ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Foodie

"Σχεδιασμός και υλοποίηση διαδικτυακής εφαρμογής αναζήτησης και δημιουργίας μαγειρικών συνταγών με βάση διαθέσιμα υλικά"

Του φοιτητή

Μαλλιότα Παναγιώτη-Κωνσταντίνου

Αρ. Μητρώου: 154486

Επιβλέπων

Βασίλειος Κώστογλου, Καθηγητής ΔΙΠΑΕ

Θεσσαλονίκη 2024

Τίτλος Π.Ε: "Foodie: Διαδικτυακή πλατφόρμα για αναζήτηση και δημιουργία μαγειρικών συνταγών με βάση διαθέσιμα υλικά"

Κωδικός Π.Ε: 22224

Ονοματεπώνυμο φοιτητή/τών: Μαλλιότας Παναγιώτης-Κωνσταντίνος

Ονοματεπώνυμο εισηγητή: Μαλλιότας Παναγιώτης-Κωνσταντίνος

Ημερομηνία ανάληψης Π.Ε: 18-04-2022

Ημερομηνία περάτωσης Π.Ε. 16-04-2024

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

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Μαλλιότα Παναγιώτη-Κωνσταντίνου που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

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

Πρόλογος

Η παρούσα πτυχιακή εργασία συντάχθηκε από τον φοιτητή Μαλλιότα Παναγιώτη-Κωνσταντίνο υπό την εποπτεία του κ. Βασίλειου Κώστογλου. Αφορά την ανάπτυξη μιας διαδικτυακής πλατφόρμας, η οποία επιτρέπει στους χρήστες να αναζητούν και να δημιουργούν μαγειρικές συνταγές βάσει των υλικών που διαθέτουν. Η παρούσα πλατφόρμα περιλαμβάνει μια ιστοσελίδα όπου οι χρήστες μπορούν να αναζητούν συνταγές ανάλογα με τα διαθέσιμα υλικά τους, καθώς και να καταχωρούν νέες συνταγές. Η ανάπτυξη της πλατφόρμας έγινε κυρίως με χρήση των τεχνολογιών NextJs, React, Prisma, TRPC, PostgresSql, Typescript και Mantine ως UI Framework. Στόχος της εργασίας μας είναι να προσφέρουμε στο ευρύ κοινό μια εύχρηστη πλατφόρμα για τη διαχείριση και την ανταλλαγή μαγειρικών συνταγών με βάση τα υλικά που διαθέτουν, παρέχοντας περισσότερες επιλογές από τις υπάρχουσες εφαρμογές. Τέλος, στόχος είναι να αυξηθεί η ευαισθητοποίηση σχετικά με την αξιοποίηση των υλικών που έχουν οι χρήστες στο σπίτι τους για τη δημιουργία υγιεινών και πεντανόστιμων γευμάτων.

Περίληψη Πτυχιακής

Η διαδικτυακή πλατφόρμα που προτείνεται στην πτυχιακή εργασία σχεδιάζεται να παρέχει δύο βασικές λειτουργίες: αναζήτηση συνταγών βάσει των διαθέσιμων υλικών και δημιουργία νέων συνταγών. Οι χρήστες μπορούν να αναζητούν συνταγές που ταιριάζουν με τα υλικά που διαθέτουν στο σπίτι τους, προσφέροντας έτσι λύσεις για το τί να μαγειρέψουν χωρίς την ανάγκη για επιπλέον αγορές. Επιπλέον, οι χρήστες μπορούν να καταχωρούν νέες συνταγές με βάση τα διαθέσιμα υλικά τους, συμβάλλοντας στη δημιουργία μιας κοινότητας ανταλλαγής συνταγών και ιδεών στον χώρο της μαγειρικής. Ταυτόχρονα, η πλατφόρμα αναμένεται να επιλύει προβλήματα που συνδέονται με τη δυσκολία εύρεσης κατάλληλων συνταγών βασισμένων στα υλικά που διαθέτουν οι χρήστες, ενθαρρύνοντας τη δημιουργικότητα στη μαγειρική και τη χρήση υλικών που ήδη υπάρχουν στο σπίτι, μειώνοντας έτσι το φαγητό σε βάρος και την σπατάλη τροφίμων. Συνολικά, η πλατφόρμα στοχεύει στη δημιουργία μιας κοινότητας που ανταλλάσσει γνώσεις, ιδέες και εμπειρίες στον τομέα της μαγειρικής, προωθώντας την ευεξία και την υγιεινή στη διατροφή.

"Foodie: Online platform for searching and creating cooking recipes based on available ingredients"

Malliotas Panagiotis-Konstantinos

Abstract

The proposed online platform outlined in the thesis is designed to offer two primary functions: recipe search based on available ingredients and recipe creation. Users can search for recipes that match the ingredients they have at home, thus providing solutions for what to cook without the need for additional purchases. Additionally, users can submit new recipes based on their available ingredients, contributing to the creation of a community for recipe exchange and ideas in the culinary field. Simultaneously, the platform is expected to address challenges associated with the difficulty of finding suitable recipes based on users' available ingredients, encouraging creativity in cooking and the use of existing household ingredients, thereby reducing food waste. Overall, the platform aims to create a community that exchanges knowledge, ideas, and experiences in the culinary domain, promoting wellness and healthy eating.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Βασίλειο Κώστογλου, καθηγητή του τμήματος
Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνές Πανεπιστημίου της Ελλάδος, ο οποίος είναι ο επιβλέπων καθηγητής της συγκεκριμένης πτυχιακής εργασίας.

Περιεχόμενα

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Η ανάγκη που οδήγησε στην ιδέα

Η ιδέα για τη δημιουργία της πλατφόρμας προήλθε από την ανάγκη που είχαμε με την φίλη μου να βρίσκουμε εύκολα και γρήγορα νέες συνταγές κατά την καθημερινή μας ζωή. Όπως πολλοί συμπολίτες μας, συχνά βρισκόμαστε μπροστά στο δίλημμα του τι να μαγειρέψουμε, κυρίως λόγω του περιορισμένου αριθμού των συνταγών που γνωρίζουμε ή των περιορισμένων υλικών που έχουμε στη διάθεσή μας.

Αυτή η καθημερινή πρόκληση μας οδήγησε στη σκέψη ότι θα ήταν χρήσιμο να υπήρχε ένα εργαλείο που θα μας βοηθούσε να ανακαλύπτουμε εύκολα και γρήγορα νέες ιδέες για γεύματα, λαμβάνοντας υπόψη τα υλικά που έχουμε ήδη στην κουζίνα μας. Επιπλέον, αντιλαμβανόμαστε ότι αυτή η δυσκολία δεν είναι μοναδική σε εμάς, αλλά είναι κάτι που αντιμετωπίζουν πολλοί άνθρωποι καθημερινά.

Έτσι, η ιδέα για μια πλατφόρμα που θα προσφέρει εύκολη πρόσβαση σε ποικίλες συνταγές, λαμβάνοντας υπόψη τα διαθέσιμα υλικά του κάθε χρήστη, φάνηκε όχι μόνο εφικτή αλλά και απαραίτητη για την καθημερινότητα πολλών ανθρώπων. Από αυτό το σημείο, ξεκίνησε η διαδικασία σχεδίασης και ανάπτυξης μιας τέτοιας πλατφόρμας, με στόχο να προσφέρει λύσεις σε ένα κοινό πρόβλημα που αντιμετωπίζουν πολλοί σήμερα.

1.2 Το μέσο για την υλοποίηση της ιδέας

Οι διαδικτυακές πλατφόρμες αποτελούν καίριο στοιχείο στον σύγχρονο κόσμο της τεχνολογίας και του επιχειρείν. Στο πλαίσιο της πτυχιακής εργασίας μου θα αναδείξουμε τον σημαντικό ρόλο που διαδραματίζουν στη δημιουργία και ανάπτυξη εφαρμογών. Μέσω αυτών, οι χρήστες μπορούν να αποκτούν νέες γνώσεις και δεξιότητες μέσω εκπαιδευτικού περιεχομένου, να επικοινωνούν με κοντινά τους άτομα που βρίσκονται αρκετά μακρυά τους ή ακόμα και να να ανταλλάσσουν συνταγές μαγειρικής.

Η χρήση των διαδικτυακών πλατφορμών συνδέεται άμεσα με την ευκολία πρόσβασης σε ποικίλα περιεχόμενα και τη δυνατότητα διαμόρφωσης ενός εξατομικευμένου περιβάλλοντος. Αυτή η πολυπλοκότητα και ποικιλία προσφοράς αποτελεί σημαντική πρόκληση και ευκαιρία για την ανάπτυξη νέων εφαρμογών και υπηρεσιών που να καλύπτουν τις ανάγκες των χρηστών με αποτελεσματικό και καινοτόμο τρόπο.

Αντιμέτωπος με αυτή την πρόκληση αποφάσισα να αναπτύξω μια διαδικτυακή πλατφόρμα που εστιάζει στη δημιουργία και την κοινοποίηση μαγειρικών συνταγών με βάση τα διαθέσιμα υλικά. Μέσω αυτής της πλατφόρμας, οι χρήστες μπορούν εύκολα να βρουν συνταγές που ταιριάζουν με τα υλικά που διαθέτουν, δίνοντάς τους τη δυνατότητα να μαγειρέψουν εύκολα και γρήγορα χωρίς περιττές αναζητήσεις. Ταυτόχρονα, παρέχει έναν χώρο κοινοποίησης και ανταλλαγής ιδεών μεταξύ των χρηστών, προσφέροντας έτσι μια καινοτόμο λύση στις ανάγκες τους.

1.3 Οργάνωση της εργασίας

Στο επόμενο κεφάλαια θα αναφερθούμε στις τεχνολογίες που επιλέχθηκαν για την ανάπτυξη της εφαρμογής, από τον τρόπο επεξεργασίας του κώδικα έως τις βιβλιοθήκες που χρησιμοποιήθηκαν. Θα εξετάσουμε τις κύριες τεχνολογίες που χρησιμοποιήθηκαν καθώς και τα αντίστοιχα εργαλεία που ενσωματώθηκαν στην πλατφόρμα.

Μεταβαίνοντας στο τρίτο κεφάλαιο, θα επικεντρωθούμε στις λειτουργικές απαιτήσεις της εφαρμογής, χρησιμοποιώντας ιστορίες χρηστών υποστηριζόμενες από διαγράμματα. Στη συνέχεια θα εξετάσουμε την αρχιτεκτονική του backend, περιλαμβανομένης της δημιουργίας API και σημείων άκρων, καθώς και της αλληλεπίδρασης με τη βάση δεδομένων και θα μελετήσουμε την αρχιτεκτονική του frontend για την λήψη και επεξεργασία δεδομένων. Στο τέλος του τρίτου κεφαλαίου, θα παρουσιάσουμε τις λεπτομέρειες υλοποίησης στο backend και στο frontend.

Στο τέταρτο κεφάλαιο, η προσοχή μας στρέφεται στη λειτουργικότητα της εφαρμογής. Μέσω διακριτών σεναρίων, θα επιδείξουμε τη λειτουργία της εφαρμογής από την πλευρά του χρήστη και από την πλευρά του διαχειριστή. Κάθε σενάριο θα συνοδεύεται από μια λεπτομερή ανάλυση, περιγράφοντας τα απαραίτητα βήματα από την αρχή μέχρι το τέλος.

Κεφάλαιο 2ο: Τεχνολογίες

Τα επόμενα υποκεφάλαια εστιάζουν στις τεχνολογίες και τις βιβλιοθήκες που επιλέχθηκαν για την υλοποίηση της διαδικτυακής πλατφόρμας. Κάθε τεχνολογία έχει επιλεγεί με προσοχή για να διασφαλιστεί η βέλτιστη απόδοση, ευελιξία και ευκολία χρήσης της πλατφόρμας. Ανάμεσα στα θέματα που θα εξετάσουμε περιλαμβάνονται οι τεχνολογίες του NextJs, React, Prisma, TRPC, PostgresSql, Typescript και Mantine ως UI Framework, καθώς και άλλα εργαλεία και βιβλιοθήκες που επιλέχθηκαν για την υλοποίηση συγκεκριμένων λειτουργιών της πλατφόρμας. Σε κάθε υποκεφάλαιο θα αναλύσουμε τη χρήση, την εγκατάσταση, τις δυνατότητες και τα πλεονεκτήματα κάθε τεχνολογίας ή βιβλιοθήκης που εφαρμόστηκε στο πλαίσιο της ανάπτυξης της πλατφόρμας.

2.1 Περιβάλλον VS Code

Το περιβάλλον VS Code [1] (Visual Studio Code) αποτελεί ένα βασικό εργαλείο ανάπτυξης λογισμικού για μια ευρεία ανοιχτή κοινότητα προγραμματιστών. Η κοινότητα αυτή συνεχώς συμβάλλει στην ανάπτυξη και βελτίωση του περιβάλλοντος VS Code μέσω της δημιουργίας επεκτάσεων, τροποποιήσεων και αναφορών προβλημάτων. Η συνεισφορά της κοινότητας διασφαλίζει τη συνεχή εξέλιξη του περιβάλλοντος ανάπτυξης, προσθέτοντας νέες λειτουργίες και βελτιστοποιώντας την απόδοση. Επιπλέον, η κοινότητα παρέχει στήριξη και ανταλλαγή γνώσεων μέσω φόρουμ, τμημάτων συζητήσεων και ανοιχτών πόρων. Με αυτόν τον τρόπο, το VS Code γίνεται ένα εργαλείο που εμπλουτίζεται συνεχώς από τη συνεργασία και την ανταλλαγή ιδεών της ανοιχτής προγραμματιστικής κοινότητας.

Μερικοί από τους πιο δημοφιλείς λόγους για να επιλέξει κάποιος να χρησιμοποιήσει το VS Code είναι οι εξής:

- **Ελαφρύ και γρήγορο**: Η αρχιτεκτονική του VS Code βασίζεται στην τεχνολογία της Electron [2], προσφέροντας ισχυρή απόδοση και γρήγορη απόκριση.
- **Σταθερότητα**: Συνεχείς βελτιώσεις και ενημερώσεις από την κοινότητα και τη Microsoft [3] εξασφαλίζουν σταθερή λειτουργία.
- **Ευέλικτο με επεκτάσεις**: Οι πολλές επεκτάσεις που προσφέρει επιτρέπουν την προσαρμογή του περιβάλλοντος εργασίας σύμφωνα με τις ανάγκες του προγραμματιστή.

2.2 HTML, CSS και JavaScript

Τα βασικά στοιχεία HTML, CSS και JavaScript αποτελούν το θεμέλιο του διαδικτυακού κόσμου και αποτελούν αναπόσπαστο μέρος του προγράμματος σπουδών της σχολής μας. Μέσα από τα μαθήματα που παρακολουθήσαμε, αποκτήσαμε βασικές γνώσεις και κατανόηση για τον τρόπο λειτουργίας αυτών των γλωσσών προγραμματισμού. Κάθε στοιχείο - από τη δομή του HTML, τη μορφοποίηση με CSS, έως τη δυναμική λειτουργία με JavaScript - έχει εξερευνηθεί και εφαρμοστεί στην πράξη μέσα από πρακτικές ασκήσεις και έργα που εκπονήθηκαν κατά τη διάρκεια των μαθημάτων. Η εξοικείωσή μας με αυτά τα βασικά εργαλεία μας επιτρέπει να αναπτύξουμε εφαρμογές και ιστοσελίδες που ανταποκρίνονται στις σύγχρονες απαιτήσεις της τεχνολογίας και του διαδικτύου.

Κάνοντας μια γρήγορη σύνοψη των τεχνολογιών, η HTML (HyperText Markup Language) χρησιμοποιείται για τη διαμόρφωση της δομής των ιστοσελίδων, καθορίζοντας τον τρόπο παρουσίασης των περιεχομένων όπως κείμενα, εικόνες, βίντεο κ.λπ. Η CSS (Cascading Style Sheets) είναι υπεύθυνη για τη μορφοποίηση και την εμφάνιση των στοιχείων που δημιουργούνται με το HTML, επιτρέποντας την προσαρμογή των χρωμάτων, των γραμματοσειρών, των διατάξεων και πολλών άλλων στοιχείων στη σελίδα. Η JavaScript είναι μια γλώσσα προγραμματισμού που χρησιμοποιείται για τη δυναμική δημιουργία και λειτουργία των ιστοσελίδων, επιτρέποντας την προσθήκη διαδραστικών στοιχείων, ανταλλαγή δεδομένων με τον διακομιστή, αλλαγή του περιεχομένου της σελίδας και άλλες πολλές λειτουργίες που βελτιώνουν την εμπειρία του χρήστη. Με την συνδυαστική χρήση αυτών των τριών γλωσσών, μπορούμε να δημιουργήσουμε εκθαμβωτικές και λειτουργικές ιστοσελίδες που ανταποκρίνονται στις ανάγκες και τις προτιμήσεις των χρηστών.

2.3 Typescript

Η TypeScript [4] αποτελεί αναπόσπαστο κομμάτι του μοντέρνου web development λόγω των πολλαπλών οφελών που προσφέρει. Με την αύξηση της πολυπλοκότητας των εφαρμογών και των ιστοσελίδων, η ανάγκη για ένα εργαλείο που να προσφέρει σταθερότητα, ασφάλεια και ευκολία στην ανάπτυξη γίνεται ολοένα και πιο σημαντική.

Η TypeScript αποτελεί μια υπερσύνολο της JavaScript, προσθέτοντας στοιχεία όπως την τύπωση και τις συνθήκες, καθιστώντας την αναπτυξιακή διαδικασία πιο ασφαλή και αποτελεσματική. Η στατική τύπωση της TypeScript επιτρέπει στους προγραμματιστές να εντοπίζουν σφάλματα και πιθανά προβλήματα κατά τη διάρκεια της ανάπτυξης, ενώ η έγκυρη χρήση των τύπων βοηθά στη διατήρηση του κώδικα πιο οργανωμένο και ευανάγνωστο. Επιπλέον, η TypeScript παρέχει επεκτασιμότητα και δυνατότητες οργάνωσης μεγάλων έργων, επιτρέποντας τη χρήση προηγμένων λειτουργιών όπως ο μοντουλαρισμός (modularity [5]) και οι διασφαλίσεις της ασφάλειας τυπων (type safety [6]). Με την αυξημένη συντακτική ανάλυση και την υποστήριξη για τα πιο πρόσφατα χαρακτηριστικά της

JavaScript, η TypeScript προσφέρει ένα ισχυρό εργαλείο για την ανάπτυξη ποιοτικού και αξιόπιστου κώδικα.

2.4 Η έννοια του διακομιστή

Ένας συμβατικός διακομιστής (server), γραμμένος για παράδειγμα σε Node.js [7] με τη χρήση του framework Express.Js [8], αποτελεί το κέντρο της λειτουργίας ενός web περιβάλλοντος. Η βασική λογική του είναι να ακούει για εισερχόμενα αιτήματα από τους πελάτες και να ανταποκρίνεται ανάλογα, παρέχοντας τα απαιτούμενα δεδομένα ή υπηρεσίες. Αυτό μπορεί να περιλαμβάνει την ανάγνωση και την εγγραφή δεδομένων από ή προς μια βάση δεδομένων, την εκτέλεση επιχειρηματικής λογικής, την επικοινωνία με άλλες υπηρεσίες ή API, και άλλες λειτουργίες που απαιτούνται για την εξυπηρέτηση των αιτημάτων των πελατών (client-browser).

Αυτό το μεμονωμένο σύστημα λειτουργεί με βάση το REST (Representational State Transfer [9]), το οποίο είναι ένα απλό πρότυπο αρχιτεκτονικής για τη διάδοση των δεδομένων σε ένα δίκτυο. Τα αιτήματα από τους πελάτες αντιμετωπίζονται με βάση τις ποικίλες διαδρομές (routes) που έχουν καθοριστεί στον διακομιστή, με κάθε διαδρομή να αντιστοιχεί σε μια συγκεκριμένη λειτουργία ή υπηρεσία. Ο διακομιστής ανταποκρίνεται σε κάθε αίτημα με τα αντίστοιχα δεδομένα ή κωδικό κατάστασης HTTP, σύμφωνα με την επεξεργασία που πρέπει να γίνει. Η διάρκεια της σύνδεσης είναι αμιγώς πρωτόκολλο HTTP, χωρίς καμία μορφή διατήρησης κατάστασης μεταξύ αιτημάτων. Αυτή η δομή επιτρέπει την απλή, αποδοτική και επεκτάσιμη επεξεργασία των αιτημάτων από τον διακομιστή.

2.5 Front End - User Interface

Τον καιρό του 2000, η ανάπτυξη ιστοσελίδων είχε έναν διαφορετικό τρόπο σε σχέση με τη σημερινή πραγματικότητα. Οι περισσότερες ιστοσελίδες κατασκευάζονταν χρησιμοποιώντας στατικά αρχεία HTML, τα οποία περιείχαν το περιεχόμενο της σελίδας και τη δομή της. Οι εικόνες και άλλα πολυμέσα ήταν συνήθως ενσωματωμένα απευθείας στο HTML. Η αλληλεπίδραση με τον χρήστη συνήθως επιτυγχάνονταν με τη χρήση φορμών HTML και με την ανακατεύθυνση του χρήστη σε νέες σελίδες κατά την πλοήγηση. Αυτή η στατική προσέγγιση είχε τα πλεονεκτήματά της σε απλότητα και σταθερότητα, αλλά έκανε δύσκολη τη διαχείριση μεγάλου όγκου περιεχομένου και περιορίζει τη δυνατότητα δημιουργίας δυναμικών και αλληλεπιδραστικών εφαρμογών. Την εποχή εκείνη, οι προγραμματιστές επικεντρώνονταν κυρίως στη δημιουργία στατικών σελίδων με HTML και CSS, ενώ η JavaScript χρησιμοποιούνταν συνήθως για μικρές ενδείξεις λειτουργικότητας όπως η επικύρωση φόρμας. Το μοντέλο ανάπτυξης εφαρμογών που χρησιμοποιούταν εκείνη την περίοδο λέγεται πολυσέλιδο (multi page [10]).

Η μετάβαση από το πολυσέλιδο στο μονοσέλιδο (single page [11]) μοντέλο εφαρμογών πηγάζει από την ανάγκη για βελτιωμένη εμπειρία χρήσης και απόκριση στις απαιτήσεις των σύγχρονων χρηστών. Τα πολυσέλιδα εφαρμογές συχνά απαιτούν ανανέωση της σελίδας κάθε φορά που ο χρήστης πλοηγείται σε μια νέα σελίδα, κάτι που μπορεί να οδηγήσει σε καθυστερήσεις και αρνητική εμπειρία χρήστη. Αντίθετα, οι μονοσέλιδες εφαρμογές διαθέτουν ένα μόνο σημείο πλοήγησης, το οποίο διαχειρίζεται δυναμικά την αλληλεπίδραση του χρήστη με την εφαρμογή. Αυτό επιτρέπει την άμεση αλληλεπίδραση και τη φόρτωση δεδομένων με την απαιτούμενη αποδοτικότητα, δημιουργώντας μια πιο ομαλή και ευχάριστη εμπειρία για τον χρήστη. Με την εκμετάλλευση των προηγμένων τεχνολογιών όπως η AJAX και η αρχιτεκτονική του REST API, τα μονοσέλιδα μοντέλα εφαρμογών επιτυγχάνουν

τη μέγιστη απόδοση και αποκρίνονται αποτελεσματικά στις ανάγκες μιας σύγχρονης, διαδραστικής και δυναμικής εμπειρίας χρήστη.

2.5.1 Η ανάγκη για τη δημιουργία UI βιβλιοθηκών (React.Js)

Η μετάβαση από την απλή χρήση HTML, CSS και JavaScript σε πιο προηγμένες βιβλιοθήκες και πλατφόρμες όπως η React.Js [12] (ή γνωστή στους προγραμματιστές ως απλά React) πηγάζει από την ανάγκη για αποτελεσματικότερη και διαχειρίσιμη ανάπτυξη εφαρμογών. Η αυξανόμενη πολυπλοκότητα των σύγχρονων ιστοσελίδων και εφαρμογών απαιτεί τη χρήση προηγμένων εργαλείων και πλατφορμών που επιτρέπουν την αποδοτική διαχείριση του κώδικα και την οργάνωση των συνιστωσών της εφαρμογής.

Η React είναι μια ανοιχτού κώδικα βιβλιοθήκη JavaScript για τη δημιουργία χρήστης διεπαφών (UI). Δημιουργήθηκε από τους προγραμματιστές της Meta [13] το 2013 και κατασκευάστηκε με στόχο την αποδοτική και ευέλικτη δημιουργία εφαρμογών με μεγάλο όγκο δεδομένων. Μία από τις βασικές έννοιες της React είναι η "component-based" [14] αρχιτεκτονική, όπου η εφαρμογή διαιρείται σε μικρά, ανεξάρτητα συστατικά (components), κάθε ένα από τα οποία είναι υπεύθυνο για μια συγκεκριμένη λειτουργία ή εμφάνιση στην οθόνη. Αυτό διευκολύνει την επαναχρησιμοποίηση κώδικα και τη διαχείριση της κατάστασης της εφαρμογής.

Ένα από τα κυριότερα χαρακτηριστικά της React είναι η εικονική DOM (Virtual DOM [15]). Αυτό σημαίνει ότι η React διαχειρίζεται μια εικονική αναπαράσταση του DOM της εφαρμογής στη μνήμη και συγκρίνει αυτή την αναπαράσταση με τον πραγματικό DOM, εντοπίζοντας τις αλλαγές και ενημερώνοντας μόνο τα απαραίτητα σημεία, κάτι που οδηγεί σε μεγάλη βελτίωση της απόδοσης και της αποκρισιμότητας της εφαρμογής.

Η React χρησιμοποιείται ευρέως σε πολλές εφαρμογές, από μικρές ιστοσελίδες έως και μεγάλα επιχειρησιακά συστήματα. Είναι συμβατή με πολλές άλλες τεχνολογίες και βιβλιοθήκες και έχει ένα ευρύ οικοσύστημα εργαλείων και πρόσθετων (add-ons) που την καθιστούν εύκολη στη χρήση και επεκτάσιμη. Επιπλέον, η αναπτυξιακή κοινότητα της React είναι ενεργή και ανοικτή, προσφέροντας πολλούς πόρους, παραδείγματα και υποστήριξη σε νέους και έμπειρους προγραμματιστές. Με όλα αυτά, η React αποτελεί μια ισχυρή και ευέλικτη επιλογή για την ανάπτυξη σύγχρονων web εφαρμογών.

Να μην ξεχάσουμε πως πίσω από όλη αυτή η λειτουργικότητα κρύβεται ένας διακομιστής που σερβίρει μια εφαρμογή React και αναλαμβάνει την εξυπηρέτηση του αρχικού HTML, των στιλ και των script που απαιτούνται για την απόδοση της εφαρμογής στον περιηγητή. Αυτός ο διακομιστής πρέπει να είναι σε θέση να κατανείμει τα αρχεία της React εφαρμογής σε απόκριση στα αιτήματα που λαμβάνει από τους πελάτες. Η παροχή του αρχικού HTML και του bundle [16] της εφαρμογής είναι ουσιαστική για την αποτελεσματική αναπαραγωγή της εφαρμογής στον περιηγητή του χρήστη. Ταυτόχρονα, αυτός ο διακομιστής πρέπει να διαχειρίζεται δυναμικά τις διαδρομές της εφαρμογής React, ώστε να επιστρέφει τις σωστές σελίδες ανάλογα με το URL που ζητά ο πελάτης.

2.5.2 Next.Js

Το Next.js [17] δημιουργήθηκε ως μια απάντηση σε κάποιες από τις προκλήσεις που αντιμετώπιζαν οι προγραμματιστές κατά την ανάπτυξη εφαρμογών με React. Αποτελεί ένα πλαίσιο εργασίας για την React, δηλαδή προσφέρει επιπρόσθετη λειτουργικότητα σε αυτή. Αν και η React παρέχει μια ισχυρή

βάση για τη δημιουργία δυναμικών και ευέλικτων εφαρμογών, η διαχείριση των πολλαπλών σελίδων, των δρομολογητών (routers), και της απόδοσης μπορούσε να γίνει περίπλοκη σε μεγαλύτερα και πιο περίπλοκα έργα.

Το Next.js προσφέρει πολλαπλά οφέλη και λειτουργίες που καθιστούν την ανάπτυξη εφαρμογών React πιο αποδοτική και ευέλικτη. Ανάμεσα στα κύρια πλεονεκτήματά του περιλαμβάνουν:

1. **Server-Side Rendering (SSR) [18]**: Το Next.js επιτρέπει την εκτέλεση του κώδικα της React στον διακομιστή, παρέχοντας προεπισκόπηση των σελίδων και βελτιώνοντας το SEO. Αυτό εξασφαλίζει πιο γρήγορη φόρτωση και απόδοση της εφαρμογής.
2. **Δυναμική Προεπισκόπηση (Pre-rendering) [19]**: Το Next.js υποστηρίζει την προεπισκόπηση σελίδων κατά το build time, επιτρέποντας τη δημιουργία στατικών αρχείων HTML για τις σελίδες που δεν αλλάζουν συχνά. Αυτό βελτιώνει την απόδοση και την εμπειρία των χρηστών.
3. **Δυνατότητα API Routes**: Το Next.js διαθέτει ενσωματωμένη υποστήριξη για δημιουργία API routes, επιτρέποντας τη δημιουργία back-end λειτουργικότητας χωρίς την ανάγκη ενός ξεχωριστού back-end server.
4. **Διαχείριση Κατάστασης (State Management)**: Το Next.js είναι συμβατό με πολλές βιβλιοθήκες διαχείρισης κατάστασης, όπως το Zustand, toRedux και το MobX, παρέχοντας λύσεις για την αποθήκευση και τη διαχείριση της κατάστασης της εφαρμογής.
5. **Ευέλικτη Διαχείριση Δρομολογητών (Routing)**: Οι ενσωματωμένοι δρομολογητές του Next.js επιτρέπουν τη διαχείριση διαδρομών και την πλοήγηση μεταξύ των σελίδων με ευκολία, βελτιώνοντας την εμπειρία του χρήστη.
6. **Αυτόματος Φορτωτής (Automatic Code Splitting [20])**: Το Next.js διαχειρίζεται τον διαχωρισμό του κώδικα σε διάφορα αρχεία, βελτιώνοντας τον χρόνο φόρτωσης και την απόδοση της εφαρμογής.

Όλα αυτά τα χαρακτηριστικά καθιστούν το Next.js ένα ιδανικό εργαλείο για τη δημιουργία προηγμένων και αποδοτικών React εφαρμογών, ενώ παράλληλα προσφέρει μια ευέλικτη και φιλική προς τον προγραμματιστή εμπειρία ανάπτυξης.

2.5.3 Ταυτοποίηση, εξουσιοδότηση και Next Auth

Η ταυτοποίηση (authentication) και η εξουσιοδότηση (authorization) είναι δύο ζωτικές διαδικασίες στον κόσμο της ασφάλειας και της πρόσβασης στο διαδίκτυο. Η ταυτοποίηση αφορά τη διαδικασία επαλήθευσης της ταυτότητας ενός χρήστη, δηλαδή τη βεβαίωση ότι ο χρήστης είναι όντως αυτό που ισχυρίζεται ότι είναι, χρησιμοποιώντας πιστοποιητικά, κωδικούς πρόσβασης, βιομετρικά δεδομένα κ.λπ. Η εξουσιοδότηση, από την άλλη πλευρά, αφορά τη διαδικασία ελέγχου των δικαιωμάτων πρόσβασης ενός χρήστη σε συγκεκριμένους πόρους ή λειτουργίες μέσα στην εφαρμογή.

Το Next Auth [21] είναι μια βιβλιοθήκη που παρέχει έτοιμες λύσεις για την ταυτοποίηση και την εξουσιοδότηση σε εφαρμογές React με τη χρήση του Next.js framework. Μέσω του Next Auth, οι προγραμματιστές μπορούν να ενσωματώσουν εύκολα και γρήγορα διάφορες μορφές ταυτοποίησης, όπως σύνδεση με κοινωνικά δίκτυα (όπως Google, Facebook, Twitter κλπ.), ή ακόμη και πιστοποιητικά της επιχείρησης, στις εφαρμογές τους. Επιπλέον, παρέχει μια εύκολη διαδικασία για τον έλεγχο των δικαιωμάτων πρόσβασης των χρηστών, καθώς και ευέλικτες επιλογές ρύθμισης των ρόλων και των αδειών πρόσβασης. Με το Next Auth, οι προγραμματιστές μπορούν να υλοποιήσουν γρήγορα και

αποτελεσματικά ασφαλείς λύσεις για τη διαχείριση ταυτοποίησης και εξουσιοδότησης στις εφαρμογές τους.

2.5.3 UI Component Libraries

Οι βιβλιοθήκες χρήστη (UI libraries) δημιουργήθηκαν για να αντιμετωπίσουν συγκεκριμένα προβλήματα που αντιμετωπίζουν οι προγραμματιστές κατά τη δημιουργία επαγγελματικών και ευανάγνωστων διεπαφών χρήστη (UIs). Η ανάπτυξη μιας εξαιρετικής διεπαφής χρήστη απαιτεί συνήθως πολλές ώρες σχεδιασμού και ανάπτυξης. Οι UI βιβλιοθήκες έρχονται να αντιμετωπίσουν αυτήν την πρόκληση παρέχοντας έτοιμα στοιχεία που μπορούν να χρησιμοποιηθούν αμέσως, χωρίς την ανάγκη για πλήρη ανάπτυξη από το μηδέν. Αυτό επιτρέπει στους προγραμματιστές να επικεντρωθούν στη λογική της εφαρμογής και στην επικοινωνία με τον χρήστη, αντί να ασχολούνται με λεπτομέρειες σχεδιασμού.

Οι UI βιβλιοθήκες συνήθως προσφέρουν ένα ευρύ φάσμα επαναχρησιμοποιήσιμων στοιχείων, τα οποία μπορούν να προσαρμοστούν και να επεκταθούν για να ανταποκριθούν στις ανάγκες της συγκεκριμένης εφαρμογής. Αυτό επιτρέπει την εύκολη δημιουργία σύνθετων διεπαφών χρήστη χωρίς την ανάγκη για εξειδικευμένες γνώσεις σχεδιασμού. Επιπλέον, οι βιβλιοθήκες αυτές παρέχουν μια ενοποιημένη εμπειρία σχεδίασης και ανάπτυξης, επιτρέποντας στους προγραμματιστές να επικεντρώνονται στην παραγωγικότητα και την ανάπτυξη υψηλής ποιότητας εφαρμογών.

Τέλος, οι βιβλιοθήκες αυτές συχνά υποστηρίζονται από μια ενεργή κοινότητα που παρέχει συνεχείς ενημερώσεις και βελτιώσεις, ενισχύοντας την αξιοπιστία και την ανάπτυξή τους στον χρόνο. Μερικές από τις πιο δημοφιλείς βιβλιοθήκες UI είναι το Material UI και το Mantine UI. Στην παρούσα εφαρμογή θα χρησιμοποιήσουμε το Mantine UI.

2.5.4 Mantine UI

Το Mantine UI είναι μια σύγχρονη βιβλιοθήκη χρήστη (UI library) για τη δημιουργία εξεζητημένων και προσιτών διεπαφών χρήστη σε React εφαρμογές. Σχεδιασμένο με έμφαση στην απλότητα και την ευελιξία, το Mantine παρέχει ένα ευρύ φάσμα έτοιμων στοιχείων UI που μπορούν να προσαρμοστούν εύκολα στις ανάγκες της εφαρμογής μας.

Το Mantine προσφέρει μια ποικιλία από στοιχεία, όπως κουμπιά, φόρμες, επιλογείς, πίνακες, καρτέλες και πολλά άλλα, τα οποία είναι σχεδιασμένα με σύγχρονο και καθαρό στιλ. Η βιβλιοθήκη επικεντρώνεται στην απλότητα χρήσης και στην ευκολία προσαρμογής, επιτρέποντας σε εμάς τους προγραμματιστές να δημιουργήσουμε ευέλικτες και χρήσιμες διεπαφές χρήστη με ελάχιστο κώδικα.
Μια από τις κύριες δυνατότητες του Mantine είναι η χρήση του απλού, αλλά ισχυρού API που παρέχει, το οποίο επιτρέπει την εύκολη προσαρμογή και επέκταση των στοιχείων UI που παρέχει. Επιπλέον, η βιβλιοθήκη προσφέρει ενσωματωμένες δυνατότητες για αντίδραση και αναπαραγωγή των στοιχείων ώστε να μπορούμε να δούμε απεθείας στην ιστοσελίδα του τις αλλαγές που θέλουμε, καθώς και υποστήριξη για θεματοποίηση και προσαρμογή του στιλ της εφαρμογής.

Η βιβλιοθήκη Mantine προσφέρει επίσης εξαιρετική τεκμηρίωση και υποστήριξη, καθώς και μια ενεργή κοινότητα που συμμετέχει στην ανάπτυξή της. Με τη συχνή ενημέρωση και τη διαρκή προσθήκη νέων χαρακτηριστικών, η βιβλιοθήκη διατηρείται σε υψηλό επίπεδο και μας παρέχει ό,τι

χρειάζόμαστε για τη δημιουργία ποιοτικών εφαρμογών. Επιπλέον, η κοινότητα του Mantine είναι ενεργή και φιλόξενη, παρέχοντας υποστήριξη και ανταλλαγή γνώσεων μεταξύ των μελών της. Με αυτά τα χαρακτηριστικά, το Mantine αποτελεί ένα ισχυρό εργαλείο για τους προγραμματιστές που επιθυμούν να δημιουργήσουν επαγγελματικές και ευέλικτες διεπαφές χρήστη με απλότητα και αποτελεσματικότητα.

2.5.5 Διαχείριση κατάστασης και Zustand

Η διαχείριση κατάστασης (state management [22]) αναφέρεται στον τρόπο με τον οποίο διαχειριζόμαστε την κατάσταση της React εφαρμογής μας. Κάθε φορά που μια κατάσταση εφαρμογής αλλάζει, όπως για παράδειγμα όταν ένα στοιχείο ενεργοποιείται ή απενεργοποιείται, χρειάζεται ένας τρόπος αποθήκευσης και ενημέρωσης αυτών των αλλαγών. Η αποτελεσματική διαχείριση της κατάστασης είναι κρίσιμη για τη διασφάλιση της συνοχής και της ακεραιότητας της εφαρμογής.

Το Zustand είναι μια βιβλιοθήκη state management για τις εφαρμογές React που προσφέρει έναν απλό και εύχρηστο τρόπο διαχείρισης της κατάστασης σε global επίπεδο. Με το Zustand, μπορούμε να δημιουργήσουμε μια γενική κατάσταση (global state) για την εφαρμογή μας και να τη μοιραστούμε μεταξύ διαφορετικών component χωρίς να υπάρχει άμεση σχέση αυτών μέσα στο δέντρο της React. Η κύρια ιδέα πίσω από το Zustand είναι η απλότητα και η ευελιξία. Μπορούμε να οργανώσουμε το state μας σε slices και να το ενημερώσουμε με ελάχιστο κώδικα.

Το Zustand προσφέρει επίσης προηγμένες λειτουργίες όπως τα devtools για ευκολότερο debugging και τη δυνατότητα χρήσης με middleware [23] για προσθήκη πρόσθετης λειτουργικότητας στο state management. Με αυτά τα χαρακτηριστικά, το Zustand είναι μια ισχυρή επιλογή για τη διαχείριση της κατάστασης σε εφαρμογές React, προσφέροντας ταυτόχρονα απλότητα και ευελιξία.

2.5.6 Επικύρωση δεδομένων και Zod

Η επικύρωση δεδομένων είναι μια σημαντική διαδικασία στην ανάπτυξη λογισμικού, που στοχεύει στη διασφάλιση της ακρίβειας και της ορθότητας των δεδομένων που χρησιμοποιούνται στην εφαρμογή. Αποτελεί έναν κρίσιμο μηχανισμό για την αντιμετώπιση πιθανών σφαλμάτων και ασφάλειας, καθώς επιτρέπει στους προγραμματιστές να ελέγχουν τη συμφωνία και τη συνέπεια των δεδομένων που εισάγονται, επεξεργάζονται ή αποθηκεύονται στην εφαρμογή.

Το Zod [24] είναι μια ελαφριά και ευέλικτη βιβλιοθήκη επικύρωσης σχημάτων για TypeScript και JavaScript, που προσφέρει έναν απλό και δυναμικό τρόπο για τον έλεγχο των τύπων δεδομένων. Χρησιμοποιώντας το Zod, μπορούμε να καθορίσουμε και να επικυρώνουμε τα σχήματα των δεδομένων στην εφαρμογή μας, εξασφαλίζοντας έτσι την ορθότητα και τη συνέπεια των δεδομένων κατά τον χειρισμό τους. Με το Zod, οι προγραμματιστές μπορούν να δημιουργήσουν εύκολα σχήματα δεδομένων, να εφαρμόσουν αυστηρούς ελέγχους επαλήθευσης και να διατηρήσουν μια ομαλή και ασφαλή λειτουργία της εφαρμογής τους. Κατά συνέπεια, το Zod αποτελεί ένα ισχυρό εργαλείο για την επιβεβαίωση της ορθότητας των δεδομένων και τη διασφάλιση της αξιοπιστίας και της ασφάλειας της εφαρμογής.

2.5.7 Διαχείριση ημερομηνιών και Day.js

Η επεξεργασία ημερομηνιών αποτελεί σημαντικό μέρος της ανάπτυξης λογισμικού και εφαρμογών, καθώς η ανάγκη για διαχείριση ημερομηνιών είναι συχνή σε πολλά πεδία, όπως η ανάλυση δεδομένων, η παρουσίαση πληροφοριών, οι χρονολογίες και πολλά άλλα. Ωστόσο, η επεξεργασία ημερομηνιών μπορεί να είναι σύνθετη και προκλητική λόγω των διαφορετικών μορφών και των διαφορετικών απαιτήσεων που προκύπτουν στην πορεία της ανάπτυξης.

Σε αυτό το πλαίσιο, η βιβλιοθήκη day.js [25] προσφέρει μια εύχρηστη και ευέλικτη λύση για την επεξεργασία ημερομηνιών σε εφαρμογές JavaScript. Με τη day.js, οι προγραμματιστές μπορούν εύκολα να δημιουργήσουν, να μετατρέψουν και να μορφοποιήσουν ημερομηνίες με ελάχιστο κώδικα και ευκολία. Επιπλέον, η day.js προσφέρει ευέλικτες λειτουργίες για τη διαχείριση της ζώνης ώρας, την εκτέλεση πράξεων μεταξύ ημερομηνιών και την προσαρμογή των ημερομηνιών σύμφωνα με τις ανάγκες της εφαρμογής.

Επιπλέον, οι προηγμένες δυνατότητες που προσφέρει η day.js, όπως η επιθεώρηση των ημερομηνιών, η προσθαφαίρεση χρόνων με εύκολο και ξεκάθαρο τρόπο σε σχέση με τα εργαλεία που διαθέτει η απλή JavaScript, καθιστούν τη βιβλιοθήκη αυτή μια ιδανική επιλογή για τη διαχείριση ημερομηνιών σε εφαρμογές JavaScript.

2.6 Back-End

Η ανάγκη για πιο διαδραστικές εφαρμογές αποτελεί έναν από τους κύριους παράγοντες που οδήγησαν στην ανάπτυξη και εξέλιξη του Back-end. Καθώς οι χρήστες αναζητούν ολοένα και πιο δυναμικές και αλληλεπιδραστικές εμπειρίες στις εφαρμογές τους, το Back-end ανέλαβε τον ρόλο της δημιουργίας αυτών των εμπειριών. Αντί οι εφαρμογές να περιορίζονται στην απλή παρουσίαση στατικού περιεχομένου, το Back-end παρέχει τη δυνατότητα για δυναμική διαχείριση δεδομένων, προσωποποίησης εμφάνισης και ανταπόκρισης στις ενέργειες των χρηστών. Επιπλέον, η ανάπτυξη ενός αποτελεσματικού Back-end επιτρέπει την υλοποίηση πολύπλοκης επιχειρηματικής λογικής, τη διαχείριση των διαφόρων αιτημάτων του Front-end και την ασφαλή αποθήκευση και διαχείριση των δεδομένων.

Το Back-end αποτελεί το καίριο κομμάτι της αρχιτεκτονικής μιας εφαρμογής και προσφέρει απαραίτητες λειτουργίες για την αποθήκευση, την επεξεργασία και την παρουσίαση δεδομένων στους χρήστες. Ένα αποτελεσματικό Back-end παρέχει ασφαλή διαχείριση της βάσης δεδομένων, επικοινωνία με το Front-end για τη μεταφορά δεδομένων και την ανταλλαγή πληροφοριών, καθώς και την υποστήριξη των απαιτήσεων ασφαλείας και αυθεντικοποίησης των χρηστών. Επιπλέον, το Back-end μπορεί να προσφέρει λειτουργίες όπως η διαχείριση χρηστών, η εκτέλεση επιχειρηματικής λογικής, η διαχείριση των ειδοποιήσεων και η γενική επεξεργασία αιτημάτων από το Front-end. Με άλλα λόγια, το Back-end αποτελεί την κινητήρια δύναμη πίσω από τη λειτουργία μιας εφαρμογής και είναι ουσιώδες για την παροχή των απαραίτητων υπηρεσιών στους χρήστες.

Στα επόμενα υποκεφάλαια, θα εστιάσουμε σε μερικές από τις βασικές πτυχές του Back-end, εξετάζοντας τις λειτουργίες, την αρχιτεκτονική και τα εργαλεία που χρησιμοποιούνται για τη δημιουργία δυναμικών και αποδοτικών εφαρμογών. Αναλύοντας την αρχιτεκτονική, θα εξετάσουμε τη

διαμόρφωση του Back-end ώστε να υποστηρίζει την αποτελεσματική διαχείριση των αιτημάτων του Front-end και τη διαχείριση των δεδομένων. Στη συνέχεια, θα εξετάσουμε τις διάφορες λειτουργίες του, όπως τη διαχείριση των χρηστών, την αυθεντικοποίηση και την εξουσιοδότηση, τη διαχείριση των δεδομένων και τις διαφορετικές υπηρεσίες που παρέχονται στο Front-end. Τέλος, θα εξετάσουμε τις τεχνολογίες και τα εργαλεία που χρησιμοποιούνται στην ανάπτυξη του Back-end, εστιάζοντας στην αποτελεσματικότητά τους και τη συμβολή τους στη δημιουργία υψηλής ποιότητας εφαρμογών.

2.6.1 Next Js ως Backend

Ο συμβατικός τρόπος ανάπτυξης λογισμικού επιφέρει τη δημιουργία ξεχωριστών servers για κάθε λειτουργικότητα, διαιρώντας την εφαρμογή σε μικρά μοναδικά τμήματα. Αυτή η προσέγγιση, ενώ έχει τα πλεονεκτήματά της, μπορεί να παρουσιάσει προκλήσεις σε ό,τι αφορά την διαχείριση και συντήρηση των επί μέρους λειτουργιών. Ωστόσο η εμφάνιση του Next.js προσφέρει μιαν ευέλικτη λύση που αποκλείει την ανάγκη για πολλαπλούς servers.

Στον πυρήνα του Next.js βρίσκεται η ιδέα του "zero configuration" [26], που επιτρέπει στους προγραμματιστές να επικεντρωθούν στην ανάπτυξη της λειτουργικότητας, αντί να ασχολούνται με την ρύθμιση και τη σύνταξη κώδικα για τη διαμόρφωση του server. Η εξασφάλιση ενός μονοστιγμιαίου server που συνδυάζει τη λειτουργικότητα του front-end και του back-end επιτρέπει τη διευκόλυνση του κύκλου ανάπτυξης, μειώνοντας τον χρόνο που απαιτείται για τη δημιουργία, εκτέλεση και συντήρηση της εφαρμογής.

Η δυνατότητα να έχουμε ένα μόνο server που υποστηρίζει τόσο το front-end όσο και το back-end συντελεί στη βελτίωση της απόδοσης και της εμπειρίας του χρήστη, αφού εξαλείφει το κόστος των πολλαπλών αιτήσεων προς διαφορετικούς servers για την απόκριση σε μια σελίδα. Το Next.js διευκολύνει την ανάπτυξη δυναμικών και διαδραστικών εφαρμογών, προσφέροντας μιαν εξελιγμένη, συνολική λύση για τη δημιουργία μοντέρνων web εφαρμογών.

2.6.2 TRPC

Το TRPC [27], ή "Typed RPC", αποτελεί ένα ισχυρό εργαλείο που συμπληρώνει το Next.js στη διαδικασία ανάπτυξης ενός πλήρους web περιβάλλοντος. Η χρησιμότητά του πηγάζει από τη δυνατότητά του να παρέχει μια δομημένη και τύπική προσέγγιση στο Remote Procedure Calls (RPC), επιτρέποντας στους προγραμματιστές να οργανώνουν και να κατανοούν πλήρως τις αλληλεπιδράσεις μεταξύ του server και του client.

Με το TRPC, η διαδικασία ανάπτυξης γίνεται ακόμα πιο αποτελεσματική και ασφαλής. Ο τύπος των δεδομένων που μεταδίδονται καθίσταται πιο σαφής, επιτρέποντας την ανίχνευση σφαλμάτων και την αντιμετώπισή τους προτού αυτά προκαλέσουν προβλήματα στην παραγωγική λειτουργία. Επιπλέον, η δυνατότητα της τύπικής επικοινωνίας μεταξύ του client και του server μειώνει τον κίνδυνο λάθους στον κώδικα και βελτιστοποιεί τη διαχείριση των αιτημάτων.

Ένα ακόμα πλεονέκτημα του TRPC είναι η ευκολία στην παραμετροποίηση και την επαναχρησιμοποίηση κώδικα. Η δομή του επιτρέπει την εύκολη δημιουργία και διαχείριση endpoints, καθώς και τη διαχωρισμό της λειτουργικότητας σε μικρότερα, πιο ευέλικτα τμήματα κώδικα. Αυτό επιτρέπει την επαναχρησιμοποίηση και τη διαχείριση των εν λόγω τμημάτων με αποτελεσματικότητα,

μειώνοντας τον χρόνο και το κόστος ανάπτυξης. Θα δούμε σε επόμενο κεφάλαιο την ευκολία χρήσης του καθώς και το πόση πολλή λειτουργικότητα μπορούμε να πετύχουμε με ελάχιστο κώδικα.

Συνολικά, το TRPC αποτελεί ένα εξαιρετικά χρήσιμο εργαλείο για το Next.js, παρέχοντας την απαιτούμενη δομική σταθερότητα, αποδοτικότητα και ασφάλεια στην ανάπτυξη λογισμικού.

2.6.3 SQL και η ανάγκη για τη δημιουργία του Prisma

Με την ανάπτυξη του Backend προέκυψε η ανάγκη για τη δημιουργία μιας μόνιμης αποθήκευσης δεδομένων, γνωστής και ως persistent storage, η οποία θα επέτρεπε τη διατήρηση των πληροφοριών του χρήστη μεταξύ διαφορετικών συνεδριών. Αυτό επιτυγχάνεται μέσω βάσεων δεδομένων, όπως οι σχεσιακές βάσεις δεδομένων SQL ή οι NoSQL αποθηκευτικοί μηχανισμοί, που επιτρέπουν την αποθήκευση, την λήψη και τη διαχείριση δεδομένων με αποτελεσματικό και ασφαλή τρόπο. Αυτός ο τύπος αποθήκευσης δεδομένων είναι κρίσιμος για τη δημιουργία λειτουργικών εφαρμογών, καθώς εξασφαλίζει τη συνέπεια και την ακεραιότητα των πληροφοριών, ενώ παράλληλα παρέχει τη δυνατότητα εύκολης πρόσβασης και αναζήτησης των δεδομένων από το Backend.

Η SQL (Structured Query Language) είναι μια γλώσσα προγραμματισμού που χρησιμοποιείται για τη διαχείριση και τον ερωτηματολόγηση δεδομένων σε σχεσιακές βάσεις δεδομένων. Αποτελεί ένα προτυποποιημένο μέσο για την αλληλεπίδραση με σχεσιακές βάσεις δεδομένων, επιτρέποντας την εκτέλεση διαφόρων ενεργειών, όπως εισαγωγή, λήψη, επεξεργασία και διαγραφή δεδομένων (CRUD που σημαίνει Create, Read, Update, Delete).

Με τη βοήθεια της SQL, οι προγραμματιστές και οι διαχειριστές βάσεων δεδομένων μπορούν να δημιουργήσουν και να διαχειριστούν πίνακες, να ορίσουν σχέσεις μεταξύ δεδομένων, και να εκτελέσουν πολύπλοκες επερωτήσεις για την λήψη συγκεκριμένων πληροφοριών. Επιπλέον, η SQL παρέχει δυνατότητες για τη διαχείριση της ασφάλειας και της απόδοσης της βάσης δεδομένων, καθώς και για τη δημιουργία αποθηκευμένων διαδικασιών (stored procedures) και συναρτήσεων για προηγμένη επεξεργασία δεδομένων.

Για τους προγραμματιστές που χρησιμοποιούσαν τη JavaScript ως βασική γλώσσα προγραμματισμού εφαρμογών, η δημιουργία ενός εργαλείου όπως το Prisma [28] ήταν φυσικό επόμενο. Το Prisma προσφέρει μια ολοκληρωμένη λύση για τη διαχείριση της βάσης δεδομένων σε ένα οικοσύστημα JavaScript, καθιστώντας την ανάπτυξη και τη συντήρηση των εφαρμογών πιο αποτελεσματικές και εύκολες. Με τη δυνατότητα να αλληλεπιδράσει απευθείας με τη βάση δεδομένων και να παρέχει σύγχρονες λειτουργίες όπως η μεταφορά δεδομένων και η εύρεση δεδομένων, το Prisma καλύπτει τις ανάγκες των προγραμματιστών που επιλέγουν τη JavaScript για την ανάπτυξη εφαρμογών. Με τον απλό τρόπο που παρέχει τη δυνατότητα διαχείρισης της βάσης δεδομένων, το Prisma αποτελεί ένα αναπόσπαστο εργαλείο για την επίτευξη ευέλικτων και αποδοτικών λύσεων στον κόσμο της JavaScript ανάπτυξης.

Σε επόμενο κεφάλαιο, θα εξετάσουμε την απλότητα της δημιουργίας μιας βάσης δεδομένων σε σχέση με την SQL με ένα απλό σχήμα χρησιμοποιώντας το Prisma. Θα δούμε πώς μπορούμε να ορίσουμε τα μοντέλα μας και να ορίσουμε τις σχέσεις μεταξύ τους με έναν εύκολο και κατανοητό τρόπο. Επίσης, θα

εξετάσουμε πώς μπορούμε να χρησιμοποιήσουμε τις διάφορες δυνατότητες που παρέχει το Prisma για τη δημιουργία και τη διαχείριση των πινάκων και των σχέσεων στη βάση δεδομένων μας.

2.7 Ο συνδυασμός των τεχνολογιών

Στο πλαίσιο της παρούσας εργασίας, θα εξετάσουμε πώς η χρήση των προαναφερθέντων τεχνολογιών μπορεί να συμβάλει στην ασφάλεια και την αξιοπιστία του κώδικα. Το TRPC, σε συνδυασμό με το Next.js, προσφέρει ένα αποδοτικό σύστημα επικοινωνίας μεταξύ client και server, ενώ το Prisma παρέχει μια ασφαλή σύνδεση με τη βάση δεδομένων και προστατεύει το σύστημα από πιθανά προβλήματα ασφαλείας που σχετίζονται με SQL ενέργειες. Το Zod, τέλος, βοηθά στην εξασφάλιση της σωστής μορφοποίησης και επαλήθευσης των δεδομένων, προσφέροντας ένα ευέλικτο σύστημα επικύρωσης δεδομένων στο πλαίσιο του TypeScript. Μέσω αυτών των τεχνολογιών, επιτυγχάνεται ένα υψηλό επίπεδο ασφάλειας και αξιοπιστίας στην ανάπτυξη κώδικα.

Κεφάλαιο 3ο: Υλοποίηση του Foodie

Κατά τη διάρκεια αυτού του κεφαλαίου, θα εξετάσουμε τη σχεδίαση και την υλοποίηση της εφαρμογής, ξεκινώντας με τη χρήση user stories για τη διαμόρφωση της δομής της. Στη συνέχεια, θα εστιάσουμε στις τεχνολογικές λύσεις που επιλέχθηκαν, αναλύοντας τις βιβλιοθήκες και τα εργαλεία που επηρέασαν την ανάπτυξη. Τέλος, θα δούμε την αλληλεπίδραση της εφαρμογής με τη βάση δεδομένων, χωρίς να παραλείψουμε να εξετάσουμε το ανέβασμα εικόνων σε αποθηκευτικό χώρο.

3.1 Σχεδίαση και Λειτουργικές απαιτήσεις

Η ιδέα για την εφαρμογή εύρεσης συνταγών πηγάζει από την ανάγκη να βρίσκουμε εύκολα και γρήγορα νόστιμες και υγιεινές συνταγές με βάση τα υλικά που έχουμε διαθέσιμα στο σπίτι μας. Ως λάτρης της μαγειρικής, συχνά έρχομαι αντιμέτωπος με το πρόβλημα του τι να μαγειρέψω, έχοντας περιορισμένο χρόνο και υλικά. Πιστεύω πως η ανάγκη αυτή αγγίζει και πολλούς άλλους χρήστες, καθώς η καθημερινότητα μπορεί να είναι γεμάτη με υποχρεώσεις, αφήνοντας ελάχιστο χρόνο για την αναζήτηση και προετοιμασία φαγητού.

Με αφετηρία αυτή την ανάγκη, στόχος μου ήταν η δημιουργία μιας εφαρμογής που θα βοηθά τους χρήστες να αξιοποιούν στο έπακρο τα υλικά που διαθέτουν, μειώνοντας παράλληλα το άγχος και τον χρόνο που αφιερώνουν στην εύρεση συνταγών.

Συγκεντρώνοντας τις πολύτιμες εισφορές των κοντινών μου ατόμων, καθορίστηκαν τα βασικά σημεία τις εφαρμογής, όπως:

- Αναζήτηση συνταγών με βάση διαθέσιμα υλικά
- Δημιουργία συνταγών από την κοινότητα
- Προβολή λεπτομερών οδηγιών για τις συνταγές
- Δυνατότητα αξιολόγησης και σχολιασμού συνταγών
- Προσωποποιημένες προτάσεις συνταγών με βάση τις προτιμήσεις του χρήστη.

3.2 User stories

Τα user stories είναι σύντομες περιγραφές των λειτουργικών απαιτήσεων μιας εφαρμογής από την πλευρά του χρήστη. Συνήθως γράφονται από το πρίσμα του χρήστη, περιγράφοντας τις ανάγκες, τις επιθυμίες ή τα προβλήματα που αντιμετωπίζει κατά τη χρήση της εφαρμογής. Βοηθούν τους προγραμματιστές και να προσανατολιστούν στις ανάγκες των χρηστών κατά τη διάρκεια του αναπτυξιακού κύκλου.

Κατά τη διάρκεια αυτής της πτυχιακής θα τα χρησιμοποιήσουμε ως εργαλείο για να παρουσιάσουμε την διαδικασία ανάπτυξης λογισμικού, δηλαδή πως μια ιδέα μέσω απλών βημάτων μπορεί να γίνει ολόκληρη εφαρμογή.

3.2.1 User Story: Χρήστης μαθαίνει για την εφαρμογή

Ο Νίκος που αναζητά συνταγές για να μαγειρέψει νόστιμα γεύματα στο σπίτι του, ακούει για μια νέα πλατφόρμα εύρεσης συνταγών. Θέλοντας να δοκιμάσει την εμπειρία, επισκέπτεται την ιστοσελίδα και εντυπωσιάζεται από την καλαίσθητη και φιλική προς τον χρήστη διεπαφή. Με ελάχιστη καθοδήγηση, αφού εξερευνήσει τις λειτουργίες της πλατφόρμας και κατανοήσει τον τρόπο λειτουργίας της, κατευθύνεται προς τη σελίδα αναζήτησης συνταγών με σκοπό να ανακαλύψει τις ιδανικές επιλογές για τις δικές του γαστρονομικές προτιμήσεις.

Το ενδιαφέρον του Νίκου για τη νέα πλατφόρμα εύρεσης συνταγών οδήγησε στην ανάγκη για τη δημιουργία μιας σελίδας φίλτρων και αναζήτησης. Αυτή η σελίδα προσφέρει έναν εύκολο και γρήγορο τρόπο για τον χρήστη να εντοπίσει τις συνταγές που ταιριάζουν με τις προτιμήσεις του. Μέσω ενός φιλτραρίσματος βάσει συγκεκριμένων κριτηρίων, όπως τα υλικά ή η κατηγορία μια συνταγής, ο χρήστης μπορεί να περιορίσει τα αποτελέσματα και να εντοπίσει τις κατάλληλες επιλογές. Με αυτόν τον τρόπο, ο χρήστης βρίσκει εύκολα και γρήγορα τις ιδανικές συνταγές που ταιριάζουν με τις δικές του γαστρονομικές προτιμήσεις, βελτιώνοντας έτσι την εμπειρία του χρήστη στην πλατφόρμα.

Για την υλοποίηση αυτής της λειτουργικότητας ήταν απαραίτητη η δημιουργία του σχήματος που θα αντιπροσωπεύει την πληροφορία με την οποία θα αναπαριστάται αυτή η συνταγή. Η σημαντικότητα αυτού του σχήματος είναι η μεγαλύτερη μέσα σε όλη αυτή την εφαρμογή αφού όλες οι επόμενες λειτουργικότητες όπως θα δούμε παρακάτω, βασίζονται σε αυτό το τελικό αποτέλεσμα.

Το παρακάτω σχήμα περιγράφει τη δομή των δεδομένων για τις συνταγές στην εφαρμογή. Ας δούμε κάποιες από τις λεπτομέρειες:

- **id**: Το μοναδικό αναγνωριστικό της συνταγής
- **userId**: Το αναγνωριστικό του χρήστη που δημιούργησε τη συνταγή
- **title**: Ο τίτλος της συνταγής
- **description**: Η περιγραφή της συνταγής
- **categoryIds**: Οι κατηγορίες στις οποίες ανήκει η συνταγή
- **servings**: Ο αριθμός των μερίδων που παράγει η συνταγή
- **images**: Οι εικόνες που σχετίζονται με τη συνταγή
- **cookingTime**: Ο χρόνος που απαιτείται για το μαγείρεμα της συνταγής

- **totalCost**: Το συνολικό κόστος της συνταγής
- **difficulty**: Το επίπεδο δυσκολίας της συνταγής
- **tags**: Οι ετικέτες που χαρακτηρίζουν τη συνταγή
- **instructions**: Οι οδηγίες για την παρασκευή της συνταγής
- **ingredients**: Τα υλικά που απαιτούνται για τη συνταγή, συμπεριλαμβανομένων των ποσοτήτων
- **averageRating**: Η μέση βαθμολογία που έχει λάβει η συνταγή
- **costPerServing**: Το κόστος ανά μερίδα της συνταγής
- **createdAt**: Η ημερομηνία δημιουργίας της συνταγής
- **updatedAt**: Η ημερομηνία ενημέρωσης της συνταγής

```
model Recipe {
    id             String            @id @default(cuid())
    userId         String
    title          String
    description    String
    categoryIds    String[]
    servings       Int
    images         Image[]
    cookingTime    Int
    totalCost      Float
    difficulty     RecipeDifficulty
    tags           String[]
    instructions   String[]
    ingredients    RecipeIngredient[]
    averageRating  Float             @default(0)
    costPerServing Float             @map("totalCost / servings")
    createdAt      DateTime          @default(now())
    updatedAt      DateTime          @updatedAt
    user           User              @relation(fields: [userId], references: [id], onDelete: Cascade)
    Comment        Comment[]
    RecipeRating   RecipeRating[]
    RecipeReaction RecipeReaction[]
    Favorite       Favorite[]
}
```

Σχήμα 3.2.1. Το σχήμα της συνταγής

Αυτό το σχήμα προσφέρει μια ολοκληρωμένη περιγραφή των χαρακτηριστικών και των συσχετίσεων μεταξύ των δεδομένων που αφορούν τις συνταγές. Με αυτόν τον τρόπο, η εφαρμογή μπορεί να αποθηκεύει, να ανακτά και να εμφανίζει πληροφορίες για τις συνταγές με οργανωμένο και αποτελεσματικό τρόπο.

3.3 Ένα βήμα πίσω - Αρχιτεκτονική της εφαρμογής

Στο προηγούμενο κεφάλαιο, παρουσιάσαμε το σχήμα της συνταγής μας στο στάδιο τελικής υλοποίησης. Ωστόσο, για να κατανοήσουμε την εξέλιξη της εφαρμογής από λειτουργικής άποψης, είναι αναγκαίο να κάνουμε ένα βήμα πίσω. Στο υπόλοιπο κεφάλαιο, θα εξετάσουμε λεπτομερώς τα βήματα που ακολουθήσαμε για την υλοποίηση των λειτουργιών της εφαρμογής που οδήγησαν στην τελική εκδοχή που βλέπει ο χρήστης. Θα διερευνήσουμε τις απαραίτητες διαδικασίες και τα βήματα που ακολουθήσαμε προκειμένου να επιτύχουμε την πλήρη υλοποίηση των λειτουργιών που προσφέρει η εφαρμογή μας.

Στα επόμενα κεφάλαια, θα εξετάσουμε τον τρόπο με τον οποίο συνδέεται η ανάπτυξη του Back-end, το οποίο είναι υπεύθυνο για τη λειτουργικότητα, με το Front-end, το οποίο διαχειρίζεται τη διεπαφή της εφαρμογής. Θα εξετάσουμε τα κύρια στοιχεία που είναι απαραίτητα για την υλοποίηση μερικών βασικών λειτουργικοτήτων, προκειμένου να διασφαλίσουμε τη συνεργασία και τη συνοχή μεταξύ των δύο μερών της εφαρμογής.

3.4 Router για την διαχείριση αιτημάτων στον Server

Ο παρακάτω κώδικας δημιουργεί έναν δρομολογητή (router [29]) για το TRPC και καθορίζει ποιοι δρομολογητές θα χρησιμοποιηθούν για κάθε συγκεκριμένο τμήμα της εφαρμογής. Οι δρομολογητές είναι υπεύθυνοι για τη διαχείριση αιτημάτων και αποστολή απαντήσεων σε συγκεκριμένα μονοπάτια του API. Με αυτόν τον τρόπο, ο δρομολογητής "admin" αναλαμβάνει τα αιτήματα που σχετίζονται με τον διαχειριστή της εφαρμογής, ενώ ο δρομολογητής "recipes" χειρίζεται αιτήματα που αφορούν συνταγές. Αυτή η διαμόρφωση βοηθά στη διαχείριση του κώδικα και την οργάνωση των λειτουργιών της εφαρμογής.

```
1  export const appRouter = createTRPCRouter({
2      admin: adminRouter,
3      upload: uploadRouter,
4      comments: commentsRouter,
5      favoriteRecipes: favoriteRecipesRouter,
6      recipes: recipeRouter,
7      recipeRatings: recipeRatingsRouter,
8  })
```

Σχήμα 3.4.1 Βασικός δρομολογητής αιτημάτων

3.5 Διαχειριστής

Στο παρακάτω υποκεφάλαιο, θα εξετάσουμε τον ρόλο και τις λειτουργίες του διαχειριστή στο πλαίσιο της εφαρμογής. Ο διαχειριστής αποτελεί ένα σημαντικό μέρος της διαδικασίας ανάπτυξης και λειτουργίας του συστήματος, καθώς έχει την ευθύνη για τη διαχείριση των δεδομένων, την επικοινωνία με τους χρήστες και την εξασφάλιση της ασφάλειας της εφαρμογής. Θα εξετάσουμε ποιες είναι οι βασικές λειτουργίες που αναμένεται να εκτελέσει ο διαχειριστής, καθώς και πώς αυτές επηρεάζουν τη λειτουργία και την εμπειρία των χρηστών. Επιπλέον, θα δούμε πώς ο διαχειριστής συμβάλλει στην ανάπτυξη και τη βελτίωση της εφαρμογής, προσφέροντας την απαιτούμενη υποστήριξη και διαχειρίζοντας τις ανάγκες των χρηστών και του συστήματος. Τέλος, θα εξετάσουμε τις τεχνικές λεπτομέρειες και τις προτεραιότητες που αφορούν την ανάπτυξη και τη λειτουργία των διαχειριστικών λειτουργιών του διαχειριστή.

3.5.1 Δημιουργία κατηγορίας

Πριν αρχίσουμε να δημιουργούμε συνταγές, είναι απαραίτητο να ορίσουμε τις κατηγορίες στις οποίες ανήκουν οι συνταγές. Αυτό το καθήκον είναι αποκλειστικά υπεύθυνο το διαχειριστής της πλατφόρμας, καθώς έχει την εξουσία να ορίζει και να επεξεργάζεται τις κατηγορίες ανάλογα με τις ανάγκες και το περιεχόμενο της εφαρμογής. Ο διαχειριστής είναι υπεύθυνος για τη δημιουργία, την επεξεργασία και τη διαχείριση των κατηγοριών, καθώς και για την ανάθεση κατάλληλων ετικετών ή ταξινομικών συστημάτων που εξυπηρετούν την καλύτερη οργάνωση και αναζήτηση των συνταγών από τους χρήστες. Η οριοθέτηση των κατηγοριών είναι κρίσιμη για την ομαλή λειτουργία της πλατφόρμας και την καλή εμπειρία των χρηστών, ενώ παράλληλα συμβάλλει στη διατήρηση της τάξης και της διαφάνειας στο σύστημα. Παρακάτω βλέπουμε τον τρόπο με τον οποίο αναπτύσσουμε αυτή τη λειτουργικότητα στο back-end μας.

```
export const adminRouter = createTRPCRouter({
    createCategory: publicProcedure
        .input(CreateCategoryModel)
        .mutation(async ({ ctx, input }) ⇒ {
            if (getAuthState(ctx.session).isAdmin === false) {
                throw new Error("User unauthorized to create category")
            }
            const category = await ctx.db.category.create({
                data: input,
            })
            return category
        })
})
```

Σχήμα 3.5.1 Δρομολογητής δημιουργίας συνταγής

Αυτό το κομμάτι κώδικα δημιουργεί έναν δρομολογητή με χρήση της βιβλιοθήκης TRPC για το Back-end της εφαρμογής. Στο προηγούμενο υποκεφάλαιο είδαμε το κεντρικό σημείο που μαζεύει συγκεντρωτικά όλους τους δρομολογητές της εφαρμογής. Συγκεκριμένα, αυτός ο κώδικας δημιουργεί έναν δρομολογητή για το διαχειριστή (admin), ο οποίος επιτρέπει τη δημιουργία νέας κατηγορίας στην εφαρμογή.

Ο δρομολογητής αυτός παρέχει ένα endpoint με το όνομα "createCategory", το οποίο είναι διαθέσιμο μόνο ως μετάλλαξη (mutation [30]). Κάθε φορά που καλείται αυτό το endpoint με μια αίτηση, ελέγχεται πρώτα αν ο χρήστης που κάνει το αίτημα έχει δικαίωμα διαχειριστή (isAdmin). Αν ο χρήστης δεν έχει δικαίωμα διαχειριστή, εκτίθεται ένα σφάλμα "User unauthorized to create category" ή αλλιώς “Ο χρήστης δεν έχει δικαιοδοσία για δημιουργία κατηγορίας”. Στη συνέχεια, εάν ο έλεγχος είναι επιτυχής, η λειτουργία δημιουργεί μια νέα κατηγορία στη βάση δεδομένων χρησιμοποιώντας τα δεδομένα που παρέχονται στο αίτημα.

Το "CreateCategoryModel" αναφέρεται στη μοντέλο εισόδου για τη δημιουργία κατηγορίας, ενώ ο "publicProcedure" προσδιορίζει ότι η συνάρτηση αυτή είναι προσπελάσιμη από το εξωτερικό κόσμο.

Η επιτυχής εκτέλεση της λειτουργίας επιστρέφει τα δεδομένα της νέας κατηγορίας που δημιουργήθηκε.

```
export const CreateCategoryModel = z.object({
    parentRecipeCategoryId: z.string().or(z.null()),
    priority: z.number(),
    slug: z.string().min(1),
    name: z.string().min(1),
})
```

Σχήμα 3.5.2 Μοντέλο δημιουργίας νέα κατηγορίας

Το παραπάνω τμήμα κώδικα ορίζει το μοντέλο εισόδου για τη δημιουργία νέας κατηγορίας στην εφαρμογή. Το μοντέλο ορίζεται χρησιμοποιώντας τη βιβλιοθήκη zod, η οποία όπως είπαμε στο κεφάλαιο 2.5.6 παρέχει μια δυνατή τύπωση δεδομένων και επικυρώνει την ακεραιότητα των δεδομένων.

Το μοντέλο CreateCategoryModel περιλαμβάνει τέσσερα πεδία:

- **name**: Το όνομα της κατηγορίας.
- **parentRecipeCategoryId**: Το αναγνωριστικό της γονικής κατηγορίας στην οποία ανήκει η νέα κατηγορία. Αυτό το πεδίο μπορεί είτε να είναι ένα string είτε να είναι null, εάν η κατηγορία δεν έχει γονική κατηγορία.
- **priority**: Η προτεραιότητας της κατηγορίας.
- **slug**: Το μοναδικό αναγνωριστικό της κατηγορίας, το οποίο χρησιμοποιείται στο URL.

Κάθε πεδίο έχει καθορισμένες προδιαγραφές σχετικά με τον τύπο του δεδομένου (π.χ., string, number) και τους περιορισμούς (π.χ., ελάχιστο μήκος). Η χρήση του zod.object επιτρέπει την επιβολή των περιορισμών αυτών στη δομή του αντικειμένου. Αν κατά το αίτημα τα πεδία που στέλνει ο χρήστης από τον client δεν είναι έγκυρα, αυτή η κλήση θα επιστρέψει ένα error.

```
model Category {
    id                   String   @id @default(cuid())
    name                 String
    parentRecipeCategoryId String?
    priority             Int      @default(0)
    slug                 String
    createdAt            DateTime @default(now())
    updatedAt            DateTime @updatedAt

    @@unique([name])
}
```

Σχήμα 3.5.3 Σχήμα της κατηγορίας στη βάση δεδομένων

Στο παραπάνω σχήμα, παρατηρούμε τον τρόπο που η κατηγορία ορίζεται μέσω του Prisma στη βάση δεδομένων. Καθώς προχωράμε, αντιλαμβανόμαστε την αξία που προσφέρουν αυτές οι τεχνολογίες, καθώς επιβεβαιώνουν τη συνέπεια σε όλα τα επίπεδα της εφαρμογής. Από τη διαχείριση της βάσης δεδομένων μέχρι τον έλεγχο της εγκυρότητας των δεδομένων στο back-end, η ομοιομορφία αυτή συμβάλλει στη δημιουργία μιας απρόσκοπτης εμπειρίας χρήστη. Σε επόμενο υποκεφάλαιο, θα εξετάσουμε πώς αυτά τα δεδομένα αξιοποιούνται στο front-end, δημιουργώντας ένα ενιαίο και συνεκτικό περιβάλλον για τους χρήστες.

3.5.2 Άλλες λειτουργικότητες κατηγοριών

Πέρα από το να δημιουργήσουμε μια κατηγορία σαν διαχειριστής, θα πρέπει να μπορούμε να ανακτήσουμε αυτή την πληροφορία όπως και να την επεξεργαστούμε. Στο παρακάτω σχήμα φαίνεται το κομμάτι κώδικα που έχει αναλάβει την λήψη αυτής της πληροφορίας.

```
getCategories: publicProcedure.query(async ({ ctx, input }) => {
    const categories = await ctx.db.category.findMany({
        include: {
            images: true
        },
        orderBy: {
            name: "asc",
        },
    })
    return categories
})
```

Σχήμα 3.5.4 Λήψη κατηγοριών

Η μέθοδος getCategories ανακτά όλες τις κατηγορίες από τη βάση δεδομένων και τις επιστρέφει, χωρίς να προκαλεί κάποια αλλαγή στα δεδομένα. Το query χρησιμοποιείται εδώ γιατί απλά ανακτά δεδομένα χωρίς να τα τροποποιεί, ενώ το mutation που είδαμε παραπάνω χρησιμοποιείται όταν θέλουμε να προκαλέσουμε κάποια αλλαγή στα δεδομένα, όπως προσθήκη, τροποποίηση ή διαγραφή.

Βλέπουμε ότι η μέθοδος που χρησιμοποιείται για να φέρουμε τα δεδομένα από στην οντότητα **category** είναι η **findMany**. Η μέθοδος αυτή θα μετατραπεί στην αντίστοιχη **select ... from ...** , που έχει η SQL, απο το Prisma. Επίσης οι παράμετροι που δέχεται η findMany θα μετατραπούν στα αντίστοιχα ερωτήματα που χρειάζεται η SQL που δουλεύει πίσω από το Prisma για να μας φέρει τα ακριβώς τα δεδομένα που ζητήσαμε. Για παράδειγμα η επιλογή include:{ images: true } θα μας φέρει μέσα στο αποτέλεσμα και όλες τις σχετιζόμενες εικόνες με την κατηγορία που έχουμε αποθηκευμένες στη βάση μας.

Για να ενημερώσουμε τα δεδομένα μια κατηγορίας ως διαχειριστής θα πρέπει να καλέσουμε την μέθοδο updateCategory. Η μέθοδος αυτή λειτουργεί σχεδόν με τον ίδιο τρόπου που λειτουργεί και η μέθοδος createCategory που είδαμε στο κεφάλαιο 3.5.2. Και οι δύο μέθοδοι δέχονται σαν είσοδο ένα μοντέλο επικύρωσης δεδομένων με την μόνη διαφορά ότι αυτή τη φορά, το πεδίο id είναι απαραίτητο. Ο λόγος που το πεδίο id είναι απαραίτητο στην μέθοδο update είναι φυσικά επειδή δεν μπορούμε να ενημερώσουμε εγγραφή που δεν υπάρχει στη βάση δεδομένων μας.

```
updateCategory: publicProcedure
    .input(UpdateCategoryModel)
    .mutation(async ({ ctx, input }) => {
        if (getAuthState(ctx.session).isAdmin === false) {
            throw new Error("User unauthorized to create category")
        }

        const category = await ctx.db.category.update({
            where: {
                id: input.id
            },
            data: input
        })
        return category
    })
```

Σχήμα 3.5.5 Μέθοδος ενημέρωσης δεδομένων κατηγορίας

Τέλος για να κλείσουμε αυτό το υποκεφάλαιο, δεν μπορούμε να παραλείψουμε την τελευταία λειτουργία CRUD, την Delete. Η λειτουργικότητα αυτή είναι σχεδόν ίδια με αυτή της update. Αυτή τη φορά το μόνο απαραίτητο στοιχείο εισόδου είναι το id της κατηγορίας με το οποίο θα την αναγνωρίσουμε στη βάση δεδομένων μας.

```
deleteCategory: publicProcedure
    .input(z.object({ id: z.string() }))
    .mutation(async ({ ctx, input }) => {
        if (getAuthState(ctx.session).isAdmin === false) {
            throw new Error("User unauthorized to create category")
        }

        const category = await ctx.db.category.delete({
            where: {
                id: input.id
            }
        })
        return category
    })
```

Σχήμα 3.5.6 Διαγραφή κατηγορίας

3.5.3 Μονάδες μέτρησης συστατικών

Πριν ετοιμαστούμε να δημιουργήσουμε μια συνταγή θα πρέπει να έχουμε κάποια συστατικά τα οποία θα περιέχει η κάθε συνταγή. Τα συστατικά όμως σε μια συνταγή μετριούνται πάντα σε συγκεκριμένες

μονάδες μέτρησης. Οι μονάδες μέτρησης στα υλικά που χρησιμοποιούνται στην εφαρμογή έχουν κρίσιμη σημασία για την ακριβή περιγραφή των συστατικών και των ποσοτήτων που απαιτούνται σε μια συνταγή. Η χρήση μονάδων μέτρησης επιτρέπει στους χρήστες να κατανοούν πιο εύκολα τις ποσότητες που χρειάζονται για την παρασκευή ενός συγκεκριμένου γεύματος ή ενός συστατικού.

Θα πρέπει λοιπόν να δημιουργήσουμε τον μηχανισμό ο οποίος διαχειρίζεται τις μονάδες μέτρησης συστατικών πριν αρχίσουμε να δημιουργούμε συστατικά. Οι μέθοδοι CRUD για την διαχείριση των μονάδων μέτρησης είναι παρόμοιοι με αυτές της διαχείρισης κατηγοριών, οπότε δεν θα αφιερώσουμε χρόνο στην λεπτομερή περιγραφή της υλοποίησής τους. Πάντα οι μέθοδοί μας περιμένουν ως είσοδο ένα επικυρωμένο σχήμα δεδομένων και ανάλογα με αυτό κάνουν μερικές κινήσεις πάνω στην βάση δεδομένων μας.

Ας εξετάσουμε λοιπόν τις αρχικές μονάδες μέτρησης που θα περιέχει η εφαρμογή και ας δούμε έναν αποτελεσματικό τρόπο να έχουμε αρχικά δεδομένα κάθε φορά που φτιάχνουμε έναν νέο server στον οποίο θα υπάρχει η εφαρμογή μας.

```
export const MEASUREMENT_UNITS_DATA = [
    { name: "Κουταλάκι του γλυκού", slug: "κ. γλυκού" },
    { name: "Κουταλιά της σούπας", slug: "κ. σούπας" },
    { name: "Κιλό", slug: "κιλ." },
    { name: "Γραμμάριο", slug: "γρ." },
    { name: "Λίτρο", slug: "λτ." },
    { name: "Μιλιλίτρα", slug: "ml." },
    { name: "Σταγόνα", slug: "σταγόνα" },
    { name: "Κιβώτιο", slug: "κιβώτιο" },
    { name: "Τεμάχιο", slug: "τεμάχιο" },
]
```

Σχήμα 3.5.6 Μονάδες μέτρησης υλικών

Οι seeders [31] στο Prisma είναι εργαλεία που χρησιμοποιούνται για την αρχική πλήρωση της βάσης δεδομένων με δεδομένα. Αυτά τα δεδομένα μπορούν να είναι σταθερά ή δεδομένα που χρησιμοποιούνται για τη δοκιμή της εφαρμογής κατά τη διάρκεια της ανάπτυξης. Οι βασικές χρήσεις των seeders στο Prisma περιλαμβάνουν:

- **Αρχική Πλήρωση Δεδομένων**: Κατά τη δημιουργία μιας νέας βάσης δεδομένων ή κατά τη μετάβαση σε ένα νέο περιβάλλον ανάπτυξης, οι seeders μπορούν να χρησιμοποιηθούν για να γεμίσουν αυτή τη βάση με αρχικά δεδομένα.

- **Δοκιμαστικά Δεδομένα**: Οι seeders μπορούν να δημιουργήσουν δεδομένα που χρησιμοποιούνται για δοκιμή λειτουργικότητας και ελέγχους κατά τη διάρκεια της ανάπτυξης της εφαρμογής.

Με τη χρήση των seeders, μπορούμε να διαχειριστούμε αποτελεσματικά την αρχική πλήρωση και τη διαχείριση των δεδομένων της εφαρμογής μας, εξασφαλίζοντας την ομαλή λειτουργία της. Το παρακάτω τμήμα κώδικα δημιουργεί έναν πίνακα από υποσχέσεις (promises [32]) με βάση έναν προκαθορισμένο πίνακα δεδομένων (MEASUREMENT_UNITS_DATA). Για κάθε στοιχείο στον προκαθορισμένο πίνακα, η μέθοδος upsert του Prisma καλείται για να εισαχθούν ή να ενημερωθούν τα δεδομένα στη βάση δεδομένων.

Αν το όνομα της μονάδας μέτρησης ήδη υπάρχει στη βάση δεδομένων, τότε τα δεδομένα ενημερώνονται με βάση τις πληροφορίες από τον προκαθορισμένο πίνακα. Διαφορετικά, δημιουργείται ένα νέο αντικείμενο μονάδας μέτρησης με τις πληροφορίες από τον προκαθορισμένο πίνακα.

Αυτός ο τρόπος εξασφαλίζει ότι τα δεδομένα των μονάδων μέτρησης είναι συνεπή σε όλο το σύστημα και είναι ενημερωμένα με τις τρέχουσες πληροφορίες που περιέχονται στον προκαθορισμένο πίνακα MEASUREMENT_UNITS_DATA.

```
const measurementUnitsPromises = MEASUREMENT_UNITS_DATA.map(measurementUnit ⇒ {
        return prisma.measurementUnit.upsert({
            create: {
                name: measurementUnit.name,
                slug: measurementUnit.slug,
            },
            where: {
                name: measurementUnit.name,
            },
            update: {
                name: measurementUnit.name,
                slug: measurementUnit.slug,
            },
        });
    })
```

Σχήμα 3.5.7 Seeder μονάδων μέτρησης

3.5.4 Συστατικά συνταγών

Αφού έχουμε ολοκληρώσει την λειτουργικότητα για την διαχείριση μονάδων μέτρησης, σειρά έχει η διαχείριση των συστατικών. Η διαχείριση των συστατικών μας επιτρέπει να καταγράφουμε τα στοιχεία κάθε συστατικού, όπως το όνομά του, τη μονάδα μέτρησής του και άλλες σημαντικές πληροφορίες. Κάθε συστατικό αντιστοιχεί σε μια μοναδική εγγραφή στη βάση δεδομένων μας, επιτρέποντάς μας να τα ανακαλούμε εύκολα κατά τη δημιουργία νέων συνταγών με τις σωστές πληροφορίες.

Φυσικά η λειτουργικότητα σε επίπεδο κώδικα δεν διαφέρει ιδιαίτερα με τις προηγούμενες υλοποιήσεις μας. Θα αναφέρουμε λοιπόν εδώ ότι για να είναι σχεσιακά συνδεδεμένες οι συνταγές με τα υλικά τους θα πρέπει να δημιουργηθεί ένα νέος πίνακας ο οποίος θα αναπαριστά αυτή τη σχέση.

```
model RecipeIngredient {
    id               String          @id @default(cuid())
    measurementQty   Int
    recipeId         String
    ingredientId     String
    measurementUnitId String
    Recipe           Recipe          @relation(fields: [recipeId], references: [id], onDelete: Cascade)
    Ingredient       Ingredient      @relation(fields: [ingredientId], references: [id], onDelete: Cascade)
    MeasurementUnit  MeasurementUnit @relation(fields: [measurementUnitId], references: [id], onDelete: Cascade)
}
```

Σχήμα 3.5.8 Πίνακας σχέσεων συνταγών, υλικών και μονάδας μέτρησης

Το παραπάνω σχήμα απεικονίζει τον τρόπο με τον οποίο η σχέση μεταξύ συνταγής και συστατικού διαχειρίζεται στη βάση δεδομένων. Κάθε εγγραφή στον πίνακα RecipeIngredient αντιστοιχεί σε ένα συγκεκριμένο συστατικό που χρησιμοποιείται σε μια συνταγή, καθώς και στις αντίστοιχες μονάδες μέτρησης για αυτό το συστατικό.

Πιο συγκεκριμένα, το σχήμα περιλαμβάνει τα εξής πεδία:

- **id**: Μοναδικό αναγνωριστικό για κάθε εγγραφή στον πίνακα RecipeIngredient.
- **measurementQty**: Η ποσότητα του συστατικού που χρησιμοποιείται στη συνταγή, σε μονάδες μέτρησης.
- **recipeId**: Το αναγνωριστικό της συνταγής που χρησιμοποιεί αυτό το συστατικό.
- **ingredientId**: Το αναγνωριστικό του συστατικού που προστίθεται στη συνταγή.
- **measurementUnitId**: Το αναγνωριστικό της μονάδας μέτρησης που χρησιμοποιείται για αυτό το συστατικό.

Επιπλέον, χρησιμοποιείται το @relation για να οριστεί η σχέση μεταξύ του RecipeIngredient και των συναφών πινάκων Recipe, Ingredient και MeasurementUnit. Αυτό επιτρέπει τη διασύνδεση των δεδομένων μεταξύ των πινάκων και την αποτελεσματική λήψη των πληροφοριών σχετικά με τις συνταγές, τα συστατικά και τις μονάδες μέτρησης που χρησιμοποιούνται σε αυτές.

Το onDelete: Cascade είναι μια δήλωση που ορίζει τον τρόπο με τον οποίο η διαγραφή ενός στοιχείου στον πίνακα που έχει αυτή τη σχέση επηρεάζει τα σχετικά στοιχεία σε άλλους πίνακες. Στη συγκεκριμένη περίπτωση, το onDelete: Cascade σημαίνει ότι όταν διαγράφεται ένα στοιχείο στον πίνακα Recipe, Ingredient ή MeasurementUnit, τότε οποιαδήποτε αντίστοιχη εγγραφή στον πίνακα RecipeIngredient που ανήκει σε αυτή τη συνταγή θα διαγραφεί αυτόματα.

Αυτό εξασφαλίζει τη συνέπεια της βάσης δεδομένων, καθώς δεν θα υπάρχουν παραμένουσες εγγραφές στον πίνακα RecipeIngredient που αναφέρονται σε συνταγές, συστατικά ή μονάδες μέτρησης που έχουν διαγραφεί. Συνεπώς, αποφεύγονται οι ασυνέπειες στα δεδομένα και διασφαλίζεται η ομαλή λειτουργία της εφαρμογής.

3.5.5 Συνταγές

Τώρα που έχουμε προετοιμάσει το έδαφος για να δημιουργήσουμε τις συνταγές, ήρθε η ώρα να δούμε πως συνδυάζεται όλη αυτή η πληροφορία που έχουμε συγκεντρώσει ως τώρα. Ας δούμε λοιπόν το σχήμα 3.5.2 σε λίγο μεγαλύτερο βάθος.

id: Αυτό το πεδίο αναπαριστά το μοναδικό αναγνωριστικό της συνταγής. Το id δημιουργείται αυτόματα και χρησιμοποιείται για να αναφερθεί σε μια συγκεκριμένη συνταγή και να την αναγνωρίσει.

userId: Το πεδίο userId αναπαριστά το μοναδικό αναγνωριστικό του χρήστη που δημιούργησε τη συνταγή.

title: Εδώ καταχωρείται ο τίτλος της συνταγής, που αναγνωρίζει το περιεχόμενό της. Στην βάση μας επιτρέπονται τίτλοι μέχρι 256 χαρακτήρες σε μήκος.

description: Η περιγραφή της συνταγής παρέχει επιπλέον πληροφορίες και λεπτομέρειες σχετικά με το περιεχόμενό της. Το πεδίο description δεν πρόκειται απλά για ένα αλφαριθμητικό πεδίο. Θα δούμε σε επόμενο κεφάλαιο ότι μέσω ενός Rich Text Editor δίνουμε την δυνατότητα στον χρήστη να φτιάξει πολύ όμορφες περιγραφές οι οποίες θυμίζουν ολόκληρη ιστοσελίδα.

categoryIds: Αυτό το πεδίο περιέχει τα αναγνωριστικά των κατηγοριών που ανήκει η συνταγή.

servings: Το πλήθος των μερίδων που παράγει η συνταγή, που βοηθά στον υπολογισμό της ποσότητας των υλικών που απαιτούνται.

images: Αυτό το πεδίο περιλαμβάνει τις εικόνες που σχετίζονται με τη συνταγή, όπως φωτογραφίες του ετοιμάζομενου φαγητού ή των υλικών. Θα δούμε σε επόμενο κεφάλαιο τον τρόπο με τον οποίο διαχειριζόμαστε το ανέβασμα εικόνων στον server μας, καθώς η αποστολή αρχείων στον server διαφέρει αρκετά από την αποστολή απλών δεδομένων.

cookingTime: Ο χρόνος προετοιμασίας και μαγειρέματος της συνταγής, που δίνει μια εκτίμηση για τον απαιτούμενο χρόνο για την παρασκευή της.

totalCost: Το συνολικό κόστος των υλικών που απαιτούνται για την παρασκευή της συνταγής.

difficulty: Το επίπεδο δυσκολίας της συνταγής, που βοηθά τους χρήστες να επιλέξουν συνταγές που ταιριάζουν στις ικανότητές τους. Η εφαρμογή μας έχει 5 επίπεδα δυσκολίας τα οποία είναι Πολύ Εύκολη, Εύκολη, Μέτρια, Δύσκολη και πολύ Δύσκολη.

tags: Οι ετικέτες που χαρακτηρίζουν τη συνταγή, που βοηθούν στην ταξινόμηση και αναζήτηση συνταγών βάσει συγκεκριμένων θεμάτων ή χαρακτηριστικών. Μπορεί να γίνει αναζήτηση συνταγών μέσω του μηχανισμού αναζήτησης με βάση τα tags.

instructions: Οι λεπτομερείς οδηγίες για την παρασκευή της συνταγής, που περιέχουν τα βήματα και τις απαιτήσεις για την επιτυχή παρασκευή του φαγητού.

ingredients: Τα συστατικά που απαιτούνται για την παρασκευή της συνταγής, που περιλαμβάνουν τα υλικά και τις ποσότητες που απαιτούνται.

averageRating: Η μέση βαθμολογία που έχει λάβει η συνταγή από τους χρήστες, που δείχνει τη δημοφιλία και την ποιότητά της. Κάθε φορά που προστίθεται μια νέα κριτική σε μία συνταγή, αυτό το πεδίο θα πρέπει να υπολογίζεται ξανά.

costPerServing: Το κόστος ανά μερίδα για τη συνταγή, που βοηθά τους χρήστες να εκτιμήσουν τον οικονομικό προϋπολογισμό της συνταγής.

createdAt: Η ημερομηνία δημιουργίας της συνταγής, που καταγράφει το χρονικό σημείο κατά το οποίο δημιουργήθηκε η συνταγή.

updatedAt: Η ημερομηνία τελευταίας ενημέρωσης της συνταγής, που καταγράφει το χρονικό σημείο κάθε φορά που ενημερώθηκε η συνταγή.

3.5.6 Ανέβασμα εικόνων

Το ανέβασμα εικόνων αποτελεί ένα από τα βασικά χαρακτηριστικά που εμπλουτίζουν την εμπειρία των χρηστών σε μια εφαρμογή. Αν και μπορεί να φαίνεται απλό στην επιφάνεια, η διαδικασία ανεβάσματος εικόνων περιλαμβάνει πολλά σημαντικά στάδια που απαιτούν προσεκτική σχεδίαση και υλοποίηση.

Κατά τη διάρκεια αυτής της διαδικασίας, οι χρήστες έχουν τη δυνατότητα να ανεβάσουν εικόνες από διάφορες πηγές, όπως οι συσκευές τους ή το διαδίκτυο, και να τις ενσωματώσουν στο περιεχόμενο της εφαρμογής. Αυτό μπορεί να συμβεί σε πολλά σημεία της εφαρμογής, όπως σε προφίλ χρηστών, αναρτήσεις συνταγών, προϊόντα ή άλλα στοιχεία που απαιτούν οπτική αναπαράσταση. Σε γενικές γραμμές το ανέβασμα εικόνων είναι σημαντική λειτουργικότητα σε μια εφαρμογή για πολλούς λόγους:

- **Βελτιωμένη χρηστικότητα**: Οι εικόνες προσθέτουν οπτικό ενδιαφέρον και καθιστούν την εφαρμογή πιο ελκυστική για τους χρήστες.
- **Αύξηση της αλληλεπίδρασης**: Οι χρήστες μπορούν να ανεβάζουν δικές τους εικόνες, προσδίδοντας έναν βαθμό εξατομίκευσης και συμμετοχής στην εφαρμογή.
- **Περιεκτική παρουσίαση πληροφοριών**: Οι εικόνες μπορούν να παρέχουν πληροφορίες και συναισθήματα που δεν μπορεί να περιέχει μόνο κείμενο, βοηθώντας τους χρήστες να κατανοήσουν καλύτερα το περιεχόμενο.
- **Ανάπτυξη κοινότητας**: Οι χρήστες μπορούν να μοιραστούν τις δικές τους εικόνες, να αλληλεπιδρούν και να συνδέονται με άλλους χρήστες μέσω του κοινού ενδιαφέροντος.
- **Επικοινωνία των πληροφοριών**: Οι εικόνες μπορούν να μεταδώσουν πληροφορίες γρήγορα και εύκολα, κάνοντας την επικοινωνία πιο αποτελεσματική και κατανοητή.

Στο συγκεκριμένο υποκεφάλαιο θα μελετήσουμε την υλοποίηση ενός μηχανισμού ανεβάσματος εικόνων σε έναν τοπικό server. Ας δούμε όμως τα απαραίτητα βήματα για την υλοποίηση αυτού του μηχανισμού:

1. **Δημιουργία φόρμας ανεβάσματος**: Δημιουργία μιας φόρμας HTML στο πρότυπο της ιστοσελίδας για το ανέβασμα εικόνων. Η φόρμα αυτή περιλαμβάνει ένα πεδίο επιλογής αρχείου (file input) όπου ο χρήστης μπορεί να επιλέξει την εικόνα που επιθυμεί να ανεβάσει.
2. **Διαχείριση ανεβασμένου αρχείου**: Με την υποβολή της φόρμας, το επιλεγμένο αρχείο εικόνας αποστέλλεται στον server. Στον erver, ένα σύστημα διαχείρισης αρχείων λαμβάνει

το αρχείο, ελέγχει τον τύπο και το μέγεθος του, και στη συνέχεια το αποθηκεύει σε κατάλληλη τοποθεσία.

3. **Αποθήκευση εικόνας στον server**: Το σύστημα διαχείρισης αρχείων αποθηκεύει το αρχείο εικόνας στον εξυπηρετητή, συνήθως σε έναν φάκελο που είναι προσβάσιμος από την εφαρμογή.
4. **Αποθήκευση στη Βάση Δεδομένων**: Αν είναι απαραίτητο, οι πληροφορίες σχετικά με την ανεβασμένη εικόνα (όπως το όνομα του αρχείου, ο τύπος, το μέγεθος κλπ.) μπορούν να αποθηκευτούν και σε μια βάση δεδομένων, προκειμένου να διατηρηθεί ιστορικό και να γίνει εύκολη η ανάκτησή τους. Στην περίπτωσή μας θα αποθηκεύσουμε τις πληροφορίες και στην βάση δεδομένων μας ώστε να γίνει η συσχέτιση με την εκάστοτε συνταγή και να μπορούμε να διαχειριστούμε μια εικόνα στο μέλλον.

Το παρακάτω τμήμα κώδικα δημιουργεί δυναμικά μια φόρμα που επιτρέπει στους χρήστες να επιλέξουν εικόνες από τη συσκευή τους. Πρώτα, ο χρήστης επιλέγει εικόνες μέσω του στοιχείου εισαγωγής αρχείων <input type="file">, οι οποίες αποθηκεύονται στη μεταβλητή form.values.selectedImages. Στη συνέχεια, ο κώδικας επεξεργάζεται κάθε μια από αυτές τις εικόνες χρησιμοποιώντας τη μέθοδο map για να δημιουργήσει ένα αντικείμενο FormData, το οποίο περιέχει την εικόνα ως μέρος του. Στη συνέχεια, κάθε αντικείμενο FormData αποστέλλεται στον διακομιστή με μια αίτηση POST χρησιμοποιώντας την axios.post. Το αποτέλεσμα κάθε αιτήματος ανέβασματος εικόνας επιστρέφεται ως υπόσχεση, η οποία περιλαμβάνει το αποτέλεσμα της αντίστοιχης λειτουργίας.

```
1  const uploadImagePromises = form.values.selectedImages.map((img) => {
2      const formData = new FormData()
3      formData.append("image", img)
4      return axios.post("/api/image", formData)
5  })
```

Σχήμα 3.5.9 Δημιουργία φόρμας ανεβάσματος εικόνας

Αφού έχουμε ετοιμάσεις τις φόρμες με τις εικόνες για να στείλουμε στον server, είμαστε έτοιμοι να τις στείλουμε στον server. Το παρακάτω κομμάτι κώδικα κώδικα αναμένει την ολοκλήρωση όλων των υποσχόμενων αιτημάτων ανεβάσματος εικόνας που προηγουμένως δημιουργήθηκαν. Χρησιμοποιεί τη μέθοδο Promise.all για να περιμένει την εκτέλεση όλων των αιτημάτων ταυτόχρονα. Μόλις όλα τα αιτήματα ολοκληρωθούν με επιτυχία, η μεταβλητή imageUploadResponse θα περιέχει έναν πίνακα από αντικείμενα. Κάθε αντικείμενο θα περιλαμβάνει τα δεδομένα της απάντησης για κάθε αίτημα ανεβάσματος εικόνας. Τα δεδομένα αυτά περιέχουν το όνομα του αρχείου και τη διαδρομή του αρχείου στον διακομιστή. Η μεταβλητή imageUploadResponse έχει τύπο δεδομένων που έχει καθοριστεί ως πίνακας αντικειμένων, με κάθε αντικείμενο να περιέχει τη δομή δεδομένων που περιγράφεται στον τύπο { data: { data: { name: string, path: string } } }.

Ας δούμε τώρα την διαχείριση αυτών των αιτημάτων από την πλευρά του server. Το παρακάτω τμήμα κώδικα αν και φαίνεται αρκετά σύνθετο στην πραγματικότητα είναι απλό. Δημιουργεί μια συνάρτηση με το όνομα readFile, η οποία δέχεται ένα αίτημα Next.js API (req: NextApiRequest) ως παράμετρο

και επιστρέφει μια υπόσχεση (Promise) με ένα αντικείμενο που περιέχει τα πεδία και τα αρχεία που έχουν ανέβει. Τα απαραίτητα βήματα για αυτή την συνάρτηση είναι τα εξής:

- Ορισμός των επιλογών για τη βιβλιοθήκη formidable. Οι επιλογές περιλαμβάνουν τον καθορισμό του φακέλου όπου θα αποθηκευτούν οι εικόνες, τον τρόπο δημιουργίας του ονόματος του αρχείου και το μέγιστο μέγεθος αρχείου που επιτρέπεται να ανέβει.
- Δημιουργία ενός αντικειμένου τύπου formidable με τις παραπάνω επιλογές.
- Χρήση της μεθόδου parse της βιβλιοθήκης formidable για να αναλύσει το αίτημα και να πάρει τα πεδία και τα αρχεία που έχουν ανέβει.
- Επιστροφή μιας υπόσχεσης με τα πεδία και τα αρχεία που προκύπτουν από τη μέθοδο parse.

```
const readFile = (
    req: NextApiRequest,
): Promise<{ fields: formidable.Fields; files: formidable.Files }> ⇒ {
    const options: formidable.Options = {}

    options.uploadDir = path.join(process.cwd(), "/public/images")
    options.maxFileSize = 1024 * 1024
    options.filename = (name, ext, path) ⇒ {
        return Date.now().toString() + "_" + path.originalFilename
    }

    const form = formidable(options)

    return new Promise((resolve, reject) ⇒ {
        form.parse(req, (err, fields, files) ⇒ {
            if (err) reject(err)
            resolve({ fields, files })
        })
    })
}
```

Σχήμα 3.5.10 Μέθοδος εγγραφής αρχείων στον αποθηκευτικό μας χωρο

Αυτή η συνάρτηση είναι χρήσιμη για τον χειρισμό αιτημάτων που περιλαμβάνουν ανέβασμα αρχείων. Χρησιμοποιεί τη βιβλιοθήκη formidable για την ανάλυση των αιτημάτων multipart/form-data, όπως αυτά που δημιουργούνται κατά την υποβολή φορμών με αρχεία. Η συνάρτηση επιστρέφει ένα αντικείμενο που περιέχει τα πεδία και τα αρχεία που έχουν ανέβει, τα οποία μπορούν να χρησιμοποιηθούν για περαιτέρω επεξεργασία ή αποθήκευση. Το κομμάτι του server που διαχειρίζεται όμως την αποθήκευση αυτής της πληροφορίας στον αποθηκευτικό χώρο αναπαριστάται στο σχήμα 3.5.11.

Το παρακάτω τμήμα κώδικα αναλαμβάνει την υλοποίηση ενός Next.js API handler. Ας δούμε τι κάνει κάθε μέρος του κώδικα:

1. **Ορισμός του handler**: Αυτός ο ορισμός καθορίζει τον τύπο του handler ως NextApiHandler, που είναι μια συνάρτηση που δέχεται δύο παραμέτρους, ένα αίτημα (req) και μια απάντηση (res), και επιστρέφει μια υπόσχεση (Promise).

2. **Δοκιμή ύπαρξης του φακέλου εικόνων**: Μέσα σε ένα προσπάθεια-παγίδευση (try-catch), το πρόγραμμα προσπαθεί να διαβάσει τον φάκελο /public/images χρησιμοποιώντας τη συνάρτηση fs.readdir. Εάν αυτό αποτύχει (δηλαδή δεν υπάρχει ο φάκελος), τότε δημιουργεί το φάκελο χρησιμοποιώντας τη συνάρτηση fs.mkdir. Αξίζει να σημειωθεί ότι όταν ο θα πάμε να ζητήσουμε την εικόνα από το front-end μας, το url που θα ζητήσουμε θα είναι το

   https://{το domain του server μας πχ foodie.gr}/images/{image id}

   Ο φάκελος public στο Next.Js θεωρείται "αυτονόητος" όταν ζητάμε αρχεία από αυτόν. Αυτό συμβαίνει επειδή τα μόνα αρχεία που μπορούμε να ζητήσουμε απευθείας από έναν Next.Js server θα πρέπει να βρίσκονται μέσα στον φάκελο public.

3. **Ανάγνωση του αρχείου που ανέβηκε**: Η μεταβλητή image περιέχει τα πεδία και τα αρχεία που ανέβηκαν στον server, τα οποία αποκτήθηκαν χρησιμοποιώντας τη συνάρτηση readFile που δημιουργήσαμε πιο πάνω.

4. **Αποστολή απάντησης**: Η συνάρτηση αποστέλλει μια απάντηση JSON που περιέχει το όνομα και τη διαδρομή του πρώτου αρχείου εικόνας που ανέβηκε. Η διαδρομή του αρχείου προκύπτει από το αντίστοιχο πεδίο newFilename που παρέχεται από το formidable κατά το ανέβασμα του αρχείου. Το όνομα αρχείου προέρχεται από το αρχικό όνομα του αρχείου που παρέχεται από τον χρήστη.

```
1  const handler: NextApiHandler = async (req, res) ⇒ {
2      try {
3          await fs.readdir(path.join(process.cwd() + "/public", "/images"))
4      } catch (error) {
5          await fs.mkdir(path.join(process.cwd() + "/public", "/images"))
6      }
7      const image: any = await readFile(req)
8      res.json({
9          error: false,
10         data: {
11             name: image.files.image[0].originalFilename.split(".")[0],
12             path: image.files.image[0].newFilename,
13         },
14     })
15 }
```

Σχήμα 3.5.11 Διαχείριση αποθήκευσης αρχείου εικόνας

Τώρα που έχουμε αποθηκεύσει τις εικόνες μας στον αποθηκευτικό μας χώρο, ας αποθηκεύσουμε όλη αυτή την πληροφορία στην βάση δεδομένων μας. Το παρακάτω τμήμα κώδικα δημιουργεί έναν

δρομολογητή για τη διαχείριση του ανεβάσματος εικόνων στον server. Ο δρομολογητής αυτός χειρίζεται τις αιτήσεις που αφορούν τις εικόνες και εκτελεί διάφορες ενέργειες πάνω σε αυτές.

Στην αρχή, καθορίζεται η διαδικασία για τη διαχείριση των εικόνων χρησιμοποιώντας τη συνάρτηση publicProcedure. Έπειτα, καθορίζεται το μοντέλο εισόδου που περιγράφει τα δεδομένα που απαιτούνται για το ανέβασμα μιας εικόνας στη βάση δεδομένων.

Στη συνέχεια, η λειτουργία μετάλλαξης εκτελείται ασύγχρονα. Αυτή η λειτουργία λαμβάνει τα δεδομένα εισόδου και το περιβάλλον εκτέλεσης και δημιουργεί μια εγγραφή για τη νέα εικόνα στη βάση δεδομένων, χρησιμοποιώντας τα δεδομένα από την είσοδο. Τέλος, επιστρέφει ένα αντικείμενο που περιέχει το αναγνωριστικό της εικόνας και ένα boolean πεδίο που υποδηλώνει ότι δεν υπήρξε σφάλμα κατά την εκτέλεση.

Σε αυτό το σημείο αξίζει να αναφέρουμε ότι ως τώρα δεν έχουμε δει κάποιον συγκεκριμένο μηχανισμό διαχείρισης σφαλμάτων στο back-end μας. Αυτό συμβαίνει επειδή το TRPC έχει ενσωματωμένη αυτή τη λειτουργικότητα. Εμείς στην εφαρμογή μας γράφουμε κώδικα για το καλύτερο δυνατό σενάριο, δηλαδή ότι είτε όλα θα πάνε καλά είτε όχι. Σε ένα πιο σύνθετο περιβάλλον ανάπτυξης εφαρμογών η σωστή διαχείριση σφαλμάτων αποτελεί πολύ σημαντικό κομμάτι της εφαρμογής γιατί μέσω αυτού, ο χρήστης μπορεί να έχει μια πιο σαφή εικόνα του τι έκανε λάθος ή οι προγραμματιστές μπορούν να δουν κενά λογικής στον κώδικά τους.

```
export const uploadRouter = createTRPCRouter({
    image: publicProcedure
        .input(SaveImageToDBModel)
        .mutation(async ({ input, ctx }) => {
            const image = await ctx.db.image.create({
                data: {
                    name: input.imageName,
                    path: input.imagePath,
                    categoryId: input.categoryId,
                    recipeId: input.recipeId,
                },
            });
            return {
                error: false,
                imageId: image.id,
            };
        })
})
```

Σχήμα 3.5.12 Αποθήκευση πληροφοριών εικόνων στην βάση δεδομένων

3.6 Χρήστες

Καμία εφαρμογή δεν μπορεί να δώσει μια εξατομικευμένη εμπειρία χρήσης αν δεν μπορεί να αναγνωρίσει τον χρήστη. Η σύνδεση χρηστών είναι ένα κρίσιμο στοιχείο σε πολλές εφαρμογές,

καθώς επιτρέπει την ασφαλή πρόσβαση σε προστατευμένους πόρους και λειτουργίες. Ενώ ορισμένοι χρήστες μπορεί να έχουν πρόσβαση μόνο σε συγκεκριμένα δεδομένα ή λειτουργίες, άλλοι μπορεί να έχουν πρόσβαση σε περισσότερα.

Μέσω της σύνδεσης χρηστών, μπορούμε να υλοποιήσουμε διάφορα σενάρια, όπως:

- **Αυθεντικοποίηση και Εξουσιοδότηση**: Οι χρήστες πρέπει να αυθεντικοποιηθούν πριν αποκτήσουν πρόσβαση σε προστατευμένους πόρους. Μετά την αυθεντικοποίηση, η εξουσιοδότηση καθορίζει ποιοι χρήστες έχουν πρόσβαση σε ποιους πόρους και με ποιες ενέργειες.

- **Προσωποποίηση**: Οι χρήστες μπορούν να αποκτήσουν προσωπικά προφίλ, να διατηρούν λίστες αγαπημένων, να αποθηκεύουν ρυθμίσεις και άλλα προσαρμοσμένα δεδομένα που εξυπηρετούν τις ανάγκες τους.

- **Καταγραφή Δραστηριότητας**: Η σύνδεση των χρηστών μας επιτρέπει να καταγράφουμε τη δραστηριότητά τους στην εφαρμογή, όπως τις ενέργειες που πραγματοποιούν, τις προτιμήσεις τους και τα δεδομένα που αναζητούν.

- **Περιορισμός Πρόσβασης**: Μπορούμε να εφαρμόσουμε περιορισμούς πρόσβασης, όπως περιορισμούς συχνότητας λήψης δεδομένων, περιορισμούς πρόσβασης σε συγκεκριμένες ώρες ή την εφαρμογή πολιτικών ασφαλείας όπως ο διπλός έλεγχος ταυτότητας.

Ο διαχειριστής θα πρέπει να αναγνωρίζεται από το σύστημα και να του δίνεται πρόσβαση στις ανάλογες αρμοδιότητες. Με την ίδια ακριβώς λογική πρέπει να λειτουργεί και ο χρήστης. Ο διαχειριστής δεν έχει την ανάγκη για μια σελίδα προφίλ ενώ ο απλός χρήστης θα πρέπει να έχει πρόσβαση στις συνταγές που έχει δημιουργήσει, στις αγαπημένες του συνταγές, στις πληροφορίες που φαίνονται στα σχόλια που αφήνει και άλλες ανάλογες λειτουργίες. Σε προηγούμενο κεφάλαιο είδαμε ότι αν πάμε να καλέσουμε μια από τις μεθόδους του διαχειριστή, θα γίνει έλεγχος από το back-end για να δει αν είμαστε διαχειριστής και αν δεν είμαστε θα πετάξει σφάλμα. Η σύνδεση χρηστών συχνά αποτελεί τη βάση για την ασφαλή και προσαρμοσμένη εμπειρία του χρήστη σε μια εφαρμογή.

3.6.1 User Story: Είσοδος χρηστών

Ο Νίκος που του άρεσε η εφαρμογή επιθυμεί να δημιουργήσει έναν λογαριασμό για να ανεβάσει και αυτός δικές του συνταγές, να αφήσει σχόλια και να αποθηκεύσει τις αγαπημένες του συνταγές. Στην αρχή, του ζητείται να εισέλθει στην πλατφόρμα, όπου ένα μενού εισόδου τον καλεί να επιλέξει μεταξύ των επιλογών "Σύνδεση με Google" ή "Σύνδεση με Github". Αφού επιλέξει την επιθυμητή μέθοδο σύνδεσης, οδηγείται σε μια ασφαλή σελίδα σύνδεσης όπου μπορεί να εισάγει τα διαπιστευτήριά του. Με την επιτυχή σύνδεση, μπαίνει στον κόσμο των συνταγών, έτοιμος να ανακαλύψει νέες γεύσεις και να μοιραστεί τις δικές του δημιουργίες.

Για να υλοποιήσουμε αυτή τη λειτουργία χρησιμοποιούμε την βιβλιοθήκη Next Auth που είδαμε στο κεφάλαιο 2.5.3. Η βιβλιοθήκη σε συνδυασμό με το Prisma μας δίνει έναν πολύ πολύ εύκολο τρόπο διαχειριστούμε την είσοδο των χρηστών μέσα στην εφαρμογή μας. Το σχήμα 3.6.1 παρουσιάζει ένα

σημείο του configuration του Next Auth που παρέχει δύο παροχείς (providers) για την εξωτερική σύνδεση των χρηστών μέσω των παρόχων GitHub και Google. Κάθε πάροχος απαιτεί ένα σύνολο διαπιστευτηρίων (client ID και client secret) για να επικοινωνήσει με τις αντίστοιχες υπηρεσίες (GitHub ή Google) και να παρέχει λειτουργίες σύνδεσης. Τα διαπιστευτήρια μπορεί να τα δημιουργήσει ο προγραμματιστής μέσω της σελίδας που δίνει ο κάθε πάροχος για τη διαχείριση διαπιστευτηρίων εφαρμογών. Για παράδειγμα η Google μας δίνει μια πλατφόρμα μέσω της οποίας οι προγραμματιστές μπορούν να ζητήσουν πρόσβαση σε λειτουργίες της Google (όπως η σύνδεση μέσω Google) και να ρυθμίσουν ποια domains τους θα έχουν πρόσβαση σε αυτές τις λειτουργίες. Με άλλα λόγια η Google πρέπει να γνωρίζει ότι κάποιος προγραμματιστής θέλει το domain foodie.gr να έχει πρόσβαση στην λειτουργία εισόδου μέσω Google.

Κάθε πάροχος λαμβάνει τα απαραίτητα διαπιστευτήρια (client ID και client secret) από το περιβάλλον (environment) και τα χρησιμοποιεί για να αιτηθεί και να λάβει πληροφορίες σύνδεσης από τον αντίστοιχο πάροχο. Μετά την επιτυχή επαλήθευση, ο πάροχος επιστρέφει ένα αναγνωριστικό του χρήστη, το οποίο μπορεί να χρησιμοποιηθεί για να αποκτηθούν περισσότερες πληροφορίες ή να αποθηκευτεί στην εφαρμογή.

Αυτό το μοντέλο εξυπηρετεί την ευελιξία και τη δυνατότητα επιλογής του παρόχου σύνδεσης από τους χρήστες. Για παράδειγμα, ένας χρήστης μπορεί να επιλέξει να συνδεθεί με το λογαριασμό του στο GitHub, ενώ ένας άλλος μπορεί να προτιμά τη σύνδεση μέσω του λογαριασμού του στη Google. Με αυτόν τον τρόπο, η εφαρμογή παρέχει τη δυνατότητα σύνδεσης μέσω διαφόρων πλατφορμών, ανάλογα με τις προτιμήσεις του κάθε χρήστη.

```
providers: [
    GitHubProvider({
        clientId: env.GITHUB_CLIENT_ID,
        clientSecret: env.GITHUB_CLIENT_SECRET
    }),
    GoogleProvider({
        clientId: env.GOOGLE_CLIENT_ID,
        clientSecret: env.GOOGLE_CLIENT_SECRET
    })
]
```

Σχήμα 3.6.1 Ρυθμίσεις Next Auth για παρόχους

3.6.2 Το σχήμα του χρήστη

Όπως και κάθε άλλη πληροφορία, οι λεπτομέρειες του χρήστη πρέπει να αποθηκευτούν στη βάση. Αυτή η διαδικασία είναι κρίσιμη για την αποτελεσματική λειτουργία της εφαρμογής μας και τη διασφάλιση της πρόσβασης σε απαραίτητα δεδομένα ανά πάσα στιγμή. Οι μηχανισμοί για αυτή τη λειτουργία μας είναι πλέον γνωστοί από προηγούμενα κεφάλαια, όπου έχουμε εξετάσει τις μεθόδους αποθήκευσης και ανάκτησης δεδομένων. Ας δούμε όμως το σχήμα του χρήστη πιο αναλυτικά, διότι περιλαμβάνει μερικά σημεία που αξίζει να αναφερθούν για να έχουμε μια πιο σφαιρική εικόνα της υποδομής της εφαρμογής μας.

```
model User {
    id              String          @id @default(cuid())
    name            String?
    email           String?         @unique
    image           String?
    role            Role[]          @default([user])
    accounts        Account[]
    sessions        Session[]
    Recipe          Recipe[]
    RecipeRating    RecipeRating[]
    Comment         Comment[]
    Favorite        Favorite[]
}
```

Σχήμα 3.6.2 Σχήμα χρήστη

Τα περισσότερα πεδία είναι κατανοητά με μια πρώτη ματιά. Κάθε χρήστης έχει ένα μοναδικό αναγνωριστικό (id), όνομα (name) και email. Επίσης, μπορεί να έχει μια εικόνα προφίλ (image). Ο χρήστης μπορεί να έχει συνδέσεις με διάφορους λογαριασμούς (accounts), όπως GitHub ή Google. Οι συνεδρίες που ανοίγει ο χρήστης καταγράφονται στο πεδίο sessions. Επιπλέον, ο χρήστης μπορεί να δημιουργεί συνταγές (recipes), να αξιολογεί συνταγές (recipeRatings), να κάνει σχόλια (comments) και να αποθηκεύει αγαπημένες συνταγές (favorites). Αυτό το μοντέλο καλύπτει όλες τις σημαντικές πληροφορίες που αφορούν τον χρήστη και τη δραστηριότητά του στην εφαρμογή μας. Τα τελευταία πεδία που δεν είναι τύπου String αποτελούν τις σχέσεις μεταξύ άλλων πινάκων.

3.6.3 Ρόλοι χρηστών

Ενδεχομένως να παρατηρήσατε ότι στην περιγραφή του σχήματος του χρήστη στο προηγούμενο κεφάλαιο παραλείψαμε να εξηγήσουμε το πεδίο role και τη χρησιμότητα αυτού. Οι ρόλοι των χρηστών αναφέρονται στις διακριτές κατηγορίες ή δικαιώματα που έχουν ορισμένοι χρήστες σε μια εφαρμογή ή σύστημα. Αναλόγως των ρόλων τους, οι χρήστες μπορεί να έχουν πρόσβαση σε διαφορετικές λειτουργίες, πληροφορίες ή δυνατότητες στην εφαρμογή.

Το πεδίο "role" στο μοντέλο του χρήστη καθορίζει τον ρόλο του χρήστη στην εφαρμογή. Αυτό μπορεί να είναι ένας από τους προκαθορισμένους ρόλους που έχουν οριστεί από τους εμάς, τους δημιουργούς της εφαρμογής. Οι ρόλοι μπορούν να είναι προκαθορισμένοι ή να προσαρμόζονται ανάλογα με τις ανάγκες της εφαρμογής. Ορισμένες κοινές κατηγορίες ρόλων περιλαμβάνουν:

- **Διαχειριστές (Admins)**: Οι διαχειριστές έχουν πλήρη πρόσβαση και έλεγχο στην εφαρμογή. Μπορούν να δημιουργούν, να επεξεργάζονται και να διαγράφουν περιεχόμενο, να διαχειρίζονται χρήστες και να εκτελούν άλλες διαχειριστικές εργασίες.

- **Συντάκτης (Editor)**: Ο συντάκτης είναι υπεύθυνος για τη δημιουργία και τη δημοσίευση περιεχομένου στην εφαρμογή. Μπορεί να δημιουργήσει νέες αναρτήσεις, να επεξεργαστεί υπάρχοντα περιεχόμενο και να διαχειριστεί τα σχόλια.

- **Χρήστες (Users)**: Οι χρήστες είναι οι τυπικοί χρήστες της εφαρμογής. Έχουν πρόσβαση σε βασικές λειτουργίες όπως η περιήγηση, η ανάγνωση περιεχομένου και η αλληλεπίδραση με την εφαρμογή.

- **Μέλη (Members)**: Οι μέλη μπορεί να έχουν κάποιες περιορισμένες δυνατότητες σε σχέση με τους χρήστες, όπως η δυνατότητα να δημιουργούν περιεχόμενο ή να συμμετέχουν σε ορισμένες δραστηριότητες της εφαρμογής.

Οι παραπάνω ρόλοι είναι μόνο μερικά παραδείγματα και μπορεί να προσαρμοστούν ανάλογα με τις ανάγκες και τις λειτουργίες της εφαρμογής. Στην περίπτωσή μας οι ρόλοι admin και user είναι αρκετοί. Στην εφαρμογή μας αξίζει να σημειωθεί ότι ένας admin θεωρείται και χρήστης. Δεν υπάρχει δηλαδή περιορισμός του admin στο να έχει και δικό του προφιλ.

3.6.4 User Story: Αποθήκευση αγαπημένων συνταγών

Ο Νίκος, πλέον συνδεδεμένος στην εφαρμογή, αναζητά πάντα νέες συνταγές για να δοκιμάσει και να μοιραστεί με τους φίλους του. Ένα από τα αγαπημένα του χαρακτηριστικά σε μια εφαρμογή μαγειρικής είναι η δυνατότητα να αποθηκεύει τις αγαπημένες του συνταγές. Έτσι, με κάθε νέα συνταγή που ανακαλύπτει, επιλέγει να την αποθηκεύσει για μελλοντική αναφορά. Αυτό του επιτρέπει να διατηρεί μια οργανωμένη λίστα με τις αγαπημένες του συνταγές, έτοιμες να τις επαναλάβει ή να τις μοιραστεί με τους φίλους του στο μέλλον.

Ήρθε λοιπόν η ώρα για να να σχεδιάσουμε και να υλοποιήσουμε έναν μηχανισμό για να καλύψουμε αυτή την ανάγκη του Νίκου. Ένας μηχανισμός αποθήκευσης αγαπημένων συνταγών θα μπορούσε να λειτουργεί ως εξής:

1. **Αποθήκευση Συνταγής στη Βάση Δεδομένων**: Κάθε φορά που ένας χρήστης επιλέγει να προσθέσει μια συνταγή στα αγαπημένα του, η εφαρμογή θα αποθηκεύει τα στοιχεία της συνταγής (όπως τίτλος, περιγραφή, υλικά, οδηγίες κλπ.) στη βάση δεδομένων.

2. **Σύνδεση με τον Χρήστη**: Κάθε αποθηκευμένη συνταγή θα συνδέεται με τον συγκεκριμένο χρήστη που την αποθήκευσε. Αυτό επιτυγχάνεται με την προσθήκη ενός πεδίου που αναφέρει το αναγνωριστικό του χρήστη στον πίνακα των αγαπημένων συνταγών.

3. **Διαχείριση Αγαπημένων Συνταγών**: Ο χρήστης θα έχει τη δυνατότητα να προσθέσει, να διαγράψει ή να επεξεργαστεί τις αγαπημένες του συνταγές από το προφίλ του στην εφαρμογή.

4. **Προβολή Αγαπημένων Συνταγών**: Οι αγαπημένες συνταγές του χρήστη θα είναι διαθέσιμες για προβολή σε μια ειδική σελίδα ή λίστα στην εφαρμογή. Αυτό θα του επιτρέπει να έχει γρήγορη πρόσβαση σε αυτές και να τις διαχειρίζεται εύκολα.

Αυτόν τον μηχανισμό έχουμε υλοποιήσει στην εφαρμογή μας. Με την υλοποίηση του μηχανισμού αποθήκευσης αγαπημένων συνταγών, προσφέρουμε στους χρήστες μας έναν τρόπο να δημιουργήσουν μια προσωπική συλλογή αγαπημένων γευμάτων και ιδεών μαγειρικής. Αυτή η λειτουργία ενισχύει τη διαδραστικότητα και τον βαθμό εμπλοκής τους με την πλατφόρμα,

προσφέροντάς τους έναν πιο εξατομικευμένο τρόπο πλοήγησης και ανακάλυψης νέων συνταγών που τους ενδιαφέρουν. Συνολικά, ο μηχανισμός αυτός συμβάλλει στη δημιουργία μιας πιο πλούσιας και εκφραστικής εμπειρίας χρήστη για τους λάτρεις της μαγειρικής και της διατροφής.

3.6.5 User Story: Αξιολόγηση συνταγής και σχόλια

Ο Νίκος καθώς περιηγείται στην πλατφόρμα μας, ανακαλύπτει μια συνταγή για νόστιμα νηστίσιμα μακαρόνια με λαχανικά. Μετά τη δοκιμή της συνταγής και το θετικό του αποτέλεσμα, αποφασίζει να αξιολογήσει τη συνταγή και να αφήσει ένα σχόλιο για την εμπειρία του. Με αυτόν τον τρόπο, συνεισφέρει στην κοινότητα μας, παρέχοντας χρήσιμα σχόλια και συμβουλές σε άλλους χρήστες που ενδιαφέρονται να δοκιμάσουν την ίδια συνταγή. Τα σχόλια και οι αξιολογήσεις του Μάριου βοηθούν επίσης άλλους χρήστες να καταλάβουν τις δυνατότητες και τα πλεονεκτήματα της συγκεκριμένης συνταγής, καθώς και να λάβουν έμπνευση για το τι να μαγειρέψουν επόμενα. Έτσι, ο Νίκος απολαμβάνει όχι μόνο τη δική του μαγειρική περιπέτεια, αλλά συμβάλλει και στην ενίσχυση της κοινότητας μας, διαμορφώνοντας ένα χώρο ανταλλαγής ιδεών και εμπειριών γύρω από τη μαγειρική τέχνη.

Η δυνατότητα αξιολογήσεων είναι ένα σημαντικό εργαλείο που επιτρέπει στους χρήστες να αξιολογούν τις συνταγές που έχουν δοκιμάσει και να μοιράζονται τις εμπειρίες τους με την κοινότητα. Μέσω αυτού του μηχανισμού, οι χρήστες μπορούν να δίνουν βαθμολογίες και να παρέχουν σχόλια για τις συνταγές που έχουν δοκιμάσει, επιτρέποντάς τους να εκφράσουν την ικανοποίησή τους ή τυχόν προβλήματα που παρουσιάστηκαν.

Ο μηχανισμός λειτουργεί με τον εξής τρόπο: Κάθε συνταγή έχει ένα σύνολο αξιολογήσεων από τους χρήστες. Οι χρήστες μπορούν να αξιολογήσουν μια συνταγή βαθμολογώντας την με έναν αριθμό από 1 έως 5 αστέρια και να προσθέτουν σχόλια για να περιγράψουν την εμπειρία τους. Αυτές οι αξιολογήσεις και τα σχόλια είναι ορατά σε άλλους χρήστες που ενδιαφέρονται να δοκιμάσουν τη συνταγή.

Η αξιολόγηση δεν είναι μόνο ένας τρόπος για τους χρήστες να μοιράζονται τις απόψεις τους, αλλά επίσης παρέχει στους χρήστες μια αξιόπιστη πηγή πληροφοριών. Μέσω των αξιολογήσεων και των σχολίων, οι χρήστες μπορούν να αποφασίσουν ποιες συνταγές να δοκιμάσουν βασιζόμενοι σε προσωπικές εμπειρίες και γνώμες άλλων μελών της κοινότητας. Επιπλέον, οι δημοσιευμένες αξιολογήσεις και τα σχόλια μπορούν να παρέχουν στους χρήστες χρήσιμες πληροφορίες και συμβουλές για την προετοιμασία των συνταγών.

Ο μηχανισμός διαχείρισης σχολίων λειτουργεί ως εξής: Κάθε συνταγή έχει ένα πεδίο σχολίων όπου οι χρήστες μπορούν να προσθέτουν τα δικά τους σχόλια και παρατηρήσεις. Οι χρήστες μπορούν να γράφουν σχόλια για τις συνταγές που έχουν δοκιμάσει, να μοιράζονται τις δικές τους προσαρμογές και να παρέχουν συμβουλές για τη βελτίωσή τους. Θα πρέπει επίσης να μπορούν να επεξεργαστούν το σχόλιό τους για δικούς τους λόγους όπως επίσης και να το διαγράψουν.

Τα σχόλια που δημοσιεύονται στις συνταγές είναι ορατά σε όλους τους χρήστες και αποτελούν πολύτιμη πληροφορία για όσους ενδιαφέρονται να δοκιμάσουν την ίδια συνταγή στο μέλλον. Οι σχολιασμοί παρέχουν μια πλατφόρμα για την ανταλλαγή γνώμης και την κοινότητα να μάθει από τις εμπειρίες των άλλων. Επιπλέον, τα σχόλια μπορούν να λειτουργήσουν ως πηγή έμπνευσης και καθοδήγησης για νέες ιδέες και προσεγγίσεις στο μαγείρεμα.

3.7 Υλοποίηση λειτουργιών στο Front-End

Στα προηγούμενα υποκεφάλαια επικεντρωθήκαμε στην υλοποίηση των λειτουργιών της εφαρμογής μας στο back-end, το οποίο αποτελεί τον πυρήνα της λογικής και των δεδομένων μας. Τώρα, σε αυτό το κεφάλαιο, θα εξετάσουμε το πρόσωπο της εφαρμογής μας που βλέπει ο χρήστης, δηλαδή το front-end. Η δουλειά εδώ επικεντρώνεται στην παρουσίαση και την αλληλεπίδραση με τις πληροφορίες που ανακτήθηκαν από το back-end.

Κατά τη διάρκεια της ανάπτυξης του front-end, χρησιμοποιήθηκαν σύγχρονα εργαλεία και τεχνολογίες όπως η React για τη δημιουργία δυναμικών και ευέλικτων διεπαφών χρήστη, το Next.js για την αποδοτική ανάπτυξη σελίδων και το Mantine UI για τη δημιουργία ευέλικτου και επαναχρησιμοποιήσιμου σχεδιασμού.

Στο front-end, ο χρήστης μπορεί να αλληλεπιδράσει με την εφαρμογή μας μέσω μιας φιλικής προς τον χρήστη διεπαφής, η οποία του επιτρέπει να περιηγηθεί σε διάφορες σελίδες, να αναζητήσει συνταγές, να προβάλει λεπτομέρειες για κάθε συνταγή, να προσθέσει συνταγές στα αγαπημένα του, να αξιολογήσει και να σχολιάσει συνταγές και πολλά άλλα. Ο στόχος είναι να παρέχεται μια ομαλή και ευχάριστη εμπειρία χρήστη που θα ενθαρρύνει τους χρήστες να εξερευνούν, να μαγειρεύουν και να αλληλεπιδρούν με την εφαρμογή μας.

3.7.1 Μερικές λεπτομέρειες για την React

Όταν ο χρήστης εισέρχεται πληκτρολογεί το url της εφαρμογής μας, ο browser τον στέλνει στον διακομιστή μας, ο οποίος επιστρέφει ένα κεντρικό αρχείο HTML που περιέχει το ριζικό σημείο της React εφαρμογής. Μέσα σε αυτό το HTML αρχείο, συνήθως χρησιμοποιούμε ένα στοιχείο HTML με ένα id, όπου η React θα "δέσει" την εφαρμογή μας. Αυτό το στοιχείο συνήθως ονομάζεται "root" και λειτουργεί ως σημείο εκκίνησης της React.

Κατά την ανάπτυξη της εφαρμογής, η React αναλαμβάνει τη δημιουργία και διαχείριση του DOM (Document Object Model) της σελίδας. Αυτό περιλαμβάνει τη δημιουργία και ενημέρωση των αντικειμένων του DOM, όπως τα στοιχεία HTML και το CSS, καθώς και τον συγχρονισμό τους με την κατάσταση της εφαρμογής. Η React χρησιμοποιεί την μια υλοποίηση της έννοιας Virtual DOM για να επιτύχει αποδοτικότητα και αποτελεσματικότητα στη διαχείριση του DOM. Το Virtual DOM είναι ένα σημαντικό χαρακτηριστικό της React που συμβάλλει στην αποδοτικότητα και την αποτελεσματικότητα της διαχείρισης του DOM σε μια εφαρμογή. Ας εξετάσουμε τρία βασικά σημεία του Virtual DOM.

1. Απεικόνιση σε εικονικό επίπεδο

- Το Virtual DOM είναι μια δομή δεδομένων που αποτελεί μια αναπαράσταση του πραγματικού DOM σε μνήμη.
- Κάθε στοιχείο του πραγματικού DOM έχει ένα αντίστοιχο στο Virtual DOM, το οποίο περιλαμβάνει πληροφορίες για τον τύπο του στοιχείου, τις ιδιότητες του, καθώς και τα παιδιά του στο δέντρο DOM.
- Κάθε φορά που υπάρχει αλλαγή στον πραγματικό DOM, η React δημιουργεί ένα νέο Virtual DOM που αντικατοπτρίζει αυτές τις αλλαγές.

2. Διαφορική ανανέωση

- Η React χρησιμοποιεί τον Virtual DOM για να συγκρίνει το προηγούμενο Virtual DOM με το νέο και να εντοπίζει τις διαφορές μεταξύ τους.
- Αυτή η διαδικασία ονομάζεται διαφορική ανανέωση και επιτρέπει στη React να εφαρμόζει μόνο τις απαραίτητες αλλαγές στον πραγματικό DOM, αντί να ανανεώνει ολόκληρο το δέντρο DOM.

3. Απόδοση και αποτελεσματικότητα

- Η χρήση του Virtual DOM συμβάλλει στη βελτίωση της απόδοσης και της αποτελεσματικότητας της εφαρμογής React.
- Επειδή η React εφαρμόζει μόνο τις αλλαγές που απαιτούνται στον πραγματικό DOM, μειώνεται ο χρόνος που απαιτείται για την ανανέωση της διεπαφής του χρήστη.
- Επιπλέον, η επιλεκτική εφαρμογή αλλαγών στο DOM συμβάλλει στη μείωση της κατανάλωσης πόρων του προγράμματος και στη βελτίωση της απόκρισης της εφαρμογής.

Όταν ο χρήστης αλληλεπιδρά με την εφαρμογή, η React αντιλαμβάνεται αυτές τις αλληλεπιδράσεις και επανασχεδιάζει δυναμικά τα αντίστοιχα τμήματα της εφαρμογής που έχουν αλλάξει. Αυτό σημαίνει ότι οι αλλαγές που γίνονται στην κατάσταση της εφαρμογής αντικατοπτρίζονται αυτόματα στο UI, χωρίς την ανάγκη για ανανέωση της σελίδας. Αυτή η αυτόματη διαχείριση των αλλαγών ονομάζεται "διαχείριση ανανεώσεων" (differential rendering [33]) και αποτελεί ένα από τα βασικά πλεονεκτήματα της React στον τομέα των Single Page Applications.

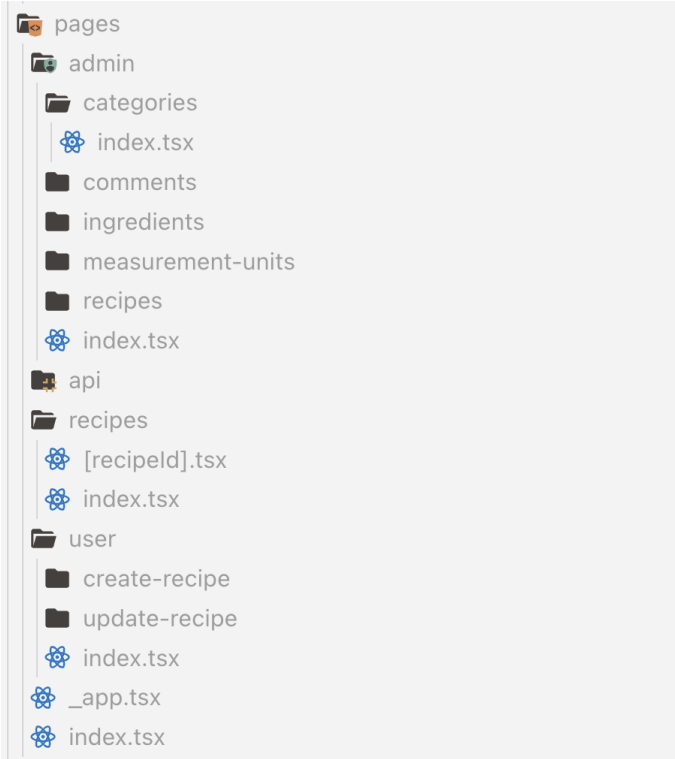
Το Next.Js που μελετήσαμε στο κεφάλαιο 2.5.2 είναι ένα framework για την React που μας δίνει μερικές ευκολίες στην υλοποίηση των εφαρμογών μας. Στα επόμενα υποκεφάλαια θα δούμε αναλυτικά την χρησιμότητα του Next.Js σε συνδυασμό με το TRPC και την ευκολία χρήσης που μας δίνει για την διαχείριση των διαδρομών μας στην εφαρμογή.

3.7.2 Δρομολόγηση - React Router DOM vs Next.Js Pages Router

Το React Router DOM [34] είναι μια βιβλιοθήκη που χρησιμοποιείται συχνά στο React για τη δρομολόγηση (routing) σε μια εφαρμογή. Το React Router DOM παρέχει ένα σύνολο στοιχείων που μπορούν να χρησιμοποιηθούν για τον ορισμό διαδρομών στην εφαρμογή, όπως <BrowserRouter>, <Route>, <Switch> και άλλα. Η διαχείριση των διαδρομών γίνεται με βάση την κατάσταση της εφαρμογής και η αλλαγή των σελίδων γίνεται χωρίς ανανέωση της σελίδας.

Από την άλλη πλευρά, το Pages Router του Next.js είναι ένα ενσωματωμένο σύστημα δρομολόγησης που παρέχεται από το Next.js για την ανάπτυξη διαδικτυακών εφαρμογών. Με το Pages Router, η δρομολόγηση γίνεται μέσω του φακέλου pages στο project Next.js. Κάθε αρχείο JavaScript ή TypeScript που τοποθετείται σε αυτόν τον φάκελο αντιστοιχεί σε μια διαδρομή στην εφαρμογή. Αυτό σημαίνει ότι κάθε αρχείο στο φάκελο pages μπορεί να είναι μια σελίδα της εφαρμογής μας.

Και τα δύο συστήματα δρομολόγησης προσφέρουν τη δυνατότητα διαχείρισης της διαδρομής και της κατάστασης της εφαρμογής, αλλά με διαφορετικούς τρόπους υλοποίησης. Ενώ το React Router DOM προσφέρει μια πιο ευέλικτη και προσαρμόσιμη προσέγγιση, το Pages Router του Next.js προσφέρει μια απλή και ευκολότερη προς χρήση λύση, ιδανική για τη γρήγορη ανάπτυξη εφαρμογών χωρίς την ανάγκη για ρύθμιση ή διαμόρφωση της δρομολόγησης.



Σχήμα 3.7.1 Δομή NextJs Pages Router

Κάθε ένα από τα παραπάνω αρχεία αντιπροσωπεύει μια αντίστοιχη διαδρομή. Για παράδειγμα για να δούμε το περιεχόμενο του αρχείου που βρίσκεται στον φάκελο admin -> categories -> index.tsx θα πρέπει στον browser μας να βάλουμε την διαδρομή /admin/categories. Σημειώνεται ότι τα αρχεία index.tsx θεωρούνται τα κεντρικά σημεία πρόσβασης σε έναν φάκελο και γι αυτό τον λόγο δεν χρειάζεται στον browser μας να πάμε στην διαδρομή /admin/categories/index.

Πέρα από τα κεντρικά αρχεία index.tsx, ο pages router υποστηρίζει και μια διαφορετική σύνταξη. Το αρχείο [recipeId].tsx στο Pages Router του Next.js αντιστοιχεί σε μια δυναμική διαδρομή (dynamic route). Αυτό σημαίνει ότι το recipeId είναι μια μεταβλητή που μπορεί να πάρει οποιαδήποτε τιμή κατά την προβολή της σελίδας.

Για παράδειγμα, αν θέλουμε να έχουμε μια σελίδα που να δείχνει τη συνταγή με βάση το αναγνωριστικό της (id), μπορούμε να χρησιμοποιήσουμε ένα αρχείο με το όνομα [recipeId].tsx. Όταν ο χρήστης προσπαθεί να προβάλει μια συγκεκριμένη συνταγή με ένα συγκεκριμένο id, το Next.js θα προσπαθήσει να αντιστοιχίσει αυτό το id στην παράμετρο recipeId και να φορτώσει το αντίστοιχο αρχείο [recipeId].tsx από τον φάκελο pages.

Έτσι, το αρχείο [recipeId].tsx μπορεί να χρησιμοποιηθεί για τη δημιουργία μιας δυναμικής σελίδας συνταγών, όπου το περιεχόμενο της σελίδας εξαρτάται από το συγκεκριμένο idτης συνταγής που

προβάλλεται. Δηλαδή όταν ο χρήστης πάει στην διαδρομή /recipes/5 θα του εμφανιστεί η συνταγή με αναγνωριστικό 5.

3.7.3 Βασικές ρυθμίσεις και συνδυασμός με TRPC

Τα θέματα (ή themes) στις εφαρμογές React είναι ουσιώδεις παράγοντες για τη διατήρηση συνοχής και στυλιστικής ενότητας σε ολόκληρο το project. Καθορίζουν την εμφάνιση των στοιχείων και τη συμπεριφορά τους σε διάφορες καταστάσεις και περιβάλλοντα. Ένα καλά ορισμένο θέμα περιλαμβάνει πληροφορίες για τα βασικά χρώματα, τα μεγέθη γραμματοσειράς, τα περιθώρια και άλλες σχετικές παραμέτρους που επηρεάζουν την εμφάνιση των στοιχείων της εφαρμογής.

Με τον παρακάτω κώδικα, δημιουργούμε ένα θέμα χρησιμοποιώντας τη βιβλιοθήκη Mantine. Ορίζουμε τα βασικά χαρακτηριστικά του θέματος, όπως τα χρώματα, το μέγεθος γραμματοσειράς, τις γραμματοσειρές, τα περιθώρια και άλλα. Η χρήση των θεμάτων μας επιτρέπει να διαχειριστούμε εύκολα την εμφάνιση των στοιχείων της εφαρμογής μας και να εξασφαλίσουμε τη συνοχή και την ευκολία στη συντήρησή τους. Επίσης, η δυνατότητα προσαρμογής του θέματος κάνει ευκολότερη την προσθήκη νέων στοιχείων ή την προσαρμογή των υπαρχόντων σύμφωνα με τις ανάγκες της εφαρμογής μας. Αυτό καθιστά τα θέματα ένα ισχυρό εργαλείο για τη δημιουργία ενός συνεκτικού και ευέλικτου σχεδιασμού.

```
export const theme = createTheme({
    colors: COLORS,
    activeClassName: classes.active,
    primaryColor: "foodie",
    defaultRadius: "md",
    cursorType: "pointer",
    breakpoints: {
        xs: "30em",
        sm: "48em",
        md: "64em",
        lg: "74em",
        xl: "90em",
        "2xl": "120em",
    },
    scale: 1.15,
    fontFamily: `${Inter.style.fontFamily}, Segoe UI, Roboto, Verdana, sans-serif`,
    fontFamilyMonospace: `${Inter.style.fontFamily}, Monaco, Courier, monospace`,
    fontSizes: {
        xs: rem(10),
        sm: rem(12),
        md: rem(14),
        lg: rem(16),
        xl: rem(20),
    }
})
```

Σχήμα 3.7.2 Το θέμα της εφαρμογής για το Mantine

Αν θέλουμε να παραμετροποιήσουμε ένα ήδη υπάρχον component που μας δίνει το Mantine, θα πρέπει να το αλλάξουμε με τον παρακάτω κώδικα. Αρχικά, γίνεται εισαγωγή των CSS κλάσεων από το αρχείο MultiSelect.module.css μέσω της εισαγωγή import classes from "./MultiSelect.module.css".

Στη συνέχεια, ορίζεται η σταθερά MultiSelectConfig, η οποία είναι μια διεύρυνση (extension) του MultiSelect με συγκεκριμένες παραμέτρους που προσαρμόζουν τη συμπεριφορά και την εμφάνισή του. Οι παράμετροι που ορίζονται είναι οι εξής:

- **defaultProps**: Ορίζει προκαθορισμένες τιμές για ορισμένες ιδιότητες του MultiSelect, όπως το στρογγυλεμένο σχήμα ("radius"), η παρουσία εικονιδίου ελέγχου ("withCheckIcon") και η απόκρυψη επιλεγμένων επιλογών ("hidePickedOptions").

- **classNames**: Καθορίζει τις CSS κλάσεις που θα εφαρμοστούν στα διάφορα στοιχεία του MultiSelect. Πιο συγκεκριμένα, ορίζει τις κλάσεις που θα εφαρμοστούν στον κύριο ριζικό κόμβο ("root"), στην ετικέτα ("label") και στις κουκκίδες ("pill"). Αυτές οι κλάσεις προέρχονται από το αρχείο CSS που εισήχθη προηγουμένως.

Αυτές οι ρυθμίσεις παρέχουν έναν εύκολο τρόπο για τον προσαρμογή της εμφάνισης και της συμπεριφοράς του MultiSelect στην εφαρμογή μας. Για να αξιοποιήσουμε το MultiSelectConfig που μόλις φτιάξαμε θα πρέπει να το περάσουμε μέσα στο θέμα μας στο πεδίο που ορίζει το Mantine για τα component overrides. Αυτό είναι το πεδίο components και μέσα σε αυτό θα πρέπει να ορίσουμε το override για το MultiSelect με τον εξής τρόπο. **components: { MultiSelect: MultiSelectConfig }**

```
1  import classes from "./MultiSelect.module.css"
2
3  export const MultiSelectConfig = MultiSelect.extend({
4      defaultProps: {
5          radius: "md",
6          withCheckIcon: false,
7          hidePickedOptions: true,
8      },
9      classNames: {
10         root: classes.root,
11         label: classes.label,
12         pill: classes.pill,
13     }
14 })
```

Σχήμα 3.7.3 MultiSelect component override

Τα επόμενα βήματα στην εφαρμογή μας εστιάζουν στο MyApp component που λειτουργεί ως βασικό σημείο εκκίνησης. Αυτό το component είναι υπεύθυνο για την αρχικοποίηση της εφαρμογής μας και την παροχή κοινών λειτουργιών σε ολόκληρο το δέντρο των συστατικών (component) της. Ένα σημαντικό στοιχείο που πρέπει να λάβουμε υπόψη είναι το θέμα της εφαρμογής μας που ορίσαμε πιο πάνω. Το θέμα καθορίζει την εμφάνιση και την αίσθηση της εφαρμογής μας, επηρεάζοντας στοιχεία όπως τα χρώματα, οι γραμματοσειρές και τα γενικά στυλ. Ας δούμε σε λεπτομέρεια τον απαραίτητο κώδικα για να ξεκινήσουμε την εφαρμογή μας, συμπεριλαμβανομένης της εισαγωγής των βασικών component και της ρύθμισης του θέματος. Αυτό θα περιλαμβάνει τη χρήση των κατάλληλων βιβλιοθηκών και εργαλείων για τη διαχείριση των στυλ και την εξασφάλιση μιας συνεπούς και επαγγελματικής εμφάνισης σε όλες τις σελίδες της εφαρμογής. Χωρίς το θέμα μας θα έπρεπε σε κάθε σημείο που χρησιμοποιούμε τα components να εφαρμόσουμε τα κοινά στυλ με το “χέρι”, πράγμα που θα έκανε τις μελλοντικές αλλαγές σε χρώματα και μεγέθη δύσκολες και χρονοβόρες.

```tsx
const MyApp = ({
    Component,
    pageProps: { session, ...pageProps },
}: AppType<{ session: Session | null }> & AppPropsWithLayout) => {
    const getLayout = Component.getLayout ?? ((page) => page)

    return (
        <SessionProvider session={session as Session}>
            <MantineProvider theme={theme}>
                <ModalsProvider>
                    <Notifications position="bottom-center" />
                    {getLayout(<Component {...pageProps} />)}
                </ModalsProvider>
            </MantineProvider>
        </SessionProvider>
    )
}

export default api.withTRPC(MyApp)
```

Σχήμα 3.7.4 Κεντρικό αρχείο εισόδου της διεπαφής

MyApp Component

Αυτή η συνάρτηση React λειτουργεί ως ο πυρήνας της εφαρμογής μας. Λαμβάνει δύο παραμέτρους: το Component, το οποίο αναπαριστά την τρέχουσα σελίδα, και το pageProps, που περιέχει πρόσθετες πληροφορίες για τη σελίδα. Με τη χρήση της getLayout μέθοδος, μπορούμε να προσαρμόσουμε τη δομή της σελίδας, ανάλογα με τη συγκεκριμένη απαίτηση. Σε πολύπλοκες εφαρμογές μπορεί να υπάρχουν διαφορετικά layouts, όπως για παράδειγμα ένα layout που θα είναι το side navigation για τον διαχειριστή και ένα που θα είναι για τον απλό χρήστη. Στην εφαρμογή μας έχουμε μόνο ένα layout το οποίο είναι κοινό για όλους. Αυτό περιλαμβάνει το header που είναι η μπάρα που βρίσκεται στο πάνω μέρος την εφαρμογής και το footer που βρίσκεται πάντα στο τέλος της σελίδας.

Providers

Χρησιμοποιούμε διάφορους providers για να παρέχουμε συγκεκριμένες λειτουργίες σε ολόκληρη την εφαρμογή. Ο SessionProvider διαχειρίζεται τη συνεδρία του χρήστη, ο MantineProvider παρέχει το θέμα του Mantine UI framework, ενώ ο ModalsProvider διαχειρίζεται την εμφάνιση και την απόκρυψη των modals. Με αυτούς τους providers, μπορούμε να διαχειριστούμε ορισμένες βασικές λειτουργίες της εφαρμογής μας με συνέπεια και αποτελεσματικότητα.

TRPC Integration

Μέσω της γραμμής κώδικα api.withTRPC, ενσωματώνουμε το TRPC στην εφαρμογή μας. Αυτό επιτρέπει τη δημιουργία αιτημάτων προς τον διακομιστή μας για δεδομένα με το TRPC. Η

ενσωμάτωση του τροποποιημένου πρωτόκολλου RPC (Remote Procedure Call) ή TRPC στην εφαρμογή μας αποτελεί ένα σημαντικό κομμάτι της υποδομής μας. Αξιοποιώντας το TRPC, μπορούμε να ανταλλάσσουμε δεδομένα και να εκτελούμε λειτουργίες μεταξύ του front-end και του back-end της εφαρμογής μας με έναν αποδοτικό τρόπο. Ένα από τα βασικά πλεονεκτήματα του TRPC είναι η αυτόματη γεννήτρια κλάσεων στον client, η οποία μας επιτρέπει να επικοινωνούμε με το back-end μας χωρίς την ανάγκη για χειροκίνητη κωδικοποίηση των αιτημάτων και των απαντήσεων. Με την REST υλοποίηση που είναι η πιο συνήθης, θα έπρεπε να δημιουργήσουμε μόνοι μας το αίτημα και να το στείλουμε στον server. Με το TRPC αυτό το αίτημα πλέον γίνεται αυτόματα μια μέθοδος που μπορούμε να καλέσουμε απευθείας από τον client. Αυτό μειώνει τον κίνδυνο σφαλμάτων και επιταχύνει την ανάπτυξη της εφαρμογής μας. Θα δούμε στο επόμενο υποκεφάλαιο την ευκολία χρήσης του TRPC σε σχέση με την αντίστοιχη REST υλοποίηση.

Με το TRPC, μπορούμε να ορίσουμε πολλούς διαφορετικούς τύπους αιτημάτων και να διαχειριζόμαστε την ασύγχρονη επικοινωνία μεταξύ του client και του server. Αυτό μας επιτρέπει να δημιουργήσουμε εφαρμογές με δυνατότητα ανταπόκρισης σε πραγματικό χρόνο και να παρέχουμε μια ομαλή και ευχάριστη εμπειρία χρήστη. Επιπλέον, η χρήση του TRPC μας επιτρέπει να δημιουργήσουμε εύκολα και αποτελεσματικά τις λειτουργίες που απαιτούνται από την εφαρμογή μας, επιτρέποντάς μας να επικεντρωθούμε στην ανάπτυξη λειτουργικού κώδικα χωρίς την ανάγκη για πολύπλοκη υποδομή.

3.7.4 TRPC ή REST

Το TRPC και το REST είναι δύο διαφορετικές προσεγγίσεις για τη διαμόρφωση της επικοινωνίας μεταξύ client και server σε μια διαδικτυακή εφαρμογή. Καθώς κάθε μοντέλο έχει τα δικά του χαρακτηριστικά και πλεονεκτήματα, η επιλογή μεταξύ τους εξαρτάται από τις ανάγκες και τις προτεραιότητες της κάθε εφαρμογής.

Το REST, με την απλότητα και την ευελιξία του, προσφέρει μια ευρέως αποδεκτή προσέγγιση για τη διαμόρφωση διαδικτυακών εφαρμογών. Με τη χρήση των HTTP μεθόδων και του JSON για τη μεταφορά δεδομένων, το REST επιτρέπει την αποδοτική αλληλεπίδραση μεταξύ client και server. Η ανάπτυξη και η συντήρηση μιας εφαρμογής REST είναι συνήθως απλή και κατανοητή, καθώς οι πόροι και οι ενέργειες περιγράφονται με σαφήνεια.

Από την άλλη πλευρά, το TRPC προσφέρει μια πιο δυναμική και ασφαλή προσέγγιση σε συνδυασμό με την Typescript. Χρησιμοποιώντας την ιδέα της κλήσης συναρτησιακών μεθόδων αντί της διαχείρισης πόρων, το TRPC επιτρέπει την ευέλικτη και ασφαλή αλληλεπίδραση μεταξύ client και server. Η χρήση της TypeScript επιτρέπει τη διασφάλιση τύπων και την εύκολη διαχείριση των δεδομένων επειδή όλα τα αρχεία που περιέχουν τους τύπους μας για το backend και για την βάση δεδομένων μας βρίσκονται στο ίδιο repository, ενώ η αυτόματη γεννήτρια κώδικα μπορεί να επιταχύνει τη διαδικασία ανάπτυξης. Η αυτόματη γεννήτρια κώδικα είναι αυτό που καθιστά τη συλλογή τεχνολογιών που χρησιμοποιούμε σε αυτή την πτυχιακή πολύ παραγωγική. Από τη στιγμή που θα ορίσουμε το μοντέλο της βάσης δεδομένων μας στο Prisma, αυτομάτως έχουμε διαθέσιμους όλους του τύπους σε κάθε συνάρτηση που έχει σχέδη με την βάση. Αυτό είναι πολύ βοηθητικό γιατί την στιγμή που θα αλλάξουμε έστω και ένα πεδίο στην βάση δεδομένων μας, αμέσως θα ξέρουμε όλα τα σημεία του κώδικάς μας που επηρεάστηκαν. Επίσης, μιας και πλέον ο κώδικάς μας έχει πολλά errors (από τις αλλαγές στο μοντέλο της βάσης), δεν θα επιτρέπεται να στείλουμε μια έκδοση του λογισμικού μας live για τους πελάτες, πράγμα που θα έκανε την εμπειρία τους άσχημη.

Ας δούμε όμως την διαφορά στην υλοποίηση του κώδικα που θα μας φέρνει μια συνταγή χρησιμοποιώντας το TRPC με την βοήθεια του Tanstack Query και την αντίστοιχη υλοποίηση που θα κάναμε αν χρησιμοποιούσαμε το REST.

```
const { data: recipe, error, isLoading } = api.recipes.getRecipe.useQuery(
    { id: recipeId },
    {
        cacheTime: 1000 * 60 * 60,
        enabled: !!recipeId
    })
```

Σχήμα 3.7.5 λήψη συνταγής με το TRPC

Η μέθοδος getRecipes έγινε αυτόματα διαθέσιμη σε εμάς στον client όταν δημιουργήσαμε την υλοποίησή της στο back-end. Το api είναι το κεντρικό σημείο που γίνεται το integration με το TRPC όπως είδαμε στο κεφάλαιο 3.7.3. Μέσα στο api λοιπόν είναι διαθέσιμος ο δρομολογητής recipes που αφορά τις λειτουργικότητες των συνταγών και μέσα σε αυτόν τον δρομολογητή υπάρχει η μέθοδός μας για να παίρνουμε συνταγές. Αυτή η μέθοδος στην πλευρά του client έχει κάποιες περισσότεες λειτουργικότητες, μία από από τις οποίες είναι η μέθοδος useQuery. Αυτή η μέθοδος δέχεται δύο παραμέτρους. Η πρώτη είναι το payload που θα στείλει στον server και η δεύτερη μερικές επιλογές για τη λειτουργία αυτής της μεθόδου. Στην περίπτωσή μας την έχουμε βάλει να τρέξει μόνο όταν υπάρχει recipeId με την χρήση του enabled: !!recipeId και της έχουμε πει να το κρατήσει στην cache για 1000 * 60 * 60 ms δηλαδή για 1 ώρα. Η useQuery επιστρέφει τρία βασικά αποτελέσματα που αποθηκεύονται στις μεταβλητές data, error και isLoading. Η μεταβλητή data περιέχει τα δεδομένα της συνταγής που ανακτήθηκαν από τον server, ενώ η μεταβλητή error περιέχει οποιοδήποτε σφάλμα που παρουσιάστηκε κατά τη διάρκεια του αιτήματος. Το isLoading δείχνει εάν το αίτημα είναι ακόμα σε εξέλιξη ή όχι.

```
type Recipe = {
    id: number
    name: string
    ..."rest of recipe fields"
}

function useGetRecipe(recipeId: number) {
    return useQuery(['recipe', recipeId], async () => {
        const response = await axios.get<Recipe>(`/api/recipes/${recipeId}`)
        return response.data
    })
}

const { data: recipe, error, isLoading } = useGetRecipe(recipeId)

```

Σχήμα 3.7.6 Λήψη συνταγής με REST

Η αντίστοιχη υλοποίηση με χρήση REST θα περιλάμβανε τη χρήση μιας βιβλιοθήκης HTTP όπως το

Axios ή το Fetch API για την αποστολή αιτήσεων HTTP στον server. Θα έπρεπε να κατασκευάσουμε ένα αίτημα GET που να αντιστοιχεί στο API endpoint για τη λήψη μιας συγκεκριμένης συνταγής βάσει του αναγνωριστικού της (recipeId). Ένα παράδειγμα υλοποίησης με το Axios φαίνεται στο σχήμα 3.7.5.

Στην υλοποίηση με το REST θα έπρεπε να διαχειριστούμε εμείς τον τύπο της απάντησης του back-end καθώς πλέον ο client και ο server είναι δύο διαφορετικά repositories που πιθανότατα είναι και γραμμένα σε διαφορετικές γλώσσες προγραμματισμού. Στην περίπτωση που το back-end άλλαζε την δομή της απάντησης δεν θα είχαμε απευθείας σφάλμα που θα μας δείξει να αλλάξουμε τα σημεία που χρησιμοποιούμε τα παλιά πεδία. Θα έπρεπε να περιμένουμε από τους προγραμματιστές του back-end να μας το πούνε, να το διαπιστώσουμε εμείς κατά τη διάρκεια της ανάπτυξης ή στο χειρότερο σενάριο να το διαπιστώσει κάποιος χρήστης μας όταν η εφαρμογή είναι σε αυτόν. Επίσης με την υλοποίηση του REST θα έπρεπε να φτιάχνουμε ένα function για κάθε endpoint του backend και να διαχειριζόμαστε τις πιθανές μελλοντικές αλλαγές αυτών των endpoints.

3.7.5 Mutations με το TRPC

Τα mutations στο TRPC αναφέρονται στις λειτουργίες που τροποποιούν τα δεδομένα στον server. Χρησιμοποιούνται για να διαχειριστούν τη δημιουργία, ενημέρωση ή διαγραφή αντικειμένων. Το TRPC παρέχει έναν ευέλικτο τρόπο για τη διαχείριση αυτών των λειτουργιών, επιτρέποντάς μας να καλούμε mutations μέσω του διακομιστή TRPC και να διαχειριζόμαστε τις αλλαγές στα δεδομένα μας με ασφάλεια και ευκολία. Ένα mutation ουσιαστικά είναι μια μέθοδος του back-end, όπως η updateCategory που είδαμε στο κεφάλαιο 3.5.2, που μας είναι διαθέσιμη μέσω του api integration με το TRPC. Παρακάτω περιγράφεται η χρήση του updateCategory mutation από την πλευρά του client.

```
const { mutateAsync: updateCategory } = api.admin.updateCategory.useMutation({
    onSuccess: async () ⇒ {
        await utils.admin.getCategories.invalidate()
        showNotification({
            title: "Επιτυχία",
            message: "Η κατηγορία ενημερώθηκε επιτυχώς",
            color: "success",
        })
        onClose()
    },
    onError: () ⇒ {
        showNotification({
            title: "Πρόβλημα",
            message: "Κάτι πήγε στραβά",
            color: "error",
        })
    }
})
```

Σχήμα 3.7.7 Mutation ενημέρωσης κατηγορίας

Αυτό το κομμάτι κώδικα εκτελεί ένα mutation για την ενημέρωση μιας κατηγορίας χρησιμοποιώντας το hook useMutation που παρέχεται από το TRPC για να καλέσει την μέθοδο updateCategory που θα ενημερώσει τα δεδομένα της κατηγορίας στο back-end.

Στην περίπτωση επιτυχούς ολοκλήρωσης, η λειτουργία onSuccess καλείται. Εδώ, πρώτα ακυρώνεται η προηγούμενη αποθηκευμένη λίστα κατηγοριών με τη χρήση της utils.admin.getCategories.invalidate(). Με αυτόν τον τρόπο σε όλες τις σελίδες της εφαρμογής μας που έχουμε λάβει δεδομένα μέσω της μεθόδου getCategories θα καθαριστεί η cache, που σημαίνει πως όταν ο χρήστης πάει σε αυτές θα χρειαστεί να πάρει τις τελευταίες πληροφορίες από την βάση δεδομένων μας και να δει την σωστή αναπαράσταση δεδομένων. Στη συνέχεια, εμφανίζεται μια επιτυχημένη ειδοποίηση στον χρήστη με το showNotification() που μας παρέχεται από το Mantine όπως το είδαμε στο κεφάλαιο 3.7.4, ενώ τέλος, το παράθυρο που χρησιμοποιείται για την ενημέρωση κατηγοριών κλείνει. Σε περίπτωση αποτυχίας, η λειτουργία onError καλείται, εμφανίζοντας μια ειδοποίηση σφάλματος στον χρήστη.

3.7.6 Μια πιο λεπτομερής ματιά στην React

Μέχρι στιγμής στο κεφάλαιο 3.7 είδαμε την υλοποίηση των λειτουργικών απαιτήσεων από την πλευρά της του πελάτη. Παρακάτω περιγράφεται μια αναπαράσταση της δομής που μέσω της τεχνικής Virtual DOM που είδαμε στο κεφάλαιο 3.7.1 μετατρέπεται σε πραγματικό DOM που μπορεί να δει ο χρήστης. Αυτό το τμήμα κώδικα αναπαριστά τη φόρμα επεξεργασίας κατηγορίας εντός ενός modal. Η χρήση του modal βοηθά στην οργάνωση και την ευκολία χρήσης της φόρμας, καθώς προσφέρει ένα πλαίσιο εργασίας που επιτρέπει στον χρήστη να επικεντρωθεί στην επεξεργασία των πληροφοριών της κατηγορίας. Το modal θα είναι ανοιχτό μόνο όταν η κατάσταση isOpen αλλάξει από τον χρήστη. Συνήθως αυτή η κατάσταση αλλάζει από το πάτημα ενός κουμπιού, δηλαδή όταν ο χρήστης πατήσει το κουμπί “Επεξεργασία κατηγορίας”.

Εδώ χρησιμοποιείται το useForm hook [35] του Mantine για τη διαχείριση της φόρμας. Το hook αυτό δέχεται ένα αντικείμενο UpdateCategoryModel που ορίζει τη δομή και τους κανόνες επικύρωσης των δεδομένων της φόρμας με χρήση του Zod. Το σχήμα επικύρωσης δεδομένων του Zod στην προκειμένη περίπτωση προϋποθέτει ότι το όνομα της κατηγορίας είναι τουλάχιστον ένας χαρακτήρας, όπως και ότι η περιγραφή της κατηγορίας είναι τουλάχιστον ένας χαρακτήρας. Σε περίπτωση που υπάρχουν σφάλματα κατά την επικύρωση ένα μήνυμα θα εμφανιστεί στον χρήστη κάτω από το αντίστοιχο στοιχείο εισόδου. Η σύνδεση της φόρμας με τα στοιχεία εισόδου γίνεται μέσω της μεθόδου form.getInputProps που συνδέει κάθε στοιχείο με το αντίστοιχο πεδίο που υπάρχει μέσα στην φόρμα.

Τα στοιχεία της φόρμας περιλαμβάνουν διάφορα είδη εισόδου, όπως text input, select και number input, για την εισαγωγή του ονόματος, της γονικής κατηγορίας, της περιγραφής και της προτεραιότητας της κατηγορίας. Οι επιλογές για το πεδίο γονική κατηγορίας είναι οι ήδη υπάρχουσες κατηγορίες. Τέλος, περιλαμβάνει ένα component για τη μεταφόρτωση εικόνων (DropzoneComponent) που επιτρέπει στον χρήστη να ανεβάσει εικόνες για την κατηγορία.

Τα δύο κουμπιά στο κάτω μέρος του modal δίνουν στον χρήστη τη δυνατότητα να επιλέξει ανάμεσα στην ακύρωση της ενέργειας ή την εφαρμογή των αλλαγών. Το κουμπί "Ενημέρωση" είναι απενεργοποιημένο μέχρις ότου η φόρμα να είναι έγκυρη, προσφέροντας έναν μηχανισμό προστασίας από ανεπιθύμητες ενέργειες. Όλα αυτά συνοδεύονται από μια λεπτομερή διαχείριση σφαλμάτων και επιτυχίας, που προσφέρουν ενημερώσεις στον χρήστη σχετικά με την κατάσταση της ενέργειας.

```tsx
const form = useForm<UpdateCategory>({
    validate: zodResolver(UpdateCategoryModel),
})

return (
    <Modal
        opened={isOpen}
        onClose={onClose}
        title="Επεξεργασία κατηγορίας"
    >
        <Stack>
            <TextInput
                label="Όνομα κατηγορίας"
                placeholder="Εισάγετε όνομα κατηγορίας"
                {...form.getInputProps("name")}
            />
            <Select
                label="Γονική κατηγορία"
                placeholder="Επιλέξτε κατηγορία"
                clearable
                data={parentCategories.map((category) =>
                    ({ value: category.id, label: category.name }))
                }
                {...form.getInputProps("parentRecipeCategoryId")}
            />
            <TextInput
                label="Περιγραφή"
                placeholder="Εισάγετε περιγραφή"
                {...form.getInputProps("slug")}

            />
            <NumberInput
                label="Προτεραιότητα"
                hideControls
                placeholder="Εισάγετε προτεραιότητα"
                {...form.getInputProps("priority")}
            />

            <DropzoneComponent />

            <Flex justify="space-between">
                <Button
                    onClick={onClose}
                    variant="outline"
                >
                    Ακύρωση
                </Button>
                <Button
                    disabled={!form.isValid()}
                    onClick={async () => {
                        await handleUpdateCategory(form.values)
                    }}
                    color="success"
                >
                    Ενημέρωση
                </Button>
            </Flex>
        </Stack>
    </Modal>
)
```

Σχήμα 3.7.8 Component UpdateCategoryModal

Κεφάλαιο 4ο: Η εφαρμογή Foodie

Στα προηγούμενα κεφάλαια είδαμε τη διαδικασία ανάπτυξης μιας πλήρους εφαρμογής από την άποψη του back-end και του front-end. Τώρα, έχοντας ολοκληρωθεί η υλοποίηση των λειτουργιών και του σχεδιασμού της εφαρμογής μας, έχουμε φτάσει στο στάδιο όπου είμαστε έτοιμοι να την φέρουμε σε λειτουργία. Σε αυτό το κεφάλαιο θα δούμε πώς η εφαρμογή μας αλληλεπιδρά με τους χρήστες και πώς μπορούμε να δοκιμάσουμε τις λειτουργίες που έχουμε αναπτύξει. Θα εξετάσουμε την πλοήγηση στην εφαρμογή, την επικοινωνία με το back-end για τη λήψη και αποστολή δεδομένων, καθώς και τη διαχείριση τυχόν σφαλμάτων και ανταποκρίσεων. Με την εφαρμογή μας πλέον σε λειτουργία, θα είμαστε έτοιμοι να παρέχουμε στους χρήστες την εμπειρία που επιθυμούν και να αντιμετωπίσουμε επιτυχώς οποιεσδήποτε προκλήσεις προκύψουν.

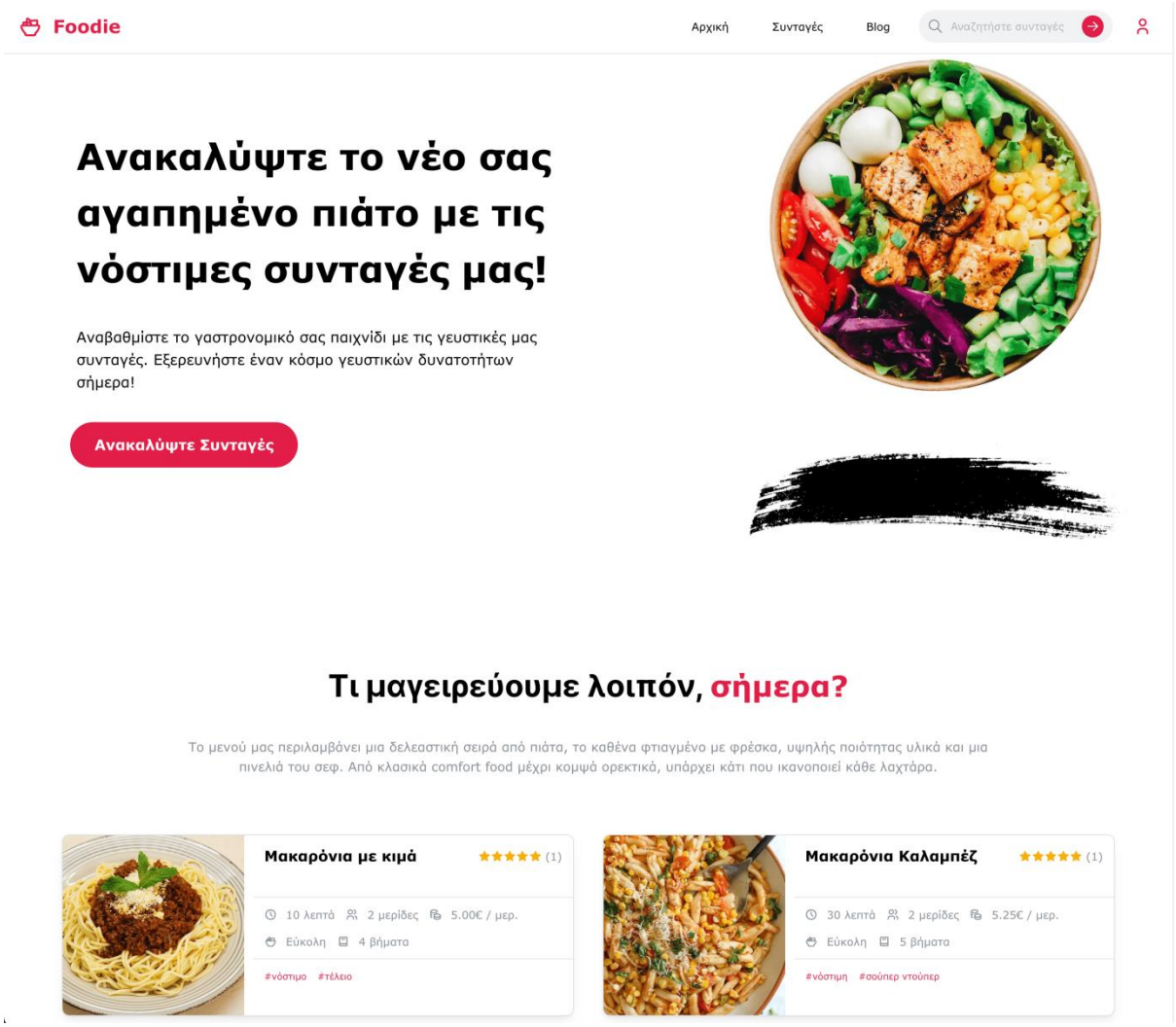
4.1 Αρχική σελίδα

Η αρχική σελίδα αποτελεί το πρώτο σημείο επαφής με τους χρήστες και είναι ένα από τα πιο σημαντικά σημεία της εφαρμογής μας. Εδώ οι χρήστες θα βρουν βασικές πληροφορίες, διαφημίσεις ή προσφορές και συνδέσμους για περαιτέρω πλοήγηση στην εφαρμογή. Η σχεδίαση και το περιεχόμενο της αρχικής σελίδας είναι κρίσιμα για να προσελκύσουμε το ενδιαφέρον των χρηστών και να τους καθοδηγήσουμε στο επόμενο βήμα της εμπειρίας τους στην πλατφόρμα μας. Μέσω της αρχικής σελίδας, προσπαθούμε να προβάλουμε το μήνυμα και τον σκοπό της εφαρμογής μας με έναν ελκυστικό και εύληπτο τρόπο, προσφέροντας ένα περιβάλλον που ενθαρρύνει την περαιτέρω εξερεύνηση και αλληλεπίδραση. Από τη σχεδίαση του layout μέχρι την επιλογή του περιεχομένου, η αρχική σελίδα παίζει έναν καθοριστικό ρόλο στη δημιουργία μιας ευχάριστης και λειτουργικής εμπειρίας χρήστη.

Στην αρχική μας σελίδα, ο χρήστης υποδέχεται μια φιλόξενη ατμόσφαιρα, όπου η εικόνα και το μήνυμα που παρουσιάζονται του επιτρέπουν να καταλάβει αμέσως τη φύση της εφαρμογής. Το στυλ και τα χρώματα που χρησιμοποιούνται αποπνέουν μια αίσθηση οικειότητας και ζεστασιάς, ενώ τα γραφικά και οι εικόνες επικεντρώνονται σε εικόνες φαγητού και εδεσμάτων, δημιουργώντας μια αμέσως αναγνωρίσιμη ατμόσφαιρα για τον χρήστη.

Τα φιλικά μηνύματα που παρουσιάζονται στην αρχική σελίδα σχεδιάζονται με γνώμονα την απλότητα και την κατανοητή γλώσσα, προσφέροντας στον χρήστη μια άμεση και φιλική επικοινωνία. Οι ελάχιστες πληροφορίες που φαίνονται στις κάρτες των συνταγών έχουν ως σκοπό να προσφέρουν μια αίσθηση ευχαρίστησης, ευκολίας και ανανέωσης στον χρήστη, δίνοντάς του την εντύπωση ότι βρίσκεται σε ένα φιλικό περιβάλλον. Μέσω αυτών των μηνυμάτων, ο χρήστης ενθαρρύνεται να εξερευνήσει την εφαρμογή μας και να ανακαλύψει τις δυνατότητές της με άνεση και ευχαρίστηση.

Σε μελλοντικό στάδιο της εφαρμογής θα μπορούσαμε να έχουμε προσωποποιημένες προτάσεις για κάθε χρήστη ανάλογα με τις συνταγές που του αρέσουν, τους λογαριασμούς που θα ακολουθεί ή ακόμα και τις πιθανές προτιμήσεις διατροφικών στοιχείων που θα έχει επιλέξει στις ρυθμίσεις του λογαριασμού του. Με αυτόν τον τρόπο η αλληλεπίδραση του χρήστη με την εφαρμογή μας θα είναι ακόμα πιο ενδιαφέρουσα καθώς θα έχει έναν παραπάνω λόγο να θέλει να την ανοίξει και να αφιερώσει χρόνο σε αυτή. Με αυτόν τον τρόπο η εικόνα του για την εφαρμογή μας θα βελτιωθεί και ενδεχομένως να τον κάνει να μοιραστεί την εμπειρία του με άλλους χρήστες, γεγονός που θα μεγαλώσει την κοινότητα και την δημοφιλία της εφαρμογής μας.



Σχήμα 4.1.1 Αρχική σελίδα

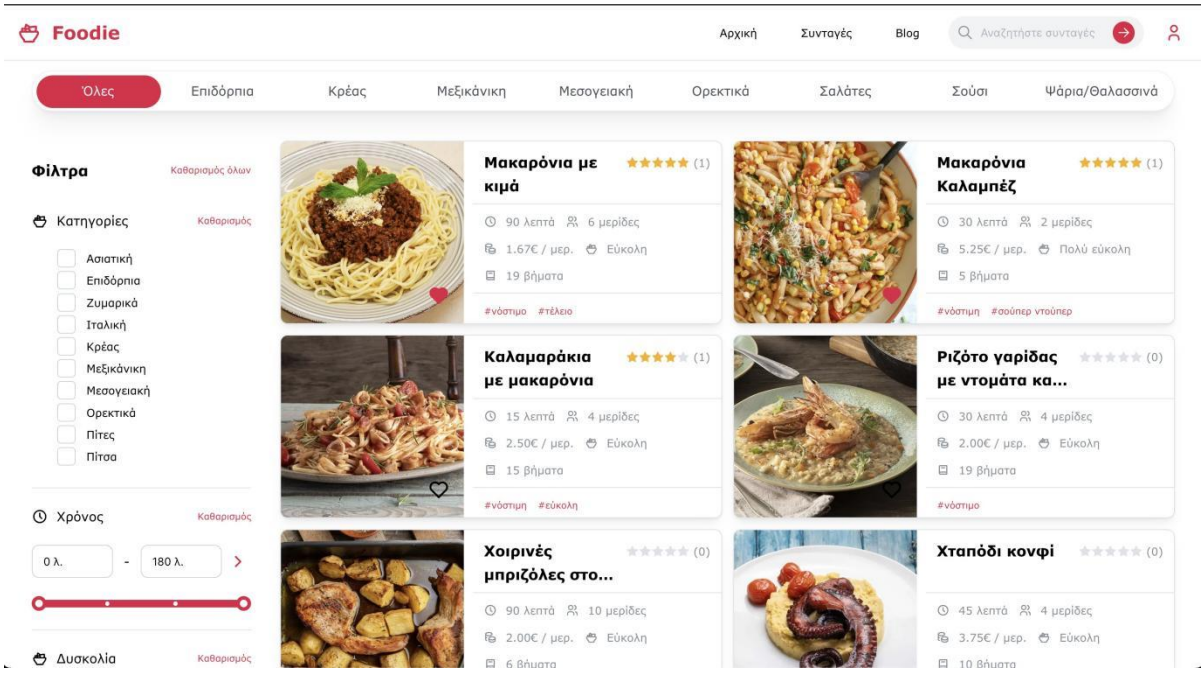
Πατώντας στο κουμπί “Ανακαλύψτε Συνταγές” ο χρήστης πλέον βρίσκεται στην σελίδα της κεντρικής ιδέας της εφαρμογής μας.

4.2 Σελίδα εύρεσης συνταγών

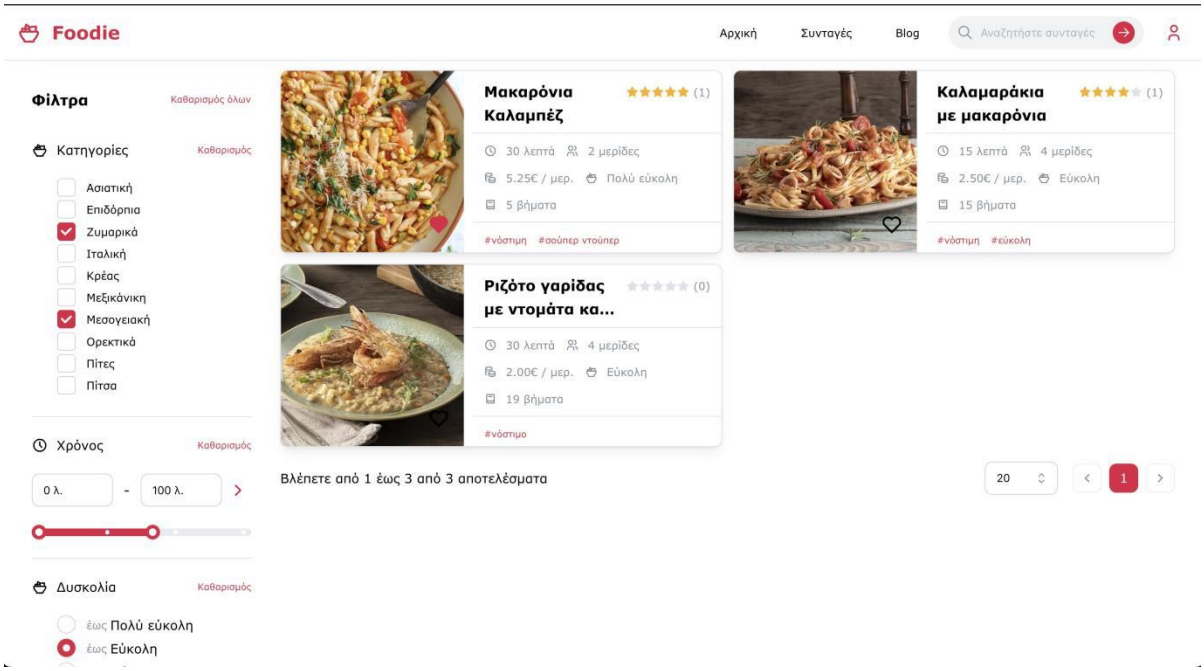
Η σελίδα εύρεσης συνταγών είναι από τις πιο σημαντικές της εφαρμογής μας, προσφέροντας στους χρήστες μια πλούσια εμπειρία αναζήτησης και εξερεύνησης για να ανακαλύψουν νέες και ενδιαφέρουσες συνταγές. Μέσω μιας εύχρηστης διεπαφής και μιας ποικιλίας φίλτρων, οι χρήστες μπορούν να περιηγηθούν σε μια μεγάλη συλλογή συνταγών, να αναζητήσουν συγκεκριμένα είδη φαγητού ή συστατικά, και να απολαύσουν πλούσιο περιεχόμενο που τους εμπνέει για την επόμενη μαγειρική τους περιπέτεια.

Η σελίδα προσφέρει επίσης εξειδικευμένες λειτουργίες, όπως τη δυνατότητα αποθήκευσης συνταγών στη λίστα αγαπημένων προσφέροντας έτσι στους χρήστες μια αληθινή διαδραστική εμπειρία κοινότητας. Με τη συνεχή ενημέρωση του περιεχομένου και την προσθήκη νέων συνταγών από τους

χρήστες και τους διαχειριστές, η σελίδα εύρεσης συνταγών παρέχει μια ατελείωτη πηγή έμπνευσης και δημιουργικότητας σε κάθε φανατικό του μαγειρέματος.



Σχήμα 4.2.2 Σελίδα συνταγών



Σχήμα 4.2.2 Αποτελέσματα φίλτρων συνταγών

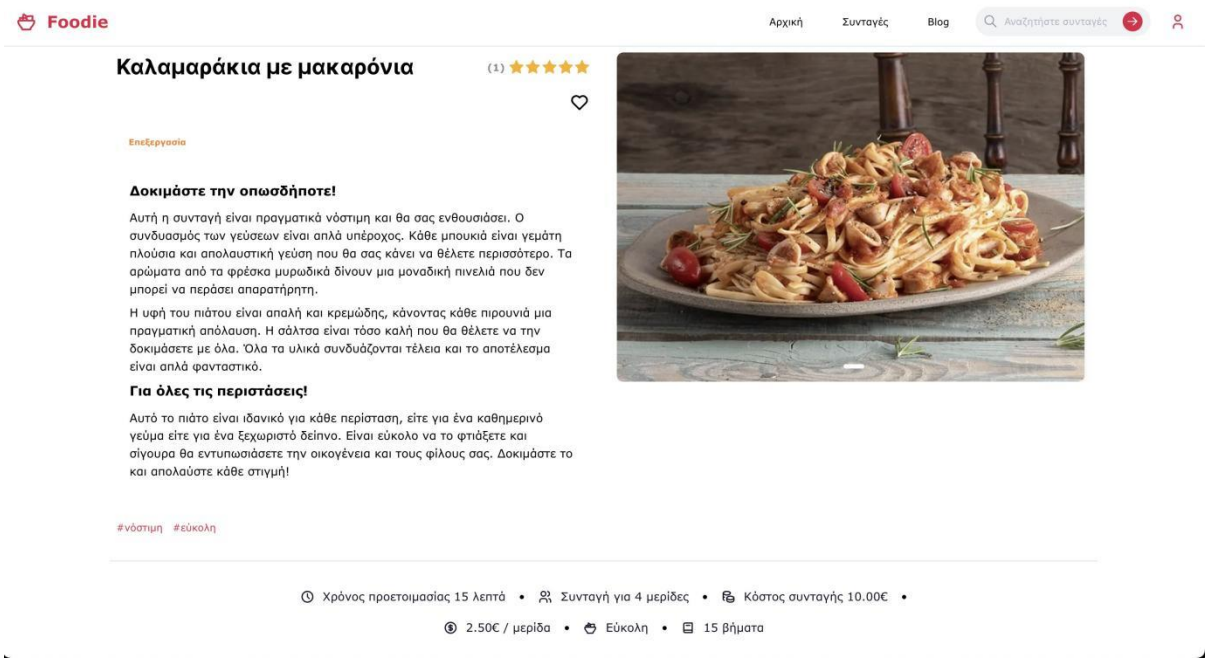
4.3 Σελίδα συνταγών

Η σελίδα μιας συνταγής αποτελεί τον πυρήνα της εφαρμογής μας, παρέχοντας στους χρήστες μια λεπτομερή προβολή και κατανόηση μιας συγκεκριμένης συνταγής. Καθώς ο χρήστης ανακαλύπτει μια συνταγή, προσφέρουμε μια εύχρηστη διεπαφή που περιλαμβάνει τις απαραίτητες πληροφορίες για τα συστατικά, τις οδηγίες μαγειρέματος και οποιεσδήποτε επιπλέον πληροφορίες είναι απαραίτητες, όπως χρόνος προετοιμασίας, δυσκολία εκτέλεσης και διατροφικές πληροφορίες.

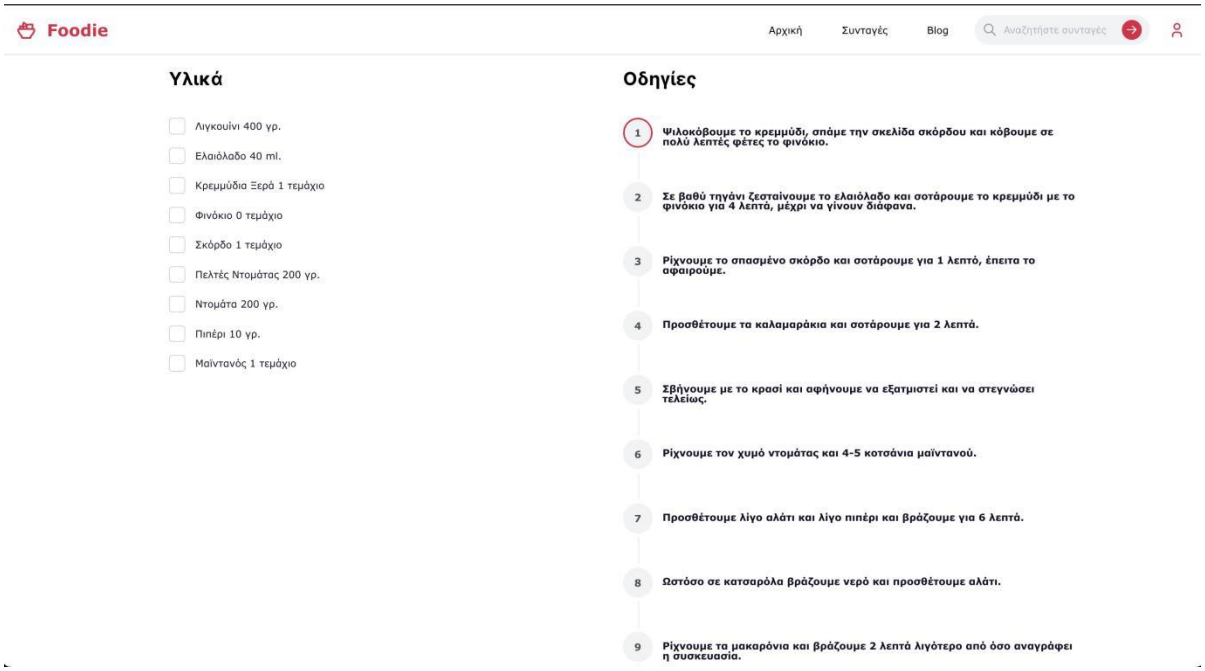
Η σελίδα παρέχει επίσης προαιρετικές λειτουργίες, όπως αξιολόγηση και σχολιασμός της συνταγής από τους χρήστες, καθώς και τη δυνατότητα αποθήκευσης στη λίστα αγαπημένων. Επίσης, παρέχουμε εναλλακτικές εκδόσεις της συνταγής για διαφορετικές διατροφικές προτιμήσεις ή διατροφικά περιορισμένους χρήστες.

Επιπλέον, έχουμε ενσωματώσει διαδραστικά στοιχεία, όπως βίντεο μαγειρέματος και συνδέσμους προς παρόμοιες συνταγές, για να εμπλουτίσουμε την εμπειρία του χρήστη. Όλα αυτά είναι διαθέσιμα στον χρήστη μέσω του Rich Text Editor που υπάρχει στη σελίδα δημιουργίας συνταγών. Αυτά τα στοιχεία όχι μόνο βοηθούν τους χρήστες να κατανοήσουν καλύτερα τη διαδικασία μαγειρέματος, αλλά και να ανακαλύψουν νέες τεχνικές και ιδέες που μπορούν να εφαρμόσουν στις δικές τους δημιουργίες. Επιπρόσθετα, οι χρήστες έχουν τη δυνατότητα να μοιραστούν τις δικές τους παραλλαγές και συμβουλές μέσω των σχολίων, δημιουργώντας μια δυναμική κοινότητα μαγείρων που μαθαίνουν και εμπνέονται ο ένας από τον άλλον.

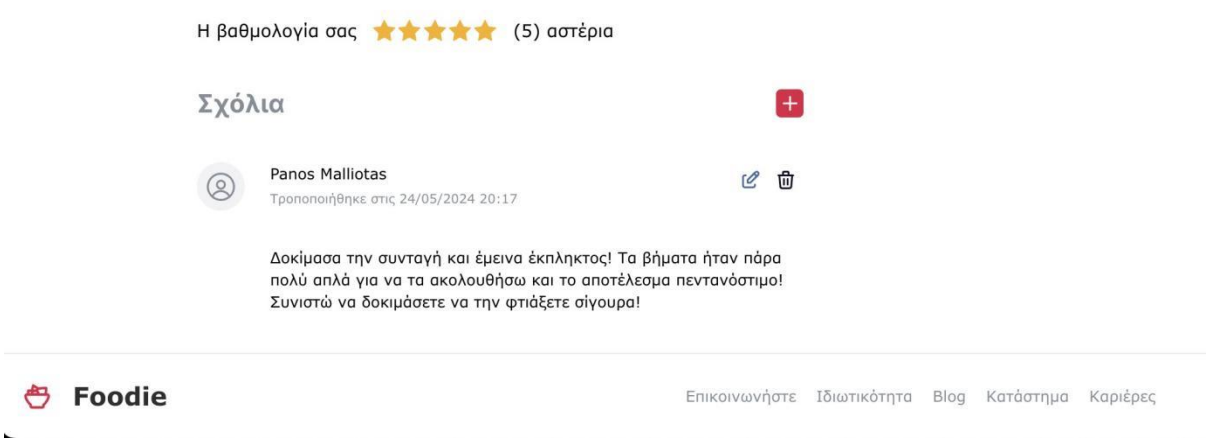
Με λεπτομερείς φωτογραφίες και πρακτικές συμβουλές, η σελίδα μιας συνταγής προσφέρει στους χρήστες μια ολοκληρωμένη εμπειρία μαγειρέματος που τους εμπνέει να δοκιμάσουν και να δημιουργήσουν νέα γεύσεις και γευστικές εμπειρίες.



Σχήμα 4.3.1 Σελίδα συνταγής (περιγραφή)



Σχήμα 4.3.2 Σελίδα συνταγής (βήματα & υλικά)



Σχήμα 4.3.3 Σελίδα συνταγής (σχόλια χρηστών & βαθμολογίες)

Τέλος, η δυνατότητα των χρηστών να προσθέτουν σχόλια και κριτικές στις συνταγές ενθαρρύνει τη συνεχή βελτίωση και την προσαρμογή των συνταγών στις ανάγκες της κοινότητας. Αυτή η αλληλεπίδραση προσφέρει πολύτιμα σχόλια στους δημιουργούς των συνταγών, επιτρέποντάς τους να βελτιώσουν τις προτάσεις τους και να προσφέρουν ακόμα πιο ποιοτικές και δοκιμασμένες λύσεις στους υπόλοιπους χρήστες. Η κοινή χρήση φωτογραφιών από τα τελικά αποτελέσματα των συνταγών προσθέτει μια οπτική διάσταση στην εμπειρία, κάνοντας την πλατφόρμα πιο ζωντανή και ελκυστική. Οι χρήστες μπορούν να συμμετέχουν σε συζητήσεις, να ζητούν συμβουλές και να μοιράζονται τις δικές τους εμπειρίες, δημιουργώντας έναν χώρο αμοιβαίας υποστήριξης και έμπνευσης.

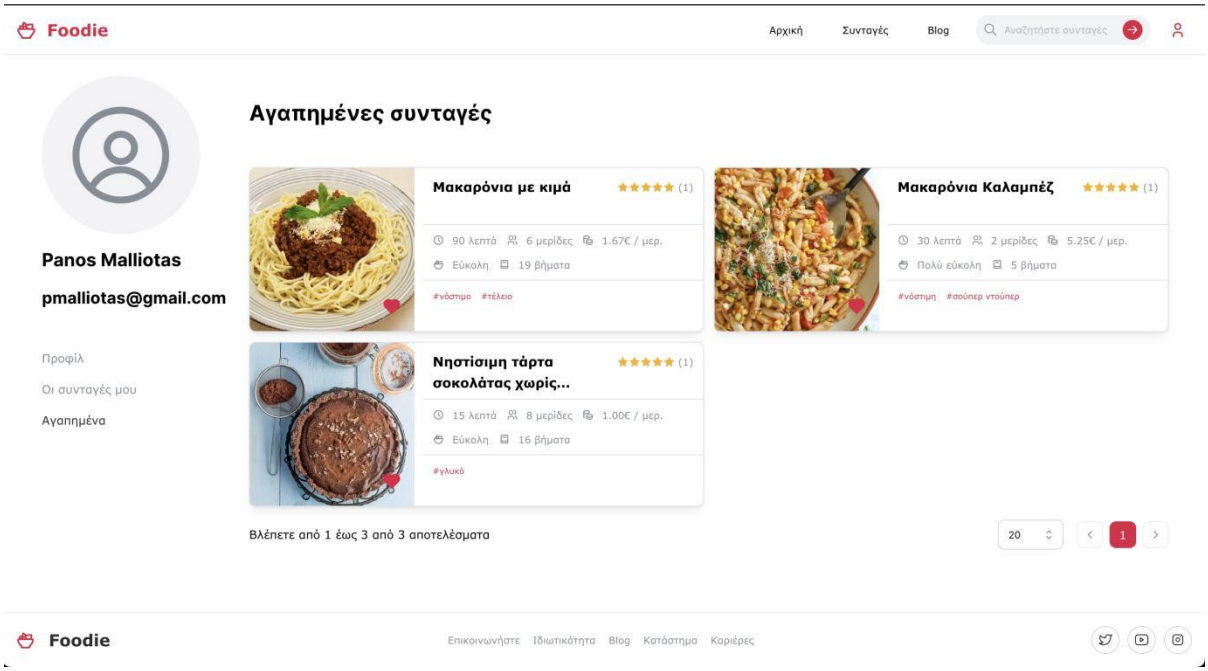
Μέσα από αυτήν την πλούσια και διαδραστική εμπειρία, οι χρήστες της εφαρμογής μας δεν περιορίζονται απλά στην ανακάλυψη και εκτέλεση συνταγών, αλλά γίνονται μέρος μιας ζωντανής και δημιουργικής κοινότητας. Η εφαρμογή μας προσφέρει ένα περιβάλλον όπου η αγάπη για τη

μαγειρική ενισχύεται και μοιράζεται, προάγοντας την καινοτομία και τη συνεχή ανακάλυψη νέων γευστικών εμπειριών.

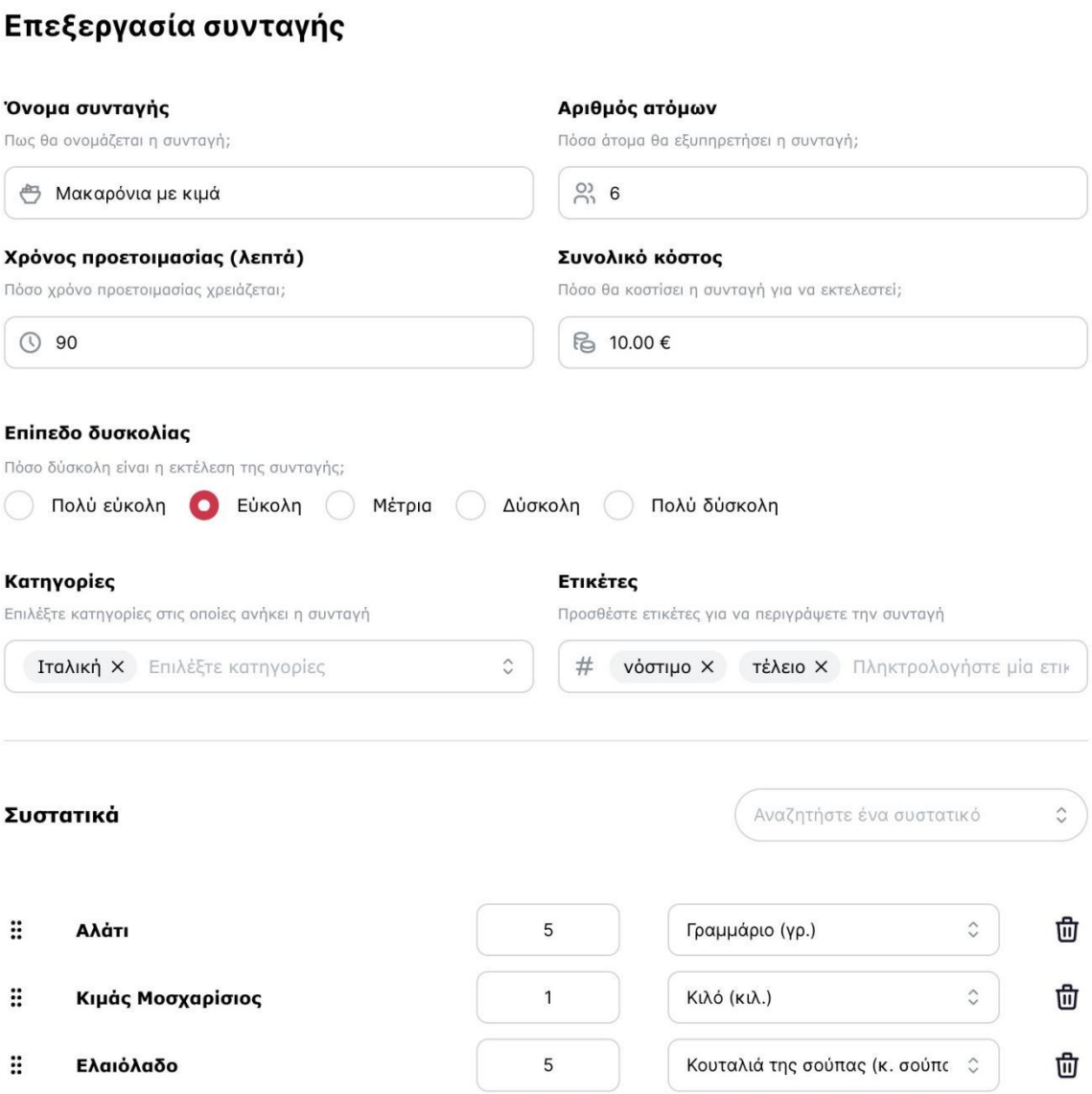
4.4 Σελίδα προφίλ χρήστη

Στη σελίδα προφίλ του χρήστη, ο κάθε χρήστης μπορεί να διαχειριστεί το προφίλ του με ευκολία και να έχει πρόσβαση σε πληροφορίες σχετικά με τις αγαπημένες του συνταγές καθώς και αυτές που έχει δημιουργήσει. Η σελίδα παρέχει μια επισκόπηση των αγαπημένων συνταγών του χρήστη, επιτρέποντάς του να περιηγηθεί στις αγαπημένες του γεύσεις και να ανακαλύψει νέες ιδέες για το μαγείρεμα.

Επιπλέον, ο χρήστης έχει τη δυνατότητα να δει όλες τις συνταγές που έχει δημιουργήσει μέσω μιας εύχρηστης και φιλικής προς τον χρήστη διεπαφής. Η πρόσβαση σε αυτές τις συνταγές επιτρέπει την ανασκόπηση και την αξιολόγηση των παλαιότερων δημιουργιών, παρέχοντας έτσι μια πλήρη εικόνα της μαγειρικής του δραστηριότητας. Οι συνταγές εμφανίζονται οργανωμένες και εύκολα προσβάσιμες, διευκολύνοντας τον χρήστη στην αναζήτηση και την ανάκτηση των επιθυμητών συνταγών. Μπορεί να επεξεργαστεί τις υπάρχουσες συνταγές ή να προσθέσει νέες σύμφωνα με τις ανάγκες και τις προτιμήσεις του. Αυτή η δυνατότητα παρέχει ευελιξία και προσαρμοστικότητα, επιτρέποντας στον χρήστη να βελτιώσει τις συνταγές του, να δοκιμάσει νέους συνδυασμούς υλικών και να εξερευνήσει διαφορετικές γαστρονομικές προσεγγίσεις. Η διαδικασία επεξεργασίας και προσθήκης νέων συνταγών είναι απλή και διασκεδαστική, καθιστώντας την μαγειρική εμπειρία ακόμα πιο ευχάριστη.



Σχήμα 4.4.1 Σελίδα αγαπημένων συνταγών χρήστη



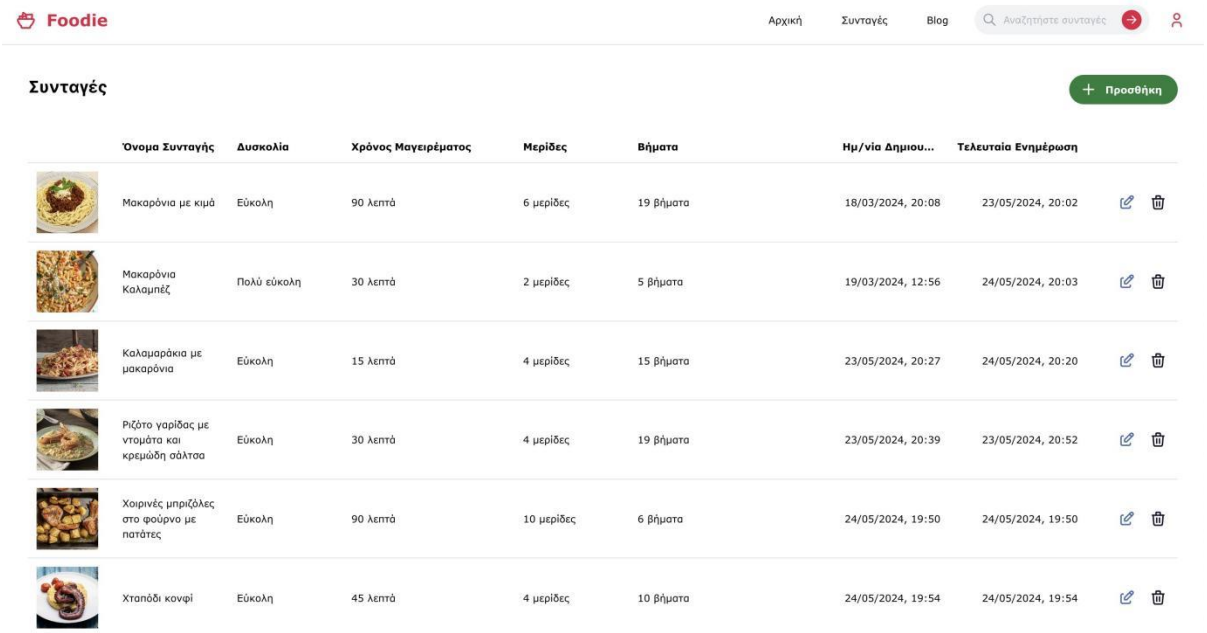
Σχήμα 4.4.2 Σελίδα δημιουργίας-επεξεργασίας συνταγής

4.5 Σελίδα διαχείρισης συνταγών διαχειριστή

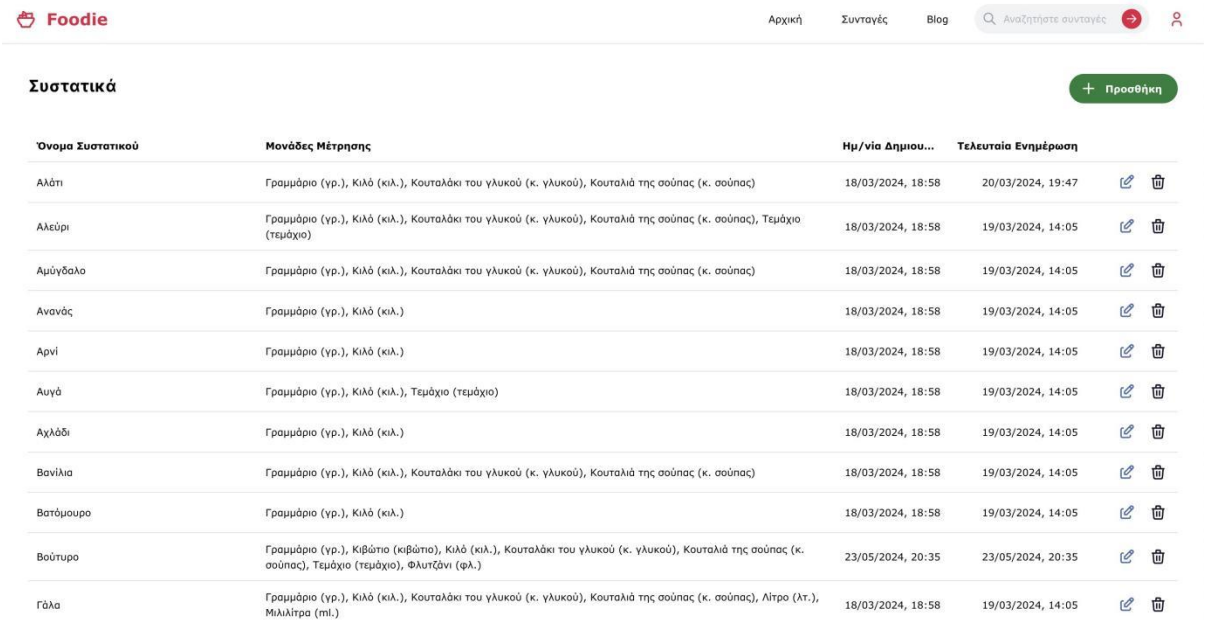
Η σελίδα διαχείρισης συνταγών του διαχειριστή αποτελεί ένα κρίσιμο τμήμα της εφαρμογής, προσφέροντας στο διαχειριστή τη δυνατότητα να διαχειρίζεται το περιεχόμενο των συνταγών με αποτελεσματικότητα και ευκολία.

Μέσω αυτής της σελίδας, ο διαχειριστής μπορεί να προσθέσει, να επεξεργαστεί ή να διαγράψει συνταγές, ενημερώνοντας τον κατάλογο των διαθέσιμων συνταγών της εφαρμογής. Να σημειωθεί ότι ο διαχειριστής μπορεί να επεξεργαστεί και συνταγές χρηστών αν το κρίνει απαραίτητο (όπως για παράδειγμά αν ο χρήστης έχει γράψει προσβλητικά ή ακατάλληλα σχόλια μέσα στην περιγραφή της συνταγής του). Επιπλέον, μπορεί να διαχειριστεί τις κατηγορίες των συνταγών και να προσθέσει νέες κατηγορίες ή να επεξεργαστεί τις υπάρχουσες, να επεξεργαστεί τις διαθέσιμες μονάδες μέτρησης και τα υλικά προσαρμόζοντας έτσι τη δομή της εφαρμογής στις ανάγκες και τις προτιμήσεις των χρηστών.

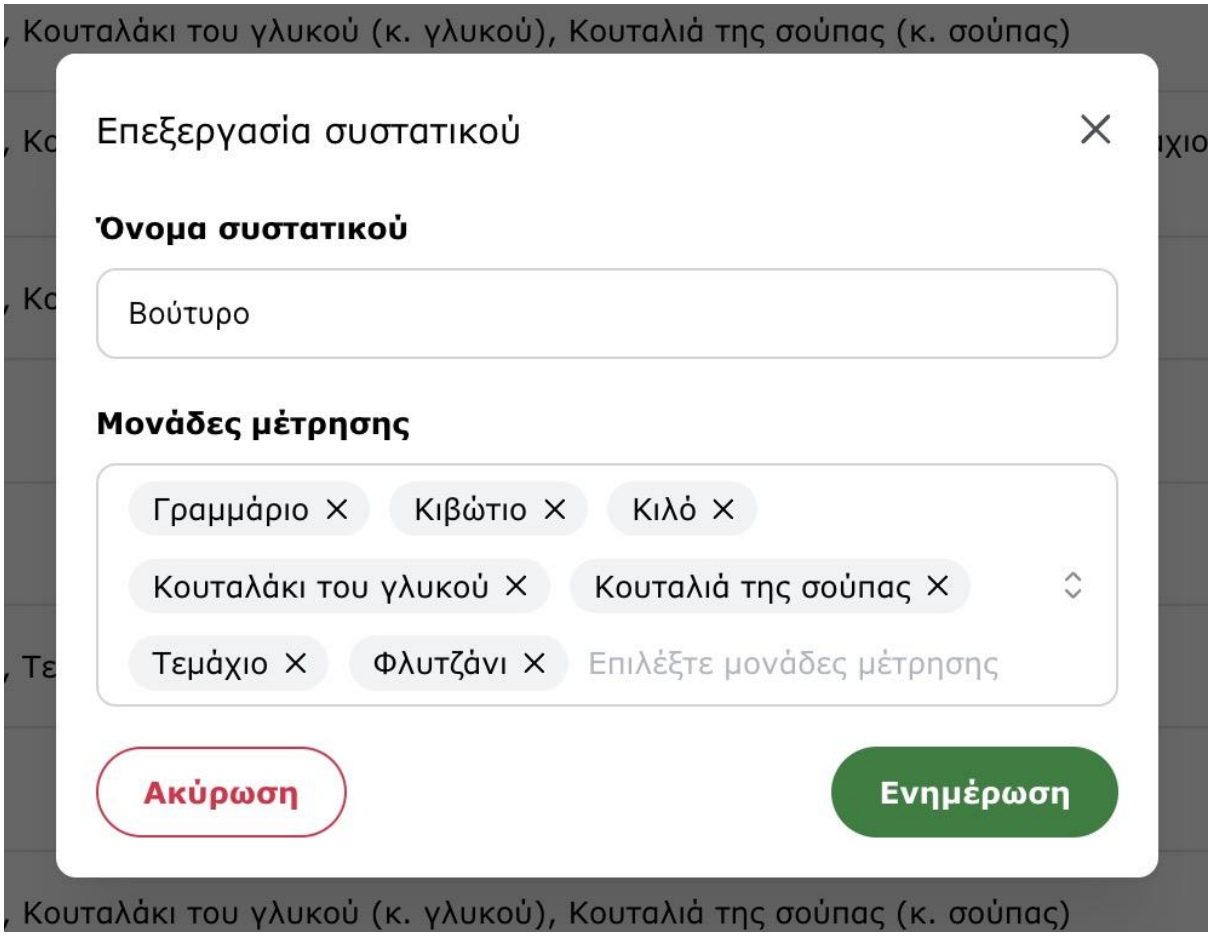
Η σελίδα αυτή προσφέρει στον διαχειριστή ένα εύχρηστο και ισχυρό εργαλείο για τη διαχείριση του περιεχομένου της εφαρμογής, ενισχύοντας έτσι την εμπειρία των χρηστών και βελτιστοποιώντας τη λειτουργικότητα της εφαρμογής στο σύνολό της.



Σχήμα 4.5.1 Σελίδα συνταγών διαχειριστή



Σχήμα 4.5.2 Σελίδα συστατικών διαχειριστή



Σχήμα 4.5.3 Modal διαχείρισης συστατικών διαχειριστή

Κεφάλαιο 5ο: Συμπεράσματα και μελλοντικές βελτιώσεις

Στο πλαίσιο της παρούσας πτυχιακής εργασίας, εστιάσαμε σε μια πλήρη και συνεκτική διαδικασία ανάπτυξης λογισμικού, ξεκινώντας από την αρχική έρευνα και ανάλυση των αναγκών του χρήστη. Μέσω της συλλογής δεδομένων και της ανάλυσης των απαιτήσεων, αποσαφηνίσαμε τους στόχους και τις προσδοκίες που καθορίζουν το πλαίσιο της εφαρμογής μας. Ακολούθησε η φάση του σχεδιασμού, κατά την οποία δημιουργήσαμε τα λειτουργικά και αισθητικά στοιχεία της εφαρμογής, λαμβάνοντας υπόψη την ευχρηστία και την εμπειρία του χρήστη. Στο επόμενο στάδιο, μεταφέραμε τον σχεδιασμό στην πράξη, χρησιμοποιώντας τις κατάλληλες τεχνολογίες και μεθόδους ανάπτυξης, προκειμένου να δημιουργήσουμε μια λειτουργική και αποτελεσματική εφαρμογή που ανταποκρίνεται στις ανάγκες των χρηστών.

Αναλύσαμε λεπτομερώς τις τεχνολογίες που χρησιμοποιήσαμε και του λόγους που επιλέχθηκαν όλες αυτές σε συνδυασμό μεταξύ τους, καθώς και τα πλεονεκτήματα που προσφέρουν σε σχέση με άλλες αντίστοιχες τεχνολογίες-βιβλιοθήκες. Στην ανάπτυξη λογισμικού είναι πολύ σημαντικό να κατανοούμε τα εργαλεία που χρησιμοποιούμε και τους λόγους για τους οποίους τα έχουμε επιλέξει. Πολλές φορές οι προγραμματιστές χρησιμοποιούμε τεχνολογίες επειδή είναι γνωστές ενώ υπάρχουν επιλογές που πιθανώς ταιριάζουν περισσότερο στην ανάγκες μας. Μέσω αυτής της πτυχιακής μελετήσαμε έναν τρόπο σκέψης για την ανάπτυξη εφαρμογών, προσπαθώντας να κάνουμε κατανοητά

τα βασικά βήματα τα οποία είναι απαραίτητα για την υλοποίηση οποιαδήποτε εφαρμογής. Οι τεχνολογίες που χρησιμοποιήσαμε δεν είναι οι ”σωστές” για κάθε περίπτωση, αλλά η προσέγγιση στην επιλογή τους που παρουσιάστηκε σε αυτή την πτυχιακή είναι αρκετά καλή για να αποτελέσει ένα παράδειγμα για την επίλυση προβλημάτων άλλων προγραμματιστών στο μέλλον.

Μέσα από αυτήν την διαδικασία, αναδείξαμε τη σημασία της ενσωμάτωσης της θεωρητικής κατανόησης με την πρακτική εφαρμογή, δημιουργώντας έτσι μια εφαρμογή που ανταποκρίνεται στις πραγματικές ανάγκες των χρηστών και προσφέρει μια ολοκληρωμένη και ικανοποιητική εμπειρία χρήστη. Πέρα από την υλοποίηση της βασικής ιδέας τη εφαρμογής η οποία είναι να αναζητά συνταγές με τα διαθέσιμα υλικά που έχει, δώσαμε στον χρήστη μερικές λειτουργίες ώστε η αλληλεπίδρασή του με την εφαρμογή μας να είναι πολύ εύκολη και ευχάριστη. Μερικές από αυτές τις λειτουργίες είναι η εύκολη σύνδεση με λογαριασμούς Google και Github, η εύκολη και λεπτομερής αναζήτηση συνταγών, η δυνατότητα αποθήκευσης συνταγών ως αγαπημένων, η δυνατότητα δημιουργίας και επεξεργασίας συνταγών, η δυνατότητα σχολιασμού άλλων συνταγών καθώς και για τους πιθανούς μελλοντικούς διαχειριστές, μια πολύ εύκολη εμπειρίας διαχείρισης της πλατφόρμας.

Στο μέλλον η εφαρμογή μπορεί να επεκταθεί με πρόσθετες λειτουργίες που θα βελτιώσουν την εμπειρία χρηστών και θα ενισχύσουν την αξία της εφαρμογής. Μερικές από αυτές τις λειτουργίες είναι δυνατότητα αντίδρασης με emoji σε συνταγές ή σχόλια, το ανέβασμα βίντεο σε κάθε συνταγή, η δημιουργία native εφαρμογών για την υποστήριξη των περιβαλλόντων Android και iOS, η δημιουργία διαφορετικών τύπων προφίλ όπως επαγγελματίας μάγειρας, ερασιτέχνης και η αξιολόγηση αυτών των προφίλ. Επίσης θα μπορούσαμε να δώσουμε την δυνατότητα σε χρήστες να ακολουθούν άλλους λογαριασμούς, όπως επίσης να επιλέγουν στις ρυθμίσεις του λογαριασμού τους υλικά στα οποία έχουν αλλεργία ώστε να μην εμφανίζονται αποτελέσματα με αυτά ή να μαρκάρονται οι συνταγές που τα περιέχουν ως “Επίφοβες λόγω αλλεργίας”.

Ως η “καλύτερη δυνατή” μελλοντική επέκταση οφείλεται να αναφερθεί ότι είναι η αλλαγή μερικών βασικών τεχνολογιών από αυτές που χρησιμοποιήσαμε όπως για παράδειγμα το TRPC και το Prisma. Όπως αναφέραμε πριν από λίγο η επιλογή των κατάλληλων τεχνολογιών για κάθε εφαρμογή είναι η αρχή και το τέλος της σωστής ανάπτυξης λογισμικού. Οι τεχνολογίες TRPC και Prisma όσο εύχρηστες, ασφαλείς και παραγωγικές και αν είναι, έχουν ένα μειονέκτημα. Σε εφαρμογές με μεγάλο όγκο χρηστών μένουν λίγο πίσω σε απόδοση σε σχέση με μια υλοποίηση ενός Backend με SQL και GoLang για παράδειγμα. Στα πλαίσια αυτής της πτυχιακής δεν θα είναι ποτέ πρόβλημα για να διορθωθεί. Αν όμως η εφαρμογή αποκτήσει μια πάρα πολύ μεγάλη κοινότητα χαρούμενων χρηστών, θα αναγκαστούμε να υλοποιήσουμε αυτή την “καλύτερη δυνατή” επέκταση.

ΒΙΒΛΙΟΓΡΑΦΊΑ

[1] Visual Studio Code ,Wikipedia. Available: https://en.wikipedia.org/wiki/Visual_Studio_Code

[2] Electron, What is Electron? Available: https://www.electronjs.org/docs/latest/

[3] Microsoft, Wikipedia. Available: https://el.wikipedia.org/wiki/Microsoft

[4] Typescript, Typescript for the new Programmer. Available: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html

[5] Modularity, Modular programming: beyond the spaghetti mess. Available: https://www.tiny.cloud/blog/modular-programming-principle/
[6] Type Safety, Wikipedia. Available: https://en.wikipedia.org/wiki/Type_safety

[7] Node.Js, Introducton to Node.Js. Available: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

[8] Express.Js, What is Express.Js? Available: https://www.codecademy.com/article/what-is-express-js

[9] REST, Wikipedia. Available: https://en.wikipedia.org/wiki/REST

[10] Multipage, Single page vs Multi page application: What is better for your project? Available: https://www.linkedin.com/pulse/single-page-application-vs-multi-page-what-better-your-/

[11] Single Page, Single page vs Multi page application: What is better for your project? Available: https://www.linkedin.com/pulse/single-page-application-vs-multi-page-what-better-your-/

[12] React, Quick start. Available: https://react.dev/learn

[13] Meta, Who are we? Available: https://about.meta.com/company-info/

[14] Component based software engineering, Wikipedia. Available: https://en.wikipedia.org/wiki/Component-based_software_engineering

[15] Virtual DOM, Virtual DOM and internals. Available: https://legacy.reactjs.org/docs/faq-internals.html

[16] Bundle, Wikipedia. Available: https://en.wikipedia.org/wiki/Product_bundling#Software

[17] Next.Js, What is Next.js? Available: https://nextjs.org/docs

[18] Server Side Rendering, What is server-side rendering: definition, benefits and risks. Available: https://solutionshub.epam.com/blog/post/what-is-server-side-rendering

[19] Static Site Generation, Static Site Generation (SSG). Available: https://nextjs.org/docs/pages/building-your-application/rendering/static-site-generation

[20] Automatic Code Splitting, Enhancing Web Performance with Automatic Code Splitting in Next.js. Available: https://www.linkedin.com/pulse/enhancing-web-performance-automatic-code-splitting-nextjs-bin-tariq-p0zmf/

[21] Next Auth, Next Auth Introduction. Available: https://next-auth.js.org/getting-started/introduction

[22] State Management, What is state management? Available: https://www.techtarget.com/searchapparchitecture/definition/state-management

[23] Middleware, What is middleware? Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-middleware

[24] Zod. Available: https://zod.dev/?id=introduction

[25] DayJs. Available: https://day.js.org/en/

[26] Zero Configuration, Wikipedia. Available: https://en.wikipedia.org/wiki/Zero-configuration_networking

[27] TRPC, Introduction. Available: https://trpc.io/docs

[28] Prisma, What is Prisma ORM? Available: https://www.prisma.io/docs/orm/overview/introduction/what-is-prisma

[29] Router, What is Routing in software engineering? Available: https://divpusher.com/glossary/routing/

[30] Mutation. Available: https://github.com/prisma/prisma1/blob/master/docs/1.1/04-Reference/03-Prisma-API/04-Mutations.md

[31] Seeders, Seeding. Available: https://www.prisma.io/docs/orm/prisma-migrate/workflows/seeding

[32] Promises. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

[33] Diffing rendering, Demystifying Re-rendering, Diffing, and Reconciliation in React. Available: https://medium.com/@islamghany3/demystifying-re-rendering-diffing-and-reconciliation-in-react-3a821ebc36c9

[34] React Router Dom, What is react-router-dom? Available: https://www.geeksforgeeks.org/what-is-react-router-dom/

[35] Hooks, Hooks at a glance. Available: https://legacy.reactjs.org/docs/hooks-overview.html