαίδευση μπορούν να δημιουργούν μικρές κοινότητες για την αλληλοϋποστήριξη στον αγώνα τους για την ολοκλήρωση των σπουδών τους.

Οι υπηρεσίες κοινωνικής δικτύωσης ενθαρρύνουν την εκμάθηση μέσω της «συμμετοχικής κουλτούρας» (participatory culture). [12] Ως συμμετοχική κουλτούρα ορίζεται η αντίθετη έννοια της καταναλωτικής κουλτούρας – με άλλα λόγια, η κουλτούρα στην οποία τα άτομα δεν λειτουργούν μόνο ως καταναλωτές, αλλά και ως παραγωγοί. [13] Στο πλαίσιο της μόρφωσης, προσφέρει ένα χώρο που επιτρέπει την εμπλοκή, την ανταλλαγή, την καθοδήγηση και την κοινωνική αλληλεπίδραση. Οι συμμετέχοντες χρήστες στις υπηρεσίες κοινωνικών δικτύων επωφελούνται αυτών των δυνατοτήτων. Τέτοιοι χώροι μπορούν, να χαρακτηριστούν ως «χώροι έλξης» (affinity spaces), μέρη όπου πραγματοποιείται μάθηση, και δημιουργείται συμμετοχή, συνεργασία, διανομή, διασπορά εμπειρογνωμοσύνης και συσχετικότητα. [14] Οι χρήστες μοιράζονται και αναζητούν γνώσεις που συμβάλλουν στην άτυπη μάθηση.

Ένα από τα ερωτήματα που εγείρονται στην απόπειρα δημιουργία μιας πλατφόρμας κοινωνικής δικτύωσης είναι η δυνατότητα αυτής να ανταγωνιστεί τις κατά πολύ μεγαλύτερες

και εδραιωμένες υπηρεσίες, όπως το Facebook και το Twitter. Όμως, μικρότερα, εξειδικευμένα μέσα κοινωνικής δικτύωσης γίνονται όλο και πιο δημοφιλή, χάρη στο αυξημένο επίπεδο διάδρασης και εμπλοκής μεταξύ των χρηστών. Σύμφωνα με έρευνες, τα αυξανόμενα ποσοστά χρηστών των μεγάλων υπηρεσιών που διαθέτουν όλο και λιγότερο χρόνο σε αυτές, ορίζουν ως κύριους λόγους ότι οι ιστότοποι αυτοί είναι «βαρετοί», «μη σχετικοί» ή «μη χρήσιμοι». [15] Οι εξειδικευμένες πλατφόρμες κοινωνικής δικτύωσης προσφέρουν έναν ειδικό χώρο που έχει σχεδιαστεί για να προσελκύσει μια πολύ συγκεκριμένη ομάδα χρηστών με ένα προκαθορισμένο σύνολο αναγκών, διότι οι σημερινοί χρήστες αναζητούν κοινωνικές συνδέσεις, αίσθηση κοινότητας και κοινές εμπειρίες. Έτσι, τα κοινωνικά δίκτυα που προσεγγίζουν άμεσα συγκεκριμένες δραστηριότητες και τρόπους ζωής έχουν μια σταθερή αύξηση δημοτικότητας. Παραδείγματα αυτών είναι: η εφαρμογή καταγραφής, διαμοιρασμού φωτογραφιών και κοινωνικής δικτύωσης σχετική με το ψάρεμα – Fishbrain, και η εφαρμογή κοινωνικής δικτύωσης και ιχνηλάτησης ασκήσεων τρεξίματος και ποδηλασίας – Strava.

1.4 Δομή Εργασίας

Η παρούσα πτυχιακή εργασία είναι χωρισμένη σε πέντε (5) κεφάλαια, τα οποία εμβυθίζουν τον/την αναγνώστη/ρια στο θεωρητικό υπόβαθρο της και τον οδηγούν μέσα από την διαδρομή του σχεδιασμού και της ανάπτυξης της πλατφόρμας κοινωνικής δικτύωσης. Συγκεκριμένα:

- Στο **Κεφάλαιο 1**, πραγματοποιείται η εισαγωγή του/της αναγνώστη/ριας στην θεματολογία της πτυχιακής εργασίας, περιγράφεται η βασική ιδέα, καθώς επίσης παρουσιάζονται τα κίνητρα πίσω από αυτήν και οι στόχοι στους οποίους αποσκοπεί.
- Στο **Κεφάλαιο 2**, γίνεται λεπτομερής εξέταση της προσέγγισης του προβλήματος που επιλύει η πτυχιακή εργασία: η ερεύνα, η μεθοδολογία, οι έννοιες, τα μοντέλα και ο συλλογισμός που προηγήθηκαν της υλοποίησης της πλατφόρμας κοινωνικής δικτύωσης, όπως και η αιτιολόγηση των επιλογών που πραγματοποιήθηκαν για την επίλυση.
- Στο **Κεφάλαιο 3**, αναλύονται τα τεχνολογικά εργαλεία τα οποία χρησιμοποιήθηκαν για την ανάπτυξη της διαδικτυακής εφαρμογής, με σκοπό την παρουσίαση των καινοτομιών που εισάγουν στον τομέα αυτό. Επιπλέον, περιγράφεται η ροή εργασίας και το περιβάλλον στο οποίο έγινε η υλοποίηση της εφαρμογής,
- Στο **Κεφάλαιο 4**, παρουσιάζεται ο σχεδιασμός της διαδικτυακής εφαρμογής, καθώς και η διαδικασία ανάπτυξης των μερών της.
- Στο **Κεφάλαιο 5,** ο/η αναγνώστης/ρια θα βρει την περιγραφή της ουσίας της πτυχιακής εργασίας, δηλαδή της εφαρμογής και των μηχανισμών της. Επίσης, μέσω σεναρίων χρήσης επιδεικνύεται η λειτουργικότητα της πλατφόρμας.
- Στο **Κεφάλαιο 6**, καλύπτονται τα αποτελέσματα και επιτεύγματα της πτυχιακής εργασίας, με τη μελέτη των πορισμάτων που αποκομίζονται, ενώ περιλαμβάνεται και μια ενατένιση στις πιθανές μελλοντικές επεκτάσεις που επιτρέπει το ολοκληρωμένο έργο.

2. Μεθοδολογία Ανάλυσης και Υλοποίησης

Για την παράδοση του front-end μιας ιστοσελίδας ή μιας διαδικτυακής εφαρμογής σε έναν χρήστη, προηγούνται πολλά συμβάντα στο παρασκήνιο της, δηλαδή στο back-end. Η κατανόηση των μερών, τόσο του front-end όσο και του back-end είναι, φαινομενικά παμμέγεθης διαδικασία, λόγω του μεγάλου αριθμού των ξεχωριστών στοιχείων καθώς και της πολυπλοκότητάς τους. Επίσης, ανάλογα με το είδος της εφαρμογής ή ιστοσελίδας, το front-end και το back-end της κάθε μίας μπορεί να διαφέρει δραματικά. Ανασκοπώντας: το front-end μιας ιστοσελίδας ή εφαρμογής απαρτίζεται από τα στοιχεία HTML, CSS, JavaScript, καθώς και στατικών στοιχείων που αποστέλλονται στον πελάτη (client), όπως είναι ένας περιηγητής (web browser). Ένας εξυπηρετητής (server) είναι μια διαδικασία που τρέχει σε κάποιον απομακρυσμένο υπολογιστή, ο οποίος «ακούει» τα εισερχόμενα αιτήματα (requests) για τα προαναφερθέντα στοιχεία και αποστέλλει αποκρίσεις (responses) μέσω του Διαδικτύου. Η αποθήκευση, προσπέλαση και διαχείριση των εν λόγω στοιχείων αποτελεί ένα μεγάλο κομμάτι του back-end μιας διαδικτυακής εφαρμογής. Επίσης, το server-side μέρος μιας διαδικτυακής εφαρμογής διαχειρίζεται και άλλες σημαντικές εργασίες, όπως η εξουσιοδότηση (authorization) και η πιστοποίηση (authentication) των χρηστών της. Τα δεδομένα αποθηκεύονται σε βάσεις δεδομένων (databases), είτε σχεσιακές (relational), είτε μη-σχεσιακές (NoSQL). Όλο και πιο συχνά, το back-end μιας εφαρμογής διαθέτει ένα Web API, που αποτελεί το τρόπο διάδρασης με τα δεδομένα μια εφαρμογής, μέσω βασικών αιτημάτων και αποκρίσεων HTTP. Συνολικά, οι τεχνολογίες που χρησιμοποιούνται για την δόμηση του front-end και του back-end μιας διαδικτυακής εφαρμογής είναι γνωστές ως στοίβα (stack), και πολλές διαφορετικές γλώσσες προγραμματισμού, καθώς και δομές ανάπτυξης, χρησιμοποιούνται για την υλοποίησης εύρωστων εφαρμογών.

Η ανάπτυξη διαδικτυακών εφαρμογών είναι ένα συνεχώς μεταλλασσόμενο πεδίο της Πληροφορικής και της ανάπτυξης λογισμικού γενικότερα – ο τρόπος με τον οποίο χτίζονται οι σύγχρονες ιστοσελίδες και εφαρμογές διαφέρει σχεδόν καθ' ολοκληρίαν από τις πρακτικές που ήταν σε χρήση πριν από μία δεκαετία, πενταετία, ή ακόμη και πριν από μερικά χρόνια. Καθώς αυτή η τάση δεν φαίνεται να αλλάξει, οι σημερινοί Μηχανικοί που ασχολούνται με τον συγκεκριμένο τομέα αντιμετωπίζουν μια ασυνήθιστη πρόκληση στο στάδιο της ανάλυσης των εργασιών τους: μέσα από την αστείρευτη πληθώρα εργαλείων που τούς είναι διαθέσιμα, να επιλέξουν ποια από αυτά θα τους εξυπηρετήσουν με τον πλέον βέλτιστο τρόπο. Για αυτόν τον σκοπό, πρέπει αν πραγματοποιηθεί η ανάλυση των επιμέρους στοιχείων της εφαρμογής, ώστε να αναγνωριστούν οι απαιτήσεις που παρουσιάζουν και να μελετηθούν τα κατάλληλα εργαλεία, τεχνολογίες, μεθοδολογίες και φιλοσοφίες σύμφωνα με αυτές.

2.1 Μέθοδος Ανάλυσης

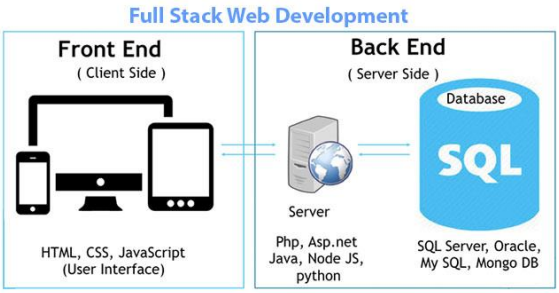
Όπως προαναφέρθηκε, οι διαδικτυακές εφαρμογές αποτελούνται από πολλά συστατικά στοιχεία. Αυτά, μπορούν να κατηγοριοποιηθούν σε δύο γενικές ομάδες: τα **συστατικά διεπαφής χρήστη** (user interface components) και το **δομικά συστατικά** (structural components). Πιο συγκεκριμένα, τα συστατικά διεπαφής χρήστη αναφέρονται στο μέρος της εφαρμογής με το οποίο ο χρήστης έρχεται σε άμεση επαφή και μπορούν να αναπτυχθούν σχεδόν αυτόνομα από τα δομικά στοιχεία της εφαρμογής, καθώς η έμφαση δίνεται στην εμπειρία του χρήστη, και όχι στην λειτουργικότητα της εφαρμογής. Τα δομικά στοιχεία αποτελούν την «καρδιά» της ανάπτυξης διαδικτυακών εφαρμογών και χωρίζονται σε τρεις (3) κατηγορίες:

1. Πελάτης (client): Το πρόγραμμα περιήγησης/πελάτης είναι η απόδοση της διεπαφής χρήστη της λειτουργικότητας της εφαρμογής, μέσω της οποίας ο

χρήστης, ουσιαστικά, αλληλοεπιδρά με την εφαρμογή. Το περιεχόμενο αυτό που παραδίδεται στον πελάτη μπορεί να αναπτυχθεί με την χρήση της HTML, CSS και JavaScript και δεν χρειάζεται προσαρμογές σχετικές με το λειτουργικό σύστημα του υπολογιστή-πελάτη.

2. Εξυπηρετητής διαδικτυακής εφαρμογής (web application server): Ο εξυπηρετητής αυτός έχει την ευθύνη της διαχείρισης της λογικής και της ορθής απόδοσης των δεδομένων. Μπορεί να αναπτυχθεί με την χρήση της Python, Ruby, Java, PHP, Node.js, μεταξύ άλλων.
3. Εξυπηρετητής βάσης δεδομένων (database server): Η βάση δεδομένων είναι αυτή που αποθηκεύει και προμηθεύει τα δεδομένα που είναι σχετικά με την λειτουργία της εφαρμογής.

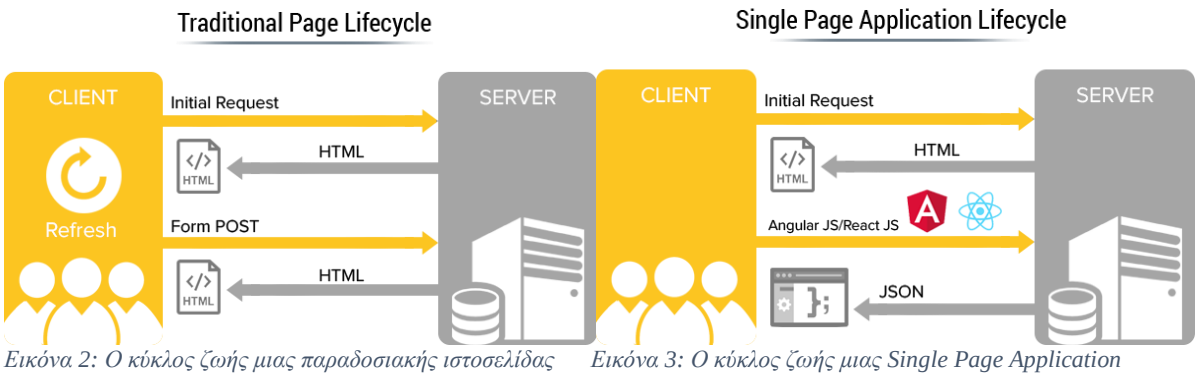
Τα τελευταία χρόνια, οι πλέον δημοφιλής αρχιτεκτονικές για την ανάπτυξη διαδικτυακών εφαρμογών είναι οι εξής δύο: το σχήμα MVC (Model-View-Controller) από την πλευρά του εξυπηρετητή (server-side) και η SPA (Single-Page-Application) από την πλευρά του πελάτη με Web API από την πλευρά του εξυπηρετητή. Οπότε, η πρώτη απόφαση για την ανάπτυξη μιας διαδικτυακής εφαρμογής είναι η επιλογή της αρχιτεκτονικής προσέγγισης, σε δεύτερο στάδιο της υλοποίηση του server-side, ύστερα του client-side και τέλος, της βάσης δεδομένων. Στη συνέχεια, ακολουθεί η ανάλυση των δομικών συστατικών που μελετήθηκαν για την παρούσα πτυχιακή εργασία.



Εικόνα 1: Η δομή του Full Stack Web Development

2.1.1 SPA και Web API

Την περίοδο σύνταξης της παρούσας πτυχιακής, ο συνδυασμός SPA και Web API έχει γίνει ο πιο δημοφιλής τρόπος για την ανάπτυξη μοντέρνων διαδικτυακών εφαρμογών. Ο κυριότερος λόγος γι' αυτή την δημοτικότητα είναι το γεγονός πως το ολόκληρο μέρος client-side της εφαρμογής φορτώνεται μόνο μία φορά, και δεν υπάρχει ανάγκη για την επαναφόρτιση της κάθε σελίδας της εφαρμογής με κλήσεις προς τον διακομιστή. Η δρομολόγηση (routing) εντός της εφαρμογής παίρνει μέρος αποκλειστικά στην πλευρά του πελάτη, ο εξυπηρετητής προμηθεύει το API για την προσπέλαση των δεδομένων.



Εικόνα 2: Ο κύκλος ζωής μιας παραδοσιακής ιστοσελίδας

Εικόνα 3: Ο κύκλος ζωής μιας Single Page Application

Αυτή η προσέγγιση παρουσιάζει μια σειρά πλεονεκτημάτων: όπως προαναφέρθηκε, λόγω της φόρτισης του ιστοχώρου εξαρχής, η πλοήγηση εντός αυτού είναι, ουσιαστικά, στιγμιαία και η εμπειρία χρήσης μια διαδικτυακής εφαρμογής πλησιάζει περισσότερο στην εμπειρία χρήσης μιας εφαρμογής εγκατεστημένης στον υπολογιστή-πελάτη (desktop application). Επίσης, ο διακριτός διαχωρισμός σε client-side και server-side δημιουργεί τις εξής διευκολύνσεις: οι προγραμματιστές που εργάζονται στην ανάπτυξη της εφαρμογής δεν χρειάζεται να είναι εμπειρογνώμονες σε εύρος full-stack, επιτρέποντάς τους να εστιάσουν και να διαπρέψουν στην ανάπτυξη το κομματιού που ανήκει στην ειδικότητά τους. Ακόμη, λόγω αυτού του διαχωρισμού, είναι ευκολότερη η αυτόνομη ανάπτυξη των front-end και back-end. Τέλος, η ανάπτυξη ενός εύρωστου Web API το καθιστά επαναχρησιμοποιήσιμο για κάθε μορφή της εφαρμογής (web, desktop, mobile).

Η αρχιτεκτονική αυτή δεν είναι δίχως κάποια ελαττώματα που πρέπει να ληφθούν υπόψιν των Μηχανικών που αναπτύσσουν την εφαρμογή: Η πρώτη φόρτωση μια τέτοια ιστοσελίδας θα είναι απαιτητική, σε θέμα χρόνου, καθώς φορτώνεται ολόκληρο το client-side. Η προετοιμασία μιας τέτοιας εφαρμογής, εκ φύσεώς της απαιτεί την ανάπτυξη ξεχωριστών έργων για τα client-side και server-side, και γι' αυτόν τον λόγο περιπλέκονται περισσότερες τεχνολογίες. Επίσης, η ροή εργασίας επιβαρύνεται με «δεσμευτές» (bundlers) και την διαμόρφωσή του για μια ευπρεπή διαδικασία προγραμματισμού, θέμα που θα αναλυθεί περαιτέρω σε ξεχωριστή ενότητα. Αυτή η επιβάρυνση, όμως, έρχεται με μερικά επιπλέον εργαλεία για την διευκόλυνση της ανάπτυξης, όπως μεταγλωττιστές της JavaScript και προγράμματα ανάλυσης λαθών στον πηγαίο κώδικα.

2.1.3 Server-side Web API

Στην περίπτωση επιλογής της αρχιτεκτονικής SPA με Web API, το επόμενο βήμα είναι η επιλογή της τεχνολογίας server-side. Ένα **API** (**A**pplication **P**rogramming **I**nterface – Διεπαφή Προγραμματισμού Εφαρμογών) είναι η διεπαφή επικοινωνίας μεταξύ του πελάτη και του εξυπηρετητή και αποσκοπεί στην απλοποίηση του προγραμματισμού της εφαρμογής, αποχωρίζοντας την υποκείμενη δομή της από την διεπαφή με την οποία αλληλοεπιδρά ο χρήστης, εκθέτοντας μόνο τις λειτουργίες που θα ορίσει ο προγραμματιστής. Ο βασικός τρόπος επικοινωνίας, όπως προαναφέρθηκε, είναι οι αιτήματα HTTP, οπότε ο εξυπηρετητής πρέπει να προμηθεύσει το API. Μια πολύ δημοφιλής υλοποίηση είναι αυτή των **RESTful APIs**, τα οποία βασίζονται στην τεχνολογία **RE**presentational **S**tate **T**ransfer και χρησιμοποιούν αιτήματα HTTP – GET, PUT, POST και DELETE, για ακριβώς αυτούς του σκοπούς, την λήψη, αποστολή, ανάρτηση και διαγραφή δεδομένων. Υπάρχουν όμως και άλλες τεχνολογίες για server-side API, όπως **Node.js** με την χρήση του web framework **Express.js** (JavaScript), **ASP .NET Web API** (C#), **Django REST** (Python), **Ruby on Rails** (Ruby).

2.1.3 SPA Framework

Με την ολοκλήρωση της μελέτης των τεχνολογιών server-side, το επόμενο στάδιο είναι η μελέτη των διαθέσιμων τεχνολογιών που θα εξυπηρετήσουν την Single-Page-Application υπό ανάπτυξη στο κομμάτι του client-side. Πριν από μερικά χρόνια, τέτοιου είδους καινούργια frameworks παρουσιάζονταν με τεράστια συχνότητα, αλλά σήμερα, αυτός ο τομέας έχει σταθεροποιηθεί αρκετά ώστε να φανούν οι δημοφιλέστερες βιβλιοθήκες και δομές ανάπτυξης. Τα δημοφιλέστερα από αυτά είναι: **React**, **Angular**, **Vue**, **Ember**, **Polymer**.

2.1.4 Database

Κάθε σύγχρονη διαδικτυακή εφαρμογή διαθέτει κάποιου είδους βάση δεδομένων. Αν και στο παρελθόν η μόνη επιλογή ήταν μεταξύ διαφόρων σχεσιακών βάσεων δεδομένων, σήμερα υπάρχουν περισσότερο εξειδικευμένες βάσεις δεδομένων, καθώς αυξάνεται το εύρος των απαιτήσεων των εφαρμογών. Η απόφαση για την βάση δεδομένων της εφαρμογής είναι εξίσου σημαντική με την επιλογή των τεχνολογιών client και server-side, και είναι πολύ σημαντική η κατανόηση των αναγκών της εφαρμογής υπό ανάπτυξη. Γενικότερα, οι βάσεις δεδομένων μπορούν να χωριστούν σε μια σειρά από κατηγορίες: **Σχεσιακές Βάσεις Δεδομένων** – οι κλασικές βάσεις δεδομένων που λειτουργούν με την χρήση σχεσιακών μοντέλων, με τις δημοφιλέστερες βάσεις δεδομένων, **Oracle**, **MySQL**, **Microsoft SQL Server** και **PostgreSQL**, να είναι όλες σχεσιακές. Οι **Μη Σχεσιακές Βάσεις Δεδομένων** δεν βασίζονται σε σχεσιακά μοντέλα, αλλά ακολουθούν άλλες φιλοσοφίες υλοποίησης, όπως η βάση δεδομένων document-store - **MongoDB**, η βάση δεδομένων key-value store - **Redis**, η βάση δεδομένων wide-column store – **Cassandra**, και η graph βάση δεδομένων - **Neo4j**.

2.1.5 Authentication και Authorization

Στην πράξη, όλες οι διαδικτυακές εφαρμογές έχουν κάποιο μηχανισμό για εγγραφή και σύνδεση των χρηστών τους. Η δυνατότητα μιας εφαρμογής να επαληθεύσει την ταυτότητα των χρηστών, ονομάζεται **authentication** – πιστοποίηση. Οι μηχανισμοί που καθορίζουν τις άδειες των χρηστών, ονομάζονται συλλογικά **authorization** – εξουσιοδότηση.

Εικόνα 4: Εικονική αναπαράσταση της διαφοράς μεταξύ authentication και authorization

Υπάρχουν αρκετοί τρόποι προσέγγισης. Ο πιο απλός τρόπος είναι μέσω της υποστήριξης των διαφόρων δομών για μηχανισμούς πιστοποίησης και εξουσιοδότησης, όπως μέσω της χρήσης JWT tokens ή βιβλιοθηκών third-party. Για πιο απαιτητικές εφαρμογές, υπάρχουν σήμερα External Identity Providers που ακολουθούν το OpenID Connect Standard. Οι υπηρεσίες που προσφέρουν, πέρα από την εγγραφή και σύνδεση των χρηστών, είναι η ενσωματωμένη υποστήριξη για εξωτερικούς παροχείς ταυτότητας (Google, Facebook), άδειες με βάση τους ρόλους των χρηστών, πολυεπίπεδη πιστοποίηση, αναλύσεις και καταγραφές. [16]

2.2 Μέθοδος Ανάπτυξης

Με την ανάλυση των απαιτούμενων μερών για την υλοποίηση της εφαρμογής, πρέπει να οριστεί μια ξεκάθαρη πορεία, όσον αφορά τα εργαλεία, τις βιβλιοθήκες και τις δομές ανάπτυξης που θα χρησιμοποιηθούν, ώστε αυτά να εξυπηρετούν με τον καλύτερο τρόπο της απαιτήσεις της εργασίας.

2.2.1 Βασικά Εργαλεία

Στην σημερινή εποχή, κανένας προγραμματιστής στον τομέα της ανάπτυξης διαδικτυακών εφαρμογών δεν πραγματοποιεί το έργο του χωρίς της χρήση των τριών βασικών κιόνων κάθε ιστοσελίδας: HTML, CSS και JavaScript. Η HTML είναι αυτή που ορίζει τον «σκελετό», την βασική δομή κάθε ιστοσελίδας, η CSS είναι αυτή που ορίζει την «εμφάνισή», την παρουσία και τον σχεδιασμό της, και η JavaScript την λειτουργικότητά και την συμπεριφορά της. Καθώς η παρούσα πτυχιακή εργασίας εστιάζει ιδιαίτερα στην μελέτη και ανάλυση τον πιο καινοτόμων λύσεων που χρησιμοποιούνται τον καιρό που συντάσσεται, ακολουθεί μια απλή, αλλά επίτιμη, σύνοψη αυτής της «τριάδας τεχνολογιών» του Διαδικτύου: [17]

HTML: Η **HTML** (**H**yper**T**ext **M**arkup **L**anguage) είναι μια γλώσσα σήμανσης. Αυτό σημαίνει πως, αντί να χρησιμοποιεί προγραμματιστικές συναρτήσεις για την απόδοση της δομής μιας ιστοσελίδας, η HTML χρησιμοποιεί ετικέτες για την αναγνώριση διαφόρων ειδών περιεχομένου εντός της ιστοσελίδας, καθώς και της χρησιμότητάς τους. [18]

CSS: Η **CSS** (**C**ascading **S**tyle **S**heets), είναι μια γλώσσα φύλλων ύψους για την περιγραφή την εμφάνισης μιας ιστοσελίδας γραμμένης με την χρήση κάποιας γλώσσας σήμανσης, όπως η HTML, συμπεριλαμβανόμενων των χρωμάτων, του σχεδιασμού και των γραμματοσειρών που εμφανίζονται. [19]

JavaScript: Γνωστή και ως **JS**, η **JavaScript** είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού υψηλού επιπέδου, που στα πλαίσια της ανάπτυξης διαδικτυακών εφαρμογών, χρησιμοποιείται για την κατασκευή της λογικής και της συμπεριφοράς των ιστοσελίδων. [20]

2.2.2 Δομές Ανάπτυξης

Η ανάπτυξη του front-end θα βασιστεί ολοκληρωτικά στη χρήση της βιβλιοθήκης **React.js** και τεχνολογιών που υποστηρίζουν αυτό το οικοσύστημα, ακολουθώντας την αρχιτεκτονική **Flux** και με την χρήση της βιβλιοθήκης **Redux**. Το σχεδιαστικό κομμάτι θα εξυπηρετήσει η **Material-UI**, μια βιβλιοθήκη ειδικά διαμορφωμένη για την δομή της **React.js**, ενώ αισθητικά και σχεδιαστικά συμμορφώνεται με την σχεδιαστική γλώσσα **Material Design**.

Για την παρούσα πτυχιακή εργασία, έγινε η επιλογή της υλοποίησής της με βάση το **server-side rendering**, ώστε η εφαρμογή να αποδίδεται στον εξυπηρετητή, παρά στην πλευρά του πελάτη. Μερικοί από του λόγους, πέρα της θέλησης για την εξερεύνηση της συγκεκριμένης τεχνολογίας, είναι πως οι εφαρμογές αυτές χαίρονται μικρότερου χρόνου απαιτούμενου για την φόρτωσή τους σε έναν περιηγητή, αλλά και από την οπτική γωνία μιας πλατφόρμας κοινωνικής δικτύωσης, χάρη στην ευκολότερη πρόσβαση στα μεταδεδομένα μιας τέτοιας εφαρμογής, ο διαμοιρασμός της σε άλλα κοινωνικά δίκτυα θα είναι εξίσου εύκολη. Επίσης, υποθετικά, αυτή η τεχνολογία προσφέρει πολύ καλύτερο **Search Engine Optimization – SEO**, απ' ότι έχει μια εφαρμογή που αποδίδεται αποκλειστικά client-side.

Για την ανάπτυξη του back-end, από την λογική της εφαρμογής μέχρι την υλοποίηση του API της, η προτίμηση ήταν ξεκάθαρα στην χρήση της **Node.js**, πάνω στο δημοφιλές framework της, **Express.js**. Με αυτές τις τεχνολογίες, τα client και server-sides μπορούν να αναπτυχθούν με την χρήση μόνο μιας γλώσσας – **JavaScript** – μειώνοντας αισθητά την ανάλωση χρόνου για την εκμάθηση των ιδιοτροπιών κάποιας άλλης γλώσσας.

Όσον αφορά τη βάση δεδομένων της εφαρμογής, η ανάπτυξή της επιλέχθηκε η **PostgreSQL.** Οι σχεσιακές βάσεις δεδομένων έχουν περάσει την δοκιμασία του χρόνου, καθώς παραμένουν οι δημοφιλέστερες σε χρήση, όντας συγχρόνως οι παλαιότερες σε

ύπαρξη. Είναι γρήγορες και αξιόπιστες, ενώ η συγκεκριμένη εξασφαλίζει την δυνατότητα της εφαρμογής να χειρίζεται περίπλοκα ερωτήματα σε SQL.

Επιπλέον, για τον σχεδιασμό των μηχανισμών authorization και authentication, χρησιμοποιήθηκαν δύο τεχνολογίες: το **Passport**, ένα Node.js middleware που προσφέρει μια πληθώρα από μηχανισμούς ταυτοποίησης, γνωστών ως «στρατηγικές» - **strategies**. Μαζί με αυτό, η εφαρμογή ακολουθεί το standard ταυτοποίησης γνωστό ως **JWT** (**J**SON **W**eb **T**okens), το οποίο βασίζεται σε κωδικοποιημένα τεκμήρια για την αναγνώριση του κάθε συνδεδεμένου χρήστη, αντί για την αποθήκευση της συνεδρίας εντός κάποιου cookie. Ιδιαίτερη εξυπηρέτηση αποτελεί το γεγονός πως μία από τις ενσωματώσεις του έρχεται στην μορφή **Node.js module**.

Όλες αυτές οι τεχνολογίες, καθώς και οι ιδιαιτερότητες της φιλοσοφίας πίσω από αυτές που επηρεάζουν την παρούσα εργασία με σημαντικό τρόπο, παρουσιάζονται με λεπτομέρεια στο **κεφάλαιο 3 – Τεχνολογίες**.

2.2.3 Βοηθητικά Εργαλεία

Η μοντέρνα ανάπτυξη διαδικτυακών εφαρμογών κάνει χρήση μιας πληθώρας βοηθητικών εργαλείων για την υποστήριξη αυτής της διαδικασίας. Για τις ανάγκες της συγκεκριμένης εργασίας, μελετήθηκαν κάποιες κατηγορίες αυτών των εργαλείων και επιλέχθηκαν μερικά, σύμφωνα με τις ανάγκες.

Σε πρώτο στάδιο, εξετάστηκε η κατηγορία των **Package Managers**. Οι Package Managers χρησιμοποιούνται για την αυτοματοποίηση των διαδικασιών εγκατάστασης, ενημέρωσης, μετατροπής και αφαίρεσης προγραμμάτων. Ως **package** ορίζεται ένα αρχείο που περιέχει τον κώδικα του λογισμικού αυτού, αρχεία για την διαμόρφωσή του και την λίστα των εξαρτήσεών του – **dependencies**, ξεχωριστών προγραμμάτων που είναι απαραίτητα για την σωστή λειτουργία του λογισμικού. Πριν την εμφάνισή τους, οι μηχανικοί που χρησιμοποιούσαν την JavaScript στο έργο τους βασίζονταν σε dependencies που διατηρούσαν τοπικά στους υπολογιστές στους οποίους εργάζονταν. Μακράν ο πιο δημοφιλής package manager, **NPM** (**N**ode **P**ackage **M**anager), εμφανίστηκε λίγο μετά την εισαγωγή της Node.js στον χώρο, και ως αποτέλεσμα, χιλιάδες open-source project εμφανίστηκαν και οι μηχανικοί σε ολόκληρο τον κόσμο είχαν την δυνατότητα να διαμοιράζονται τον κώδικά τους σε βαθμό που δεν είχε συμβεί στο παρελθόν. Όμως, για τις ανάγκες αυτής της εργασίας, επιλέχθηκε η χρήση του **Yarn**.

Για την ανάπτυξη της εμφάνισης μιας εφαρμογής, πολύ συχνά χρησιμοποιούνται **CSS Pre-processors** και **CSS Frameworks**. Οι pre-processors είναι προγράμματα τα οποία επιτρέπουν στον προγραμματιστή την παραγωγή κώδικα μέσω της χρήσης του ιδιαίτερου συντακτικού του δοθέντος pre-processor. Οι περισσότεροι pre-processors προσφέρουν περισσότερες δυνατότητες απ' όσες είναι διαθέσιμες με την βασική έκδοση της CSS. [21] Μερικοί ιδιαίτερα δημοφιλείς pre-processors είναι οι **Sass**, **LESS** και **Stylus**. Όσον αφορά τα CSS frameworks, είναι βιβλιοθήκες οι οποίες προσφέρουν την δυνατότητα ευκολότερου σχεδιασμού της εμφάνισης μιας ιστοσελίδας, με τον σχεδιασμό αυτό να συμμορφώνεται με τα σύγχρονα standards. Αν και κάποια frameworks περιέχουν κάποια λειτουργικότητα μέσω της JavaScript, τα περισσότερα εστιάζουν στον τομέα του σχεδιασμού και της διεπαφής χρήστη. Μερικά από τα πιο γνωστά frameworks είναι τα **Bootstrap**, **Semantic UI** και **Foundation**. Για τις ανάγκες της συγκεκριμένης εργασίας, η επιλογή είναι η **Material-UI**, κυρίως λόγω της ιδιαίτερης υποστήριξης για την δομή της **React**.js, αλλά και την δυνατή προσκόλληση στο **Material Design**.

Σαν τελική κατηγορία βοηθητικών εργαλείων, εξετάστηκε αυτή των **Build Tools**. Αυτά τα εργαλεία βοηθούν στην ανάπτυξη και κατασκευή εφαρμογών σε JavaScript. Περιλαμβάνει τρεις υποκατηγορίες: τους **Task Runners**, **Module Bundlers** και **Linters**. Η

κύρια λειτουργία των bundlers είναι η συνένωση όλων των JavaScript modules της εφαρμογής σε ένα ενιαίο και σειριακά λογικό αρχείο JavaScript που θα εκτελέσει ο περιηγητής, καθώς φροντίζει για όλες τις εξαρτήσεις που υπάρχουν μεταξύ τους. Για τις ανάγκες αυτής της εργασίας, επιλέχθηκε το **Webpack**, αν και υπάρχουν και άλλοι δημοφιλείς, όπως το **Rollup** και **Parcel**.

Ως **Task Runners** ορίζονται τα εργαλεία που έχουν δημιουργηθεί με σκοπό την εκτέλεση συγκεκριμένων εργασιών, με βάση κάποια κριτήρια όπως, για παράδειγμα, την εκτέλεση μιας εργασίας κάθε φορά που γίνονται αλλαγές σε κάποιο αρχείο, έτσι ώστε να μην είναι απαραίτητη η μεταγλώττισή του εκ νέου, ή η επανεκκίνηση του εξυπηρετητή κάθε φορά που γίνονται αλλαγές στη βάση δεδομένων. Για την Node.js υπάρχουν δύο ιδιαίτερα δημοφιλείς task runners, οι **Grunt** και **Gulp**, αλλά για τις ανάγκες της συγκεκριμένης εργασίας, η λειτουργικότητα του bundler που επιλέχθηκε, Webpack, καλύπτει πλήρως τις ανάγκες για την αυτοματοποίηση κάποιων εργασιών κατά την διάρκεια ανάπτυξης αυτής της εφαρμογής.

Τέλος, **Linters** ονομάζονται τα εργαλεία που αναλύουν τον πηγαίο κώδικα μιας εφαρμογής και υποδεικνύουν προγραμματιστικά και στιλιστικά λάθη, bugs και πιθανά «ασταθή» σημεία. Την συγκεκριμένη εργασία θα εξυπηρετήσει το **ESLint**.

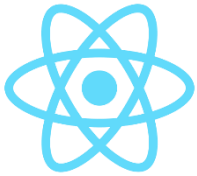
Αυτά τα βοηθητικά εργαλεία αναλύονται περαιτέρω ως μέρος του **κεφαλαίου 4 – Σχεδιασμός και Υλοποίηση Εργασίας.**

3. Τεχνολογίες

3.1 Τεχνολογίες Front-end

Η React θα αποτελέσει εκτενές κομμάτι του σχεδιασμού, καθώς αποτελεί μια νέα προσέγγιση στον χώρο της ανάπτυξης διαδικτυακών εφαρμογών και πέραν του γεγονότος πως κατακτά όλο και μεγαλύτερο μερίδιο του τομέα, σήμερα αποτελεί τον πρωτεύων τρόπο για την ανάπτυξη μεγάλων, γρήγορων και επεκτάσιμων εφαρμογών.

3.1.1 React.js



Εικόνα 5:Το λογότυπο της βιβλιοθήκης React.js

Η **React.js** είναι μια front-end βιβλιοθήκη της JavaScript ανοικτού κώδικα, που εξυπηρετεί την δόμηση διεπαφών χρήστη, καθώς και την διευκόλυνση της ανάπτυξης των εν λόγω διεπαφών. Δημιουργήθηκε, αρχικά, από τον Jordan Walke, έναν μηχανικό λογισμικού της εταιρίας Facebook, από την οποία σήμερα διατηρείται. [22] Η React υπερτερεί στην εμφάνιση και την ενημέρωση «συστατικών» (components - ουσιαστικά, στοιχεία της ιστοσελίδας) που βασίζονται σε και εμφανίζουν δεδομένα, το οποίο είναι ιδανικό για ιστότοπους που εμφανίζουν πολλά, μεταβαλλόμενα δεδομένα.

Για την βέλτιστη κατανόησή της, ακολουθεί λεπτομερής ανάλυση των κεντρικών εννοιών και της φιλοσοφίας της React.

3.1.1.1 JSX

Η JSX αποτελεί μια επέκταση συντακτικού της JavaScript. Προτείνεται για χρήση μαζί με την React για την περιγραφή της εμφάνισης της διεπαφής χρήστη. Παρόλο που μοιάζει με γλώσσα προτυποποίησης, φέρει όλη την λειτουργικότητα της JavaScript και μέσω αυτής παράγονται τα στοιχεία της React.

Το νόημα της χρήσης της JSX μπορεί να φανεί με μια κοντινότερη εξέταση της φιλοσοφίας της React: Η React αγκαλιάζει το γεγονός πως η λογική της απόδοσης (rendering logic) είναι εγγενώς συνδεδεμένη με την λογική της διεπαφής χρήστη, όπως στην περίπτωση του χειρισμού των γεγονότων (events), της αλλαγής της κατάστασης (state) με την πάροδο του χρόνου και της προετοιμασίας των δεδομένων για προβολή. Αντί του τεχνητού διαχωρισμού των τεχνολογιών με την τοποθέτηση της σήμανσης (markup) και της λογικής σε ξεχωριστά αρχεία, η React διαχωρίζει τις ανησυχίες (concerns) με χαλαρά συζευγμένες μονάδες που ονομάζονται συστατικά (components) και περιέχουν και τα δύο. Η React δεν απαιτεί την χρήση της JSX, αλλά αποτελεί χρήσιμο οπτικό βοήθημα κατά την διάρκεια εργασίας με την διεπαφή χρήστη εντός κώδικα JavaScript. Επίσης, επιτρέπει στην React να εμφανίζει περισσότερα χρήσιμα μηνύματα λάθους και προειδοποίησης.

Στο παρακάτω παράδειγμα, δηλώνουμε μια μεταβλητή και την χρησιμοποιούμε εντός της JSX, μέσω της χρήσης άγκιστρων:

Πίνακας 1: Παράδειγμα χρήσης JavaScript εντός της JSX

```
const name = 'Arnold Dziudzik';
const element = <h1>Hello, {name}!<h1>;
```

Όλες οι ορθές εκφράσεις της JavaScript μπορούν να τοποθετηθούν εντός αγκίστρων στην JSX. Επίσης, είναι δυνατή η διαίρεσή τους σε πολλαπλές γραμμές για την εξυπηρέτηση της ευανάγνωσής τους.

Μετά την μεταγλώττιση (compiling), οι εκφράσεις JSX γίνονται κανονικές κλήσεις συναρτήσεων της JavaScript και διατιμώνται σε αντικείμενα (objects) της JavaScript. Αυτό σημαίνει ότι η JSX μπορεί να χρησιμοποιηθεί εντός δομών ελέγχου (δηλώσεις if, βρόχοι for), μπορεί να εκχωρηθεί σε μεταβλητές, να χρησιμοποιηθεί ως όρισμα και να επιστραφεί από συναρτήσεις:

Πίνακας 2: Χρήση της JSX εντός συναρτήσεων

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

Για τον ορισμό κυριολεκτικών χαρακτήρων ως ιδιοτήτων, μπορούν να χρησιμοποιηθούν εισαγωγικά, και τα άγκιστρα για την ένθεση εκφράσεων JavaScript ως ορισμάτων, όπως φαίνεται στο παρακάτω παράδειγμα:

Πίνακας 3: Προσδιορισμός ιδιοτήτων με την χρήση της JSX

```
const element = <div>tabIndex="0"</div>
const element = <img src={user.avatarUrl}></img>
```

Εάν μια ετικέτα (tag) είναι κενή, μπορεί να τερματιστεί αμέσως, παρόμοια με την γλώσσα σήμανσης XML, ενώ επίσης οι ετικέτες JSX μπορούν να έχουν δικά τους παιδιά (children):

Πίνακας 4: Χαρακτηριστικά των ετικετών JSX

```
const element = <img src={user.avatarUrl} />;

const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

Όταν δημιουργείται ένα στιγμιότυπο της JSX, ουσιαστικά, δημιουργείται ένα αντικείμενο (object). Με αυτή την έννοια, στο παρακάτω παράδειγμα, οι δύο εκχωρήσεις στην μεταβλητή μπορούν να θεωρηθούν πανομοιότυπα:

Πίνακας 5: Δημιουργία αντικειμένων μέσω της JSX

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);

const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);

// Η συνάρτηση React.createElement δημιουργεί ένα αντικείμενο της παρακάτω
μορφής

const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

Αυτά τα αντικείμενα ονομάζονται στοιχεία της React (React elements). Σημασιολογικά, είναι σαν περιγραφές των στοιχείων που αποδίδονται στην οθόνη. Η React διαβάζει αυτά τα αντικείμενα και τα χρησιμοποιεί για την κατασκευή και ενημέρωση του DOM. [23]

3.1.1.2 React Elements

Τα elements αποτελούν το μικρότερο δομικό στοιχείο των συναρτήσεων που δομούνται με την χρήση της React. Ένα τέτοιο στοιχείο περιγράφει αυτό που θα αποδοθεί στην οθόνη, όπως αναφέρθηκε στην προηγούμενη ενότητα, σε αντίθεση όμως με τα DOM στοιχεία ενός φυλλομετρητή, τα στοιχεία της React είναι απλά αντικείμενα. Το DOM της React φροντίζει για την ενημέρωση του DOM ώστε να ταιριάζει στα στοιχεία της.

Οι εφαρμογές που έχουν αναπτυχθεί με την χρήση της React συνήθως έχουν ένα, μοναδικό root DOM node. Ονομάζεται έτσι επειδή όλα τα περιεχόμενα του κόμβου αυτού θα διαχειρίζονται από την React:

Πίνακας 6: Απόδοση ενός React Element

```
<div id="root"></div>
const element = <h1>Hello, World</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

Τα elements της React είναι αμετάβλητα (immutable). Αυτό σημαίνει πως δεν μπορούν να τροποποιηθούν μετά την δημιουργία τους, άρα μπορούν να βρίσκονται σε μόνο

μία κατάσταση (state) και δεν μπορούν να αλλάξουν τα παιδιά και οι ιδιότητές τους. Αυτό τα καθιστά πιο ασφαλή, καθώς δεν είναι δυνατή η ανατροπή τους λόγω λαθών στον κώδικα της εφαρμογής. Έτσι, η React ενημερώνει τα elements και τα παιδιά τους μόνο όταν αυτό είναι απαραίτητο, ώστε να έρθει το DOM στην απαιτούμενη κατάσταση, κάνοντας απλή σύγκριση της παλιάς κατάστασης με την καινούργια. Στον τομέα ανάπτυξης διαδικτυακών εφαρμογών, η αφαίρεση της έγνοιας για το πώς θα αλλάξει η διεπαφή χρήστη στο μήκος του χρόνου και η εστίαση στην ζητούμενη εμφάνισή της, απαλλάσσει τους προγραμματιστές από μια ολόκληρη κατηγορία των bugs που αντιμετωπίζουν. [24]

3.1.1.3 Components και Props

Στην React, τα **components** επιτρέπουν τον χωρισμό της διεπαφής χρήστη σε ανεξάρτητα, επαναχρησιμοποιήσιμα κομμάτια, επιτρέποντας στον προγραμματιστή τους να τα σχεδιάζει χωρίς να τα περιπλέκει μεταξύ τους. Εννοιολογικά, τα components λειτουργούν σαν συναρτήσεις της JavaScript, καθώς δέχονται αυθαίρετα ορίσματα, που ονομάζονται **props** (από την αγγλική λέξη properties - ιδιότητες) και επιστρέφουν elements της React, καθορίζοντας έτσι τι θα εμφανίζεται στην διεπαφή.

Ο πιο απλός τρόπος για τον ορισμό ενός component είναι η σύνταξη μιας συνάρτησης JavaScript:

Πίνακας 7: Βασική Διαχείριση Props

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Όπως φαίνεται στο παραπάνω παράδειγμα, η συνάρτηση δέχεται ένα και μοναδικό όρισμα, τα props, που περιέχει τα δεδομένα και επιστρέφει ένα React element. Τέτοια components ονομάζονται **function components**, επειδή είναι, κυριολεκτικά, συναρτήσεις της JavaScript.

Τα components προσφέρουν μια πληθώρα από λειτουργίες. Μέσω της σύνθεσής τους (component composing), μπορούν να αναφέρονται σε άλλα, κάτι που επιτρέπει την χρήση του ίδιου component abstraction για κάθε επίπεδο λεπτομέρειας. Ένα κουμπί ή μια φόρμα στην οθόνη, όλα μπορούν να εκφραστούν ως components. Μέσω της εξαγωγής τους (component extraction), αν κάποιο component είναι τόσο περίπλοκο που καθίσταται δύσκολη η διαχείρισή του, είναι απόλυτα δυνατό να χωριστεί σε μικρότερα components. Σε εφαρμογές μεγάλου μεγέθους, η πρόσβαση σε μια μεγάλη παλέτα από επαναχρησιμοποιήσιμα components μπορεί να αποδειχθεί κερδοφόρο. Ένας καλός γενικός κανόνας είναι, σε περίπτωση που κάποιο μέρος της διεπαφής χρήστη χρησιμοποιείται πολλές φορές (κουμπιά, πεδία κειμένου, πλαίσια εικόνων) ή είναι αρκετά περίπλοκο (feed, post), να θεωρηθούν ως επαναχρησιμοποιήσιμα components.

Πίνακας 8: Παράδειγμα component composition

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
```

```
  return (
    <div>
      <Welcome name="Arnold" />
      <Welcome name="Giannis" />
    </div>
  );
}
```

Πίνακας 9: Ένα μεγάλο component είναι καλός υποψήφιος για extraction...

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
          src={props.author.avatarUrl}
          alt={props.author.name}
        />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

Πίνακας 10:... κάνοντας extract κάποια elements του..

```
function Avatar(props) {
  return (
    <img className="Avatar"
      src={props.user.avatarUrl}
      alt={props.user.name}
    />

  );
}

function UserInfo(props) {
```

```
  return (
    <div className="UserInfo">
      <Avatar user={props.user} />
      <div className="UserInfo-name">
        {props.user.name}
      </div>
    </div>
  );
}
```

Πίνακας 11:... είμαστε σε θέση να απλοποιήσουμε αισθητά την αρχική μορφή του.

```
function Comment(props) {
  return (
    <div className="Comment">
      <UserInfo user={props.author} />
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

Όσον αφορά τα props, αξίζει να σημειωθεί πως είναι μόνο για ανάγνωση (read-only). Αυτό σημαίνει πως έναν component δεν πρέπει ποτέ να μεταβάλλει τα δικά του props. Τέτοιες συναρτήσεις χαρακτηρίζονται ως pure (αμιγές), διότι πάντα επιστρέφουν το ίδιο αποτέλεσμα όταν δέχονται το ίδιο όρισμα, σε αντίθεση με τις impure, που επιχειρούν να αλλάξουν την είσοδό τους. Παρά το γεγονός πως η React χαρακτηρίζεται για την ευελιξία της, υπάρχει ένας αυστηρός κανόνας στην χρήση της: όλα τα components πρέπει να συμπεριφέρονται σαν pure συναρτήσεις, όσον αφορά τα props τους. [25]

3.1.1.4 State και Lifecycle

Ιδανικά, όλα τα components της React πρέπει να είναι σε θέση να ενημερώνουν τον εαυτό τους. Για την υλοποίηση αυτής της ιδιότητας, κάθε component χρειάζεται μια «κατάσταση» - **state**. Το state είναι παρόμοιο με τα props, αλλά είναι private και ελέγχεται αποκλειστικά από το component μέσω της συνάρτησης setState(). Για την ορθή χρήση του state, πρέπει να παρατηρούνται τρία πράγματα:

1. Το ίδιο το state δεν πρέπει να τροποποιείται άμεσα, μόνο μέσω της συνάρτησης setState(). Το μόνο σημείο όπου το state μπορεί να εκχωρηθεί, είναι στον constructor του component.
2. Οι ενημερώσεις του state μπορεί να είναι ασύγχρονες, καθώς η React μπορεί να ομαδοποιήσει πολλές κλήσεις της setState() για λόγους επίδοσης.

3. Οι ενημερώσεις του state συγχωνεύονται. Όταν καλείται η συνάρτηση setState(), η React συγχωνεύει το object που παρέχεται στο ισχύον state. Αν το component περιέχει πολλές, ανεξάρτητες μεταβλητές, μπορούν να ενημερωθούν επίσης ανεξάρτητα, καθώς οι αλλαγές σε αυτές δεν επηρεάζουν το component ως έχει.

Αξίζει να σημειωθεί πως τα components που δεν διαθέτουν κάποιο state ονομάζονται stateless. Αυτή η διευκρίνηση είναι σημαντική, διότι εντός μιας εφαρμογής σε React κανέναν component, γονέας ή παιδί, δεν είναι σε θέση να γνωρίζει αν κάποιο άλλο component είναι stateful ή stateless, καθώς το state δεν είναι διαθέσιμο σε κανέναν άλλο component πέρα από αυτό στο οποίο ανήκει και το ενημερώνει – γι' αυτόν τον λόγο, το state θεωρείται **local** ή **encapsulated**. Στις εφαρμογές σε React, η ύπαρξη ενός component ως stateful ή stateless θεωρείται απλά ως ένα χαρακτηριστικό της υλοποίησής του που μπορεί να αλλάξει σύμφωνα με τις ανάγκες. Επίσης, τα stateless components μπορούν να είναι παιδιά stateful components, και αντιστρόφως.

Έτσι, η συμπεριφορά αναμεσά τους δεν πρέπει να λαμβάνει υπόψιν αυτό το κριτήριο, αν και ένα component-γονέας έχει την επιλογή να περάσει το state του ως props στα παιδιά του. Αυτή η ροή των δεδομένων ονομάζεται **unidirectional** – μονής κατεύθυνσης ή **top-down** – «από πάνω προς τα κάτω». Καθώς το state ανήκει σε μόνο ένα component, οποιαδήποτε δεδομένα αντλούνται από αυτό μπορούν να επηρεάσουν μόνο components που είναι κατώτερα στην ιεραρχία.

Τέλος, είναι ιδιαίτερα σημαντικό, όσο αυξάνεται ο αριθμός των components μια εφαρμογής να λαμβάνεται υπόψιν η απελευθέρωση πόρων, όταν κάποια components δεν χρησιμοποιούνται πλέον. Αυτό γίνεται μέσω του **mounting** και του **unmounting** των components μέσω των **lifecycle methods**, componentDidMount() και componentWillUnmount(), οι οποίες μπορούν να εκτελέσουν κώδικα τη στιγμή που αποδίδεται ένα component και τη στιγμή που ένα component καταστρέφεται, αντίστοιχα. [26]

Πίνακας 12: Παράδειγμα λανθασμένης και ορθής διαχείρισης του state

```
// Λανθασμένη χρήση
this.state.comment = 'Hello';

// Ορθή χρήση
this.setState({comment: 'Hello'});
```

3.1.1.5 Event Handling

Το **event handling** στα React elements είναι παρόμοιο με το event handling σε DOM elements. Υπάρχουν, όμως, μερικές συντακτικές διαφορές, όπως η χρήση του camelCase για την ονομασία των event στη React και, μέσω της χρήσης της JSX, το πέρασμα μιας συνάρτησης ως event handler, αντί για μιας γραμματοσειράς. Επίσης, δεν μπορεί να επιστραφεί η τιμή false για την αποτροπή της προκαθορισμένης συμπεριφοράς του event – αυτόν τον σκοπό εξυπηρετεί η συνάρτηση preventDefault(). Γενικότερα, με την χρήση της React δεν υπάρχει η ανάγκη για κλήση της συνάρτησης addEventListener() μετά την δημιουργία ενός element, εφόσον έχει αποδοθεί με έναν listener εξαρχής. [27]

Πίνακας 13: Σύγκριση ενός τυπικού event handling της HTML και της React

```
<a href="#" onclick="console.log('The link was clicked.'); return false">
  Click me!
```

```
</a>

function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }

  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

3.1.1.6 Conditional Rendering

Με την χρήση της React, είναι δυνατή η δημιουργία διακριτών components που ενθυλακώνουν την απαιτούμενη συμπεριφορά. Σε εξάρτηση από το state, συγκεκριμένα components μπορούν να αποδοθούν, ενώ σε άλλα μπορεί να αποτραπεί η απόδοση. Αυτό ονομάζεται **conditional rendering**, και λειτουργεί με τον ίδιο τρόπο που λειτουργούν οι συνθήκες της JavaScript. Με την χρήση τελεστών της JavaScript, είναι δυνατή η δημιουργία elements που αντιπροσωπεύουν το ισχύον state και η React φροντίζει να ενημερώνει την διεπαφή έτσι ώστε να ταιριάζει.

Επίσης, είναι δυνατή η χρήση μεταβλητών για την αποθήκευση elements, κάτι που επιτρέπει την απόδοση ενός μέρους κάποιου component σύμφωνα με κάποιες συνθήκες:

Πίνακας 14: Εκχώρηση React Element σε μεταβλητή

```
if (isLoggedIn) {
      button = <LogoutButton onClick={this.handleLogoutClick} />;
    } else {
      button = <LoginButton onClick={this.handleLoginClick} />;
    }
```

Σε αυτό το σημείο, γίνεται ιδιαίτερα χρήσιμη η JSX. Για παράδειγμα, εφόσον γνωρίζουμε πως όλες οι εκφράσεις της JavaScript μπορούν γίνουν ένθετες στην JSX, συμπεριλαμβανόμενων λογικών τελεστών, μπορούμε να κάνουμε το εξής:

Πίνακας 15: Χρήση της JSX για το Conditional Rendering ενός React Element

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Hello!</h1>
      {unreadMessages.length > 0 &&
```

```
        <h2>
          You have {unreadMessages.length} unread messages.
        </h2>
      }
    </div>
  );
}
```

Για την απόκρυψη κάποιου component, ακόμη και όταν αυτό έχει αποδοθεί από κάποιο άλλο, υπάρχει η δυνατότητα επιστροφής της τιμής null:

```
function WarningBanner(props) {
  if (!props.warn) {
    return null;
  }

  return (
    <div className="warning">
      Warning!
    </div>
  );
}
```

Αξίζει να σημειωθεί πως αυτή η μέθοδος δεν θα αποτρέψει την εκκίνηση των lifecycle methods του component. [28]

3.1.1.7 Hooks

Στην React, με την έννοια των Hooks εννοούνται οι συναρτήσεις η οποίες επιτρέπουν σε components το «γάντζωμα» (εξού και η ονομασία, hook – γάντζος), του state και των lifecycle methods του – με άλλα λόγια, επιτρέπει την σύνδεση επαναλαμβανόμενης συμπεριφοράς σε components. Μέσω των Hooks, η stateful λογική μπορεί να εξαχθεί από ένα component για να χρησιμοποιηθεί αλλού, χωρίς κάποια επιρροή στην ιεραρχία των components. Οι λύσεις που χρησιμοποιούνταν στο παρελθόν για την επίλυση αυτού του προβλήματος απαιτούσαν εκτενείς επεμβατικές διαδικασίες στα components και κατέληγαν συχνά σε κάτι γνωστό ως “wrapper hell”, components εμφωλευμένα μέσα σε components διαφόρου βαθμού αφαίρεσης (render props, higher-order components, providers, consumers). Τα βασικότερα Hooks που προσφέρει οι React είναι οι useState και useEffect, αλλά οι προγραμματιστές που δουλεύουν με την χρήση της βιβλιοθήκης μπορούν να σχεδιάσουν τα δικά τους Hooks. [29]

Πίνακας 16: Παράδειγμα χρήσης ενός React Hook

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

function Example() {
```

```
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

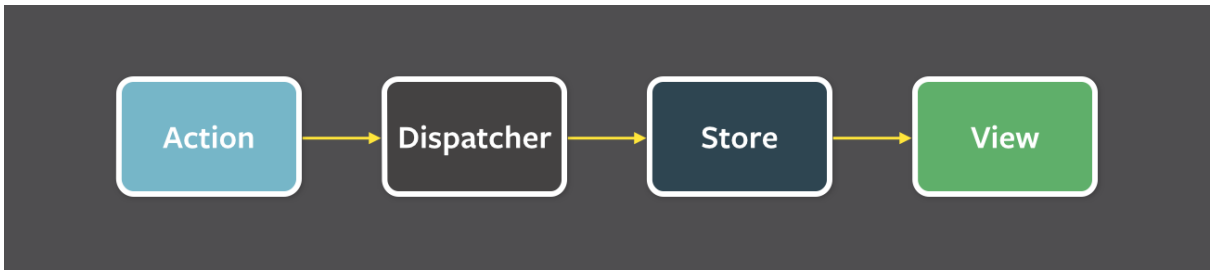
3.1.2 Flux

Εικόνα 6: Το λογότυπο της αρχιτεκτονικής Flux

Η **Flux** είναι μια αρχιτεκτονική εφαρμογών για την ανάπτυξη του client-side διαδικτυακών εφαρμογών. Παρόλο που παρουσιάστηκε στο κοινό ως αρχιτεκτονική σχετική με την βιβλιοθήκη React, δεν εξαρτάται από κάποιο τεχνολογία ή γλώσσα προγραμματισμού. Ο σκοπός ύπαρξής της είναι η επίλυση προβλημάτων που αντιμετωπίζονται σε μεγάλες και περίπλοκες εφαρμογές ανεπτυγμένες πάνω στο σχήμα MVC. Ο ιδιαίτερος τρόπος με τον οποίο συμπληρώνει την React είναι χάρη στην φύση των composable components της, εκμεταλλευόμενη την unidirectional ροή δεδομένων. Οπότε, μπορεί να θεωρείται περισσότερο ως ένα pattern και όχι ως ένα επιπλέον framework.

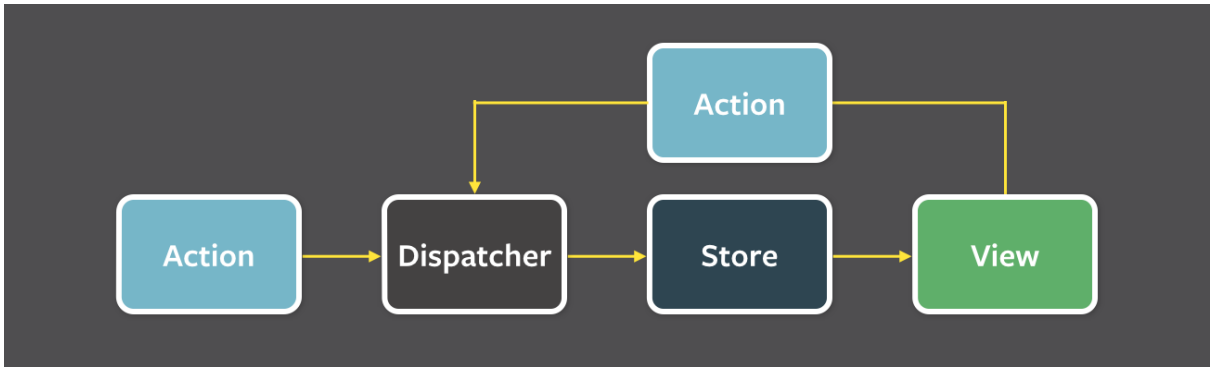
Οι εφαρμογές που ακολουθούν το σχήμα Flux έχουν τρία χαρακτηριστικά μέρη, τον «αποστολέα» (**dispatcher**), τις αποθήκες (**stores**) και τις όψεις (**views**), με τις τελευταίες να είναι, ουσιαστικά, React components. Πρέπει να δοθεί ιδιαίτερη προσοχή σε αυτό το σημείο: πρέπει να αποφευχθεί η σύγχυση με τις έννοιες model, view και controller του σχήματος MVC. Παρόλο που υπάρχουν controllers στην Flux, είναι controller-views – views που βρίσκονται την κορυφή της ιεραρχίες των components και ανακτούν δεδομένα από τα stores πριν τα περάσουν στα παιδιά τους. Επιπλέον, σαν έναν τέταρτο χαρακτηριστικό, υπάρχουν οι **action creators**, βοηθητικές μέθοδοι του dispatcher που «περιγράφουν» τις αλλαγές που είναι δυνατές στην εφαρμογή.

Η φιλοσοφία της Flux είναι η εξής: όταν ένας χρήστης αλληλοεπιδρά με μια όψη React, η όψη αυτή μεταδίδει ένα action στον κεντρικό dispatcher, ο οποίος κάνει το ίδιο προς όλα τα stores που διατηρούν την λογική και τα δεδομένα της εφαρμογής, τα οποία με την σειρά τους ενημερώνουν όλες τις όψεις που επηρεάζονται από την αλλαγή. Ιδιαίτερα χαρακτηριστική είναι η αναστροφή του ελέγχου μέσω της χρήσης των stores. Τα stores είναι αυτά που δέχονται τις ενημερώσεις, με έναν προκαθορισμένο τρόπο, και κάνουν τους απαραίτητους συμβιβασμούς, οπότε δε βασίζονται σε κάτι εξωτερικό για την σταθερή ενημέρωση των δεδομένων. Τίποτα εκτός του store δεν έχει την δυνατότητα να γνωρίζει με ποιον τρόπο διαχειρίζονται τα δεδομένα, επιτρέποντας ακόμη μεγαλύτερο διαχωρισμό των εγνοιών (separation of concerns).



Εικόνα 7: Η unidirectional ροή δεδομένων στο σχήμα Flux

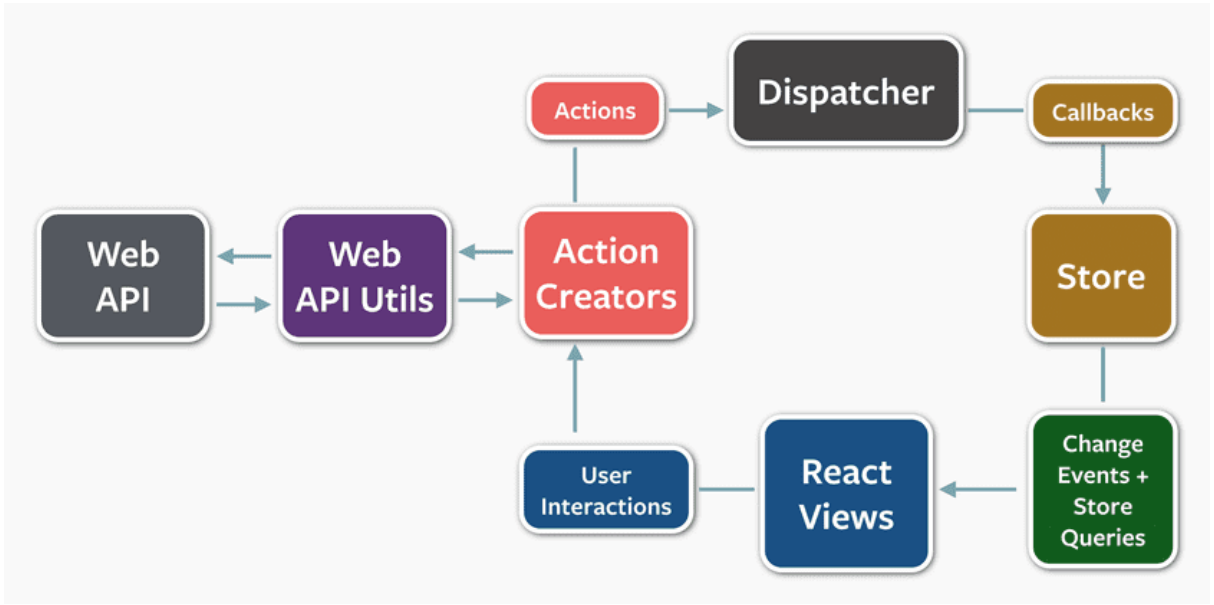
Πιο συγκεκριμένα, ο dispatcher, τα stores και τα views είναι ανεξάρτητοι κόμβοι, με ξεχωριστές εισόδους και εξόδους. Τα actions, είναι απλά αντικείμενα που περιέχουν νέα δεδομένα και ένα type με το οποίο ταυτίζονται. Όμως, τα views μπορούν να μεταδώσουν νέα actions σε ανταπόκριση στην αλληλοεπίδραση με τον χρήστη:



Εικόνα 8: Τα Views ανταποκρίνονται σε ενέργειες του χρήστη

Από το παραπάνω διάγραμμα φαίνεται πως όλα τα δεδομένα ρέουν μέσω του dispatcher, ο οποίος με την σειρά του επικαλείται της συναρτήσεις callback που έχουν καταγραφεί για κάθε store, και αυτά, ανταποκρίνονται σε αυτά τα actions τα οποία είναι σχετικά με το state που διατηρούν. Στη συνέχεια, τα controller-views ανακτούν τα δεδομένα από τα stores μέσω ενός event handler και επαναποδίδονται, μαζί με όλα τα παιδιά τους.

Αυτός ο τρόπος σκέψης για τον τρόπο λειτουργίας μιας εφαρμογής θυμίζει συναρτησιακό προγραμματισμό. [30]



Εικόνα 9: Ολοκληρωμένη απόδοση του σχήματος Flux

3.1.2.1 Dispatcher

Ο dispatcher είναι ο κεντρικός κόμβος που διαχειρίζεται ολόκληρη την ροή δεδομένων σε μια εφαρμογή Flux. Ουσιαστικά, είναι μια καταγραφή από συναρτήσεις callback για τα stores και δεν διαθέτει κάποια λογική, λειτουργώντας σαν ένας απλός μηχανισμός για την διανομή των actions προς όλα τα stores. Σε περίπτωση ανάγκης τα stores μπορούν να περιμένουν άλλα stores να ενημερωθούν πριν κάνουν το ίδιο.

Η χρησιμότητα του dispatcher είναι περισσότερο εμφανής όταν η εφαρμογή αρχίζει και μεγαλώνει. Σε μια τέτοια κατάσταση, οι εξαρτήσεις μεταξύ των stores είναι σχεδόν αναπόφευκτες: κάποιο store θα πρέπει να περιμένει την ενημέρωση κάποιου άλλου, προτού γνωρίσει τον τρόπο με τον οποίο θα ενημερωθεί το ίδιο. [31]

3.1.2.2 Stores

Τα stores περιλαμβάνουν το state και την λογική της εφαρμογής. Ο ρόλος τους είναι παρόμοιος με αυτόν του model στο σχήμα MVC, με την κύρια διαφορά πως διαχειρίζονται την κατάσταση πολλών αντικειμένων – ενός τομέα (**domain**) της εφαρμογής. Ένας επιπλέον τρόπος σύγκρισης είναι να χαρακτηριστεί ένα store ως μια συλλογή από models και συγχρόνως ένα ατομικό model από την άποψη λογικής.

Όπως προαναφέρθηκε, κάθε store καταχωρείται στον dispatcher και του παρέχει ένα callback. Αυτό το callback δέχεται κάποιο action ως όρισμα. Επιτρέποντας στα stores να αυτοενημερώνονται, η εφαρμογή απαλλάσσεται από μια πληθώρα περιπλοκών που συναντιόνται σε μεγάλες εφαρμογές σε MVC, όπως το ασταθές state. Τα ξεχωριστά μέρη μια εφαρμογής Flux είναι άκρως αποσυνδεδεμένα, με ισχυρή προσκόλληση στο Νόμο της Δήμητρας (**Law of Demeter**), την αρχή σύμφωνα με την οποία κάθε αντικείμενο εντός ενός συστήματος πρέπει να γνωρίζει ή να υποθέτει το ελάχιστο για τα υπόλοιπα αντικείμενα. Αυτό έχει αποτέλεσμα λογισμικό το οποίο συντηρείται, προσαρμόζεται, δοκιμάζεται, και εκμαθαίνεται σε νέους προγραμματιστές με πιο εύκολο τρόπο. [32]

3.1.2.3 Views και Controller-Views

Με τον όρο views χαρακτηρίζονται οι συνθέσεις των React components. Η React έχει την ιδιαιτερότητα πως παρέχει αυτό το ιδιαίτερο είδος των views που συνθέτονται και επαναχρησιμοποιούνται, που εξυπηρετούν το σχήμα Flux. Κοντά στην κορυφή της ιεραρχίας των components αυτών, βρίσκεται ένα ειδικό είδος view που «ακούει» τα events που μεταδίδονται από τα stores από τα οποία εξαρτάται. Αυτό το view ονομάζεται **controller-view**, καθώς αυτό είναι που θα ανακτήσει τα δεδομένα από τα stores και θα τα περάσει χαμηλότερα στην ιεραρχία.

Με τον πιο συνήθη τρόπο λειτουργίας, περνιέται ολόκληρο το state στα παιδιά του controller-view, ως ένα αντικείμενο, επιτρέποντας τους να χρησιμοποιήσουν το κομμάτι που χρειάζονται, διατηρώντας τα ελεύθερα από συναρτήσεις και μειώνοντας τον αριθμό των props που χρειάζονται διαχείριση. [30]

3.1.2.4 Actions

Τα actions ορίζουν τις μεθόδους τις οποίες θα επικαλεσθεί κάποιο view. Αν και το store μπορεί να διερμηνεύσει events, τα actions είναι οι κύριοι υπεύθυνοι για αυτήν τη λειτουργικότητα. Όταν λοιπόν εισάγονται νέα δεδομένα στο σύστημα, είτε μέσω της αλληλεπίδρασης με τον χρήστη, είτε μέσω κάποιο API call, αυτά τα δεδομένα

«πακετάρονται» σε ένα action – κυριολεκτικά, ένα αντικείμενο που περιέχει τα νέα πεδία με τα δεδομένα και ένα action type για την ορθή διερμηνεία του στο store.

Στην συνέχεια, ο dispatcher θα εκθέσει την μέθοδο που θα εκκινήσει την μετάδοση στα stores συμπεριλαμβάνοντας τα δεδομένα, δηλαδή το action. Όμως, επειδή σε μια εφαρμογή Flux τα stores ελέγχουν τον εαυτό τους, τα actions καταλήγουν σε αυτά μέσω callback functions και όχι setter methods. [30]

3.1.3 Redux

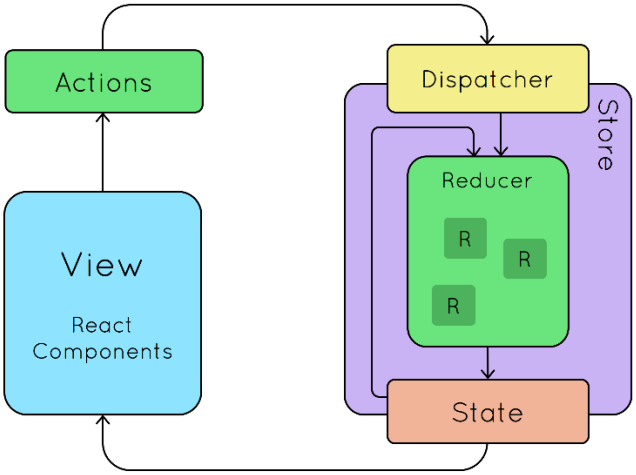
Εικόνα 10:Το λογότυπο της βιβλιοθήκης Redux

Η **Redux** είναι μια βιβλιοθήκη βασισμένη στην αρχιτεκτονική Flux, που εξυπηρετεί την διαχείριση προβλεπόμενης κατάστασης (**predictable state container**), και βοηθάει στην ανάπτυξη εφαρμογών σε JavaScript που συμπεριφέρονται σταθερά, ανεξαρτήτως τους περιβάλλοντος στο οποίο «τρέχουν» (client, server). Η Redux μπορεί να χρησιμοποιηθεί μαζί με οποιαδήποτε άλλη βιβλιοθήκη, όχι αποκλειστικά με την React. [33]

Καθώς οι απαιτήσεις των SPA σε JavaScript γίνονται ολοένα και πιο περίπλοκες, η διαχείριση του state της εφαρμογής γίνεται ολοένα και πιο δύσκολη. Η Redux αποσκοπεί να βοηθήσει τους προγραμματιστές που αναπτύσσουν τέτοιες εφαρμογές να μην χάσουν τον έλεγχο και την επίγνωση της λειτουργίας τους, όπως το πότε, γιατί και πως αλλάζει το state. Όταν ένα σύστημα καταλήγει αδιαφανές και μη ντετερμινιστικό, είναι πολύ δύσκολος ο εντοπισμός λαθών προς διόρθωση και η προσθήκη νέας λειτουργικότητας. Ακολουθώντας τα βήματα της αρχιτεκτονικής Flux, η Redux αποπειράται να κάνει αυτές τις αλλαγές στο state προβλέψιμες, επιβάλλοντας μια σειρά από περιορισμούς στο πως και πότε μπορούν να λάβουν μέρος αυτές. [34] Αυτοί οι περιορισμοί προκύπτουν από τις τρεις (3) βασικές αρχές της Redux:

1. Μια ενιαία πηγή αλήθειας: Το state ολόκληρης της εφαρμογής αποθηκεύεται σε ένα object tree εντός ενός μοναδικού store.
2. Το state είναι μόνο για ανάγνωση: Ο μόνος τρόπος για την αλλαγή του state είναι η μετάδοση ενός action, ενός αντικειμένου, δηλαδή, που περιγράφει το συμβάν, κάτι που εξασφαλίζει πως τίποτα δεν θα είναι σε θέση να επηρεάσει άμεσα το state. Επίσης, επειδή οι αλλαγές αυτές εκτελούνται σε σειρά, δεν υπάρχουν race conditions το οποία πρέπει να ληφθούν υπόψιν.
3. Οι αλλαγές εκτελούνται μόνο μέσω pure συναρτήσεων: Ο προσδιορισμός των αλλαγών στο state tree, γίνεται μόνο μέσω pure functions που ονομάζονται **reducers**. Οι reducers λαμβάνουν ως όρισμα το προηγούμενο state και το action που συνέβη, και επιστρέφουν το επόμενο state. [35]

Ανακεφαλαιώνοντας, ακολουθώντας αυτές τις αρχές, το state χαρακτηρίζεται ως ένα απλό object, το οποίο θυμίζει το model της αρχιτεκτονικής MVC, μόνο που δεν υπάρχουν setter methods. Αυτό συμβαίνει για να μην είναι δυνατές οι αυθαίρετες αλλαγές στο state. Για να αλλάξει κάτι, πρέπει πρώτα να αποσταλεί ένα action, το οποίο είναι ένα απλό object. Αυτή η επιβολή της παρουσίασης των αλλαγών ως actions επιτρέπει την εύκολη αναγνώριση των συμβάντων εντός της εφαρμογής. Τέλος, για την ένωση των actions με το state, μια συνάρτηση, ο reducer, τα δέχεται ως ορίσματα και επιστρέφει το νέο state. [36]



Εικόνα 11: Μια σφαιρική εικόνα της ροής των δεδομένων με την χρήση της Redux

3.1.4 Next.js

Εικόνα 12: Το λογότυπο της Next.js

Η **Next.js** είναι ένα framework της JavaScript που επιτρέπει την δημιουργία **server-side rendered** εφαρμογές. Το server-side rendering είναι μια δημοφιλής τεχνική για την απόδοση client-side single page applications στον εξυπηρετητή, πριν αποσταλεί η πλήρως κατασκευασμένη ιστοσελίδα στον πελάτη. Το πιο δελεαστικό πλεονέκτημα αυτής της τεχνικής είναι πως μια εφαρμογή μπορεί «ανιχνευτεί» για το περιεχόμενό της από **web crawlers** - διαδικτυακά bots που συστηματικά εξερευνούν το Παγκόσμιο Διαδίκτυο, κυρίως για τον σκοπό της ευρετηρίασής του – πριν ακόμη αποδοθεί, κάτι που βελτιώνει σε μεγάλο βαθμό το **SEO** της εφαρμογής (**S**earch **E**ngine **O**ptimization), ενώ μπορεί επίσης να παρέχει μεταδεδομένα (**metadata**) για αυτήν. Πέρα από αυτά, η Next.js προσφέρει επίσης ένα διαισθητικό σύστημα δρομολόγησης που χτίζεται στην πλευρά του πελάτη, αυτόματα βελτιστοποιεί τα στατικά σημεία των σελίδων της εφαρμογής και αυτόματα χωρίζει τον κώδικα για την εξυπηρέτηση γρηγορότερων επαναφορτόσεων των σελίδων.

3.1.5 Material-UI

Εικόνα 13: Το λογότυπο της βιβλιοθήκης Material-UI

Η **Material-UI** είναι μια open-source βιβλιοθήκη διεπαφής χρήστη που παρέχει React components τα οποία εφαρμόζουν την γλώσσα σχεδιασμού **Material Design** της Google. Ο σχεδιασμός της διεπαφής χρησιμοποιεί πλέγματα, ανταποκριτικά animations και μεταβάσεις, padding και εφέ που προσδίδουν βάθος, όπως φωτισμοί και σκιές.

3.2 Τεχνολογίες Back-end

Από άποψη προγραμματισμού, το back-end είναι ο κώδικα που τρέχει στον εξυπηρετητή μια διαδικτυακής εφαρμογής, ο οποίος δέχεται αιτήματα από τους πελάτες και χρησιμοποιώντας την προγραμματισμένη λογική του, στέλνει πίσω τις κατάλληλες απαντήσεις. Ουσιαστικά, ο εξυπηρετητής είναι ο υπολογιστής που τρέχει την εφαρμογή. Το back-end επίσης, περιλαμβάνει τη βάση δεδομένων, η οποία διατηρεί όλα τα δεδομένα της εφαρμογής.

3.2.1 Node.js

Εικόνα 14: Το λογότυπο της Node.js

Για ένα πολύ μεγάλο χρονικό διάστημα, ένας περιηγητής ήταν το μόνο περιβάλλον στο οποίο μπορούσε να τρέξει κώδικας γραμμένος σε JavaScript. Γι' αυτό, οι προγραμματιστές που ήθελαν να αναπτύξουν κάποια ιστοσελίδα έπρεπε να χρησιμοποιήσουν διαφορετικές γλώσσες προγραμματισμού για το front-end και το back-end. Παρόλο που υπήρξαν κάποιες απόπειρες για την δημιουργία περιβαλλόντων για της εκτέλεση κώδικα JavaScript, η Node.js ήταν η πρώτη που έγινε δημοφιλής και σήμερα χρησιμοποιείται από πολλές κορυφαίες εταιρίες, όπως η Netflix, η PayPal και η Uber. Η Node.js χαρακτηρίζεται ως JavaScript **runtime** ή αλλιώς, ένα περιβάλλον που επιτρέπει την εκτέλεση κώδικα JavaScript εκτός του περιηγητή. Ουσιαστικά, ένα runtime μετατρέπει κώδικα υψηλού επιπέδου σε κώδικα χαμηλού επιπέδου τον οποίο μπορεί να τρέξει ένας υπολογιστής.

Αν και η Node.js δημιουργήθηκε με σκοπό την ανάπτυξη εξυπηρετητών και διαδικτυακών εφαρμογών με την χρήση της JavaScript, μπορεί να χρησιμοποιηθεί και για την ανάπτυξη εφαρμογών για desktop περιβάλλον.

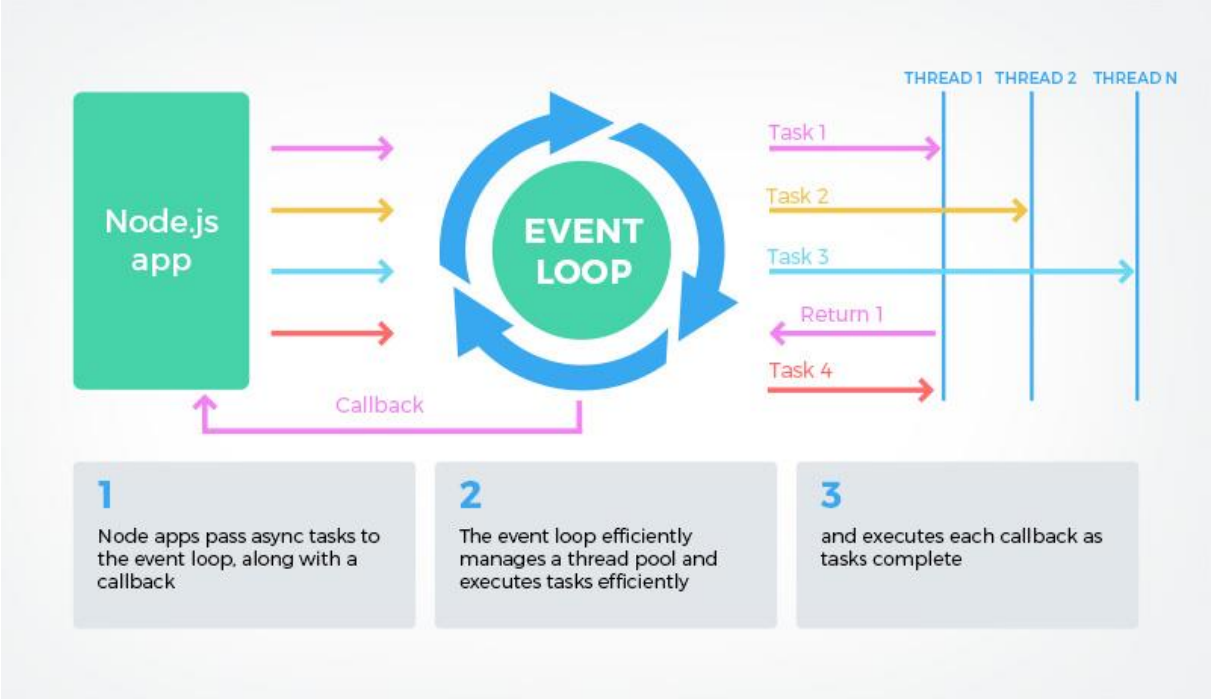
Ένα ιδιαίτερα χρήσιμο χαρακτηριστικό της Node.js είναι η δομοστοιχειώτητά της (**modularity**). Ως modularity χαρακτηρίζεται η τεχνική σχεδιασμού λογισμικού στην οποία, ένα πρόγραμμα έχει ξεχωριστά και διακριτά μέρη τα οποία προμηθεύουν από ένα κομμάτι της ολικής λειτουργικότητας της εφαρμογής. Με την πάροδο του χρόνου, αυτή η έννοια γίνεται ολοένα και πιο ουσιώδης για την δημιουργία επεκτάσιμων εφαρμογών που ενσωματώνουν χρήσιμες βιβλιοθήκες και frameworks και διαχωρίζουν τις έγνοιες σε μικρότερα, περισσότερο διαχειρίσιμα κομμάτια.

Ένα **module** είναι ένα κομμάτι κώδικα που βρίσκεται σε ένα ξεχωριστό αρχείο. Αντί να εμπεριέχεται ολόκληρος ο κώδικας της εφαρμογής σε ένα αρχείο, οργανώνεται σε ξεχωριστά αρχεία και όπου χρειάζεται συνδυάζεται μέσω της συνάρτησης require().

Η Node.js διαθέτει κάποια βασικά modules για την εξυπηρέτηση των αναγκών των προγραμματιστών που την χρησιμοποιούν, το μεγαλύτερο πλεονέκτημα που προσφέρει είναι η δυνατότητα χρήσης **third-party modules**. Χάρη στην χρήση κώδικα ανεπτυγμένου από άλλους προγραμματιστές, κάτι που σήμερα αποτελεί συντελεστικό παράγοντα στην δημιουργία απαιτητικών εφαρμογών, δεν είναι αναγκαία η ανάλωση χρόνου και προσπάθειας για την εφεύρεση μηχανισμών, κάθε φορά που υπάρχει η ανάγκη για την επέκταση της λειτουργικότητας μιας εφαρμογής, διότι, ίσως ήδη να υπάρχουν modules που να εξυπηρετούν αυτές τις ανάγκες. Για τις ανάγκες αυτής της πτυχιακής, και κυρίως για μέρη της λειτουργικότητας της εφαρμογής που δεν ήταν στο εύρος του αντικειμένου ενασχόλησης, χρησιμοποιήθηκαν μια σειρά modules, γνωστών και ως **packages**, τα οποία αναλύονται σε μεγαλύτερο βαθμό στο υποκεφάλαιο της ροής εργασίας.

Η Node.js χαρακτηρίζεται από το γεγονός πως ακολουθεί μια **event-driven** αρχιτεκτονική. Στον παραδοσιακό προστακτικό προγραμματισμό, ο προγραμματιστής δίνει στον υπολογιστή μια σειρά από εντολές για να εκτελέσει, σε μια προκαθορισμένη σειρά. Όμως, δημιουργώντας μια διαδικτυακή εφαρμογή, πρέπει να υπάρχει η δυνατότητα να διαχειρίζεται καταστάσεις χωρίς να είναι γνωστό πότε θα λάβουν μέρος αυτές, όπως κάποια δράση του χρήστη.

Η ανάπτυξη του server-side μιας διαδικτυακής εφαρμογής περιλαμβάνει την υλοποίηση λειτουργιών που συνήθως χρειάζονται χρόνο: ανάγνωση αρχείων ή αποστολή ερωτημάτων στη βάση δεδομένων. Η ιδιαιτερότητα της Node.js στην διαχείριση αυτής της λειτουργικότητας, διαφέρει από τις τυπικές προσεγγίσεις του παρελθόντος, όπως παύσεις στην εκτέλεση του κώδικα μέχρι την ολοκλήρωση των διαδικασιών ή τη χρήση πολλαπλών νημάτων (threads). Ο σχεδιασμός της Node.js εφαρμόζει έναν «βρόχο συμβάντων» - **event loop**, όπως αυτό που χρησιμοποιεί η JavaScript όταν τρέχει σε έναν περιηγητή. Το event loop επιτρέπει την εκτέλεση ασύγχρονων actions, που διαχειρίζονται με **non-blocking** τρόπο, δηλαδή οι υπόλοιπες διεργασίες δεν είναι αναγκασμένες να περιμένουν την ολοκλήρωση κάποιας άλλης για να κάνουν τη δική τους εργασία.



3.2.2 Express.js

Εικόνα 15: Το λογότυπο της Express.js

Η **Express.js** είναι ένα framework για, *de facto*, την Node.js – αν και, τυπικά, λειτουργεί εντός της Node.js ως module – και εξυπηρετεί την ανάπτυξη του back-end διαδικτυακών εφαρμογών. Στην πράξη, η δημιουργία ενός back-end από την αρχή είναι μια αρκετά απαιτητική διαδικασία και παρόλο που ένας προγραμματιστής θα μπορούσε να σχεδιάσει ένα REST API μόνο με την χρήση της Node.js, η Express.js επιτρέπει την ανάπτυξη ενός σε πολύ λιγότερο χρόνο και πολύ λιγότερο εκτενή κώδικα, κάτι που διευκολύνει την συντήρηση και επέκτασή του.

Η Express, παρά το γεγονός πως είναι ένα framework – και η λέξη framework υποδεικνύει ένα βαθμό ακαμψίας, λόγω της εκ φύσεως δογματικής προσέγγισης στην επίλυση προβλημάτων – δεν περιορίζει τον προγραμματιστή σε κάποια βέλτιστη πρακτική, αλλά του προσφέρει την ευελιξία και την προσαρμοστικότητα για τον σχεδιασμό της δικής του λύσης. Είναι ιδιαίτερα λιτή, με μεγάλο κομμάτι του εύρους της λειτουργικότητάς της να είναι διαθέσιμη, όπως απαιτείται, σε μορφή plugins.

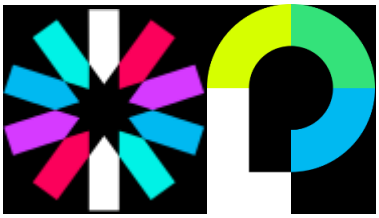
3.2.3 PostgreSQL

Εικόνα 16: Το λογότυπο της RDBMS PostgreSQL

Η **PostgreSQL** (αρχικά γνωστή ως **Postgres**), θεωρείται ως η πιο ισχυρή και ανεπτυγμένη open-source, αντικειμενοσχεσιακή βάση δεδομένων βασισμένη στην SQL. [37] Αποσκοπεί στην ασφαλή διατήρηση των δεδομένων κάθε είδους εφαρμογής, προσφέροντας δυνατότητες επέκτασης ανεξάρτητα από τον βαθμό πολυπλοκότητας των δεδομένων. Επίσης, φημίζεται για την αρχιτεκτονικής της, την αξιοπιστία της, ανθεκτικότητα στα λάθη, αλλά ιδιαίτερα για την υποστήριξη περίπλοκων ερωτημάτων SQL.

Η Postgres χαρακτηρίζεται από την αποδοτικότητά της στην σύγχρονη διαχείριση πολλαπλών εργασιών (**concurrency**), κάτι που καταφέρνει χωρίς την χρήση read locks, μέσω της μεθόδου **Multiversion Concurrency Control – MVCC**, η οποία εγγυάται την ατομικότητα των εργασιών της, την συνέπειά τους, την απομόνωσή τους καθώς και την μονιμότητά τους, που σημαίνει πως συμμορφώνεται με το σύνολο ιδιοτήτων **ACID** (**A**tomicity, **C**onsistency, **I**solation, **D**urability), που εγγυάται την εγκυρότητα της βάσης δεδομένων σε περίπτωση λαθών ή τεχνικών βλαβών. [38]

3.2.4 Passport και JWT



Εικόνα 17: Τα λογότυπα των JWT και Passport

Το **Passport** είναι ένα middleware – λογισμικό το οποίο συνδέει τα αιτήματα που παράγονται από το front-end μιας εφαρμογής, με απαίτηση κάποια δεδομένα που βρίσκονται στο back-end, εξού και η ονομασία. Χρησιμοποιείται σε εφαρμογές ανεπτυγμένες πάνω στο framework της Express.js και επιτρέπει την σύνδεση των χρηστών σε αυτές με την χρήση διαφόρων μηχανισμών διαπιστευτηρίων, γνωστών ως **strategies**. Το μέγεθος του Passport δεν επιβαρύνει με κάποιο αισθητό τρόπο την λειτουργίας της εφαρμογής και μπορεί να τροποποιηθεί για να εξυπηρετεί τις ιδιαίτερες απαιτήσεις της.. Επίσης, προσφέρει την δυνατότητα εγγραφής και σύνδεσης των χρηστών, με χρήση των στοιχείων τους από άλλες υπηρεσίες, όπως οι λογαριασμοί του στις υπηρεσίες των Google, Facebook, Amazon και Microsoft, χωρίς την χρήση των passwords τους εντός της εφαρμογής, μέσω της εφαρμογής του ανοιχτού standard **OAuth**.

Το **JWT** (**J**SON **W**eb **T**oken) είναι ένα ανοιχτό standard για την ασφαλή μετάδοση πληροφορίας, σε μορφή αρχείου JSON. Ένα JWT αποτελείται από τρία μέρη: το **header** (κεφαλή), το **payload** (φορτίο) και το **signature** (υπογραφή). Στην κεφαλή ορίζεται το είδος του αρχείου JSON, που προφανώς είναι JWT, καθώς και ο αλγόριθμος κρυπτογράφησης της πληροφορίας (π.χ., RSA, HS252, HMAC SHA256). Στο φορτίο βρίσκονται τα δεδομένα που μεταδίδονται και, τέλος, στην υπογραφή, διατηρείται το άθροισμα της κωδικοποιημένη κεφαλής, του φορτίου και ενός «μυστικού» προσωπικού κλειδιού. Η υπογραφή εξασφαλίζει την ασφάλεια της μεταδιδόμενης πληροφορίας. Αν και η κεφαλή και τα δεδομένα θα μπορούσαν να αποκρυπτογραφηθούν, δεν ισχύει το ίδιο για την υπογραφή: αν τροποποιηθούν η κεφαλή και τα δεδομένα, ενώ το κλειδί δεν είναι σύμφωνο με αυτό του αναμενόμενου αποστολέα, η διαδικασία επαλήθευσης αποτυγχάνει και έτσι αποτρέπεται η μη εξουσιοδοτημένη πρόσβαση στα δεδομένα.

Εικόνα 18: Παράδειγμα παραγόμενου JWT από το header, payload και signature ενός αρχείου JSON

4. Σχεδιασμός και Υλοποίηση Εργασίας

Έχοντας μελετήσει όλα τα εργαλεία, τεχνολογίες και αρχιτεκτονικές που θα χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής, ως επόμενο βήμα έπρεπε να σχεδιαστεί ο τρόπος λειτουργίας της εφαρμογής. Επειδή για του εκπονητές της εργασίας, πολλές πτυχές της υλοποίησης ήταν σχετικά ανεξερεύνητο πεδίο, η διαδικασία ανάπτυξης της λειτουργικότητας της εφαρμογής απαιτούσε φάσεις πειραματισμού και βελτιστοποίησης. Υπάρχουν τόσες διαφορετικές λύσεις για τις απαιτήσεις μιας εφαρμογής, και το πεδίο των πρακτικών στη βιομηχανία τόσο μεταλλασσόμενο, που δεν έχουν καθιερωθεί ακόμη απόλυτα βέλτιστες πρακτικές· η σωστή και ομαλή λειτουργικότητα μιας εφαρμογής είναι σε μεγάλο βαθμό στην κρίση του προγραμματιστή.

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, λόγω την αποσύνδεσης των client-side και server-side της εφαρμογής, αναπτύχθηκαν δύο ξεχωριστά projects: το swotlog για την υλοποίηση του front-end και το swotlog-api για την υλοποίηση του back-end.

Το όνομα της εφαρμογής, Swotlog, είναι συνδετικός όρος από τις αγγλικές λέξεις swot και log. Η έκφραση “to swot” είναι μια άτυπο βρετανικό ρήμα που σημαίνει «μελετώ εντατικά», ενώ το “log” μεταφράζεται ως «βιβλίο καταγραφής». Άρα, η ονομασία μπορεί να είναι κατανοητή σαν «καταγραφή εντατικής μελέτης», καθώς αυτόν τον σκοπό θα εξυπηρετεί για τους χρήστες της.

4.1 Ροή εργασίας

Η ροή εργασίας μιας ομάδας προγραμματιστών περιλαμβάνει μια σειρά από στάδια. Μέχρι αυτό το κεφάλαιο, έχει ολοκληρωθεί το στάδιο της **ανάλυσης** και του **προσδιορισμού των απαιτήσεων** της εφαρμογής, καθώς και αυτό της **τεχνολογικής προσέγγισης**. Τα επόμενα στάδια περιλαμβάνουν τον **σχεδιασμό και υλοποίηση της βάσης δεδομένων**, τον **σχεδιασμό και την ανάπτυξη του back-end** της εφαρμογής, **τον σχεδιασμό και την υλοποίηση του front-end** και την ενσωμάτωση των δύο κατά την ανάπτυξή τους.

Για μια αποδοτική, οργανωμένη και μη επιβαρυμένη από μηδαμινές δραστηριότητες ροή εργασίας, οι μοντέρνοι προγραμματιστές χρησιμοποιούν μια σειρά από εργαλεία που υποστηρίζουν το έργο τους, ακόμη και όταν αυτό δεν συνδέεται άμεσα με το κομμάτι του προγραμματισμού. Στην περίπτωση της παρούσας εργασίας, μεταξύ άλλων, χρησιμοποιήθηκε η εφαρμογή **Trello** για την διαχείριση των μερών της ερευνάς, της συλλογής πηγών και υλικού, της σύνταξης, της ανάπτυξης της εφαρμογής και για την αναθέτηση έργου μεταξύ των εκπονητών. Για τον σχεδιασμό των σκαριφημάτων χρησιμοποιήθηκε η εφαρμοργή **Balsamiq Mockups**, επιτρέποντας έτσι στους εκπονητές, αφού συμφωνήσουν για την εμφάνιση και τον σχεδιασμό της εφαρμογής, να επικεντρωθούν στο πιο κρίσιμι κομμάτι, δηλαδή την ανάπτυξη. Στην συνέχεια αναλύονται μερικά εργαλεία που επηρέασαν σε μεγάλο βαθμό αυτή την διαδικασία.

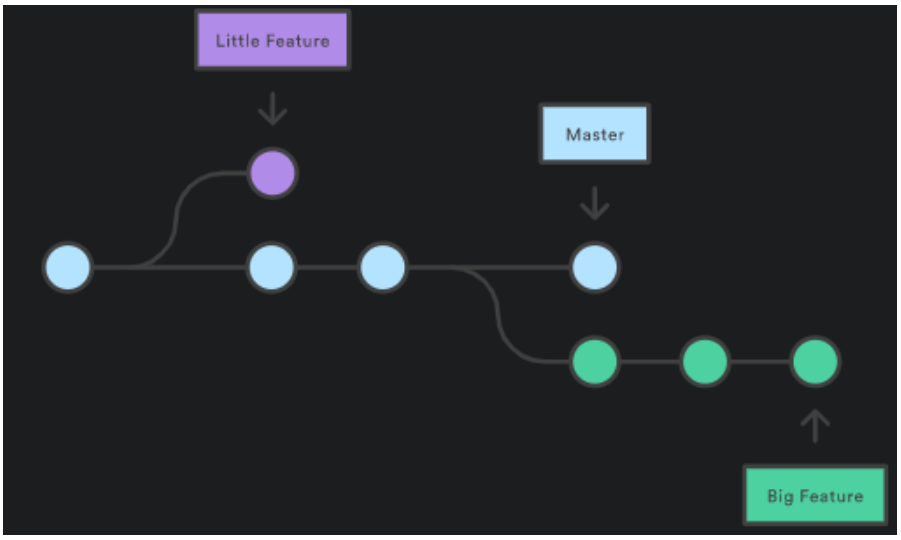
4.1.1 Version Control

Τα συστήματα **version control** είναι μια ειδική κατηγορία λογισμικού που επιτρέπει σε μια ομάδα μηχανικών λογισμικού, όπως στην περίπτωση των εκπονητών της εργασίας, να διαχειρίζονται τις αλλαγές που γίνονται στον πηγαίο κώδικα κατά την διάρκεια της ανάπτυξης της εφαρμογής. Ακόμα και σε περίπτωση λαθών, είναι δυνατή η αναίρεση των αλλαγών και η επαναφορά της πιο πρόσφατης, λειτουργικής έκδοσης του λογισμικού. Η ανάπτυξη λογισμικού χωρίς την χρήση κάποιου **VCS** (**V**ersion **C**ontrol **S**ystem) δεν είναι συνετή, διότι κατά την ανάπτυξη μιας μεγάλης εφαρμογής, η έλλειψη των αντιγράφων

ασφάλειας και της καταγραφής των εκδόσεων του λογισμικού μπορεί να αποδειχθεί μοιραία, σε περίπτωση που χαθεί ο πηγαίος κώδικας. Για μια ομάδα προγραμματιστών, ο πηγαίος κώδικας είναι η πολυτιμότερη περιουσία τους. Η χρησιμότητά του είναι τόσο μεγάλη, που μπορεί να χρησιμοποιηθεί και για έργα που δεν σχετίζονται με λογισμικό, όπως η σύνταξη αυτής της πτυχιακής εργασίας.

Πέρα από την δυνατότητα της λεπτομερής καταγραφής όλων των αλλαγών στον πηγαίο κώδικα, τα VCS επιτρέπουν την καταγραφή λαθών (bug tracking). Επίσης, επιτρέπουν την διακλάδωση (branching) του πηγαίου κώδικα, δίνοντας την δυνατότητα στους προγραμματιστές να αναπτύξουν καινούργια χαρακτηριστικά της εφαρμογής τους, χωρίς να εισβάλουν στον ήδη λειτουργικό και σταθερό κομμάτι της. Όταν τα νέα χαρακτηριστικά εγκριθούν για την συμβατότητα και σταθερότητά τους, τότε, αυτές οι εντατικές αλλαγές μπορούν να συγχωνευτούν με τον κύριο κλάδο (master branch). Ο πηγαίος κώδικας διατηρείται σε ένα **repository** (αποθήκη) και μέσω της ενεργειών γνωστών ως **pull** (από την πλευρά του αποδέκτη της ενημέρωσης και **push** (από την πηγή της ενημέρωσης), γίνεται την αντιγραφή της νεότερης αναθεώρησης του κώδικα στο τοπικό περιβάλλον ανάπτυξης. Για την εισαγωγή αλλαγών, χρησιμοποιείται η ενέργεια **commit**.

Αναμφισβήτητα, το δημοφιλέστερο VCS αυτήν την περίοδο είναι το **Git**. Πρόκειται για ένα open source project που αναπτύχθηκε αρχικά από τον Linus Torvalds. Επιπλέον, για το hosting της εφαρμογής χρησιμοποιήθηκε η υπηρεσία **Github**.



Εικόνα 19: Διάγραμμα κλάδων ανάπτυξης πηγαίου κώδικα με την χρήση VCS

4.1.2 Dependencies

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, dependencies ονομάζονται τα ξεχωριστά προγράμματα τα οποία είναι απαραίτητα για την λειτουργία της εφαρμογής. Αυτά τα προγράμματα μπορούν να μοιραστούν σε δύο γενικές κατηγορίες: τα κανονικά **dependencies** από τα οποία εξαρτάται κάποια λειτουργικότητα της εφαρμογής, και τα **dev dependencies**, προγράμματα τα οποία υποστηρίζουν την διαδικασία ανάπτυξης του λογισμικού.

Στην παρούσα εργασία χρησιμοποιήθηκαν dependencies και από τις δύο κατηγορίες. Αυτά ήταν:

Dev Dependencies:

- **nodemon**: αυτό το εργαλείο επανεκκινεί αυτόματα την εφαρμογή, όταν αυτή βρίσκεται στο στάδιο ανάπτυξης, κάθε φορά που εντοπίζονται αλλαγές στον κώδικα.

Έτσι, ο προγραμματιστής μπορεί να δει άμεσα τις αλλαγές που εισάγει να αποδίδονται στην εφαρογμή.

- **babel**: ένας μεταφραστής της JavaScript που είναι συμβατή με τους νεότερους προσδιορισμούς της – ECMAScript2015+ - για την προσαρμογή της σε παλαιότερες εκδόσεις, έτσι ώστε να είναι αναγνώσιμη από διαφορετικά προγράμματα περιήγησης, καθώς και παλαιότερες εκδόσεις αυτών.

Dependencies:

- **bcrypt:** μια βιβλιοθήκη συναρτήσεων που επιτρέπει τον κρυπτογραφικό κατατεμαχισμό των passwords που εισάγουν οι χρήστες κατά την εγγραφή τους.
- **morgan:** αυτό το middleware επιτρέπει την καταγραφή λεπτομερειών σχετικών με τα αιτήματα που καταφθάνουν στην εφαρμογή. Ιδιαίτερα χρήσιμο κατά την ανάπτυξη του API στη Node.js και με την χρήση του framework Express.js.
- **pg:** ο client για την σύνδεση με την βάση δεδομένων PostgreSQL. Ακολουθεί την ίδια non-blocking λειτουργία που εφαρμόζει η Node.js.
- **body-parser:** ακόμη ένα middleware επιτρέπει την ανάλυση του body κάθε αιτήματος.
- **cors:** το CORS (Cross-Origin Resource Sharing) είναι ένας μηχανισμός που επιτρέπει την πρόσβαση στο περιεχόμενο μιας ιστοσελίδας από ένα domain εκτός αυτού στο οποίο βρίσκεται το περιεχόμενο. Το συγκεκριμένο middleware επιτρέπει τον ορισμό, όχι μόνο ποια αιτήματα μπορούν να έχουν πρόσβαση στο περιεχόμενο, αλλά επίσης και με ποιο τρόπο.
- **redux-saga**: ένα middleware για την διαχείριση παράπλευρων ενεργειών στο πλαίσιο της Redux. Για την συνοχή του κειμένου και την καλύτερη κατανόηση μιας αρκετά περίπλοκης έννοιας, η ανάλυση του γίνεται σε μεγαλύτερο βαθμό στην περιγραφή της λειτουργικότητας του front-end.

4.2 Σχεδιασμός και Ανάπτυξη Back-end

4.2.1 Σχεδιασμός και Ανάπτυξη Βάσης Δεδομένων

Σαν πρώτο βήμα για την υλοποίηση του back-end της εφαρμογής, έπρεπε να σχεδιαστεί η βάση δεδομένων, ώστε να εξυπηρετεί τις ανάγκες τις εφαρμογής. Εντός των πλαισίων της, ορίστηκαν τις εξής βασικές οντότητες:

- person: η βασική οντότητα του χρήστη της εφαρμογής. Στον πίνακα αυτόν αποθηκεύονται οι βασικές πληροφορίες του κάθε χρήστη, όπως το ονοματεπώνυμό του και η διεύθυνση ηλεκτρονικού ταχυδρομείου του. Η επιλογή της λέξης person για την ονομασία της προκύπτει από το γεγονός πως η λέξη user είναι δεσμευμένη κωδικολέξη της Postgres. [39]
- students: η οντότητα αυτή χαρακτηρίζει τους χρήστες που διατηρούν αποκλειστικά την ιδιότητα του φοιτητή, και ο κάθε χρήστης έχει την αντίστοιχη αναφορά του στον πίνακα αυτόν.
- tutor: στον πίνακα αυτόν αναφέρονται οι χρήστες οι οποίοι, επιπλέον, επέλεξαν να εμφανίζονται ως πρόθυμοι για την βοήθεια συναδέλφων τους μέσω την επίλυσης απορίων.
- class: η οντότητα για τα μαθήματα τα οποία μπορεί να παρακολουθήσει ο κάθε χρήστης της εφαρμογής.
- class_tutor: σε αυτόν τον πίνακα διατηρούνται οι αναφορές στα μαθήματα στα οποία κάποιος χρήστης έχει δηλωθεί ως πρόθυμος βοηθός.

- class_student: στον πίνακα αυτόν βρίσκονται οι αναφορές μεταξύ των φοιτητών και των μαθημάτων που παρακολουθούν, καθώς και η κατάσταση επιτυχίας τους σε αυτό.
- post: το σύνολο των αναρτήσεων των χρηστών, με αναφορές, είτε στο μάθημα στο οποίο ανήκουν, είτε στην ομάδα χρηστών μεταξύ των οποίων γίνονται οι αναρτήσεις αυτές.
- discourse: η οντότητα αυτή εκφράζει τις ομαδικές συζητήσεις μεταξύ των χρηστών/φοιτητών.
- comment: το σύνολο των σχολίων που γίνονται στα posts, με αναφορές στον χρήστη που τα δημιούργησε και στην συζήτηση στην οποία ανήκουν.
- likes: τα "likes" των χρηστών που έγιναν σε comments και αναφέρονται σε αυτά.
- follow: ο πίνακας που διατηρεί τις σχέσεις "follow" μεταξύ των χρηστών.
- groups: οι ομάδες που δημιουργούν οι χρήστες, με αναφορά στον δημιουργό τους.
- group_person: ο πίνακας που διατηρεί τις σχέσεις μεταξύ χρηστών και ομάδων.
- task: οι εργασίες που αναθέτονται στα μέλη μιας ομάδας.

Παρακάτω, παρουσιάζονται οι εντολές SQL οι οποίες χρησιμοποιήθηκαν για την δημιουργία των απαραίτητων πινάκων, έτσι ώστε να είναι δυνατή η ανάκτηση των πληροφοριών μέσω του API της εφαρμογής.

Πίνακας 17: Οι εντολές SQL για την υλοποίηση της βάσης δεδομένων της εφαρμογής

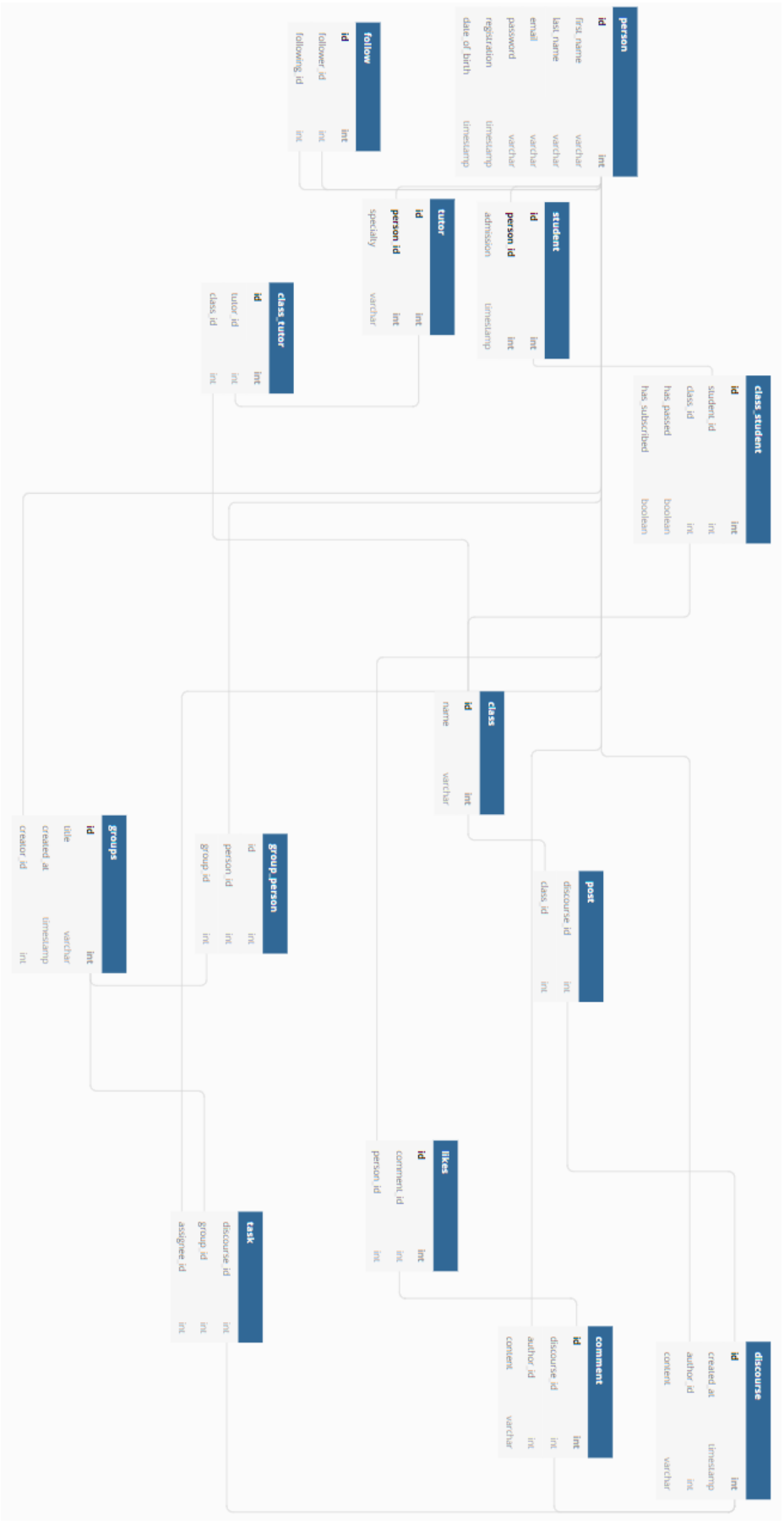
```
CREATE TABLE person (
  id SERIAL PRIMARY KEY,
  first_name varchar,
  last_name varchar,
  email varchar,
  password varchar NOT NULL,
  registration timestamp DEFAULT now(),
  date_of_birth timestamp,
  UNIQUE (email)
);

CREATE TABLE tutor (
  id SERIAL PRIMARY KEY,
  person_id int REFERENCES person (id) ON DELETE cascade NOT NULL,
  specialty varchar,
  UNIQUE (person_id)
);

CREATE TABLE student (
  id SERIAL PRIMARY KEY,
  person_id int REFERENCES person (id) ON DELETE cascade NOT NULL,
  admission timestamp,
  UNIQUE (person_id)
```

```
);

CREATE TABLE class (
  id SERIAL PRIMARY KEY,
  name varchar NOT NULL,
  UNIQUE (name)
);

CREATE TABLE class_tutor (
  class_id int REFERENCES class (id) NOT NULL,
  tutor_id int REFERENCES tutor NOT NULL,
  CONSTRAINT class_tutor_id PRIMARY KEY (class_id, tutor_id)
);

CREATE TABLE class_student (
  class_id int REFERENCES class (id) NOT NULL,
  student_id int REFERENCES student (id) NOT NULL,
  has_passed BOOLEAN,
  has_subscribed BOOLEAN,
  CONSTRAINT class_student_id PRIMARY KEY (class_id, student_id)
);

CREATE TABLE discourse (
        id serial PRIMARY KEY,
        created_at timestamp DEFAULT NOW(),
        author_id int REFERENCES person(id),
        content varchar NOT NULL
);

CREATE TABLE post (
        discourse_id int REFERENCES discourse(id),
        class_id int REFERENCES class(id),
        CONSTRAINT post_id PRIMARY KEY (discourse_id)
);

CREATE TABLE comment (
  id SERIAL PRIMARY KEY,
  discourse_id int REFERENCES discourse (id) NOT NULL,
  author_id int REFERENCES person (id) NOT NULL,
  content varchar
);

CREATE TABLE likes (
  id SERIAL,
  comment_id int REFERENCES comment (id) NOT NULL,
```

```
  person_id int REFERENCES person (id) NOT NULL,
  CONSTRAINT comment_like_id PRIMARY KEY (person_id, comment_id)
);

CREATE TABLE follow (
  id SERIAL,
  follower_id int REFERENCES person (id) NOT NULL,
  following_id int REFERENCES person (id) NOT NULL,
  constraint follow_connection_id PRIMARY KEY (follower_id, following_id)
);

CREATE TABLE groups (
  id SERIAL PRIMARY KEY NOT NULL,
  title varchar NOT NULL,
  created_at timestamp DEFAULT now(),
  creator_id int REFERENCES person (id) NOT NULL
);

CREATE TABLE group_person (
        id SERIAL,
        person_id int REFERENCES person (id) NOT NULL,
        group_id int REFERENCES groups (id) NOT NULL,
        CONSTRAINT
);

CREATE TABLE task (
        discourse_id int REFERENCES discourse (id) NOT NULL,
        group_id int REFERENCES groups (id) NOT NULL,
        assignee_id int REFERENCES person (id),
        CONSTRAINT task_id PRIMARY KEY (discourse_id)
);
```

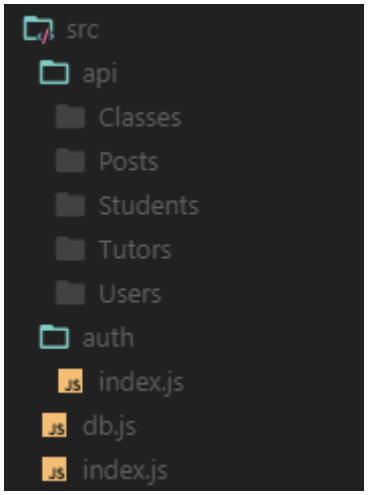
Η απόφαση για την χρήση ξεχωριστών πινάκων για την απόδοση των σχέσεων μεταξύ tutors και students με τα classes, προκύπτει από τα γεγονός πως πρόκειται για σχέσεις N-N, οπότε, η χρήση ξεχωριστών πινάκων για την αποθήκευση αυτών των σχέσεων καθιστά την βάση δεδομένων ευκολότερη στην διάσχιση και περισσότερο δομοστοιχειωτή, άρα επεκτάσιμη για το μέλλον.

Εικόνα 20: Το διάγραμμα των οντοτήτων της βάσης δεδομένων



4.2.1 Σχεδιασμός και Ανάπτυξη API

Η βασική δομή των αρχείων του API είναι η εξής:



Εικόνα 21: Η δομή των αρχείων του Web API

Η εφαρμογή τρέχει από το αρχείο index.js, όπου εισάγονται όλα τα απαραίτητα modules της Node.js, όπως τα dependencies/και middleware που θα χρησιμοποιηθούν. Αυτό το μοτίβο επαναλαμβάνεται σε όλα τα μέρη της εφαρμογής που χρειάζονται κάποια εξωτερική λειτουργικότητα. Ο κύριος ρόλος αυτού του αρχείου είναι η προετοιμασία του περιβάλλοντος της εφαρμογής και της δρομολόγησης όλων των αιτημάτων που θα δεχτεί κατά την διάρκεια της λειτουργίας της ακούγοντας στο port το οποίο έχει οριστεί στον εξυπηρετητή.

Για την καλύτερη περιγραφή της λογικής του API, παρακάτω εξετάζεται αυτό το κομμάτι της που είναι υπεύθυνο για την δημιουργία ενός νέου χρήστη. Αρχικά, όμως, ακολουθεί η δομή του φακέλου, όπου διατηρείται, σε ξεχωριστό αρχείο ο κάθε ένας, ο κώδικας ο οποίος διαχειρίζεται όλα τα αιτήματα που ανήκουν στην έννοια του χρήστη:

Εικόνα 22: Τα modules που σχετίζονται με την λογική του χρήστη

Φαίνεται έτσι πως έχουμε χωρίσει την λογική στο κομμάτι υπεύθυνο για την δημιουργία ενός νέου χρήστη στη βάση δεδομένων, στην διαγραφή του χρήστη και στην ανάκτηση του χρήστη καθώς και των μαθημάτων τα οποία παρακολουθεί και τα οποία έχει περάσει από την βάση δεδομένων, και, τέλος, για τη σύνδεσή του στη εφαρμογή (που θα εξετάσουμε περαιτέρω στην υποενότητα του μηχανισμού authorization/authentication) Το δικό του αρχείο index.js είναι απλά υπεύθυνο για την συλλογή αυτών των στοιχείων και την εξαγωγή τους για την πρόσβαση σε αυτά από το υπόλοιπο μέρος της εφαρμογής, όπως φαίνεται εδώ:

Πίνακας 18Ο τρόπος με τον οποίο ένα αρχείο index.js συλλέγει και εξάγει modules

```
import get from './get';
import del from './delete';
import create from './create';

export {
  get,
  del,
  create,
};
```

Εντός του κεντρικού αρχείου index.js του API, τα modules που δημιουργήθηκαν «απαιτούνται» μέσω της συνάρτησης require και δίνοντας της ως όρισμα την διαδρομή του αρχείου για το οποίο ενδιαφερόμαστε να αποθηκευτεί στην μεταβλητή. Αξίζει να σημειωθεί πως, σύμφωνα με το παρακάτω παράδειγμα, η διαδρομή api/Users είναι για την Node.js η ίδια με το api/Users/index.js, οπότε δεν υπάρχει ανάγκη να διευκρινίσουμε το αρχείο αυτό. Εξού, όμως, η χρήση ενός ξεχωριστού αρχείο index για την εξαγωγή των modules.

Πίνακας 19: Απαίτηση module

```
const Users = require('api/Users');
```

Στην συνέχεια, ας παρατηρήσουμε την συνάρτηση με την οποία θα μπει σε κίνηση η διαδικασία δημιουργίας του νέου χρήστη

Πίνακας 20: Παράδειγμα αιτήματος HTTP σε Node.js

```
app.post('/users/create', Users.create);
```

Στην μεταβλητή **app** έχουμε αποθηκεύσει το module της Express.js, η οποία διαθέτει το δικό της module **post** για την διαχείριση HTTP Requests του τύπου POST. Αυτό, δέχεται σαν ορίσματα το URL της εφαρμογής στο οποίο θα έρθει το αίτημα, καθώς και το module που πρέπει να χρησιμοποιηθεί. Εξετάζοντας την λογική του module, βήμα βήμα:

Πίνακας 21:Παράδειγμα εισαγωγής modules με και χωρίς destructuring

```
import { pool } from 'db';
import bcrypt from 'bcrypt';
```

Αρχικά, εισάγουμε τα δύο modules που θα χρειαστούν για την διεκπεραίωση αυτού του αιτήματος, **db** και **bcrypt**. Όπως αναφέρθηκε και στην υποενότητα της ροής εργασίας, το bcrypt θα είναι υπεύθυνο για την κρυπτογράφηση του κωδικού που θα ορίσει ο χρήστης, ενώ μέσω του db, διαχειριζόμαστε το pooling των συνδέσεων στη βάση δεδομένων. Αξίζει να σημειωθεί πως αν κάποιο module εξάγει περισσότερες από μία συναρτήσεις, ο προγραμματιστής έχει την δυνατότητα να κάνει **destructuring** με την χρήση αγκίστρων (import { pool } from 'db')– να εισάγει μεμονωμένες συναρτήσεις, χωρίς την ανάγκη

φόρτωσης ολόκληρου του module, κάτι ιδιαίτερα χρήσιμο όταν πρόκειται για ένα που είναι μεγάλο σε μέγεθος. Περαιτέρω, αυτό επιτρέπει την μεμονωμένη κλήση της κάθε συνάρτησης που εισάγεται με αυτόν τον τρόπο:

Πίνακας 22: Η λογική του module db.js

```
import { Pool } from 'pg';

const config = {
  user: process.env.DATABASE_USER,
  host: process.env.HOST,
  database: process.env.DATABASE,
  password: process.env.DATABASE_PASSWORD,
  port: process.env.DATABASE_PORT,
};

const pool = new Pool(config);

pool.on('connect', () => {
  console.log('Connected to Database!');
});

pool.on('error', (err, client) => {
  console.error('Unexpected error on idle client', err);
  process.exit(-1);
});

export {
  pool,
};
```

Όπως φαίνεται, εισάγουμε το module **pg**, που χρησιμοποιείται για την επικοινωνία μεταξύ εφαρμογών ανεπτυγμένων με τη Node.js και της βάσης δεδομένων Postgres.

Στην μεταβλητή config αποθηκεύεται η δομή στην οποία διατηρούνται οι απαραίτητες πληροφορίες για το περιβάλλον στο οποίο τρέχει η βάση δεδομένων, αλλά και των credentials που θα επιτρέψουν την επιτυχή σύνδεση: host, port, η ονομασία της βάσης, του χρήστη που έχει δικαιώματα πρόσβασης στη βάση δεδομένων και το password της.

Στις διαδικτυακές εφαρμογές όπου πιθανότατα θα πραγματοποιούνται πολλαπλά ερωτήματα στη βάση δεδομένων, συνιστάται η χρήση ενός pool (συγκέντρωσης) συνδέσεων [40], διότι η Postgres μπορεί να διαχειρίζεται,, εξ' ορισμού, μόνο ένα ερώτημα τη φορά ανά σύνδεση, οπότε και καλύπτεται αυτή η ανάγκη. Με αυτή τη μέθοδο, εξασφαλίζεται η σταθερή σύνδεση του API στη βάση δεδομένων:

Εξετάζοντας πάλι την λογική στο αρχείο index.js, το API χτίζει μια δομή από το body του αιτήματος, για την σωστή μορφοποίηση των δεδομένων πριν αυτά αποσταλθούν στη βάση δεδομένων:

Πίνακας 23: Δόμηση των δεδομένων εντός του αιτήματος

```
const create = (req, res) => {
  const {
    firstName,
    lastName,
    email,
    password,
  } = req.body;
```

Στη συνέχεια, εδραιώνεται η σύνδεση με την βάση δεδομένων και πραγματοποιείται μια σειρά από validations των δεδομένων που στάλθηκαν. Σε περίπτωση που αυτά είναι εσφαλμένα, επιστρέφονται οι ανάλογες αποκρίσεις και κωδικοί κατάστασης:

Πίνακας 24: Εδραίωση σύνδεσης με τη βάση δεδομένων

```
pool.connect((err, client, done) => {
    if (err) throw err;

    if (!email || !password) {
      done();

      return res.status(400).send('Required field is missing.');
    }

    if (typeof password !== 'string' || password.length <= 4) {
      done();

      return res.status(400).json({
        error: 'Password length should be at least 4 characters',
      });
    }
```

Στο επόμενο βήμα, και εφόσον έχει εδραιωθεί η σύνδεση, στέλνονται τα ερωτήματα στην βάση δεδομένων. Όπως φαίνεται στο παρακάτω κομμάτι κώδικα, το πρώτο ερώτημα, ανάλογα με την απάντησή του ή έλλειψη αυτής, επιτρέπει τον έλεγχο για την προϋπαρξη κάποιο χρήστη με την ίδια ηλεκτρονική διεύθυνση, η οποία πρέπει να είναι μοναδική ανά χρήστη. Αν η διεύθυνση δεν είναι ήδη εγγεγραμμένα, κωδικοποιείται το password με την χρήση του bcrypt και εκτελείται άλλο ένα ερώτημα, που αποσκοπεί στη εισαγωγή των νέων δεδομένων στον πίνακα person και αμέσως μετά άλλο ένα που προσθέτει την εισαγωγή αυτή στον πίνακα student. Αν η διαδικασία είναι επιτυχής, η απόκριση περιέχει το αντίστοιχο κωδικό κατάστασης και ένα μήνυμα επιβεβαίωσης. Αξίζει να σημειωθεί πως σε κάθε βήμα της λογικής, έχουν ληφθεί οι κατάλληλες προφυλάξεις για την απόδοση των πιθανών λαθών που μπορούν να εμφανιστούν:

Πίνακας 25: Εκτέλεση ερωτημάτων προς τη βάση δεδομένων

```
client.query(
      'SELECT * from person\
      WHERE email = $1',
      [email],
      (error, results) => {
        if (error) throw error;

        if (results.rowCount) {
          res.status(409).send('User already exists');
          done();

          return;
        }

        bcrypt.hash(password, 10, (_err, hashedPassword) => {
          if (_err) {
            done();

            return res.status(500).json({
              error: 'Something went wrong',
            });
          }

          client.query(
            `INSERT INTO person
            (first_name, last_name, email, password)
             VALUES
             ($1, $2, $3, $4) RETURNING id`,
            [
              firstName,
              lastName,
              email,
              hashedPassword
            ], (_error, _results) => {
              if (_error) {
                throw _error;
              }
              const id = _results.rows[0].id;

              client.query(
                `INSERT INTO student (person_id) values ($1)`,
                [id],
                (__err, _) => {
                  if (__err) {
```

```
                    throw __err;
                  }
                }
              );
            }
          );
        });

        done();
        res.status(201).json({
          message: 'User signed up!',
        });
      }
    );
  });
};

export default create;
```

Έχοντας αναλύσει αυτόν τον μηχανισμό, είναι κατανοητή η γενική γραμμή του σχεδιασμού και της υλοποίησης του back-end, καθώς όλα τα μέρη του API ακολουθούν την ίδια λογική.

4.2.1 Σχεδιασμός και Ανάπτυξη Μηχανισμού Authentication και Authorization

Για την υλοποίηση της εργασίας, αναπτύχθηκε ένα απλό σύστημα πιστοποίησης των χρηστών, με βάση την ηλεκτρονική τους διεύθυνση και το προσωπικό password τους, με την χρήση του middleware Passport και των JWT, όπως αναφέρθηκε προηγουμένως. Παρακάτω, γίνεται η ανάλυση αυτού του μηχανισμού, ο οποίος όμως δεν είναι αποκλειστικός για αυτή την μέθοδο πιστοποίησης, καθώς μπορεί να χρησιμοποιηθεί και με άλλες στρατηγικές.

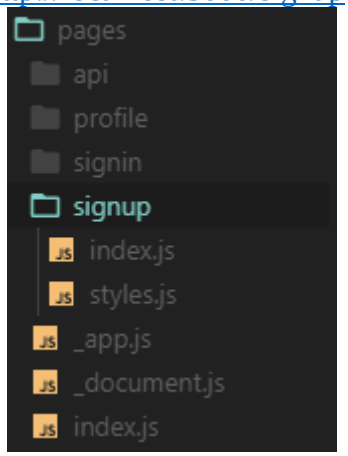
Σε γενικές γραμμές, ο μηχανισμός λειτουργεί ως εξής:
Όταν ο χρήστης συνδέεται στην εφαρμογή, το back-end της επιστρέφει ένα «υπογεγραμμένο» (βλέπε υποενότητα **3.2.4 Passport και JWT**) token στην απόκρισή του.
Το client-side της εφαρμογής αποθηκεύει αυτό το token τοπικά, και το αποστέλλει ξανά στο back-end κάθε φορά που γίνεται κάποιο αίτημα που απαιτεί πιστοποίηση. Έτσι, όλα τα αιτήματα περνούν μέσω του middleware, το οποίο είναι υπεύθυνο για τον έλεγχο των tokens και επιτρέπει την εκτέλεση του αιτήματος μόνος όταν τα tokens επαληθευτούν. Στο παρακάτω παράδειγμα φαίνεται η πρόσδεση του middleware στο framework της Express και η εφαρμογή του σε όλες τις μεθόδους HTTP:

Πίνακας 26: Ενσωμάτωση του middleware Passport

```
app.use(passport.initialize());

const initializeAuth = require('./auth/index').default;

initializeAuth(passport);

app.all('*', passport.authenticate('jwt', { session: false }));
```

4.3 Σχεδιασμός και Ανάπτυξη του Front-end

Όπως αναφέρθηκε στο προηγούμενα κεφάλαιο, το front-end της εφαρμογής αναπτύχθηκε πάνω στο React framework της Next.js. Σε αυτό, όλες οι σελίδες της εφαρμογής βρίσκονται σε έναν φάκελο, pages, και έτσι το framework είναι σε θέση να κάνει την δρομολόγηση σε αυτή. Για παράδειγμα, για την πρόσβαση στην σελίδα signup, αρκεί η πρόσθεση του ονόματος που δώσαμε στην ιστοσελίδα στο URL της εφαρμογής: στο τοπικό περιβάλλον ανάπτυξης, θα ήταν http://localhost:3000/signup:



Εικόνα 23: Το file structure του front-end

Ας σημειωθεί ξανά πως η διαδρομή pages/signup/index.js είναι ισοδύναμη με την διαδρομή pages/signup. Ακολουθεί η ανάλυση των γενικών αρχείων σε αυτή τη δομή:

- app.js: ο πυρήνας της λειτουργίας της εφαρμογής. Σε αυτό το αρχείο περιέχεται η βασική αρχικοποίηση της ιστοσελίδας, πάνω στην οποία χτίζονται εξ ορισμού όλα τα επιπλέον στοιχεία της. Αν και η Next.js παρέχει το δικό της βασικό app.js σε αυτήν την εφαρμογή έχει δημιουργηθεί ξεχωριστό, για την υπερίσχυση του βασικού. Αυτό επιτρέπει μια σειρά από λειτουργίες όπως η διατήρηση της δομής και του state της εφαρμογής, παρά τις αλλαγές των σελίδων από την πλευρά του πελάτη.
- document.js: όπως και στην περίπτωση του app.js, το document.js είναι επίσης βασικό στοιχείο της Next.js επί του οποίου έγινε η επιλογή για υπερίσχυση. Το συγκεκριμένο αρχείο αποδίδεται μέσω server-side rendering, οπότε δεν αποδίδεται στην πλευρά του πελάτη. Εξυπηρετεί την επαύξηση των tags που βρίσκονται στην εφαρμογή, για παράδειγμα, υποστηρίζοντας server-side rendering για τις βιβλιοθήκες που χρησιμοποιούνται για το styling των components.
- index.js: αυτό το αρχείο, ουσιαστικά, λειτουργεί σαν την πρώτη σελίδα στην οποία έχει πρόσβαση ο χρήστης, με μια σημαντική λεπτομέρεια: σε περίπτωση που το σενάριο χρήσης περιλαμβάνει τον χρήστη ως guest, θα του προβάλει το view που προορίζεται για χρήστες που δεν είναι συνδεδεμένοι/εγγεγραμμένοι (με prompts προς την σύνδεση και την εγγραφή), και σε περίπτωση που ο χρήστης έχει ήδη μια εν δυνάμει συνεδρία, θα του προβάλει το προσωπικό του feed.

Στην συνέχεια, εξετάζεται η γενική περίπτωση της υλοποίησης του μέρους που είναι υπεύθυνο για την εγγραφή του χρήστη, άρα η σελίδα signup/index.js.

Πίνακας 27: Εισαγωγή modules για τη σελίδα signup

```
import { connect } from 'react-redux';
import Avatar from '@material-ui/core/Avatar';
import Button from '@material-ui/core/Button';
import TextField from '@material-ui/core/TextField';
import Grid from '@material-ui/core/Grid';
import SchoolIcon from '@material-ui/icons/School';
import Typography from '@material-ui/core/Typography';
import Link from 'components/Link';
import { signUp } from 'actions/user';
import useStyles from './styles';
```

Αρχικά, ας μελετηθούν οι παραπάνω εισαγωγές των modules:

- **{ connect }/react-redux**: Η συνάρτηση connect() εξυπηρετεί την σύνδεση React Components σε stores της Redux. Πιο συγκεκριμένα, προμηθεύει το συνδεδεμένο Component με τα δεδομένα που χρειάζεται από το store, καθώς και τις συναρτήσεις που μπορεί να χρησιμοποιήσει για να αποστείλει actions στο store.
- **Avatar, Button, TextField, Grid, SchoolIcon, Typography/@material-ui**: Όλα αυτά είναι components που μας προμηθεύει το Material-UI:
    - **Avatar**: ένα βασικό avatar για εικόνες, γράμματα και εικονίδια.

    - **Button**: το τυπικό κουμπί που συναντάται σχεδόν σε όλες τις διαδικτυακές εφαρμογές.
    - **TextField**: το συνηθές πεδίο κειμένου για την εισαγωγή δεδομένων από τον χρήστη.
    - **Grid**: Στο Material Design, το Grid είναι μια από τις βασικότερες έννοιες – η αποκριτική διάταξη της ιστοσελίδας που προσαρμόζεται αυτόματα στο μέγεθος και στον προσανατολισμό της οθόνης, αποδίδοντας έτσι τον σχεδιασμό της ιστοσελίδας με συνεπή και σταθερό τρόπο.
    - **SchoolIcon**: ένα απλό εικονίδιο, σε χρήση για λόγους αισθητικής, που συμμορφώνεται με την γλώσσα του Material Design.
    - **Typography**: μια από τις πιο χρήσιμες σχεδιαστικές δυνατότητες του Material-UI, το Typography προσαρμόζει το περιεχόμενο ενός component, εισάγοντας μια τυπογραφική κλίμακα. Έτσι, διατηρείται μια συνοχή μεταξύ των styles και των μεγεθών των fonts για την βέλτιστη σχεδιαστική απόδοση.
- **Link/components**: το Link είναι ένα Component που χτίστηκε για τις ανάγκες τις εφαρμογής. Όλα τα Components που δημιουργήθηκαν διατηρούνται σε δικό τους ξεχωριστό φάκελο.
- **{ singUp }/actions/user**: ένα από τα Actions που αντιστοιχούν στα σενάρια του user. Όλα τα actions βρίσκονται εντός του store.
- **useStyles/./styles**: στο αρχείο styles.js που συνοδεύει το index.js της σελίδας signup, εμπεριέχεται η μορφοποίηση της ιστοσελίδας. Παρόλο που είναι αρχείο JavaScript, εμπεριέχει απλό CSS εντός μιας συνάρτησης. Αυτό το προγραμματιστικό παράδειγμα ονομάζεται **CSS-in-JS** (CSS μέσα σε JavaScript) και επιτρέπει μεγαλύτερο έλεγχο στο styling ενός component. Ας σημειωθεί πως ο συμβολισμός ./ (τελεία και κάθετος) υποδεικνύει πως η θέση του αρχείο βρίσκεται στον ίδιο φάκελο με το αρχείο index.js – θυμίζει την πλοήγηση σε ένα περιβάλλον Unix!

Πίνακας 28: Αρχειοθέτηση του state του signup και η συνάρτηση διαχείρισης του submit

```
const SignUp = ({ signUp }) => {
  const classes = useStyles();
  const [data, setData] = React.useState({
    firstName: '',
    lastName: '',
    email: '',
    password: '',
  });

  const handleSubmit = event => {
    event.preventDefault()
    signUp(data);
  }
}
```

Όπως φαίνεται στο παραπάνω παράδειγμα, το state της σελίδας αρχικοποιείται και περιέχει όλα τα πεδία που θα πρέπει να διαχειριστεί η εφαρμογή για την εκτέλεση της

εγγραφής – firstName, lastName, email, password. Το επόμενο κομμάτι κώδικα διαχειρίζεται το event, δηλαδή τί θα συμβεί την στιγμή που ο χρήστης προσπαθήσει να κάνει submit τα δεδομένα που έχει εισάγει. Εδώ ας παρατηρηθεί, πως το action που εισάχθηκε πριν δέχεται ως όρισμα τα δεδομένα στο state.

Στην συνέχεια, εξετάζεται ένα από τα πολυάριθμα components που αποδίδονται εντός της σελίδας, συγκεκριμένα το TextField που χρησιμοποιείται για την εισαγωγή της ηλεκτρονικής διεύθυνσης του χρήστη:

Πίνακας 29: Η μελέτη ενός component της Material-UI

```
<TextField
      variant="outlined"
      required
      fullWidth
      id="email"
      label="Email Address"
      name="email"
      autoComplete="email"
      type="email"
      value={data.email}
      onChange={e => setData({
         ...data,
         [e.target.id]: e.target.value
      })}
/>
```

Εντός του component ορίζουμε μια σειρά από props που παρέχει το API του TextField:

- το **variant** καθορίζει την παραλλαγή του component (στο Material-UI, πολλά components έχουν πολλαπλά variants),
- το **required** καθορίζει απαραίτητη την εισαγωγή δεδομένων από τον χρήστη (κάτι προφανές, καθώς δεν είναι δυνατή η εγγραφή χωρίς την εισαγωγή της ηλεκτρονικής διεύθυνσης, όπως φάνηκε στην ανάπτυξη του API της εφαρμογής),
- το **fullwidth** επιτρέπει στο component να καταλάβει ολόκληρο τον χώρο του container στο οποίο βρίσκεται,
- το **id** είναι ακριβώς αυτό, το id του element,
- το **label** ορίζει το περιεχόμενο,
- το **name** είναι το name attribute του element,
- το **autoComplete** επιτρέπει την χρήση του Autofilling της HTML για την ευκολότερη συμπλήρωση μιας φόρμας, [41]
- το HTML5 input **type**,
- το **value** ορίζει την τιμή που περιέχει το element και,
- το **onChange** είναι η συνάρτηση που διαχειρίζεται τις αλλαγές στο περιεχόμενο:
    - **…data**: Στην React.js, και συγκεκριμένα στην JSX, οι τρείς τελείες ονομάζονται **spread operator**. Με αυτήν την σύνταξη, είναι δυνατό το «άπλωμα» των ιδιοτήτων, εκεί που αναμένονται. Ουσιαστικά, η

σύνταξη …data είναι ισοδύναμη με: data.firstName, data.lastName, data.email, data.password, αλλά είναι δυναμική, οπότε συμπεριλαμβάνονται όλα τα πεδία, χωρίς να χρειάζεται να αναφερθούμε σε αυτά ιδιαιτέρως.

- **[e.target.id]: e.target.value**: ανακτά την τιμή του input στο οποίο αντιστοιχεί το id στο οποίο έγινε η κλήση του event.

Ας αναλύσουμε τώρα την λειτουργικότητα για την εγγραφή, υπό την εμβέλεια του front-end. Όταν ο χρήστης κάνει submit την φόρμα, καλείται το action του signUp και δίνονται ως όρισμα τα δεδομένα. Με την χρήση του Redux, όταν ενεργοποιείται ένα action, αλλάζει το state της εφαρμογής. Όμως, όταν συμβαίνει αυτό μπορεί να υπάρχει κάποια δράση που θα θέλαμε να κάνουμε και που χρειάζεται δεδομένα που πηγάζουν από το state της εφαρμογής, αλλά δεν είναι άμεσα συνδεδεμένη με αυτό, όπως μια ασύγχρονη κλήση. Στη λειτουργικότητα που περιγράφουμε, αυτή η δράση είναι η αποστολή ενός αιτήματος HTTP στο API της εφαρμογής, που θα αναλάβει την εκτέλεση της λογικής για την εγγραφή του χρήστη όπως είδαμε στο προηγούμενο κεφάλαιο. Σε αυτό θα μας χρησιμεύσει το **redux-saga**.

Πίνακας 30: Εισαγωγή ενός Redux Saga

```
import  { SIGN_UP } from '.';

export const signUp = data => ({
  type: SIGN_UP,
  data,
});
```

Στο παραπάνω παράδειγμα γίνεται η εισαγωγή του saga υπεύθυνου για την διαχείριση αυτής της παράπλευρης δράσης. Εντός του index.js των sagas, η συνάρτηση έχει την παρακάτω μορφή:

Πίνακας 31: Η λογική του Redux Saga

```
function* signUp({ data }) {
        try {
                const done = yield call(api.doFetch, {
                        internalRoute: '/auth/signup',
                        data
                });

                yield put({
                        type: SIGN_UP_SUCCESS,
                        message: done.message
                })

                Router.push('/signin');
        } catch (error) {
                yield put({ type: ERROR_MESSAGE, message: error });
        }
```

```
}
```

Εδώ φαίνεται πως δρομολογούνται τα δεδομένα μέσω του μηχανισμού πιστοποίησης του χρήστη, διότι με την εγγραφή του χρήστη, προφανώς θα εδραιωθεί η συνεδρία του με την εφαρμογή. Εντός αυτού του αρχείου, η συνάρτηση που είναι τελικά υπεύθυνη για την αποστολή του αιτήματος προς το API, έχει την εξής μορφή:

Πίνακας 32: Αποστολή ενός async call προς το API της εφαρμογής

```
export default async (req, res) => {
  let response;

  const request = await fetch(`http://192.168.1.106:3000/users/create`, {
                method: 'POST',
                headers: {
                        'Content-Type': 'application/json; charset=utf-8',
    },
                body: JSON.stringify({
      ...req.body
                }),
  });

  response = await normalizeResponse(request)

  res.status(request.status).json(response);
}
```

Σε αυτό το στάδιο, βλέπουμε τον τρόπο αλληλεπίδρασης του front-end με το back-end, κλείνοντας έτσι τον κύκλο που ξεκινήσαμε να διαγράφουμε από το προηγούμενο κεφάλαιο.

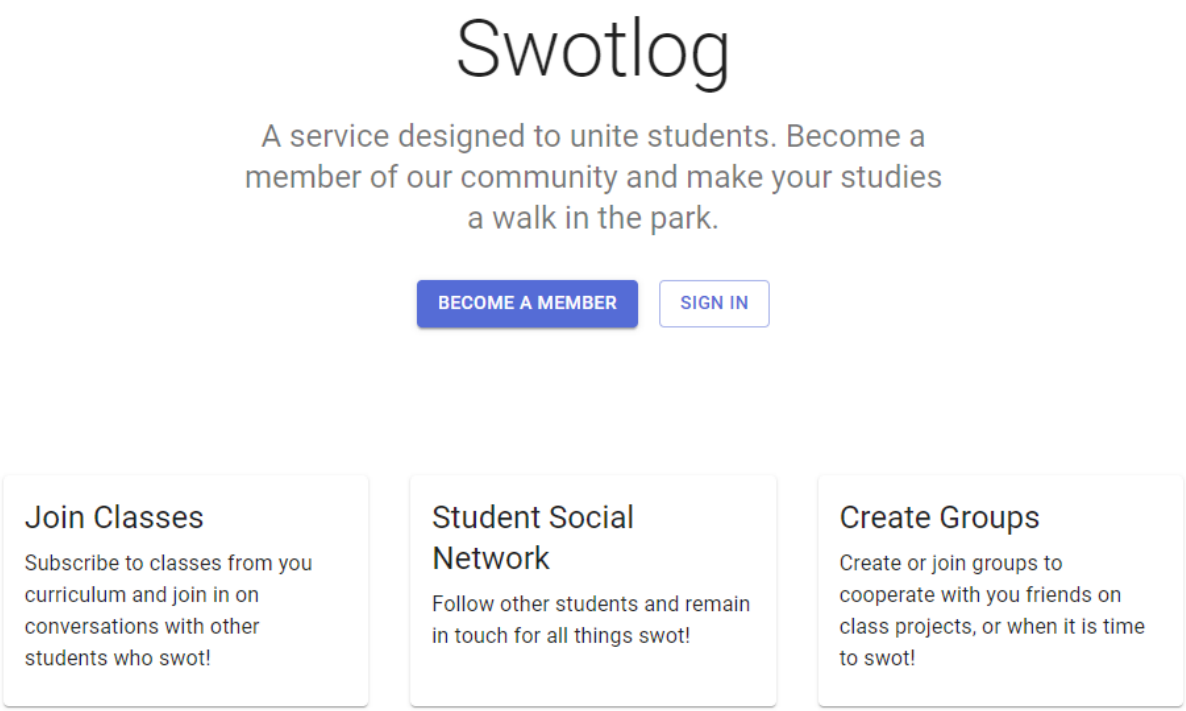
Όλα τα μέρη του front-end, λιγότερο και περισσότερο περίπλοκα στην υλοποίησή τους ακολουθούν την δομή και την λογική που αναφέρθηκε σε αυτό το κεφάλαιο.

5. Σενάρια Χρήσης

Όλοι οι χρήστες της εφαρμογής χωρίζονται σε δύο γενικές κατηγορίες: τους χρήστες – επισκέπτες (guests) και του χρήστες – μέλη της κοινωνικής πλατφόρμας. Σε αυτό το κεφάλαιο θα γίνει η εξέταση των σεναρίων χρήσης για την κάθε ομάδα χρήστη, και στην περίπτωση του χρήστη μέλους, η χρήση όλων των λειτουργιών της εφαρμογής.

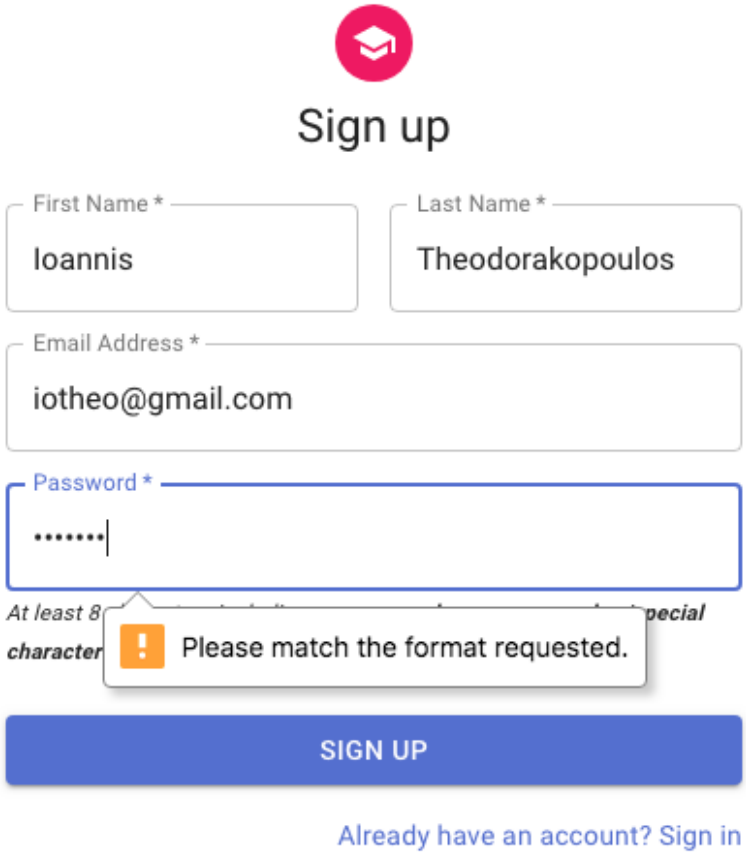
5.1 Πλοήγηση Χρήστη "Guest" – Δημιουργία Προφίλ

Με την είσοδό του στην εφαρμογή, ο χρήστης – επισκέπτης θα βρεθεί στην κεντρική σελίδα, όπου, πέρα από μια βασική παρουσίαση της υπηρεσίας, υπάρχουν οι επιλογές εγγραφής και σύνδεσης στην εφαρμογή.



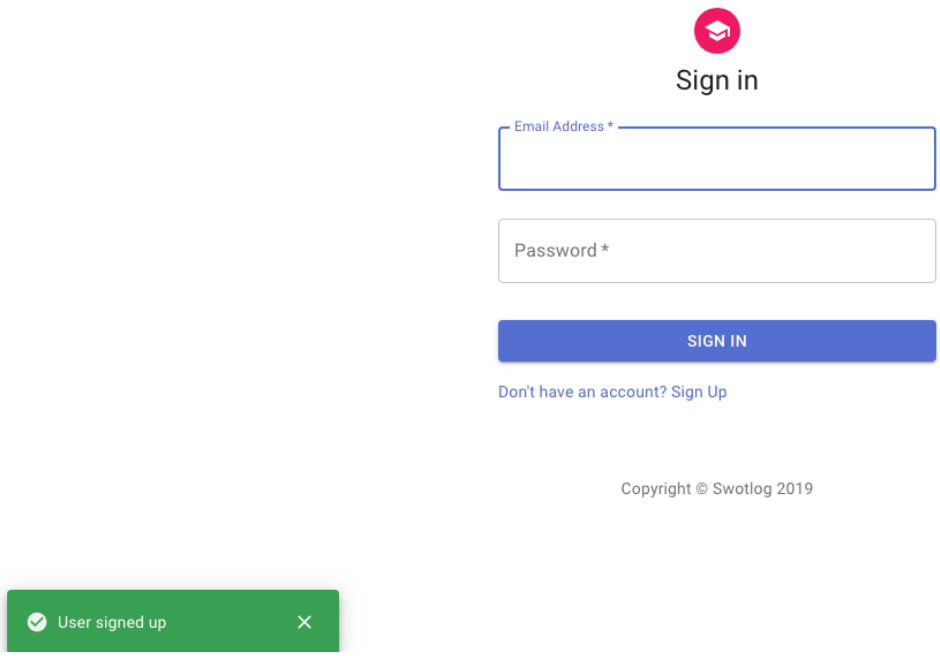
Εικόνα 24:Η κεντρική σελίδα της υπηρεσίας κοινωνικής δικτύωσης

Για τις ανάγκες του παρόντος σεναρίου, θεωρείται πως ο guest δεν διαθέτει λογαριασμό στην υπηρεσία, οπότε μονόδρομος είναι η εγγραφή του σε αυτή, μέσω του κουμπιού "Become a Member". Ο επισκέπτης θα μεταφερθεί στην σελίδα όπου μπορεί να κάνει την εγγραφή του εισάγοντας τις απαραίτητες πληροφορίες. Σε περίπτωση που κάποιο εισαχθέν δεδομένο δεν είναι στην απαιτούμενη μορφή, η εφαρμογή θα δείξει τα ανάλογα προειδοποιητικά μηνύματα.



Εικόνα 25: Η σελίδα εγγραφής - ο χρήστης - επισκέπτης εισάγει τα στοιχεία του

Μετά την επιτυχή εγγραφή, ο επισκέπτης μεταφέρεται στην σελίδα από την οποία μπορεί να συνδεθεί στην υπηρεσία, όπου του εμφανίζεται επιπλέον το μήνυμα επιτυχής εγγραφής.

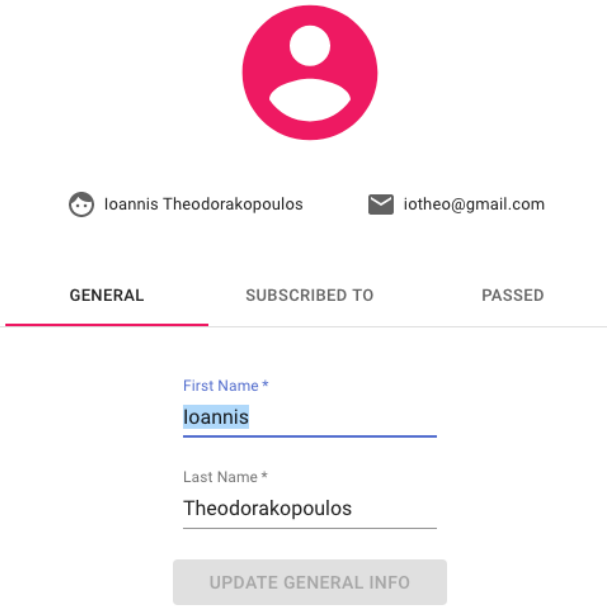


Εικόνα 26: Ανακατεύθυνση στη σελίδα σύνδεσης και μήνυμα επιτυχής εγγραφής

5.2 Πλοήγηση Χρήστη – Μέλους

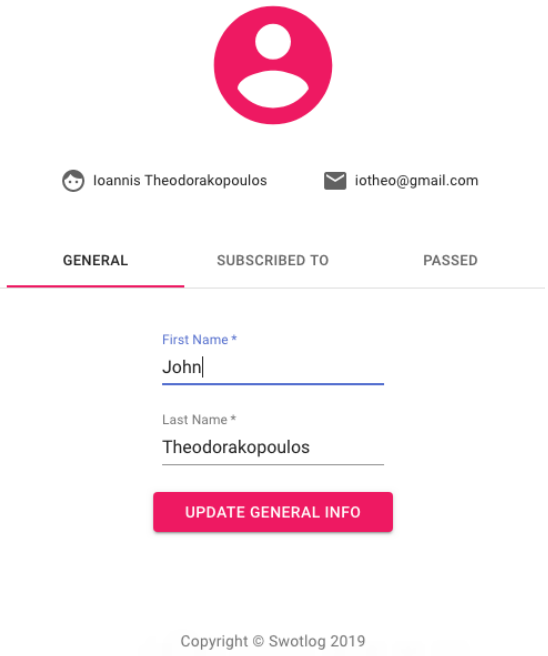
5.2.1 Αλλαγή Στοιχείων Προφίλ

Μετά από την επιτυχή σύνδεση στην εφαρμογή, ο χρήστης – μέλος θα είναι σε θέση να πλοηγηθεί στην σελίδα του προφίλ του. Εκεί, επιλέγοντας το tab “General” θα μπορέσει να αλλάξει τα στοιχεία που έχει δώσει για την αναγνώρισή του.

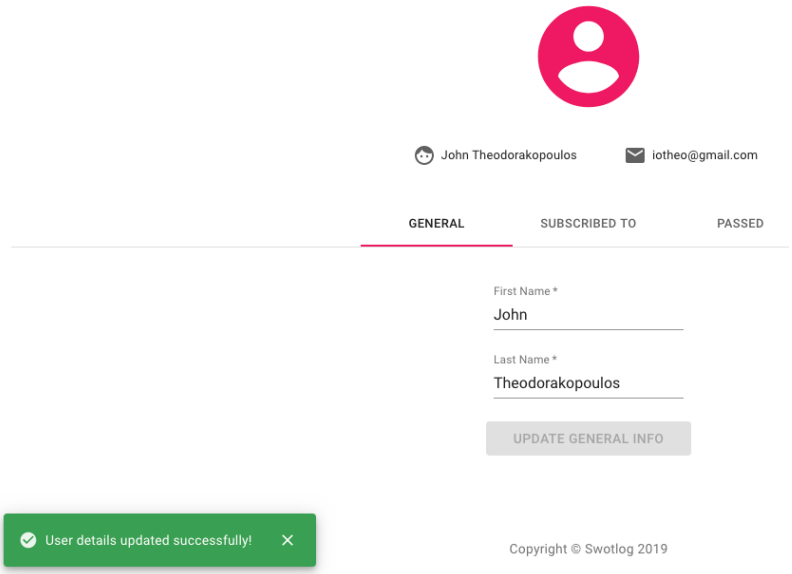


Εικόνα 27: Το tab των γενικών στοιχείων του χρήστη - κατάσταση πριν την εισαγωγή αλλαγών

Αφού εισαχθούν τα νέα στοιχεία ενεργοποιείται το κουμπί μέσω του οποίου γίνεται η ενημέρωση των στοιχείων του χρήστη. Αξίζει να σημειωθεί πως τα στοιχεία του χρήστη εμφανίζονται στη νέα τους μορφή σε πραγματικό χρόνο, συγχρόνως με το μήνυμα επιτυχίας.



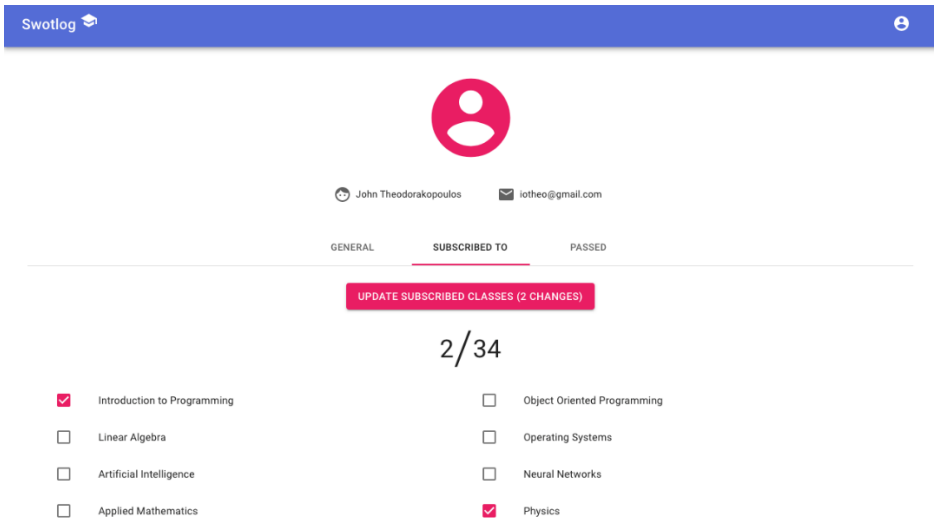
Εικόνα 28: Το tab των γενικών στοιχείων του χρήστη - κατάσταση μετά την εισαγωγή αλλαγών



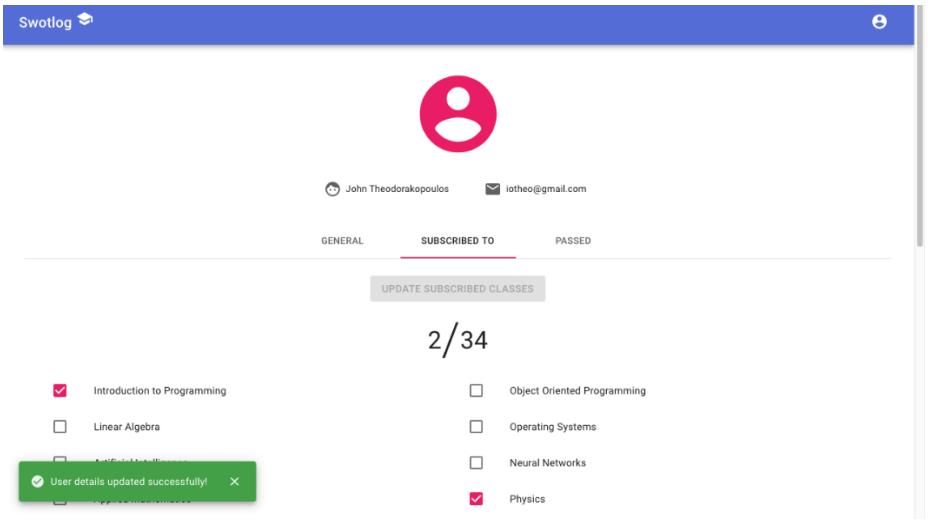
Εικόνα 29: Το tab των γενικών στοιχείων του χρήστη - επιτυχής ενημέρωση στοιχείων

5.2.2 Εγγραφή σε Μαθήματα

Στο tab "Subscribed to" ο χρήστης μπορεί να δει τη λίστα των διαθέσιμων μαθημάτων προς εγγραφή. Παρομοίως με την αλλαγή των στοιχείων, με την οποιαδήποτε αλλαγή στις επιλογές του χρήστη, ενεργοποιείται το κουμπί της ενημέρωσής των μαθημάτων του, ενώ με την επιλογή του κουμπιού, οι αλλαγές αποστέλλονται στη βάση δεδομένων και εμφανίζεται το μήνυμα της επιτυχής ενημέρωσης.



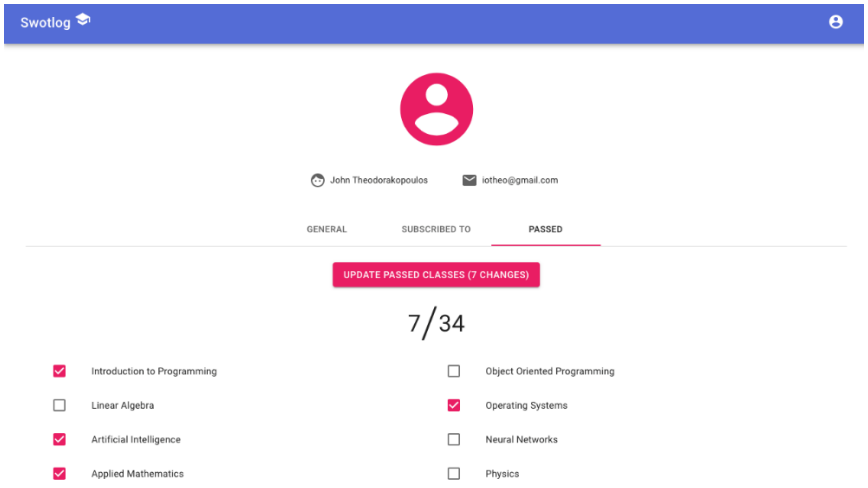
Εικόνα 30: Εγγραφή σε μαθήματα - κατάσταση με νέες επιλογές



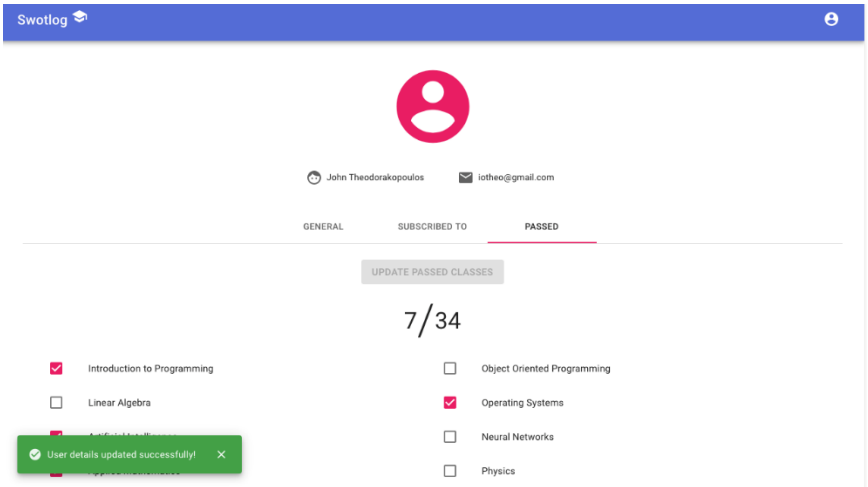
Εικόνα 31: Εγγραφή σε μαθήματα - επιτυχής δήλωση

5.2.3 Σημείωση Περασμένων Μαθημάτων

Στο επόμενο tab, "Passed", ο χρήστης μπορεί να ορίσει τα μαθήματα τα οποία έχει περάσει επιτυχώς. Η λειτουργία αυτή είναι παρόμοια με την επιλογή των μαθημάτων υπό παρακολούθηση.



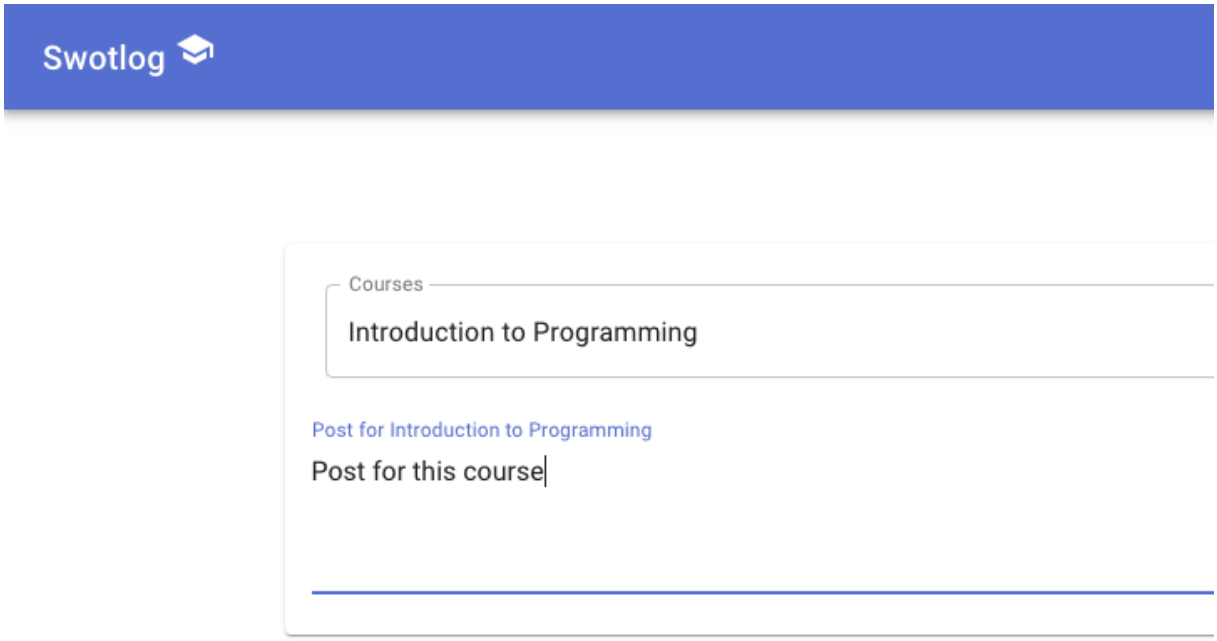
Εικόνα 32: Ενημέρωση περασμένων μαθημάτων - κατάσταση με νέες επιλογές



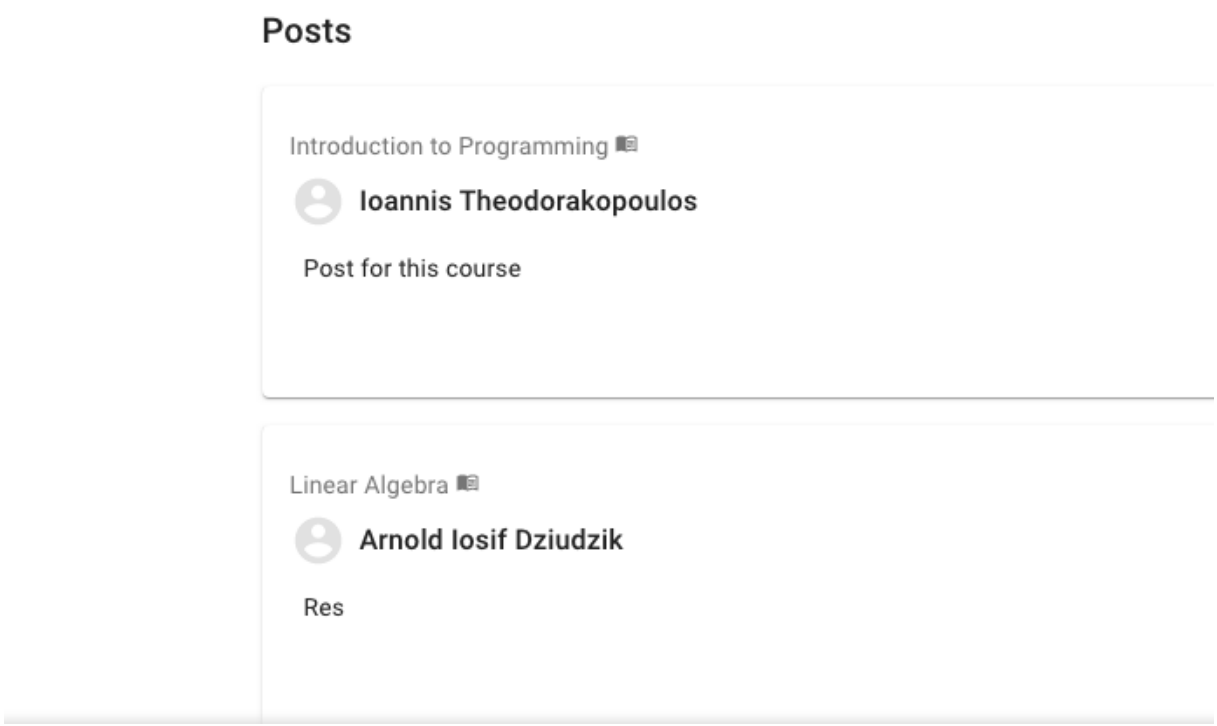
Εικόνα 33: Επιτυχής ενημέρωση περασμένων μαθημάτων

5.2.4 Δημιουργία Αναρτήσεων

Για την ανάρτηση ενός post, ο χρήστης πρέπει πρώτα να επιλέξει το μάθημα στα πλαίσια του οποίου θα γίνει η ανάρτηση. Στη συνέχεια, μπορεί να συντάξει το περιεχόμενό της ανάρτησης, και, με την ανάρτησή της, αυτή θα εμφανιστεί στο feed που έχουν στην διάθεσή τους οι χρήστες για την παρακολούθηση όλων των posts.



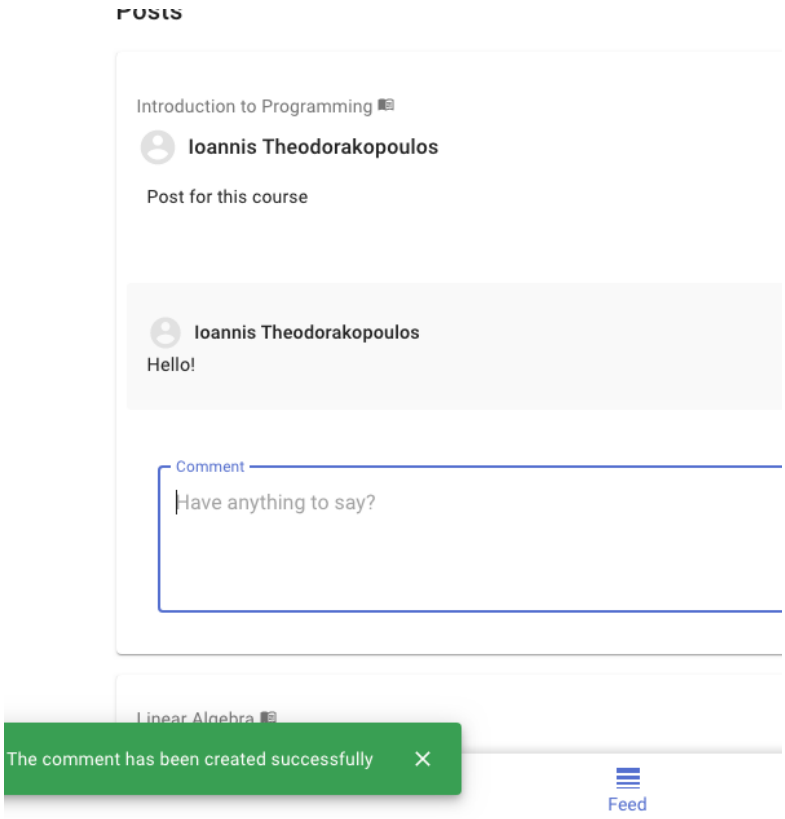
Εικόνα 34: Δημιουργία νέας ανάρτησης



Εικόνα 35: Η εμφάνιση του feed με τις αναρτήσεις των χρηστών

5.2.5 Σύνταξη Σχολίων

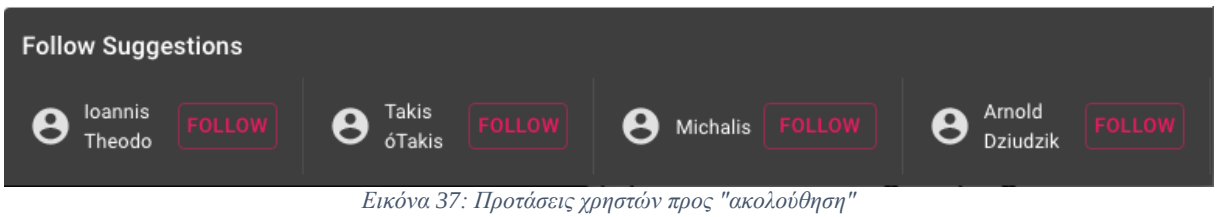
Σε κάθε ανάρτηση που βρίσκεται στο κεντρικό feed της εφαρμογής, ο χρήστης μπορεί να συντάξει κάποιο σχόλιο σχετικό με την ανάρτηση. Αυτό, με την ολοκλήρωσή του, θα εμφανίζεται κάτω από την ανάρτηση στην οποία ανήκει και συγχρόνως θα εμφανίζεται ένα μήνυμα επιτυχίας.



Εικόνα 36: Σύνταξη ενός comment

5.2.6 Μηχανισμός Follow

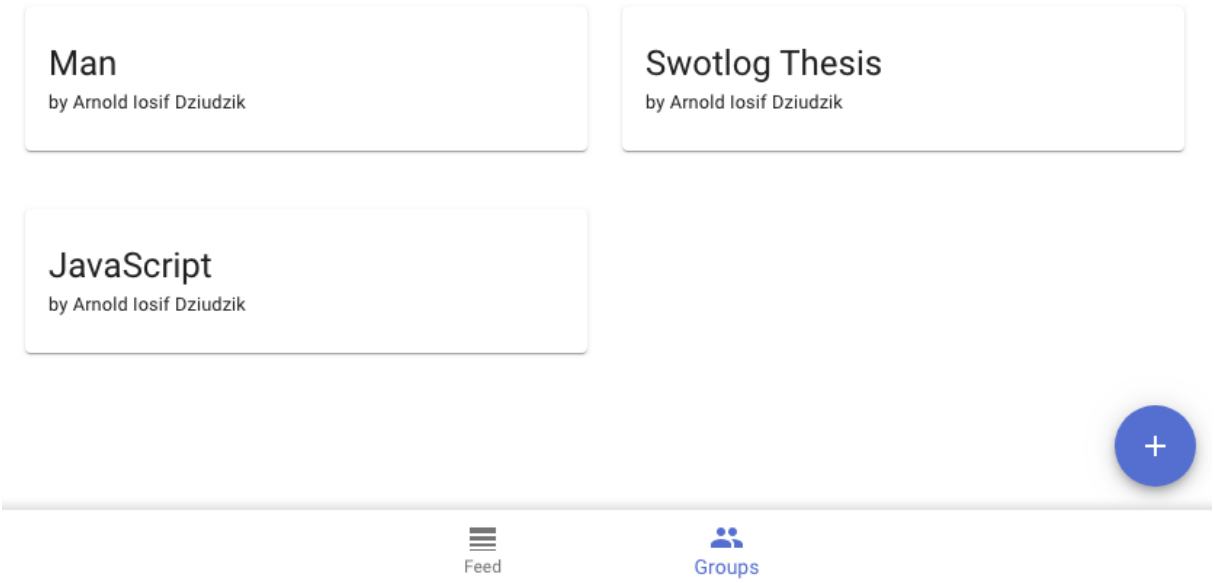
Η εφαρμογή προτείνει άλλους χρήστες της για την δικτύωση μέσω του μηχανισμού "Follow". Επιλέγοντας το κουμπί "Follow", οι επιλεγμένοι χρήστες προστίθενται στο σύνολο των «κοινωνικών συνδέσεων» του ενεργού χρήστη, και έτσι οι χρήστες μπορούν να έρθουν σε επαφή και να δημιουργήσουν ομάδες για συνεργασία.



Εικόνα 37: Προτάσεις χρηστών προς "ακολούθηση"

5.2.7 Δημιουργία Ομάδας

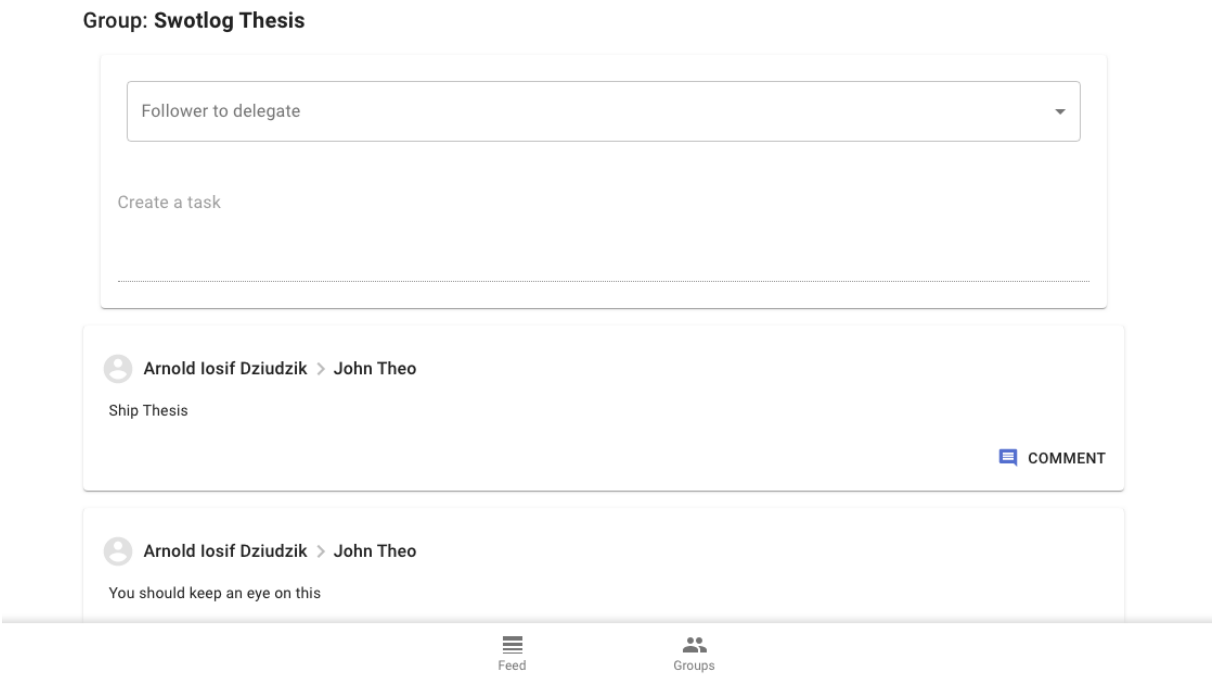
Ο χρήστης μπορεί να πλοηγηθεί στο πλαίσιο της εφαρμογής "Groups", όπου μπορεί να δει τις ομάδες στις οποίες ανήκει, καθώς και τους δημιουργούς της κάθε ομάδας. Επίσης, μπορεί να δημιουργήσει καινούργιες ομάδες, κατά ανάγκη και προτίμηση του.



Εικόνα 38: Το πλαίσιο των ομάδων

5.2.8 Ανάθεση Έργου

Εντός του περιβάλλοντος της ομάδας, ο χρήστης μπορεί να αναθέσει έργα σε χρήστες με του οποίους είναι δικτυωμένος. Για να συμβεί αυτό, επιλέγει πρώτα τον χρήστη στον οποίο θέλει να αναθέσει το έργο και το συνοδεύει με το μήνυμα που θέλει να μεταδώσει στον εκπονητή του έργου. Ένα τέτοιο "task" υποστηρίζεται από μηχανισμό comments, για την απαραίτητη συζήτηση πάνω στο θέμα της εκπόνησης, για ιδέες, πηγές και διευκρινήσεις.



Εικόνα 39: Ανάθεση έργου σε άλλον χρήστη

6. Αποτελέσματα

6.1 Συμπεράσματα

Η ανάπτυξη μιας πλατφόρμας κοινωνικής δικτύωσης είναι μια πολυεπίπεδη διαδικασία, που απαιτεί τη εμβάθυνση σε περισσότερα πεδία από την απλή ανάπτυξη του λογισμικού, γεγονός που υποδεικνύει γιατί οι μεγάλες υπηρεσίες κοινωνικής δικτύωσης προσελκύουν ταλαντούχους ανθρώπους από πολλά επιστημονικά πεδία και ειδικότητες. Από την ερεύνα πάνω στο target audience, την μελέτη της εμπειρίας των χρηστών, τον προσδιορισμό μια στρατηγικής ανάπτυξης με βάση τους χρήστες, μέχρι και την προώθησή τους ως ένα προϊόν το οποία εξυπηρετεί την παραγωγή κερδών για την ιδιοκτήτρια εταιρία, ο ρόλος του μηχανικού λογισμικού φαίνεται, επιφανειακά, όχι και τόσο σημαντικός. Όμως, σε όλα τα στάδια ανάπτυξης της πλατφόρμας, από το scoping, το wireframing και τον σχεδιασμό, μέχρι και την συντήρηση, η συμμετοχή του μηχανικού απαραίτητα, καθώς το δικό του έργο θα φέρει όλον αυτόν τον σχεδιασμό στην πραγματικότητα. Οι πλατφόρμες κοινωνική δικτύωσης είναι στενά συνδεδεμένες με του χρήστες τους και μεγαλώνουν μαζί με αυτούς, οπότε θα μπορούσε να υποστηριχθεί πως η ζωή τους ξεκινάει με την εκκίνηση της λειτουργίας τους για το κοινό, όπου με την ενσωμάτωση και την χρήση κατάλληλων μετρήσεων γίνεται ο έλεγχος της απόδοσης αλλά και της δημοσιότητας της πλατφόρμας. Έτσι, μπορεί να καθοδηγηθεί η συνεχής εξέλιξή της, ώστε να παραμένει «φρέσκια» και επίκαιρη.

Η υλοποίηση της παρούσας πτυχιακής εργασίας ήταν μια διαδικασία απαιτητική σε όλα τα στάδια της: στην μελέτη και τον σχεδιασμό της, στην εξοικείωση με την μεγάλη παλέτα εργαλείων και αρχιτεκτονικών που χρησιμοποιούνται σήμερα από εταιρίες που ηγούνται την ανάπτυξης στον τομέα της ανάπτυξης διαδικτυακών εφαρμογών. Κάθε βήμα ήταν μια ξεχωριστή πρόκληση που χρειαζόταν ιδιαίτερη προσέγγιση και συχνά, πολλαπλές αποτυχημένες απόπειρες στην υπερίσχυση. Η διαδικασία αυτή μας κράτησε σε εγρήγορση και η τιμωρία για οποιαδήποτε απόπειρα μηχανικής εκτέλεσης οποιουδήποτε κομματιού της εργασίας ήταν άμεση. Ως αποτέλεσμα, όμως, ως εκπονητές της εργασίας αυτής, είχαμε την δυνατότητα να διευρύνουμε, τόσο τις γνώσεις, όσο και τις δεξιότητές μας πέρα από την προετοιμασία που παρέχει το Τμήμα μας, τόσο στον τομέα του web development αλλά και όλων των πτυχών του αντικειμένου, όπως οι υπηρεσίες κοινωνικής δικτύωσης. Παρόλο που αυτή είναι μόλις η αρχή της σταδιοδρομίας μας, αισθανόμαστε περισσότερη αυτοπεποίθηση, τόσο στις ικανότητές μας, όσο και στην δυνατότητά μας να αναπτυσσόμαστε συνεχώς στην πορεία της. Επίσης, είχαμε την ευκαιρία να αναπτύξουμε αυτήν την εφαρμογή μέσω των συνεργατικών πρακτικών που εφαρμόζονται αυτήν την περίοδο σε όλες της ανταγωνιστικές εταιρίες, μια ικανότητα η οποία εγγυημένα θα φανεί χρήσιμη στο μέλλον.

6.2 Μελλοντική Εργασία και Επεκτάσεις

Σε αυτή την υποενότητα, αναλύονται οι πιθανές μελλοντικές εργασίες και επεκτάσεις, σύμφωνα με τις παρατηρούμενες τάσεις στο σημερινό τοπίο των υπηρεσιών κοινωνικής δικτύωσης.

User Engagement/Εμπλοκή Χρηστών: Όπως αναφέρθηκε και στην εισαγωγή, η πλειοψηφία των μέσων χρηστών του Διαδικτύου ήδη αναλώνει το μεγαλύτερο κομμάτι του χρόνου τους σε αυτό χρησιμοποιώντας πλατφόρμες κοινωνικής δικτύωσης, και μάλιστα, η συντριπτική πλειοψηφία επικεντρώνεται μόνο στις πιο δημοφιλείς. Αντί να γίνει κάποια προσπάθεια για την απόσπαση χρηστών από τις υπηρεσίες οι οποίες ήδη τους είναι οικείες,

πολύ πιο αποδοτική θα ήταν η εμπλοκή τους στην πλατφόρμα του Swotlog μέσω της σύνδεσής κάποιου προϋπάρχοντος προφίλ, όπως αυτό του Facebook, στην εφαρμογή.

In-app messaging/Επικοινωνία εντός της εφαρμογής: Από την στιγμή που θα σημειωνόταν επιτυχία στην προσέλκυση χρηστών στην πλατφόρμα, ο επόμενος στόχος θα ήταν η συγκράτηση των χρηστών στο περιβάλλον της πλατφόρμας για την εξυπηρέτηση των αναγκών τους. Αυτόν τον καιρό, σχεδόν κάθε υπηρεσία κοινωνικής δικτύωσης ενσωματώνει κάποιο μέσω επικοινωνίας που έχει αναπτυχθεί εσωτερικά, ώστε να μην προσφεύγουν οι χρήστες σε ανταγωνιστικές υπηρεσίες. Οπότε, η ανάπτυξη ενός συστήματος instant ή private messaging θα βοηθούσε σε αυτόν τον σκοπό.

User Binding/Δεσμός Χρηστών: Ο στόχος των υπηρεσιών κοινωνικής δικτύωσης είναι ακριβώς αυτός: η δικτύωση των χρηστών τους. Η ανάπτυξη ενός συστήματος που θα μπορούσε να εντοπίζει κοινά στοιχεία μεταξύ των χρηστών, όπως, εξάμηνο εισαγωγής, τόπος καταγωγής, εξωσχολικά ενδιαφέροντα, ιστορικό παρακολούθησης μαθημάτων, ποσοστά επιτυχίας/αποτυχίας σε μαθήματα υπό παρακολούθηση, θα προσέφερε την δυνατότητα της πρότασης τους μεταξύ τους.

Recommender Systems/ Συστήματα Προτάσεων: Ίσως το πιο φιλόδοξο και απαιτητικό κομμάτι για την επέκταση της εφαρμογής θα ήταν η ανάπτυξη ενός συστήματος προτάσεων με βάση τις αξιολογήσεις και τις προτιμήσεις των χρηστών, όπως παρατηρείται σήμερα σε πολλές υπηρεσίες streaming, είτε αφορά βίντεο, είτε μουσική, το λιανικό εμπόριο και, πρόσφατα, τις υπηρεσίες μεταφοράς προσώπων (Uber, Lyft). Για παράδειγμα, η εφαρμογή θα μπορούσε να προτείνει χρήστες που προσφέρονται για βοήθεια, ανάλογα με την επίδοσή τους στο παρελθόν, μαθήματα για παρακολούθηση με βάση την ικανοποίηση του φοιτητή με το αντικείμενο και το παράδοση του καθηγητή και συναδέλφους για συνεργασία στις ομαδικές εργασίες με βάση την ευσυνειδησία τους. Όμως, οι απαιτήσεις είναι αρκετά μεγάλες, καθώς αυτή η επέκταση βυθίζεται στο πεδίο της μηχανικής μάθησης και της τεχνητής νοημοσύνης, και η εκπαίδευση ενός τέτοιου συστήματος χρειάζεται αρκετό χρονικό διάστημα και τεράστια σύνολα δεδομένων που, τουλάχιστον στα πλαίσια ενός κοινωνικού δικτύου σαν το Swotlog, δεν έχουν συλλεχθεί.

Σύνδεση με προϋπάρχουσες υπηρεσίες της Τριτοβάθμιας Εκπαίδευσης: Μέσω της ανάπτυξης προσαρμοσμένων APIs, θα ήταν δυνατή η σύνδεση του Swotlog με άλλες υπηρεσίες της Τριτοβάθμιας Εκπαίδευσης. Έτσι, θα ήταν δυνατή η ανάκτηση, από επίσημες πηγές, των μαθημάτων στο πρόγραμμα σπουδών κάθε σχολής, του υπεύθυνου διδακτικού προσωπικού, οι βαθμολογίες και ο μέσος όρος του κάθε χρήστη, καθώς και τα μαθήματα που βρίσκονται στην δήλωσή του ανά εξάμηνο, για αυτόματη εγγραφή σε αυτά εντός της πλατφόρμας.

Freelance Educational Services – Donations/Ανεξάρτητες Υπηρεσίες Εκπαίδευσης - Δωρήματα**:** Η γενική λειτουργικότητα της εφαρμογής θα μπορούσε να επεκταθεί για μια επιπλέον ομάδα χρηστών, των ιδιωτικών εκπαιδευτικών που ασχολούνται με την επιμόρφωση φοιτητών της Τριτοβάθμιας Εκπαίδευσης που δυσκολεύονται σε κάποιο μάθημα του προγράμματος λόγω κενών που δημιουργήθηκαν στην πορεία τους στην Δευτεροβάθμια Εκπαίδευση. Μέσω αυτής της επέκτασης, θα μπορούσαν να διαφημίσουν τις υπηρεσίες τους, οι χρήστες θα μπορούσαν να αφήνουν κριτικές για αυτές και θα μπορούσε να ενσωματωθεί κάποιο σύστημα πληρωμών για αυτές. Επίσης, σε αυτό το πλαίσιο, για του φοιτητές που προσφέρουν την βοήθειά τους στους συναδέλφους τους, θα μπορούσε να εφαρμοστεί ένα σύστημα μικρού «δωρήματος» για τον κόπο και την συμπαράστασή τους.

Event Organizing/Οργάνωση Συμβάντων: Επιπλέον, οι φοιτητές, και σε μελλοντική επέκταση τα Ιδρύματα Τριτοβάθμιας Εκπαίδευσης, θα μπορούσαν να οργανώνουν συμβάντα και εκδηλώσεις σχετικές με την φοιτητική ζωή, σε ένα περιβάλλον που εξυπηρετεί σχεδόν αποκλειστικά τον πυρήνα αυτής, δηλαδή την μάθηση.

Mobile Version: Κατά την τωρινή περίοδο, το μεγαλύτερο μέρος της κίνησης στο Διαδίκτυο προέρχεται πια από το mobile περιβάλλον. [42] Έτσι, μια λογική κίνηση προς την επέκταση της εφαρμογής θα ήταν η ανάπτυξη της έκδοσής της και για κινητά, για τα λειτουργικά συστήματα Android και iOS.

Admin Page/Σελίδα Διαχειριστή: Στο παρόν στάδιο, όλες οι αλλαγές και το νέο περιεχόμενο που πρέπει να εισαχθεί στην εφαρμογή γίνεται «χειροκίνητα» με άμεση επέμβαση στον πηγαίο κώδικα. Με την δημιουργία μιας σελίδας διαχειριστή, η διαχείριση του περιεχομένου και των χρηστών της σελίδας θα ήταν μια ορθολογική και βελτιστοποιημένη διαδικασία.

6.3 Κατακλείδα

Η σημερινή κατάσταση του Web Development χαρακτηρίζεται από πρωτοφανής κίνηση και εξέλιξη. Αυτό το γεγονός, όμως, καθιστά τον τομέα αρκετά αποθαρρυντικό για τους νέους προγραμματιστές που θα ήθελαν να ασχοληθούν με το αντικείμενο. Καινούργιες τεχνολογίες εμφανίζονται, φαινομενικά, σε εβδομαδιαία βάση και η ποσότητα νέας πληροφορίας που πρέπει να προσλάβει ο μαθητευόμενος φαίνεται γιγάντια, με ακατάληπτους ορισμούς και συντομογραφίες. Μόνο για τις ανάγκες της παρούσας πτυχιακής εργασίας αναφέρθηκαν και μελετήθηκαν πράγματα τα οποία ήταν ανήκουστα πριν μία δεκαετία, και ίσως ξεχαστούν εντός της επόμενης. Πολλοί νέοι μηχανικοί λογισμικού ή και παλαιότεροι επαγγελματίες που θα ήθελαν να αναβαθμίσουν τις ικανότητές τους σύμφωνα με τις σύγχρονες τάσεις μπορούν να δημιουργήσουν στη συνείδησή τους την άποψη πως είναι απόλυτα αναγκαίο να κατανοήσουν και να κατέχουν στο έπακρο κάθε τεχνολογική τάση και εργαλείο. Η αλήθεια, όμως, φαίνεται να είναι διαφορετική: η παραπάνω προσέγγιση είναι περισσότερο επιβλαβής, παρά βοηθητική, διότι ο ρόλος αυτών των εργαλείων είναι ο ίδιος με αυτόν του μηχανικού που τα χρησιμοποιεί, δηλαδή η επίλυση προβλημάτων στον πραγματικό κόσμο, και, χωρίς αυτόν, τα εργαλεία δεν συνεισφέρουν τίποτα στον τομέα. Έτσι, ο καλύτερος τρόπος για την ανακάλυψη της βέλτιστης προσέγγισης για την ανάπτυξη μιας διαδικτυακής εφαρμογής δεν είναι η επίπονη διείσδυση στις μηδαμινές τεχνικές λεπτομέρειες της κάθε βιβλιοθήκης ή framework, αλλά η πρακτική προσέγγιση, η κατασκευή νέων εφαρμογών, μικρών και μεγάλων – όταν κάποιο κομμάτι της υλοποίησης φαίνεται να είναι δυσκολότερο απ’ όσο θα έπρεπε να είναι, τότε αξίζει η αναζήτηση νέων τεχνολογιών για την βελτίωση της εμπειρίας του ίδιου του προγραμματιστή. Αυτή ήταν και η προσέγγιση που χρησιμοποιήθηκε για την υλοποίηση της παρούσας πτυχιακής εργασίας.

Οι νέοι προγραμματιστές διαδικτυακών εφαρμογών δεν έχουν πια την πολυτέλεια για την αργή και εργαλειοκεντρική προσέγγιση την οποία χαίρονταν οι παλαιότεροι προγραμματιστές στο, όχι τόσο μακρινό, παρελθόν. Ο πολύτιμος και ανάρπαστος web developer του μέλλοντος θα είναι αυτός που θα είναι πρόθυμος να μάθει καινούργια πράγματα, να τα μάθει γρήγορα και, προπαντός, αυτός που θα έχει κάποιο όραμα για την εφαρμογή των νέων γνώσεων του, άρα, στον πυρήνα της επαγγελματικής ζωής του ευδαιμονεί ως Μηχανικός, βρίσκει νόημα στο αντικείμενό του, αναζητεί την προσωπική ανάπτυξη, στοχεύει ψηλά στο έργο του και επιδιώκει την βαθύτερη κατανόηση τόσο του εαυτού του. Παραμένει, όμως, μια δύσκολη, απαιτητική και συνεχώς μεταλλασσόμενη διαδρομή προς αυτή την επίτευξη.

Κανείς δεν μπορεί να αρνηθεί πως το σημερινό τοπίο χαρακτηρίζεται από πολυπλοκότητα: σχεδιαστική πολυπλοκότητα, τεχνική πολυπλοκότητα, με θέματα συνεχής αύξησης των επιδόσεων και του περιεχομένου. Αυτός ο τομέας δεν ήταν ποτέ εύκολος στην ενασχόληση και προφανώς δεν θα γίνει εύκολος ποτέ. Η έννοια της απλότητας στην ανάπτυξη διαδικτυακών εφαρμογών, δυστυχώς ή ευτυχώς, περιορίζεται μόνο στην εμπειρία του χρήστη που χαίρεται τους καρπούς της δύσκολης και συχνά απαρατήρητης προσπάθειας

της πληθώρας των προγραμματιστών που βρίσκονται πίσω από αυτές – οι εποχές όπου ένας άνθρωπος μπορούσε να δημιουργήσει μόνος του μια πλήρης λειτουργική και ανταγωνιστική διαδικτυακή εφαρμογή φαίνονται επίσης μια ανάμνηση του παρελθόντος. Αλλά η παρούσα εποχή φέρει την δική της ιδιαιτερότητα: το Διαδίκτυο έχει αλλάξει την όψη της ανθρωπότητας, έχει βελτιώσει τις ζωές αμέτρητων ανθρώπων και το μέλλον του υπαινίσσεται μια συνεχή άνθιση. Εμείς, ως Μηχανικοί και προγραμματιστές διαδικτυακών εφαρμογών, έχουμε το προνόμιο να είμαστε ανάμεσα στους ανθρώπους που θα πλάσουν αυτό το μέλλον και θα κάνουν το Διαδίκτυο πιο προσβάσιμο και περισσότερο εξυπηρετικό για του συνανθρώπους μας· αυτός είναι ο σκοπός μας, η εκπλήρωσή του η ανώτερη τιμή που μπορούμε να προσμένουμε και η θέληση αυτού μια πραγματική αρετή.

«Τις είναι θέλεις σαυτώ πρώτον ειπέ. Είθ' όυτως ποίει ά ποιείς.»

«Πρώτα πες στον εαυτό σου ποιος θέλεις να είσαι· μετά να κάνεις αυτά που κάνεις σύμφωνα με αυτό.»

- Επικτήτου Διατριβαί

Αναφορές

[1] J. H. Kietzmann, K. Hermkens, I. P. McCarthy και B. S. Silvestre, «Social Media? Get Serious! Understanding the Functional Building Blocks of Social Media,» *Business Horizons,* τόμ. 54, αρ. 3, pp. 241-251, 2011.

[2] J. A. Obar και S. Wildman, «Social media definition and the governance challenge: An introduction to the special issue,» *Telecommunications Policy,* τόμ. 39, αρ. 9, pp. 745-750, 2015.

[3] A. M. Kaplan και M. Haenlein, «Users of the world, unite! The challenges and opportunities of Social Media,» *Business Horizons,* τόμ. 53, αρ. 1, pp. 59-68, 2010.

[4] D. M. Boyd και N. B. Ellison, «Social Network Sites: Definition, History, and Scholarship,» *Journal of Computer-Mediated Communication,* τόμ. 13, αρ. 1, pp. 210-230, 2007.

[5] Nielsen Company, «Social Networks Blogs Now Account for One in Every Four and a Half Minutes Online,» 2010.

[6] G. S. O'Keeffe, K. Clarke-Pearson και C. o. C. Media, «The Impact of Social Media on Children, Adolescents, and Families,» *Pediatrics,* τόμ. 127, αρ. 4, pp. 800-804, 2011.

[7] C. J., «Most popular social networks worldwide as of October 2019, ranked by number of active users (in millions),» Statista, 2019.

[8] S. M. Al-Zoubi και M. A. Bani Younes, «Low Academic Achievement: Causes and Results,» *Academy Publication,* τόμ. 5, αρ. 11, pp. 2262-2268, 2015.

[9] S. Xi και X. Tian, «Academic Culture and Campus Culture of Universities,» *Higher Education Studies,* τόμ. 2, αρ. 2, pp. 61-65, 2012.

[10] J. Clement, «Number of social network users worldwide from 2010 to 2021 (in billions),» Statista, 14 August 2019. [Ηλεκτρονικό]. Available: https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/.

[11] J. Clemens, «Most popular social networks worldwide as of October 2019, ranked by number of active users (in millions),» Statista, 21 November 2019. [Ηλεκτρονικό]. Available: https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/.

[12] H. Jenkins, Confronting the challenges of participatory culture : media education for the 21st century, Cambridge, MA: Cambridge, MA : The MIT Press, 2009.

[13] C. Fuchs, «Social Media as Participatory Culture,» σε *Social Media: A Critical Introduction*, SAGE Publications Ltd, 2014.

[14] J. P. Gee, Situated language and learning : a critique of traditional schooling, 2004.

[15] A. Oreskovic, «Facebook comments, ads don't sway most users: poll,» *Reuters,* 5 June 2012.

[16] OpenID, «Welcome to OpenID Connect».

[17] D. Flanagan, JavaScript - The definitive guide, 6th Edition, O'Reilly Media, Inc., 2016.

[18] T. Berners-Lee, «Information Management: A Proposal,» 1990.

[19] World Wide Web Consortium, «What is CSS?,» World Wide Web Consortium, [Ηλεκτρονικό]. Available: https://www.w3.org/standards/webdesign/htmlcss#whatcss.

[20] J. Harband και K. Smith, «ECMAScript® 2020 Language Specification,» ECMA International, 22 November 2019. [Ηλεκτρονικό]. Available: https://tc39.es/ecma262/#sec-overview.

[21] Mozilla; Individual Contributors, «CSS preprocessor,» Mozilla, 2019. [Ηλεκτρονικό]. Available: https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor.

[22] A. Papp, «The History of React.js on a Timeline,» Rising Stack, 2018.

[23] Facebook Inc., «Introducing JSX,» Facebook Inc., 2019. [Ηλεκτρονικό]. Available: https://reactjs.org/docs/introducing-jsx.html. [Πρόσβαση 2 August 2019].

[24] Facebook Inc., «Rendering Elements,» Facebook Inc., 2019. [Ηλεκτρονικό]. Available: https://reactjs.org/docs/rendering-elements.html. [Πρόσβαση 2 August 2019].

[25] Facebook Inc., «Components and Props,» Facebook Inc., 2019. [Ηλεκτρονικό]. Available: https://reactjs.org/docs/components-and-props.html. [Πρόσβαση 2 August 2019].

[26] Facebook Inc., «State and Lifecycle,» Facebook Inc., 2019. [Ηλεκτρονικό]. Available: https://reactjs.org/docs/state-and-lifecycle.html. [Πρόσβαση 2 August 2019].

[27] Facebook Inc., «Handling Events,» Facebook Inc., 2019. [Ηλεκτρονικό]. Available: https://reactjs.org/docs/handling-events.html. [Πρόσβαση 2 August 2019].

[28] Facebook Inc., «Conditional Rendering,» Facebook Inc., 2019. [Ηλεκτρονικό]. Available: https://reactjs.org/docs/conditional-rendering.html. [Πρόσβαση 2 August 2019].

[29] Facebook Inc., «Hooks,» Facebook Inc., 2019. [Ηλεκτρονικό]. Available: https://reactjs.org/docs/hooks-intro.html. [Πρόσβαση 7 August 2019].

[30] Facebook Inc., «In-Depth Overview,» Facebook Inc., 2019. [Ηλεκτρονικό]. Available: https://facebook.github.io/flux/docs/in-depth-overview/. [Πρόσβαση 30 July 2019].

[31] Facebook Inc., «Flux: Actions and the Dispatcher,» 2019. [Ηλεκτρονικό]. Available: https://reactjs.org/blog/2014/07/30/flux-actions-and-the-dispatcher.html. [Πρόσβαση 30 July 2019].

[32] K. Lieberherr, «Law of Demeter: Principle of Least Knowledge,» College of Computer and Information Science, Northeastern University, [Ηλεκτρονικό]. Available: http://www.ccs.neu.edu/home/lieber/LoD.html. [Πρόσβαση 30 July 2019].

[33] D. Abramov και R. D. Authors, «Getting Started with Redux,» 2018. [Ηλεκτρονικό]. Available: https://redux.js.org/introduction/getting-started. [Πρόσβαση 31 July 2019].

[34] D. Abramov και R. D. Authors, «Motivation,» 2019. [Ηλεκτρονικό]. Available: https://redux.js.org/introduction/motivation. [Πρόσβαση 31 July 2019].

[35] D. Abramov και R. D. Authors, «Three Principles,» 2018. [Ηλεκτρονικό]. Available: https://redux.js.org/introduction/three-principles. [Πρόσβαση 31 July 2019].

[36] D. Abramov και R. D. Authors, «Core Concepts,» 27 january 2019. [Ηλεκτρονικό]. Available: https://redux.js.org/introduction/core-concepts. [Πρόσβαση 31 July 2019].

[37] The PostgreSQL Global Development Group, «PostgreSQL: The World's Most Advanced Open Source Relational Database,» The PostgreSQL Global Development Group, [Ηλεκτρονικό]. Available: https://www.postgresql.org/.

[38] T. Haerder και A. Reuter, «Principles of transaction-oriented database recovery,» *ACM Computing Surveys,* τόμ. 15, αρ. 4, pp. 287-317, 1983.

[39] The PostgreSQL Global Development Group, «Appendix B. SQL Key Words,» The PostgreSQL Global Development Group, [Ηλεκτρονικό]. Available: https://www.postgresql.org/docs/7.3/sql-keywords-appendix.html.

[40] Carlson και Brian, «Pooling,» [Ηλεκτρονικό]. Available: https://node-postgres.com/features/pooling.

[41] «HTML Specifications - Autofilling,» [Ηλεκτρονικό]. Available: https://html.spec.whatwg.org/multipage/form-control-infrastructure.html#autofill.

[42] J. Clement, «Mobile internet traffic as percentage of total web traffic in August 2019, by region,» Statista, 9 September 2019. [Ηλεκτρονικό]. Available: https://www.statista.com/statistics/306528/share-of-mobile-internet-traffic-in-global-regions/.