ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

“Διαδικτυακή εφαρμογή παραγγελίας προϊόντων που δεν έχουν πωληθεί μέχρι το τέλος της ημέρας από επιχειρήσεις”

Του φοιτητή
Παπαϊωάννου Μιχαήλ
Αρ. Μητρώου: 516178

Επιβλέπων
Ουγιάρογλου Στέφανος, Επίκουρος
Καθηγητής

26 Φεβρουαρίου 2023

Τίτλος Δ.Ε.: Διαδικτυακή εφαρμογή παραγγελίας προϊόντων που δεν έχουν πωληθεί μέχρι το τέλος της ημέρας από επιχειρήσεις
Κωδικός Δ.Ε. 22308
Ονοματεπώνυμο φοιτητή/ών: Παπαϊωάννου Μιχαήλ
Ονοματεπώνυμο εισηγητή: Παπαϊωάννου Μιχαήλ
Ημερομηνία ανάληψης Δ.Ε.: 29-10-2022
Ημερομηνία περάτωσης Δ.Ε.: 29-01-2023

Πρόλογος

Η παρούσα εργασία συντάχθηκε από τον φοιτητή Παπαϊωάννου Μιχαήλ υπό την εποπτεία του κ. Στέφανου Ουγιάρογλου. Αφορά την δημιουργία μιας διαδικτυακής εφαρμογής, η οποία επιτρέπει στους χρήστες την αγορά προϊόντων που δεν έχουν πωληθεί μέχρι το τέλος της ημέρας από διάφορες επιχειρήσεις. Η παρούσα εφαρμογή περιλαμβάνει μία ιστοσελίδα, η οποία παρέχει την δυνατότητα στις επιχειρήσεις να κάνουν Upload τα προϊόντα τα οποία δεν έχουν πωληθεί στο τέλος της εργάσιμης ημέρας. Και ένα Mobile Application, το οποίο είναι συμβατό τόσο με Android όσο και με iOS συσκευές, μέσω του οποίου οι χρήστες έχουν την δυνατότητα να παραγγείλουν τα συγκεκριμένα προϊόντα. Η διαδικτυακή εφαρμογή έχει υλοποιηθεί στο περιβάλλον του Visual Studio Code, χρησιμοποιώντας το Flutter και Laravel Framework. Στόχος μας, είναι η δημιουργία ενός μέσου από το οποίο θα μειώσουμε την σπατάλη φαγητού και θα παρέχουμε πρόσβαση σε πιθανούς καταναλωτές, οι οποίοι θα έχουν την ευκαιρία να αγοράσουν καλά προϊόντα σε πολύ καλύτερες τιμές. Παράλληλα θα δώσουμε την δυνατότητα στις επιχειρήσεις να κάνουν μεγιστοποίηση του κέρδους, ενώ ταυτόχρονα μειώνοντας τον αριθμό των τροφίμων που θα πεταχτούν. Από όλα τα τρόφιμα που παράγονται στον κόσμο κάθε χρόνο, σχεδόν το ένα τρίτο χάνεται ή σπαταλιέται σύμφωνα με εκτιμήσεις από τον Οργανισμό Τροφίμων και Γεωργίας των Ηνωμένων Εθνών. Ένα μεγάλο ποσοστό του παγκόσμιου φαγητού χάνεται μεταξύ της συγκομιδής και της λιανικής πώλησης, και πολύ περισσότερο σπαταλιέται στο λιανικό εμπόριο και σε επίπεδο καταναλωτή. Η απώλεια και η σπατάλη τροφίμων έχει σαν συνέπεια την επιδείνωση της κλιματικής αλλαγής. Όταν τα τρόφιμα καταλήγουν σε χωματερές, παράγουν μεθάνιο, το οποίο ενισχύει σημαντικά το φαινόμενο του θερμοκηπίου. Ελπίζουμε πως μέσω αυτής της διπλωματικής εργασίας, εγώ μαζί με τον κ. Ουγιάρογλου θα συνεισφέρουμε ένα μικρό λιθαράκι προς την επίλυση αυτού του μεγάλου προβλήματος.

Περίληψη

Στην συγκεκριμένη διπλωματική εργασία, θα κάνουμε αναφορά στα περιβαλλοντικά και οικονομικά προβλήματα της σπατάλης τροφίμων, καθώς και σε κάποιες εφαρμογές, οι οποίες βοηθούν στην καταπολέμηση αυτών των προβλημάτων. Στην συνέχεια θα κάνουμε αναφορά στις τεχνολογίες που χρησιμοποιήθηκαν για την δημιουργία της εφαρμογής, όπως το Flutter Framework το οποίο έχει χρησιμοποιηθεί για την δημιουργία του Mobile Application, και το laravel framework το οποίο μας βοήθησε στην δημιουργία του Backend και της Βάσης Δεδομένων. Επίσης θα γίνει ανάλυση με διαγράμματα και παραδείγματα κώδικα ο τρόπος με τον οποίο έχουν υλοποιηθεί οι αρχιτεκτονικές της εφαρμογής, όπως μια παραλλαγή της αρχιτεκτονικής MVC, η οποία έχει εφαρμοστεί για την σχεδίαση και υλοποίηση του Backend, και η αρχιτεκτονική Controller-Repository-API που βοήθησε στην υλοποίηση του Mobile Application. Έπειτα θα γίνει εκτενής ανάλυση του κώδικα των αρχείων που είχαν καθοριστικό ρόλο για την επίτευξη της μεταφοράς των δεδομένων από το FrontEnd στον server. Στην συνέχεια θα γίνει παρουσίαση ενός σεναρίου χρήσης της εφαρμογής από την προοπτική του πελάτη, και ενός σεναρίου χρήσης από την προοπτική της επιχείρησης, ενώ παράλληλα θα γίνει αναφορά σε όλα τα απαραίτητα βήματα που πρέπει να κάνει ο πελάτης και η επιχείρηση για να επιτευχθεί η λειτουργία του όλου συστήματος. Τέλος, στα πλαίσια της παρούσας διπλωματικής εργασίας θα αναφέρουμε τα συμπεράσματα καθώς και κάποιες βελτιστοποιήσεις που μπορούν να γίνουν στο μέλλον οι οποίες θα συνεισφέρουν στην μεγαλύτερη επιτυχία της εφαρμογής σε εμπορική χρήση.

«Web Application for ordering products that have not been sold by the end of the day from businesses»

«Michail Papaioannou»

Abstract

This thesis was written by the student Papaioannou Michael under the supervision of Mr. Stefanos Ouiaroglou. This project concerns the creation of an online application, which allows users to purchase products that have not been sold by the end of the day from various businesses. This application includes, a web page, which provides the possibility for businesses to upload the products that have not been sold at the end of the working day. And a Mobile Application, which is compatible with both Android and iOS devices, through which users have the possibility to order the specific products. The web application has been implemented in the Visual Studio Code environment, using Flutter and Laravel Frameworks. This thesis aims to create an Application that will reduce food waste and provide access to potential consumers, who will have the opportunity to purchase good products at much better prices. At the same time, we will enable businesses to maximize profit, while at the same time reducing the amount of food that will be thrown away. Of all the food produced in the world each year, nearly a third is lost or wasted according to estimates by the Food and Agriculture Organization of the United Nations. A large percentage of the world's food is lost between harvest and retail, and much more is wasted at the consumer level. The loss and waste of food results in the worsening of climate change. When food ends up in landfills, it produces methane, which greatly enhances the greenhouse effect. We hope that through this thesis, I, together with Mr. Uyaroglou, will contribute towards the solution of this big problem.

Ευχαριστίες

Ξεκινώντας, θα ήθελα να ευχαριστήσω τον κ. Στέφανο Ουγιάρογλου, Επίκουρο καθηγητή του τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνές Πανεπιστημίου της Ελλάδος, ο οποίος είναι ο επιβλέπων καθηγητής της συγκεκριμένης διπλωματικής εργασίας. Με τη καθοδήγησή του και τη συνεχή προσοχή που μου προσέφερε, με βοήθησε να ξεπεράσω οποιαδήποτε θέματα είχαν προκύψει κυρίως κατά τη συγγραφή της εργασίας, αλλά και κατά την ανάπτυξη της εφαρμογής. Τον ευχαριστώ για τον χρόνο τον οποίο διέθεσε. Έπειτα, θα ήθελα να ευχαριστήσω την οικογένειά μου, και τους δικούς μου ανθρώπους, καθώς μου στάθηκαν καθ'όλη την διάρκεια δημιουργίας και εκπόνησης της συγκεκριμένης διπλωματικής εργασίας. Τέλος χωρίς τη ψυχολογική στήριξη που έχει προσφέρει η μητέρα μου, δε θα μπορούσα σε καμία περίπτωση να τα καταφέρω, και για αυτό την ευχαριστώ.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Το πρόβλημα της σπατάλης τροφίμων

Οι επιχειρήσεις στις μέρες μας τείνουν να πετάνε καλά τρόφιμα γιατί δεν έχουν πωληθεί μέχρι το τέλος της εργάσιμης ημέρας. Η σπατάλη τροφίμων έχει γίνει ένα από τα ζητήματα στον κόσμο που οι περισσότεροι άνθρωποι εξακολουθούν να αναζητούν τη λύση τους. Σύμφωνα με τον FAO (Food and Agriculture Organization of the United Nations), περίπου 1,3 δισεκατομμύρια τόνοι ετησίως που ισοδυναμούν με το ένα τρίτο των τροφίμων που παράγονται για ανθρώπινη κατανάλωση στον κόσμο μετατρέπονται σε απόβλητα. Η αρχική παραγωγή είναι ο κύριος συντελεστής της σπατάλης τροφίμων αλλά όχι ο μοναδικός, ένα σημαντικό ποσοστό αυτής της σπατάλης έρχεται μέσα από αρκετά σημεία όπου συναντάμε συνήθως στα εστιατόρια, φούρνους, κουζίνες οικιακές ή εμπορικές και τελευταία στις καφετέριες.

Επιπλέον, η σπατάλη τροφίμων από επιχειρήσεις επηρεάζουν τη διαθεσιμότητα των τροφίμων σε άλλους. Το ένα τρίτο του κόσμου παράγει τρόφιμα και έχει σπατάλη, αντιθέτως υπάρχουν περίπου 868 εκατομμύρια άνθρωποι που εξακολουθούν να ζουν σε συνθήκες πείνας και ανεπαρκούς υποσιτισμού. Υπάρχουν εκατομμύρια άνθρωποι σε όλο τον κόσμο που εξακολουθούν να ψάχνουν και να αγωνίζονται για να βρουν αρκετή τροφή για να φάνε, αλλά εν τω μεταξύ, υπάρχουν εκατομμύρια φαγητά που χάνονται κάθε χρόνο. Έχει γίνει μεγάλο ζήτημα ειδικά στις ανεπτυγμένες χώρες καθώς θα οδηγήσει σε σοβαρό οικονομικό και περιβαλλοντικό πρόβλημα. Αυτό μπορεί να αποδειχθεί όταν ανακάλυψαν ότι 60 εκατομμύρια μετρικοί τόνοι τροφίμων χάνονται κάθε χρόνο στις Ηνωμένες Πολιτείες (New York Times, 2015). Άλλη μελέτη διαπίστωσε ότι τα μεγαλύτερα στερεά απόβλητα είναι τα απόβλητα τροφίμων που βρίσκονται στις χωματερές EPA (Environmental Protection Agency). Σε άλλο μέρος του κόσμου, είναι περίπου 280-300 κιλά απορριμμάτων τροφίμων το έτος ανά κάτοικο που εντοπίστηκαν στην Ευρώπη και τη Βόρεια Αμερική (Garrone, Melacini, Perego, 2014).

Σύμφωνα με την Υπηρεσία Προστασίας του Περιβάλλοντος (EPA), η σπατάλη τροφίμων είναι επίσης σημαντική αιτία περιβαλλοντικών προβλημάτων όπως η κλιματική αλλαγή και υπερθέρμανση του πλανήτη. Το μεθάνιο το οποίο παράγεται από τα απόβλητα τροφίμων στις χωματερές είναι χειρότερο όσον αφορά το φαινόμενο του θερμοκηπίου ακόμα και από το διοξείδιο του άνθρακα. Αυτά τα αέρια συμβάλλουν σημαντικά στην υπερθέρμανση του πλανήτη και την κλιματική αλλαγή μέσω της οποίας υποφέρει ολόκληρος ο πλανήτης. Η σπατάλη τροφίμων είναι επίσης έμμεσα σπατάλη φυσικών υδάτινων πόρων, επειδή το νερό που έχει χρησιμοποιηθεί για την δημιουργία της τροφής που έχει σπαταληθεί είναι επίσης χαμένο. Υπολογίζεται ότι εάν σπαταλήσετε 1 κιλό βοδινού κρέατος, σημαίνει ότι σπαταλάτε 45000 λίτρα νερού, γιατί η παραγωγή αυτού του 1 κιλού κρέατος κοστίζει 45000 λίτρα νερό.

Η σπατάλη τροφίμων είναι ένα ηθικό πρόβλημα και δεν προέρχεται μόνο από κοινωνικό, οικονομικό και περιβαλλοντικό επίπεδο, αλλά χρειάζεται όλοι οι άνθρωποι να παραμείνουν σε εγρήγορση και να λάβουν σοβαρά υπόψη το ζήτημα. Ως εκ τούτου, το αποτέλεσμα της μείωσης της σπατάλης τροφίμων έχει κερδίσει την προσοχή πολλών ιδιαίτερα σε διεθνές, περιφερειακό και εθνικό επίπεδο. Υπάρχουν πολλοί που έλαβαν σοβαρά υπόψη αυτό το ζήτημα, όπως κυβερνήσεις, ερευνητικά ιδρύματα και καταναλωτές.

1.2 Εφαρμογές για την αντιμετώπιση του προβλήματος

1. Too Good To Go

Το Too Good To Go είναι μια υπηρεσία με μια εφαρμογή για κινητά που συνδέει τους πελάτες με εστιατόρια και καταστήματα που έχουν πλεόνασμα απούλητων τροφίμων. Η υπηρεσία καλύπτει μεγάλες ευρωπαϊκές πόλεις, και από τον Οκτώβριο του 2020 ξεκίνησε να λειτουργεί στη Βόρεια Αμερική. Το 2022 το Too Good To Go ήταν η ταχύτερα αναπτυσσόμενη εφαρμογή βιώσιμων τροφίμων με βάση τον αριθμό των λήψεων. Από τον Αύγουστο του 2022, ισχυρίζεται ότι 164.000 επιχειρήσεις, που εξυπηρετούν 62 εκατομμύρια χρήστες, έχουν εξοικονομήσει 155 εκατομμύρια σακούλες με τρόφιμα [1].

2. CozZo

Με έδρα τα Σόφια της Βουλγαρίας, ο προγραμματιστής λογισμικού Ivo Dimitrov κυκλοφόρησε για πρώτη φορά την εφαρμογή CozZo το 2017 ως ένα σύστημα διαχείρισης κουζίνας. Η εφαρμογή στοχεύει στην αντιμετώπιση της σπατάλης τροφίμων επιτρέποντας στους χρήστες να δημιουργούν λίστες αγορών, να διαχειρίζονται αποθέματα ψυγείων και ντουλαπιών, να παρακολουθούν ημερομηνίες λήξης, να προγραμματίζουν γεύματα και να ανακαλύπτουν συνταγές χρησιμοποιώντας συστατικά που βρίσκονται στο σπίτι. Το CozZo έχει επεκτείνει την εμβέλειά του πέρα από τη Βουλγαρία και τώρα έχει χρήστες σε όλη τη Βόρεια Αμερική, την Ευρώπη, τη Νοτιοανατολική Ασία και την Αυστραλία. Σύμφωνα με τον Dimitrov, η εφαρμογή επέτρεψε στους χρήστες να μειώσουν τη σπατάλη φαγητού τους κατά 50% εντός του πρώτου ενός έτους [2].

3. EroeGo

Το 2021, ο Daniel Solomon και ο John Werner κυκλοφόρησαν την πρώτη εφαρμογή των Ηνωμένων Αραβικών Εμιράτων που σχεδιάστηκε για την καταπολέμηση της σπατάλης τροφίμων και της κλιματικής αλλαγής. Το EroeGo παρέχει μια διαδικτυακή πλατφόρμα παντοπωλείου όπου οι χρήστες μπορούν να έχουν πρόσβαση σε φρέσκα είδη παντοπωλείου που έχουν οριστεί να λήξουν και προσφέρονται σε μειωμένες τιμές. Η εφαρμογή προσφέρει επίσης μια υπηρεσία παράδοσης που βασίζεται σε μια δομή προμήθειας, η οποία στοχεύει να ωφελήσει τους οδηγούς παράδοσης. Το EroeGo ευθυγραμμίζει την αποστολή του με τους Στόχους Βιώσιμης Ανάπτυξης του 2030 του ΟΗΕ, με σκοπό την εξάλειψη της πείνας και την παροχή βασικής διατροφικής εκπαίδευσης και πρόσβασης σε φρέσκα τρόφιμα [2].

4. Kitche

Η start-up με έδρα το Λονδίνο Kitche, που ιδρύθηκε το 2018, δημιούργησε μια εφαρμογή που βοηθά τους χρήστες να εξοικονομήσουν χρήματα και να μειώσουν τη σπατάλη τροφίμων από την άνεση του σπιτιού τους. Μέσω της δωρεάν εφαρμογής, οι χρήστες πρώτα σαρώνουν και ανεβάζουν προϊόντα από τις αποδείξεις αγορών τους. Στη συνέχεια, η Kitche κατηγοριοποιεί τα αντικείμενα, παρέχει υπενθυμίσεις για να διασφαλίσει ότι τα τρόφιμα δεν θα σπαταληθούν και επιτρέπει στους χρήστες να φιλτράρουν χιλιάδες συνταγές με βάση τα συστατικά που έχουν ήδη στο σπίτι. Η εφαρμογή κέρδισε πρόσφατα High Commended για την Καλύτερη Επωνυμία Zero Waste στα Βραβεία Αειφορίας της Marie Claire [2].

5.Seva Kitchen

Ο φιλάνθρωπος Khushroo Poacha από το Ναγκπούρ ίδρυσε το Seva Kitchen, μια εφαρμογή διανομής φαγητού που προέρχεται από άλλους ανθρώπους και συνδέει τους δωρητές φρέσκων, ασφαλών, σπιτικών γευμάτων με τους παραλήπτες σε πραγματικό χρόνο. Η εφαρμογή επιδιώκει να αντιμετωπίσει την τεράστια ποσότητα φαγητού που πάει χαμένη σε πάρτι, φεστιβάλ και μεγάλες συγκεντρώσεις, συνδέοντας τους δωρητές με ανθρώπους και οργανισμούς που χρειάζονται φαγητό σε κοντινή απόσταση. Η Seva Kitchen ξεκίνησε επίσης την πρωτοβουλία Neki Ka Pitara, ή αλλιώς Fridge of Kindness, η οποία στοχεύει να παρέχει φρέσκο φαγητό σε άτομα σε σχολεία και νοσοκομεία σε όλη την Ινδία. Με 20 ψυγεία που βρίσκονται σε πόλεις όπως η Μπανγκαλόρ, το Χαϊντεραμπάντ, το Ναγκπούρ και το Νέο Δελχί, οι δωρητές μπορούν να παρέχουν εκ περιτροπής φρούτα, γάλα, αυγά και συσκευασμένα τρόφιμα. [2].

1.3 Κίνητρο και Συνεισφορά της εργασίας

Όπως έχουμε δει στο Κεφάλαιο 1.2, υπάρχουν πληθώρα εφαρμογών στο εξωτερικό για την καταπολέμηση του προαναφερθέντος προβλήματος. Το μειονέκτημα είναι ότι όλες αυτές οι εφαρμογές εξυπηρετούν κατά κύριο λόγο τις τοπικές κοινότητες και επιχειρήσεις, σαν αποτέλεσμα να μην υπάρχουν παρόμοιου τύπου εφαρμογές που να απευθύνονται σε επιχειρήσεις με έδρα στην Κύπρο. Αυτός είναι και ο κύριος λόγος που μας έδωσε το κίνητρο για την δημιουργία της συγκεκριμένης εφαρμογής.

Η Συνεισφορά της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη μιας εφαρμογής IOS και Android προκειμένου να μειώνει τη σπατάλη τροφίμων από επιχειρήσεις στην Κύπρο. Επιπρόσθετα θα δώσει την δυνατότητα σε πολλούς πιθανούς καταναλωτές που δεν έχουν ικανοποιητικές οικονομικές συνθήκες να προμηθευτούν καλά τρόφιμα σε μικρότερο κόστος. Επιπλέον θα δώσει την ευκαιρία σε κάποιες μικρομεσαίες επιχειρήσεις να εξασφαλίσουν κάποιο κέρδος στα προϊόντα που θα είχαν αλλιώς πετάξει. Οι επιχειρήσεις θα μπορούν μέσω μιας διαδικτυακής ιστοσελίδας να ανεβάζουν τα προϊόντα που τους έχουν απομείνει μέχρι το τέλος της ημέρας στην εφαρμογή, στην οποία θα έχουν πρόσβαση οι καταναλωτές μέσω της IOS ή Android συσκευής τους. Οι εν λόγω καταναλωτές θα βλέπουν τις διάφορες επιχειρήσεις οι οποίες συνεργάζονται με την εφαρμογή και τα προϊόντα που τους έχουν απομείνει. Έπειτα θα μπορούν να κάνουν διαδικτυακές παραγγελίας για τα συγκεκριμένα προϊόντα. Επίσης με την μείωση των τροφίμων που θα πετάγονται στα σκουπίδια και στην συνέχεια στις χωματερές, η παρούσα εργασία συνεισφέρει στην καταπολέμηση της κλιματικής αλλαγής. Στόχος μας είναι να δώσουμε και στην Κύπρο ένα εργαλείο με το οποίο θα μπορέσει να κάνει βελτίωση σε όλους τους τομείς που προαναφέραμε στο Κεφάλαιο 1.1.

1.4 Οργάνωση της εργασίας

Στο επόμενο κεφάλαιο θα αναφέρουμε όλες τις τεχνολογίες που έχουν χρησιμοποιηθεί για την υλοποίηση της εφαρμογής και ιστοσελίδας, αρχίζοντας από τον source code editor και προσομοιωτή. Στην συνέχεια θα εξηγήσουμε τι είναι η εφαρμογή Postman, η οποία έχει χρησιμοποιηθεί κυρίως για δοκιμές και αποσφαλμάτωση. Ακολούθως θα εξηγήσουμε την λειτουργία του Server XAMPP, γιατί έχει χρησιμοποιηθεί, και θα αναφέρουμε κάποια από τα βασικά πλεονεκτήματα και μειονεκτήματα του. Επίσης θα αναφέρουμε το πρόγραμμα phpMyAdmin το οποίο είναι υπεύθυνο για την υλοποίηση της βάσης δεδομένων. Κάποιες άλλες από τις τεχνολογίες που χρησιμοποιήθηκαν και θα αναφέρουμε είναι το Flutter Framework που χρησιμοποιεί γλώσσα προγραμματισμού Dart και έχει χρησιμοποιηθεί κυρίως για την δημιουργία του Frondend, και το Laravel framework που χρησιμοποιεί γλώσσα προγραμματισμού PHP και έχει χρησιμοποιηθεί κυρίως για το Backend. Επίσης αξίζει να σημειωθεί πως η ανάγνωση αυτού του κεφαλαίου είναι απαραίτητη για την κατανόηση κάποιον ορολογιών που θα χρησιμοποιηθούν επανειλημμένα κατά την διάρκεια των επόμενων κεφαλαίων.

Στο τρίτο Κεφάλαιο θα εξηγήσουμε πώς έγινε η υλοποίηση της εφαρμογής. Αρχίζοντας θα αναλύσουμε τα διαγράμματα τα οποία παρουσιάζουν την αρχιτεκτονική, η οποία έχει χρησιμοποιηθεί για την δημιουργία της λογικής των διαφόρων τμημάτων της εφαρμογής. Επιπλέον θα αναφέρουμε την αρχιτεκτονική της παραλαβής και επεξεργασίας δεδομένων του Frondend. Παράλληλα θα εξηγήσουμε με ποιο τρόπο έχει γίνει η δημιουργία των APIs και EndPoints στο Backend. Ακολούθως θα εξηγήσουμε πώς έχει κατασκευαστεί το Registration System, καθώς και τους μηχανισμούς αυθεντικοποίησης των εγγεγραμμένων χρηστών. Επιπλέον θα δούμε την Βάση Δεδομένων, η οποία περιλαμβάνει τις σχέσεις μεταξύ των πινάκων και τα δεδομένα που αποθηκεύονται στις στήλες τους. Για παράδειγμα, τις κοινοποιήσεις των επιχειρήσεων και τους εγγεγραμμένους χρήστες. Επίσης θα εξηγήσουμε τα βασικά κομμάτια κώδικα που αποτελούν την όλη λογική της εφαρμογής, καθώς και κάθε κομμάτι που έχει συμβάλλει σημαντικά στην δημιουργία της.

Στο τέταρτο Κεφάλαιο θα παρουσιάσουμε την λειτουργία της εφαρμογής. Θα κάνουμε ένα σενάριο για να δείξουμε με ποιο τρόπο λειτουργεί η εφαρμογή από την προοπτική του πελάτη. Θα αναφέρουμε όλα τα απαραίτητα βήματα που πρέπει να κάνει ο πελάτης, από την δημιουργία λογαριασμού μέχρι την παραγγελία. Έπειτα θα κάνουμε ένα σενάριο για να παρουσιάσουμε την λειτουργία της εφαρμογής από την προοπτική της επιχείρησης. Θα εξηγήσουμε την δημιουργία λογαριασμού και την διαδικασία κοινοποίησης προϊόντων.

Τέλος, στο πέμπτο Κεφάλαιο θα αναφέρουμε κάποια συμπεράσματά από την εκπόνηση της συγκεκριμένης εργασίας. Τελειώνοντας γίνονται προτάσεις ιδεών για μελλοντική επέκταση και ανάπτυξη δυνατοτήτων που θα καταστήσει την εφαρμογή πιο εύχρηστη και λειτουργική.

Κεφάλαιο 2ο: Τεχνολογίες

Στα πλαίσια αυτού του κεφαλαίου θα αναφέρουμε αναλυτικά τις τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της διαδικτυακής εφαρμογής. Επίσης θα αναφέρουμε τα προγράμματα που μας βοήθησαν για επαλήθευση, αποσφαλμάτωση και προσομοίωση στο στάδιο κατασκευής της. Παράλληλα θα εξηγήσουμε τις βασικές ορολογίες οι οποίες είναι απαραίτητες για την κατανόηση των επομένων κεφαλαίων.

2.1 VS Code και Android Emulator

2.1.1 VS Code

Η εφαρμογή έχει υλοποιηθεί στο περιβάλλων του Visual Studio Code το οποίο είναι ένα πρόγραμμα επεξεργασίας πηγαίου κώδικα που δημιουργήθηκε από τη Microsoft με το Electron Framework για Windows, Linux και macOS. [3] Οι δυνατότητες περιλαμβάνουν υποστήριξη για εντοπισμό σφαλμάτων, επισήμανση σύνταξης, έξυπνη συμπλήρωση κώδικα, snippets, ανακατασκευή κώδικα και ενσωματωμένο Git. Οι χρήστες μπορούν να αλλάξουν: Το θέμα, τις συντομεύσεις πληκτρολογίου, τις προτιμήσεις και να εγκαταστήσουν επεκτάσεις που προσθέτουν επιπλέον λειτουργίες.

2.1.2 Android Emulator

Για τις προσομοίωσης της εφαρμογής έχει χρησιμοποιηθεί το Android Emulator το οποίο προσομοιώνει συσκευές Android στον υπολογιστή, ώστε να μπορούμε να δοκιμάσουμε την εφαρμογή σε διάφορες συσκευές και επίπεδα Android API χωρίς να χρειάζεται να έχουμε κάθε φυσική συσκευή.

Κάποια από τα πλεονεκτήματα της χρήσης του εξομοιωτή είναι :

Ευελιξία: Εκτός από τη δυνατότητα προσομοίωσης μιας ποικιλίας συσκευών και επιπέδων API Android, ο εξομοιωτής διαθέτει προκαθορισμένες διαμορφώσεις για διάφορα τηλέφωνα Android, tablet, Wear OS και συσκευές Android TV.

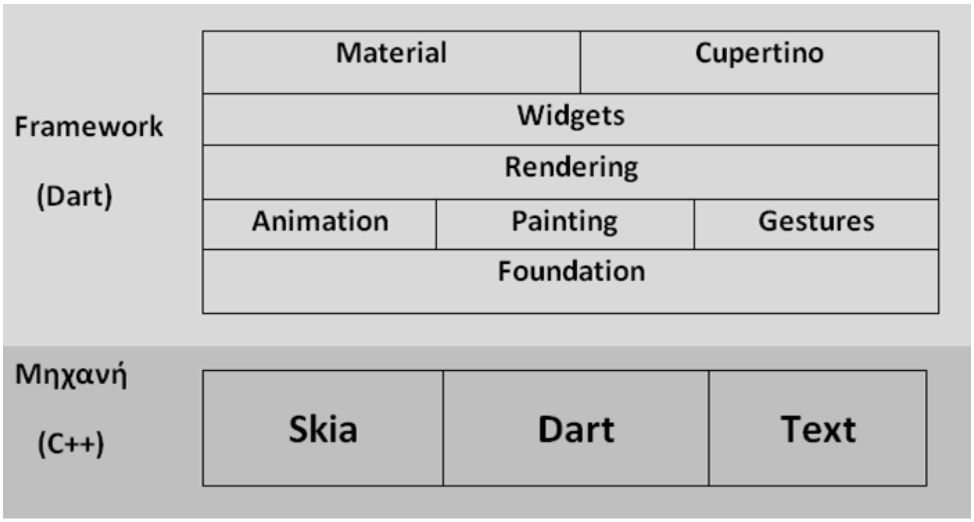
Υψηλή πιστότητα: Ο εξομοιωτής παρέχει σχεδόν όλες τις δυνατότητες μιας πραγματικής συσκευής Android. Μπορούμε να προσομοιώσουμε εισερχόμενες τηλεφωνικές κλήσεις και μηνύματα κειμένου, να καθορίσουμε τη θέση της συσκευής, να προσομοιώσουμε διαφορετικές ταχύτητες δικτύου, να αποκτήσουμε πρόσβαση στο Google Play Store και πολλά άλλα.

Ταχύτητα: Η δοκιμή της εφαρμογής στον εξομοιωτή είναι κατά κάποιο τρόπο ταχύτερη και ευκολότερη από ό,τι σε μια φυσική συσκευή. Για παράδειγμα, μπορούμε να μεταφέρουμε δεδομένα πιο γρήγορα στον εξομοιωτή παρά σε μια συσκευή συνδεδεμένη μέσω USB.

Για τις απαιτήσεις της συγκεκριμένης εφαρμογής, το Android Emulator μου έχει καλύψει όλες τις ανάγκες. Επιπλέον η εγκατάσταση και η συμβατότητα του με το VS Code ήταν πολύ απλή.

2.2 Flutter Framework

Το Flutter Framework είναι η κύρια τεχνολογία με την οποία έχει υλοποιηθεί το Frontend κομμάτι της εφαρμογής. Το Flutter είναι ένα SDK ανοιχτού κώδικα για την ανάπτυξη εφαρμογών για συσκευές iOS και Android με την ίδια βάση κώδικα. Το Flutter χρησιμοποιεί τη γλώσσα προγραμματισμού Dart για τη δημιουργία στοιχείων και τη μηχανή γραφικών Skia 2D για να ζωντανέψει τον κώδικα. Ένα react-style framework περιλαμβάνεται επίσης στο Flutter. Το περιεχόμενο του framework απεικονίζεται στο Σχήμα.2.1. Το Skia χρησιμοποιείται για να κάνει render το UI (User Interface), της εφαρμογής στο χαμηλότερο επίπεδο. Το Flutter χρησιμοποιεί ένα lightweight Dart virtual machine για να εκτελέσει το μεγαλύτερο μέρος του πλαισίου και του κώδικα εφαρμογής του. Επίσης η rendering μηχανή είναι γραμμένη σε C++, ενώ ο κώδικας πλαισίου είναι γραμμένος σε Dart. Επιπλέον το Flutter, δημιουργεί το δικό του περιβάλλον εργασίας χρήστη. [4]



Σχήμα 2.1: Flutter framework

Υπάρχουν πολλά σχέδια σχεδίασης που χρησιμοποιούν οι προγραμματιστές για το Flutter. Ο στόχος ενός μοτίβου σχεδίασης είναι να παρέχει ένα καθαρό πρότυπο για το πώς θα οργανωθεί η εργασία και πώς θα αλληλεπιδράσουν τα στοιχεία μεταξύ τους, επίσης να παρέχει χωριστά επίπεδα ώστε η αλλαγή στο ένα να είναι διαφανή για τα άλλα. Τέλος το πιο σημαντικό είναι να προωθήσει την επαναχρησιμοποίηση κώδικα.

Μια από τις σύγχρονες προκλήσεις που έχει ένας προγραμματιστής κινητών εφαρμογών είναι η δημιουργία λογισμικού σε πολλές πλατφόρμες, όπως το Android και iOS, λόγω του γεγονότος ότι ο προγραμματιστής πρέπει να διατηρεί ξεχωριστή βάση κώδικα για κάθε πλατφόρμα. Το Flutter αντιμετωπίζει αυτό το ζήτημα επιτρέποντας στους προγραμματιστές να δημιουργούν εφαρμογές για κινητά τόσο για συσκευές iOS όσο και για συσκευές Android. Το Flutter χρησιμοποιεί μια high-performance rendering engine για να κάνει render κάθε στοιχείο προβολής. Από την άποψη της αρχιτεκτονικής, ο κώδικας της C ή C++ engine περιλαμβάνει μεταγλώττιση με NDK για Android και LLVM για iOS αντίστοιχα, επίσης κατά τη διαδικασία μεταγλώττισης ο κώδικας Dart μεταγλωττίζεται σε native code. Ο Tim Sneath, διευθυντής προϊόντων ομάδας στην Google, το ορίζει ως ”a powerful general-purpose open UI toolkit” [5]. Η ενσωματωμένη υποστήριξη (built-in support), είναι προς το παρόν για τις πλατφόρμες κινητών iOS και Android όπως φαίνεται στο Σχήμα.2.2.

Σχήμα 2.2: Δομή εφαρμογής Flutter

Στο Flutter, σχεδόν τα πάντα είναι ένα widget, το widget είναι ένας τρόπος για να δημιουργήσουμε το UI, και για την δημιουργία μιας εγγενούς πλατφόρμας για Android και iOS. Το μόνο που απαιτεί το Flutter από την πλατφόρμα είναι ένας καμβάς στον οποίο θα γίνονται render τα γραφικά στοιχεία (widgets), ώστε να μπορούν να εμφανίζονται στη οθόνη της συσκευής και να έχουν πρόσβαση σε events και services [6].

Επιπλέον, στο flutter, τα widget έχουν διαφορετική διάρκεια ζωής. Είναι αμετάβλητα και υπάρχουν μόνο μέχρι να χρειαστεί να αλλάξουν. Κάθε φορά που αλλάζουν τα widgets ή οι καταστάσεις τους, το Flutter's framework δημιουργεί ένα νέο δέντρο από widget instances. Συγκριτικά, ένα Android view σχεδιάζεται μία φορά και δεν επανασχεδιάζεται μέχρι να κληθεί το invalidate. Στο iOS, τα περισσότερα από αυτά που δημιουργούν οι προγραμματιστές στο UI, γίνονται χρησιμοποιώντας αντικείμενα προβολής (view objects). Συγκριτικά, μια προβολή iOS δεν δημιουργείται εκ νέου όταν αλλάζει, αλλά είναι μια μεταβλητή οντότητα που σχεδιάζεται μία φορά και δεν επανασχεδιάζεται μέχρι να ακυρωθεί. Επιπλέον, το Flutter ακολουθεί μια διαφορετική προσέγγιση για την αποφυγή προβλημάτων απόδοσης που προκαλούνται από την ανάγκη γέφιρας (bridge), χρησιμοποιώντας την μεταγλωττισμένη (compiled), γλώσσα προγραμματισμού Dart. [4]

2.2.1 State management

Τι είναι Κατάσταση;

Σύμφωνα με τον Matt Caroll, state είναι "behavior of an app at any given moment. Άρα, κατάσταση είναι η συλλογή μεταβλητών σε ένα ορισμένο χρονικό διάστημα και η στιγμή που ένα πρόγραμμα εμφανίζει το τρέχον περιεχόμενό τους. Αν και στα πλαίσια της συγκεκριμένης διπλωματικής εργασίας μιλάμε για το flutter framework, η διαχείριση κατάστασης (State management), δεν είναι συγκεκριμένη μόνο για το Flutter. [7]

Τώρα ας επιστρέψουμε στην διαχείριση κατάστασης στο Flutter. Το πρώτο πράγμα που πρέπει να σημειωθεί είναι ότι το Flutter είναι δηλωτικό, που σημαίνει ότι το UI εμφανίζεται ως συνάρτηση από την κατάσταση (state). Έτσι, όταν εκτελούμε μια συγκεκριμένη συνάρτηση με την κατάσταση, παίρνουμε το UI ως σχέση. Το GetX δημιουργήθηκε ως μια απλή λύση για την διαχείριση κατάστασης στην ανάπτυξη εφαρμογών flutter.

2.2.2 Flutter Getx Package

Τώρα που έχουμε εξοικειωθεί με όλες τις βασικές δομές, θα πρέπει να είμαστε σε θέση να κατανοήσουμε με ποιο τρόπο επικοινωνούν αυτά τα επίπεδα μεταξύ τους. Το Getx package είναι ένα ”lightweight and powerful” flutter package που προσφέρει στους προγραμματιστές τη δυνατότητα να χειρίζονται περίπλοκα state management, route management, και ενσωματώνουν dependency injection. Επίσης το Getx Package είναι πραγματικά χρήσιμο όταν πρόκειται για τον διαχωρισμό σε στρώματα κώδικα. Κάθε επίπεδο, ή σύνολο κλάσεων, είναι υπεύθυνο για μια συγκεκριμένη εργασία. [8]

Η λύση που λύνει το GetX περιλαμβάνει την βελτίωση αναγνωσιμότητας κώδικα, διαχείριση και ενημερώσεις σύνθετων αλλαγών εφαρμογών, μείωση κώδικα με λίγες γραμμές που χρησιμοποιούνται για την εκτέλεση πολύπλοκων λειτουργιών σε σύγκριση με άλλες τεχνικές διαχείρισης κατάστασης που χρησιμοποιούνται στο Flutter, όπως το πακέτο BLOC, το οποίο είναι δύσκολο να ρυθμιστεί, και απαιτεί πολλές γραμμές κώδικα, ή το Widget Provider, το οποίο έχει λιγότερες γραμμές κώδικα από το πακέτο BLOC, αλλά έχει πολύ πιο πολύπλοκη λειτουργικότητα. Επίσης το Flutter παρέχει εργαλεία, όπως το GetConnect που ανήκει στο Getx package, με το οποίο μπορούμε να στείλουμε dynamic headers, παραμέτρους, requests και responses με προσαρμοσμένο και ασφαλή τρόπο.

2.3 Dart

Η Dart είναι μια γλώσσα προγραμματισμού που έχει σχεδιαστεί για client development, όπως για web και mobile Apps. Αναπτύσσεται από την Google και μπορεί επίσης να χρησιμοποιηθεί για την κατασκευή server και desktop εφαρμογών. Πρόκειται για μια object-oriented, class-based, garbage-collected γλώσσα με σύνταξη τύπου C. Μπορεί να μεταγλωττιστεί είτε με machine code ή JavaScript και υποστηρίζει interfaces, mixins, abstract classes, reified generics και type inferences. [9]

Η Dart αποκαλύφθηκε στο συνέδριο GOTO στο Aarhus, Δανία, 10-12 Οκτωβρίου 2011. Το έργο ιδρύθηκε από τους Lars Bak και Kasper Lund. Η Dart 1.0 κυκλοφόρησε στις 14 Νοεμβρίου 2013. Η Dart είχε αρχικά μια μικτή υποδοχή και η πρωτοβουλία DART έχει επικριθεί από μερικούς για την κατακερματισμό του web, λόγω των αρχικών σχεδίων για να συμπεριληφθεί ένα DART VM στο Chrome. Αυτά τα σχέδια έπεσαν το 2015 με την Dart 1.9 release για να επικεντρωθεί αντ’αυτού compiling Dart στην JavaScript. Η Dart 2.6 εισήγαγε μια νέα επέκταση, Dart2native, η οποία επεκτείνει native compilation στις επιφάνειας εργασίας Linux, MacOS και Windows. Οι προηγούμενοι προγραμματιστές θα μπορούσαν να δημιουργήσουν νέα εργαλεία χρησιμοποιώντας μόνο συσκευές Android ή iOS. Με αυτή την επέκταση γίνεται επίσης δυνατή η σύνταξη ενός προγράμματος σε αυτόνομα εκτελέσιμα. Σύμφωνα με τους εκπροσώπους της εταιρείας, δεν είναι πλέον απαραίτητο να εγκατασταθεί το DART SDK, καθώς τα αυτόνομα εκτελέσιμα μπορούν τώρα να αρχίσουν να τρέχουν σε λίγα δευτερόλεπτα. Η νέα επέκταση είναι επίσης ενσωματωμένη στο Toolkit Flutter, καθιστώντας δυνατή τη χρήση του μεταγλωττιστή σε μικρές υπηρεσίες (για παράδειγμα, υποστήριξη backend). [9]

Για να τρέξει σε mainstream web browsers, η Dart βασίζεται σε έναν source-to-source compiler σε JavaScript. Σύμφωνα με τον ιστότοπο της Dart, η Dart σχεδιάστηκε ”To be easy to write development tools for, well-suited to modern app development, and capable of high-performance implementations.” [9]. Σε έναν web browser, ο κώδικας είναι precompiled σε Javascript χρησιμοποιώντας τον dart2js

compiler, καθιστώντας το συμβατό με όλους τους web browsers χωρίς να χρειάζεται να το υιοθετήσουν. Με τη βελτιστοποίηση της compiled JavaScript output για να αποφευχθούν ακριβοί έλεγχοι και λειτουργίες, ο κώδικας που γράφτηκε σε Dart μπορεί, σε ορισμένες περιπτώσεις, να τρέξει ταχύτερα από τον ισοδύναμο κώδικα σε JavaScript.

Η γλώσσα Dart χρησιμοποιεί έλεγχο στατικού τύπου για να διασφαλίσει ότι η τιμή μιας μεταβλητής ταιριάζει πάντα με τον στατικό τύπο της μεταβλητής. Μερικές φορές, αυτό αναφέρεται ως Sound typing. Αν και οι τύποι είναι υποχρεωτικοί, οι σχολιασμοί τύπων είναι προαιρετικοί λόγω των συμπερασμάτων τύπου (type inference). Το σύστημα πληκτρολόγησης Dart είναι επίσης ευέλικτο, επιτρέποντας τη χρήση ενός δυναμικού τύπου σε συνδυασμό με ελέγχους χρόνου εκτέλεσης, ο οποίος μπορεί να είναι χρήσιμος κατά τη διάρκεια του πειραματισμού ή για κώδικα που πρέπει να είναι ιδιαίτερα δυναμικός. Η Dart προσφέρει Sound Null Safety, που σημαίνει ότι οι τιμές δεν μπορούν να είναι null, εκτός και αν πείτε ότι μπορούν να είναι. Με την Sound Null Safety, η Dart μπορεί να μας προστατεύσει από null exceptions κατά το χρόνο εκτέλεσης μέσω ανάλυσης στατικού κώδικα. Σε αντίθεση με πολλές άλλες null-safe γλώσσες, όταν η Dart καθορίζει ότι μια μεταβλητή είναι non-nullable, αυτή η μεταβλητή είναι πάντα non-nullable. [10]

2.3.1 Build in Dart και Flutter Βιβλιοθήκες

1. Ενσωματωμένοι τύποι, συλλογές και άλλες βασικές λειτουργίες για κάθε πρόγραμμα Dart. (dart:core).
2. Μαθηματικές σταθερές και συναρτήσεις και δημιουργία τυχαίων αριθμών. (dart:math).
3. Υποστήριξη ασύγχρονου προγραμματισμού, με κλάσεις τύπου όπως Future και Stream. (dart:async)
4. Στοιχεία HTML και άλλοι πόροι για web-based εφαρμογές όπου αλληλεπιδρούν με τον browser και το Document Object Model (DOM). (dart:html)
5. Κωδικοποιητές και αποκωδικοποιητές για μετατροπή μεταξύ διαφορετικών αναπαραστάσεων δεδομένων, συμπεριλαμβανομένων των JSON και UTF-8. (dart:convert)
6. Η Material Βιβλιοθήκη είναι ένα προσαρμόσιμο σύστημα σχεδίασης, που υποστηρίζεται από κώδικα ανοιχτού κώδικα, που βοηθά τους προγραμματιστές να δημιουργήσουν εύκολα ψηφιακές εμπειρίες υψηλής ποιότητας. Από τις οδηγίες σχεδίασης έως τα στοιχεία προγραμματιστών, η Material Βιβλιοθήκη μπορεί να βοηθήσει τους προγραμματιστές να δημιουργήσουν προϊόντα πιο γρήγορα.(flutter:material)
7. Όταν γράφετε μια εφαρμογή Material στο Flutter, έχει την εμφάνιση και την αίσθηση Material σε όλες τις συσκευές, ακόμα και στο iOS. Η βιβλιοθήκη Cupertino κάνει την εφαρμογή να μοιάζει με μια τυπική εφαρμογή σε style iOS. (flutter:cupertino)
8. Αυτή η βιβλιοθήκη εκθέτει τις υπηρεσίες χαμηλότερου επιπέδου που χρησιμοποιεί το Flutter framework για την εκκίνηση εφαρμογών, όπως κλάσεις για την οδήγηση των υποσυστημάτων εισόδου, graphics text και rendering υποσυστήματα. (dart:ui)

9. Το SharedPreferences είναι αυτό που χρησιμοποιούν οι εφαρμογές Android και iOS για την αποθήκευση απλών δεδομένων σε έναν local storage. Αυτά τα δεδομένα υπάρχουν ακόμα και όταν η εφαρμογή τερματίζεται και ξεκινά ξανά. Τα δεδομένα που είναι αποθηκευμένα στο SharedPreferences μπορούν να επεξεργαστούν και να διαγραφούν.(flutter: shared preferences)

2.3.2 Dart runtime

Ανεξάρτητα από την πλατφόρμα που χρησιμοποιούμε ή τον τρόπο με τον οποίο μεταγλωττίζουμε τον κώδικά , η εκτέλεση του κώδικα απαιτεί runtime. Αυτός ο χρόνος εκτέλεσης είναι υπεύθυνος για τις ακόλουθες κρίσιμες εργασίες:

Διαχείριση μνήμης: Η Dart χρησιμοποιεί ένα μοντέλο διαχειριζόμενης μνήμης, όπου η αχρησιμοποίητη μνήμη ανακτάται από έναν garbage collector (GC). Επιβολή του συστήματος τύπου της Dart: Αν και οι περισσότεροι έλεγχοι στην Dart είναι τύπου static (compile-time), ορισμένοι έλεγχοι είναι τύπου dynamic (runtime). Για παράδειγμα, ο Dart runtime επιβάλλει δυναμικούς ελέγχους τύπου check και cast operators. Σε native πλατφόρμες, ο Dart runtime περιλαμβάνεται αυτόματα σε αυτόνομα εκτελέσιμα αρχεία και αποτελεί μέρος του Dart VM που παρέχεται από την εντολή ”dart run”. [10]

2.4 Laravel Framework

Το Laravel είναι ένα δωρεάν και open-source PHP web framework, που δημιουργήθηκε από τον Taylor Otwell και προορίζεται για την ανάπτυξη διαδικτυακών εφαρμογών σύμφωνα με το αρχιτεκτονικό μοτίβο model–view–controller (MVC), και βασίζεται στο Symfony. Μερικά από τα χαρακτηριστικά του Laravel είναι ένα modular packaging system με αφιερωμένο dependency manager, διαφορετικοί τρόποι πρόσβασης σε σχεσιακές βάσεις δεδομένων, βοηθητικά προγράμματα που βοηθούν στην ανάπτυξη και συντήρηση εφαρμογών, και ο προσανατολισμός του προς τη συντακτική ζάχαρη. [11] (Syntactic Sugar στην επιστήμη των υπολογιστών είναι η σύνταξη σε μια γλώσσα προγραμματισμού που έχει σχεδιαστεί για να διευκολύνει την ανάγνωση ή την έκφραση των πραγμάτων. Κάνει τη γλώσσα πιο «γλυκιά» για ανθρώπινη χρήση, τα πράγματα μπορούν να εκφραστούν πιο ξεκάθαρα, πιο συνοπτικά ή με ένα εναλλακτικό ύφος που κάποιοι μπορεί να προτιμήσουν).

Ο Taylor Otwell δημιούργησε το Laravel ως μια προσπάθεια να παρέχει μια πιο προηγμένη εναλλακτική λύση στο CodeIgniter framework, το οποίο δεν παρείχε ορισμένες δυνατότητες όπως ενσωματωμένη υποστήριξη για user authentication (αυθεντικοποίηση), και authorization (εξουσιοδότηση). Η πρώτη έκδοση του Laravel έγινε διαθέσιμη στις 9 Ιουνίου 2011, ακολουθούμενη από την κυκλοφορία του Laravel 1 αργότερα τον ίδιο μήνα. Το Laravel 1 περιλάμβανε authentication, localisation, models, views, sessions, routing και άλλους μηχανισμούς, αλλά δεν είχε υποστήριξη για controllers που το εμπόδιζαν να είναι ένα πραγματικό MVC framework.

Το Laravel 2 κυκλοφόρησε τον Σεπτέμβριο του 2011, φέρνοντας διάφορες βελτιώσεις. Τα κύρια νέα χαρακτηριστικά περιελάμβαναν την υποστήριξη για controllers, που έκαναν το Laravel 2 ένα framework πλήρως συμβατό με το MVC. Ως αρνητικό στοιχείο, η υποστήριξη για πακέτα τρίτων καταργήθηκε στο Laravel 2.

Το Laravel 3 κυκλοφόρησε τον Φεβρουάριο του 2012 με ένα σύνολο νέων χαρακτηριστικών συμπεριλαμβανομένης της διεπαφής γραμμής εντολών cmd command-line interface (CLI), με το όνομα Artisan, ενσωματωμένη υποστήριξη για περισσότερα συστήματα διαχείρισης βάσεων δεδομένων, database migrations ως μορφή ελέγχου έκδοσης για διατάξεις βάσεων δεδομένων, υποστήριξη για χειρισμός συμβάντων και ένα packaging system που ονομάζεται Bundles.

Το Laravel 4, με την κωδική ονομασία Illuminate, κυκλοφόρησε τον Μάιο του 2013. Έγινε ως πλήρης επανεγγραφή του Laravel framework, μεταφέροντας τη διάταξή του σε ένα σύνολο ξεχωριστών πακέτων που διανέμονται μέσω του Composer, το οποίο χρησιμεύει ως application-level package manager. Μια τέτοια διάταξη βελτίωσε την επεκτασιμότητα του Laravel 4. Άλλες νέες δυνατότητες στην έκδοση Laravel 4 περιλαμβάνουν τη δημιουργία βάσης δεδομένων για τον αρχικό πληθυσμό των βάσεων δεδομένων, την υποστήριξη για ουρές μηνυμάτων (message queues), την ενσωματωμένη υποστήριξη για την αποστολή διαφορετικών τύπων email και την υποστήριξη για καθυστερημένη διαγραφή εγγραφών βάσης δεδομένων που ονομάζεται soft deletion. [12]

Το Laravel 5 κυκλοφόρησε τον Φεβρουάριο του 2015 ως αποτέλεσμα εσωτερικών αλλαγών που κατέληξαν στην αναρίθμηση της τότε μελλοντικής έκδοσης Laravel 4.3. Οι νέες δυνατότητες στην έκδοση του Laravel 5 περιλαμβάνουν υποστήριξη για τον προγραμματισμό περιοδικά εκτελούμενων εργασιών μέσω ενός πακέτου που ονομάζεται Scheduler, ενός στρώματος αφαίρεσης (bstraction layer), που ονομάζεται Fly system που επιτρέπει την απομακρυσμένη αποθήκευση να χρησιμοποιείται με τον ίδιο τρόπο όπως τα τοπικά συστήματα αρχείων, βελτιωμένος χειρισμός των package assets μέσω του Elixir και απλοποιημένος έλεγχος αυθεντικοποίησεις μέσω του προαιρετικού πακέτου Socialite. Το Laravel 5 εισήγαγε επίσης μια νέα δομή εσωτερικού καταλόγου (directory tree structure), για ανεπτυγμένες εφαρμογές.

Το Laravel 7 κυκλοφόρησε στις 3 Μαρτίου 2020, με νέα χαρακτηριστικά όπως Laravel Sanctum, Custom Eloquent Casts, Blade Component Tags, Fluent String Operations και Route Model Binding βελτιώσεις. [13]

Και μετά από αυτή την ιστορική αναδρομή έχουμε φτάσει στην έκδοση που έχει χρησιμοποιηθεί για την δημιουργία του Backend της εφαρμογής. Το Laravel 8 κυκλοφόρησε στις 8 Σεπτεμβρίου 2020, με νέες δυνατότητες όπως το Laravel Jetstream (το Jetstream παρέχει την υλοποίηση για τη σύνδεση (login), την εγγραφή (registration), την επαλήθευση email, την αυθεντικοποίηση δύο παραγόντων (two-factor authentication), τη διαχείριση περιόδων σύνδεσης (session management), το API μέσω Laravel Sanctum και προαιρετικές δυνατότητες διαχείρισης ομάδας (team management)). [13]

2.4.1 Laravel admin

Το laravel-admin είναι ένα πρόγραμμα δημιουργίας διοικητικής διεπαφής για το laravel το οποίο με έχει βοηθήσει να δημιουργήσω ένα backend CRUD, με μόνο λίγες γραμμές κώδικα. Τι είναι το backend CRUD; Στα παρασκήνια, τα API συχνά χρειάζεται να χειρίζονται δεδομένα ως μέρος της λειτουργίας τους. Συνήθως, αυτές οι λειτουργίες δεδομένων που ονομάζονται εν συντομία CRUD, εκτελούνται σε backend βάσεις δεδομένων. Το CRUD σημαίνει Create, Read, Update και Delete, τα οποία αποτελούν τις τέσσερις θεμελιώδεις λειτουργίες.

Εγγαταστασή του Laravel admin [14]

Αρχικά, εγκαταστήστα το laravel και βεβαιωθείκα ότι οι ρυθμίσεις σύνδεσης της βάσης δεδομένων είναι σωστές. Στη συνέχεια, εκτέλεσα την πιο κάτω εντολή από το cmd στον κατάλογο όπου έχω αποθηκευμένο το Laravel Project για να δημοσιεύσω τα στοιχεία και ρυθμίσεις:

```
php artisan vendor:publish --provider="Encore\Admin\AdminServiceProvider"
```

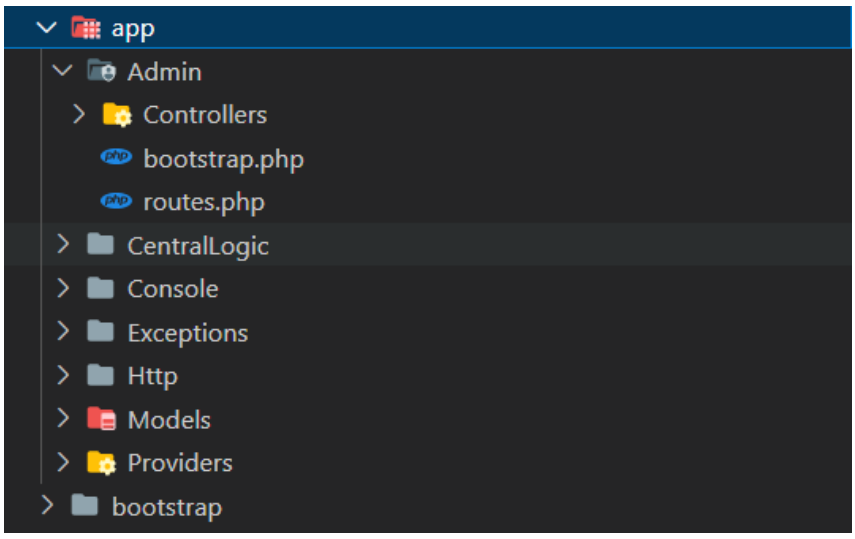
Μετά την εκτέλεση της εντολής πρέπει να βρούμε το αρχείο admin.php στον φάκελο config/admin.php. Σε αυτό το αρχείο μπορούμε να αλλάξουμε τον κατάλογο εγκατάστασης, τη σύνδεση db ή τα ονόματα των πινάκων. Στο τέλος εκτελούμε την παρακάτω εντολή για να ολοκληρώσουμε την εγκατάσταση.

```
php artisan admin:install
```

Αρχεία που δημιουργήθηκαν

Μετά την ολοκλήρωση της εγκατάστασης, δημιουργούνται τα ακόλουθα αρχεία στο project directory:

1. Configuration file: Αφού ολοκληρωθεί η εγκατάσταση, όλες οι ρυθμίσεις παραμέτρων βρίσκονται στο αρχείο config/admin.php.
2. Admin files: Μετά την εγκατάσταση, δημιουργείται το directory app/Admin, το οποίο περιέχει το μεγαλύτερο μέρος του admin panel όπως φαίνεται στο Σχήμα 2.4.



Σχήμα 2.3: Laravels directory Admin.

Το app/Admin/routes.php χρησιμοποιείται για τον καθορισμό διαδρομών. Ο κατάλογος app/Admin/Controllers χρησιμοποιείται για την αποθήκευση όλων των ελεγκτών. Τέλος το αρχείο HomeController.php περιλαμβάνει τον κώδικα που χρησιμοποιείται για τον χειρισμό του αρχικού αιτήματος του διαχειριστή.

3. Static assets: Τα front-end στατικά αρχεία βρίσκονται στον κατάλογο /public/packages/admin.

2.5 API

Ένα application programming interface (API) είναι ένας τρόπος για δύο ή περισσότερα προγράμματα υπολογιστών να επικοινωνούν μεταξύ τους. Είναι ένας τύπος διεπαφής λογισμικού, που προσφέρει υπηρεσία σε άλλα κομμάτια λογισμικού. [15] Ένα έγγραφο ή πρότυπο που περιγράφει τον τρόπο δημιουργίας ή χρήσης μιας τέτοιας σύνδεσης ή διεπαφής ονομάζεται προδιαγραφή API. Ένα σύστημα υπολογιστή που πληροί αυτό το πρότυπο λέγεται ότι υλοποιεί ή εκθέτει ένα API. Ο όρος API μπορεί να αναφέρεται είτε στην προδιαγραφή είτε στην υλοποίηση.

Σε αντίθεση με μια διεπαφή χρήστη (UI), η οποία συνδέει έναν υπολογιστή με ένα άτομο, μια διεπαφή προγραμματισμού εφαρμογών (API), συνδέει υπολογιστές ή κομμάτια λογισμικού μεταξύ τους. Δεν προορίζεται να χρησιμοποιηθεί απευθείας από άτομο (τον τελικό χρήστη) εκτός από προγραμματιστή υπολογιστή που το ενσωματώνει στο λογισμικό. Ένα API αποτελείται συχνά από διαφορετικά μέρη που λειτουργούν ως εργαλεία ή υπηρεσίες που είναι διαθέσιμα στον προγραμματιστή. Ένα πρόγραμμα ή ένας προγραμματιστής που χρησιμοποιεί ένα από αυτά τα μέρη λέγεται ότι καλεί αυτό το τμήμα του API. Οι κλήσεις που απαρτίζουν το API είναι επίσης γνωστές ως υπορουτίνες, μέθοδοι (methods), αιτήματα (requests), ή τελικά σημεία (endpoints). Μια προδιαγραφή API ορίζει αυτές τις κλήσεις, που σημαίνει ότι εξηγεί πώς να τις χρησιμοποιήσουμε ή να τις εφαρμόσουμε. [16]

Ένας σκοπός των API είναι να κρύψουν τις εσωτερικές λεπτομέρειες του τρόπου λειτουργίας ενός συστήματος, εκθέτοντας μόνο εκείνα τα μέρη που θα θεωρήσει χρήσιμα ο προγραμματιστής. Ένα API μπορεί να είναι προσαρμοσμένο για ένα συγκεκριμένο ζεύγος συστημάτων ή μπορεί να είναι ένα κοινό πρότυπο που επιτρέπει τη διαλειτουργικότητα μεταξύ πολλών συστημάτων.

Ο όρος API χρησιμοποιείται συχνά για να αναφέρεται σε web API, τα οποία επιτρέπουν την επικοινωνία μεταξύ υπολογιστών που συνδέονται μέσω του Διαδικτύου. Υπάρχουν επίσης API για γλώσσες προγραμματισμού, βιβλιοθήκες λογισμικού και λειτουργικά συστήματα υπολογιστών. Τα API προέκυψαν τη δεκαετία του 1940, αν και ο όρος εμφανίστηκε τις δεκαετίες του 1960 και του 1970. Οι πρόσφατες εξελίξεις στα API οδήγησαν στην αύξηση της δημοτικότητας των μικροϋπηρεσιών, οι οποίες είναι χαλαρά συνδεδεμένες υπηρεσίες που προσπελάζονται μέσω δημόσιων API. [17]

2.6 Postman

Το Postman είναι μια εφαρμογή που χρησιμοποιείται για δοκιμές API, είναι ένας πελάτης HTTP που δοκιμάζει αιτήματα HTTP, χρησιμοποιώντας ένα γραφικό περιβάλλον χρήστη μέσω του οποίου λαμβάνουμε διαφορετικούς τύπους απαντήσεων οι οποίες πρέπει στη συνέχεια να επικυρωθούν.

Μεθόδοι

Το Postman προσφέρει πολλές μεθόδους αλληλεπίδρασης τελικού σημείου (endpoint). Οι παρακάτω είναι μερικες από τις πιο επαναχρησιμοποιούμενες μεθόδους του Postman, συμπεριλαμβανομένων των λειτουργιών τους:

1. GET: Απόκτηση πληροφοριών
2. POST: Προσθήκη πληροφοριών
3. PUT: Αντικατάσταση πληροφοριών
4. PATCH: Ενημέρωση πληροφοριών
5. DELETE: Διαγραφή πληροφοριών

Κωδικοί απόκρισης (Response Codes)

Κατά τη δοκιμή των API με τον Postman, συνήθως λαμβάνουμε διαφορετικούς κωδικούς απόκρισης. Μερικοί από τους πιο κοινούς περιλαμβάνουν:

1. Series 100 : Χρονικές αποκρίσεις, για παράδειγμα, «102 Processing».
2. 200 Series : Απαντήσεις όπου ο πελάτης αποδέχεται το αίτημα και ο διακομιστής το επεξεργάζεται με επιτυχία, για παράδειγμα, «200 Ok».
3. 300 Series : Απαντήσεις που σχετίζονται με την ανακατεύθυνση URL, για παράδειγμα, ”301 Moved Permanently.”
4. 400 Series : Αποκρίσεις σφαλμάτων πελάτη, για παράδειγμα, «400 Bad Request».
5. 500 Series : Αποκρίσεις σφαλμάτων διακομιστή, για παράδειγμα, ”500 Internal Server Error”.

Επίσης το Postman δίνει τη δυνατότητα ομαδοποίησης διαφορετικών αιτημάτων. Αυτή η δυνατότητα είναι γνωστή ως «συλλογές» και βοηθά στην οργάνωση των δοκιμών. Αυτές οι συλλογές είναι φάκελοι όπου αποθηκεύουν τα αιτήματα και μπορούν να δομηθούν με όποιον τρόπο προτιμά η ομάδα. Δίνεται επίσης η δυνατότητα εξαγωγής-εισαγωγής τους. Ο Postman μας επιτρέπει επίσης να δημιουργήσουμε διαφορετικά περιβάλλοντα μέσω της δημιουργίας/χρήσης μεταβλητών. Για παράδειγμα, μια μεταβλητή URL που στοχεύει σε διαφορετικά περιβάλλοντα δοκιμής (dev-QA), επιτρέποντάς μας να εκτελούμε δοκιμές σε διαφορετικά περιβάλλοντα χρησιμοποιώντας υπάρχοντα αιτήματα. [18]

Αυτή ήταν μια γρήγορη περίληψη των μεθόδων και των σφαλμάτων δοκιμής API κατά τη χρήση του Postman η οποία ήταν καθοριστική για την δημιουργία της συγκεκριμένης εφαρμογής, μας επέτρεψε την υλοποίηση δοκιμών για τα endpoints του backend καθώς και GET - POST δοκιμές αιτημάτων στον server.

2.7 XAMPP

Το XAMPP είναι ένας Cross-Platform web server που είναι δωρεάν και ανοιχτού κώδικα. Επίσης το XAMPP επιτρέπει στους προγραμματιστές να γράφουν και να ελέγχουν τον κώδικά τους σε έναν local web server. Δημιουργήθηκε από τους Apache Friends και το κοινό μπορεί να αναθεωρήσει ή να τροποποιήσει τον εγγενή πηγαίο κώδικα του (native source code). Περιλαμβάνει MariaDB, διακομιστή HTTP Apache και διερμηνείς για PHP και Perl, μεταξύ και άλλων γλωσσών. Λόγω της απλότητας ανάπτυξης

του XAMPP, ένας προγραμματιστής μπορεί γρήγορα και εύκολα να εγκαταστήσει μια στοίβα WAMP ή LAMP σε ένα λειτουργικό σύστημα, με το πρόσθετο πλεονέκτημα ότι μπορούν επίσης να φορτωθούν κοινές πρόσθετες εφαρμογές όπως το WordPress και το Joomla. [19] Το XAMPP χρησιμοποιείται για τη δοκιμή πελατών ή websites πριν από τη δημοσίευσή τους σε έναν απομακρυσμένο web server. Σε έναν τοπικό υπολογιστή, το λογισμικό του XAMPP παρέχει ένα κατάλληλο περιβάλλον για τη δοκιμή έργων MYSQL, PHP, Apache και Perl. Επειδή οι περισσότερες αναπτύξεις web server στον πραγματικό κόσμο μοιράζονται τα ίδια στοιχεία με το XAMPP, η μετάβαση από έναν τοπικό δοκιμαστικό server σε έναν ζωντανό server είναι απλή.

2.7.1 Πλεονεκτήματα και μειονεκτήματα του XAMPP

Πλεονεκτήματα:

1. Είναι Multi Cross-Platform, πράγμα που σημαίνει ότι λειτουργεί τόσο σε Windows όσο και σε Linux.
2. Με μία μόνο εντολή, μπορούμε να ξεκινήσουμε και να σταματήσουμε ολόκληρη τη στοίβα web server και βάσης δεδομένων.
3. Διατίθενται τόσο μια πλήρης όσο και μια τυπική έκδοση του XAMPP.
4. Διαθέτει έναν πίνακα ελέγχου που περιέχει κουμπιά έναρξης και διακοπής για συγκεκριμένους μηχανισμούς, όπως το Apache, το οποίο εκτελείται μέσω του Πίνακα Ελέγχου.
5. Περιλαμβάνει επίσης OpenSSL, phpMyAdmin, MediaWiki, Joomla, WordPress και πολλές πρόσθετες εφαρμογές.

Μειονέκτημα:

Σε σύγκριση με τον διακομιστή WAMP, η διαμόρφωση (configuration) και οι ρυθμίσεις (settings) είναι πιο δύσκολες.

2.7.2 Εξαρτήματα του XAMPP

Cross-Platform: Διαφορετικά λειτουργικά συστήματα εγκαθίστανται σε ξεχωριστές διαμορφώσεις (configurations), σε διαφορετικά τοπικά συστήματα. Το Cross-platform έχει συμπεριληφθεί για τη βελτίωση της λειτουργικότητας και της εμβέλειας του πακέτου διανομής Apache. Λειτουργεί με μια ποικιλία πλατφορμών, συμπεριλαμβανομένων των πακέτων Windows, Linus και MAC OS. [19]

Apache: Ο Apache είναι ένας cross-platform HTTP web server. Χρησιμοποιείται για τη μεταφορά υλικού Ιστού σε όλο τον κόσμο. Εάν κάποιος ζητήσει αρχεία, φωτογραφίες ή έγγραφα χρησιμοποιώντας το πρόγραμμα περιήγησής του, οι HTTP servers θα προβάλλουν τέτοια στοιχεία στους πελάτες.

MySQL Database: Το XAMPP περιλαμβάνει MySQL DBMS. Η MySQL είναι ένα από τα πιο ευρέως χρησιμοποιούμενα συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων. Παρέχει υπηρεσίες αποθήκευσης, χειρισμού, ανάκτησης, διαχείρισης και διαγραφής δεδομένων μέσω του Διαδικτύου. [19]

PHP: Η πλήρης μορφή της PHP είναι Hypertext Preprocessor. Η PHP είναι μια γλώσσα προγραμματισμού backend που χρησιμοποιείται πιο συχνά στην ανάπτυξη Ιστού. Οι χρήστες μπορούν να χρησιμοποιήσουν την PHP για να δημιουργήσουν δυναμικούς ιστότοπους και εφαρμογές. Υποστηρίζει μια ποικιλία συστημάτων διαχείρισης βάσεων δεδομένων και μπορεί να εγκατασταθεί σε οποιαδήποτε πλατφόρμα.

phpMyAdmin: Είναι ένα εργαλείο διαχείρισης βάσης δεδομένων MySQL.(Θα δούμε περισσότερα για το phpMyAdmin στο επόμενο υποκεφάλαιο)

Πίνακας ελέγχου XAMPP: Ο Πίνακας Ελέγχου XAMPP είναι ένας πίνακας που βοηθά στη λειτουργία και τη ρύθμιση άλλων στοιχείων XAMPP.

Filezilla: Είναι ένας διακομιστής πρωτοκόλλου μεταφοράς αρχείων (FTP Server), που διευκολύνει και υποστηρίζει διαδικασίες μεταφοράς αρχείων. [19]

Στα πλαίσια της συγκεκριμένης εφαρμογής το XAMPP έχει παίξει ένα πολύ σημαντικό ρόλο, καθώς χωρίς αυτό δεν θα μπορούσα να κάνω τις προσομοιώσεις και τις δοκιμές τοπικά στον υπολογιστή μου, οι οποίες είναι απαραίτητες για να διαπιστώσω αν έχει υλοποιηθεί με επιτυχία η επικοινωνία των τριών τομέων της εφαρμογής (Back-End, Front-End και Βάσης Δεδομένων). Το XAMPP έχει κάνει μια πλήρη αντικατάσταση ενός πραγματικού web server στα πλαίσια της δοκιμαστικής περιόδου της εφαρμογής.

2.8 phpMyAdmin

Το phpMyAdmin είναι ένα δωρεάν και ανοιχτού κώδικα εργαλείο διαχείρισης για MySQL και MariaDB. Επίσης το phpMyAdmin είναι μια πλήρως ικανή εφαρμογή web, γραμμένη σε php, η οποία επιτρέπει την διαχείριση και επεξεργασίας βάσεων δεδομένων. Παρέχει μια ισχυρή διεπαφή με πολλές επιλογές για τη δημιουργία βάσεων δεδομένων και πινάκων, διαχείριση λογαριασμών βάσης δεδομένων και μη αυτόματη προσθήκη δεδομένων σε μια βάση δεδομένων. [20] Το phpMyAdmin είναι ένα πλήρως εξοπλισμένο εργαλείο διαχείρισης βάσεων δεδομένων.

Ακολουθεί μια σύντομη λίστα με μερικά πράγματα που μπορεί να κάνει η εφαρμογή phpMyAdmin:

1. Δημιουργία, επεξεργασία και διαγραφή βάσεων δεδομένων
2. Διαχείριση λογαριασμών χρηστών και όλων των αδειών τους
3. Εκτέλεση προσαρμοσμένων εντολών SQL απευθείας στην εφαρμογή
4. Μη αυτόματη προσθήκη δεδομένων σε πίνακες βάσης δεδομένων

Οι τύποι βάσεων δεδομένων που μπορούμε να διαχειριστούμε με το phpMyAdmin, είναι MySQL και MariaDB. Η βάση δεδομένων της συγκεκριμένης εφαρμογής στο phpMyAdmin είναι τύπου MySQL.

2.9 Βιβλιοθήκες

Τι είναι Βιβλιοθήκες στον προγραμματισμό ;

Με λίγα λόγια, μια βιβλιοθήκη είναι μια συλλογή προγραμμένου κώδικα που μπορούν να χρησιμοποιήσουν οι προγραμματιστές για τη βελτιστοποίηση εργασιών. Αυτή η συλλογή επαναχρησιμοποιούμενου κώδικα συνήθως στοχεύει σε συγκεκριμένα κοινά προβλήματα.

Στο Κεφάλαιο 2.3 είχαμε αναφέρει τις έτοιμες βιβλιοθήκες που μας παρείχαν η Dart και το Flutter framework. Σε αυτό το υποκεφάλαιο θα εξηγήσουμε τις βιβλιοθήκες (Που δεν είναι τίποτα άλλο παρά Dart κλάσεις) τις οποίες έχω υλοποιήσει στα πλαίσια της συγκεκριμένης εφαρμογής.

Dimensions

Ο λόγος ο οποίος δημιούργησα την βιβλιοθήκη Dimensions είναι για να ομαδοποιήσω όλες τις δυναμικές διαστάσεις, που έχουν σαν σκοπό την δημιουργία μιας δυναμικής διαδικτυακής εφαρμογής (Responsive Design), σε οποιεσδήποτε διαστάσεις οθόνης. Ο τρόπος με τον οποίο μπορούμε να προσθέσουμε την βιβλιοθήκη Dimensions, είναι να την κάνουμε import σε οποιοδήποτε άλλο κομμάτι κώδικα θέλουμε να εφαρμοστεί, γράφοντας την ακόλουθη γραμμή κώδικα :

```
    import 'libraries path.../dimensions.dart';
```

Στον πιο κάτω κώδικα απεικονίζεται η υλοποίηση της dimensions :

```
import 'package:get/get.dart';
class Dimensions {

  static double screenHeight = Get.context!.height;
  static double screenWidth = Get.context!.width;

  //dynamic height

  static double Height10 = screenHeight / 73.745;
  static double Height180 = screenHeight / 4.096;

  //dynamic width

  static double Width8 = screenWidth / 49.09;
  static double Width260 = screenWidth / 1.51;

}
```

Όσον αφορά την βιβλιοθήκη dimensions, εισάγει το Getx package όπως βλέπουμε στην πρώτη γραμμή του κώδικα, το οποίο έχουμε αναφέρει στο κεφάλαιο 2.2.2'Flutter Getx Package'. Στην συνέχεια έχουμε την δήλωση των στατικών μεταβλητών screenHeight και screenWidth, και ανάθεση τιμών, τις οποίες

παίρνουμε από την μέθοδο height και width του Getx package, οι οποίες μας επιστρέφουν το ύψος και μήκος της οθόνης που χρησιμοποιούμε. Επιπλέον δηλώνουμε τις μεταβλητές Height10, Height180 και Width8, Width260 οι οποίες παίρνουν τιμές με βάση το μέγεθος που επιθυμούμε να αντιπροσωπεύει η συγκεκριμένη μεταβλητή, π.χ: Η μεταβλητή Height180 θέλουμε να αντιπροσωπεύει 180 pixels στο ύψος οθόνης του emulator στο οποίο τρέχω τον κώδικα της εφαρμογής. Το ύψος και το μήκος του emulator είναι 737.45 και 392.72 Pixels αντίστοιχα, αυτό είναι και το ύψος που βρίσκεται στην μεταβλητή screenHeight για το emulator. Τώρα, για να πάρουμε αυτό το μέγεθος που αντιστοιχεί με 180 pixels στην συγκεκριμένη οθόνη, το μόνο που έχουμε να κάνουμε είναι να διαιρέσουμε το 737.45 με τον αριθμό 180, το οποίο θα μας δώσει ένα αποτέλεσμα 4.096 που αν το διαιρέσουμε με το μέγεθος της οθόνης θα μας δώσει το επιθυμητό μέγεθος που αντιστοιχεί σε 180 pixels σε οθόνη μεγέθους 737.45 pixels. Με αυτόν τον τρόπο μπορώ να δίνω δυναμικές διαστάσεις σε στοιχεία του κώδικα οι οποίες θα μεταβάλλονται με βάση το μέγεθος της οθόνης που τρέχει η εφαρμογή.

Colors

Η βιβλιοθήκη Colors, ομαδοποιεί κάποια βασικά χρώματα, στα οποία έχω άμεση πρόσβαση από οποιοδήποτε άλλο σημείο του κώδικα, χωρίς να χρειάζεται να ψάχνω η να θυμάμαι την συγκεκριμένη κωδικοποίηση χρώματος που θέλω να χρησιμοποιήσω. Επίσης τα χρώματα παίζουν ένα κύριο ρόλο για τον σχεδιασμό και την μοναδικότητα της εφαρμογής. Ο τρόπος με τον οποίο μπορούμε να προσθέσουμε την βιβλιοθήκη Colors, είναι να την κάνουμε import σε οποιοδήποτε άλλο κομμάτι κώδικα θέλουμε να εφαρμοστεί, γράφοντας την ακόλουθη γραμμή κώδικα :

```
    import 'libraries path.../colors.dart';
```

Στον πιο κάτω κώδικα απεικονίζεται η υλοποίηση της Colors :

```
import 'dart:ui';

class AppColors {
  static final Color textColor = const Color(0xFFccc7c5);
  static final Color paraColor = const Color(0xFF8f837f);
  static final Color signColor = const Color(0xFFa9a29f);
  static final Color titleColor = const Color(0xFF5c524f);
  static final Color mainBlackColor = const Color(0xFF332d2b);
  static final Color buttonBackgroundColor = Color.fromARGB(24, 247, 246, 244);
  static final Color backgroundcolor = Color.fromARGB(255, 51, 51, 51);
  static final Color mainred = Color.fromARGB(255, 147, 0, 0);
  static final Color maiblue = Color.fromARGB(255, 56, 128, 254);
  static final Color liteblack = Color.fromARGB(255, 100, 100, 100);
}
```

Η βιβλιοθήκη Colors, εισάγει την ui βιβλιοθήκη της dart , την οποία έχουμε αναφέρει στο κεφάλαιο 2.3.1'Build in Dart και Flutter Βιβλιοθήκες'. Στην συνέχεια έχουμε την κλάση AppColors, και μέσα σε

αυτήν έχουμε την δήλωση των στατικών μεταβλητών οι οποίες είναι τύπου Color (ο τύπος Color προέρχεται από την UI βιβλιοθήκη της dart, μέσα στην οποία υπάρχει η κλάση Color). Επιπλέον έχουμε την καταχώρηση τιμών σε αυτές τις μεταβλητές, κάποιες από αυτές τις τιμές είναι σε δεκαεξαδικό χρωματικό κώδικα, και κάποιες άλλες είναι σε RGB κώδικα.

Small Text

Η βιβλιοθήκη Small Text μας παρέχει μία προσαρμοσμένη για τις ανάγκες της εφαρμογής μικρή γραμματοσειρά, η οποία θα επαναχρησιμοποιείται καθ'όλη την διάρκεια δημιουργίας της εφαρμογής. Με αποτέλεσμα την μείωση περιττών επαναλήψεων κώδικα και χάσιμο χρόνου. Ο τρόπος με τον οποίο μπορούμε να προσθέσουμε την βιβλιοθήκη Small Text, είναι να την κάνουμε import σε οποιοδήποτε άλλο κομμάτι κώδικα θέλουμε να εφαρμοστεί, γράφοντας την ακόλουθη γραμμή κώδικα :

```
import 'libraries path.../Small_Text.dart';
```

Στον πιο κάτω κώδικα απεικονίζεται η υλοποίηση της Small Text :

```
import 'package:flutter/cupertino.dart';
import 'package:meal_savers/utils/dimensions.dart';

class SmallText extends StatelessWidget {
  Color? color;
  final String text;
  double size;
  double height;

  SmallText(
      {super.key,
      this.color = const Color.fromARGB(255, 188, 188, 188),
      required this.text, this.size = 0, this.height = 1.2});

  @override
  Widget build(BuildContext context) {
    return Text(text, style: TextStyle(
           height: height, color: color, fontFamily: 'Roboto',
           fontWeight: FontWeight.w400,
           fontSize: size == 0 ? Dimensions.Height12 : size));
  }
}
```

Η βιβλιοθήκη Small Text, εισαγωγή την βιβλιοθήκη cupertino του flutter framework, την οποία έχουμε αναφέρει στο κεφάλαιο 2.3.1'Build in Dart και Flutter Βιβλιοθήκες', και την βιβλιοθήκη dimensions που έχουμε εξηγήσει σε αυτό το υποκεφάλαιο. Στην συνέχεια έχουμε την κλάση SmallText, η οποία κάνει επέκταση της κλάσης StatelessWidget (η κλάση StatelessWidget είναι τα Widgets (στα οποία αναφερθήκαμε στο κεφάλαιο 2.1 Flutter Framework), των οποίων η κατάσταση δεν μπορεί να αλλάξει όταν

κατασκευαστούν. Αυτά τα Widgets είναι αμετάβλητα μόλις κατασκευαστούν, δηλαδή οποιαδήποτε αλλαγή στις μεταβλητές, τα εικονίδια, τα κουμπιά ή η ανάκτηση δεδομένων δεν μπορεί να αλλάξει την κατάσταση της εφαρμογής). Στο σώμα της SmallText έχουμε την δήλωση των μεταβλητών: color, text, size και height, το '?' στον τύπο Color της μεταβλητής color κάνει την μεταβλητή να είναι προαιρετική μέσα στον Constructor. Ο Constructor της κλάσης SmallText, αρχικοποιεί και δίνει τιμές στις πιο πάνω μεταβλητές, η παράμετρος text είναι required επειδή δεν αρχικοποιείται στον Constructor, αλλά της δίνει τιμή ο προγραμματιστής όταν δημιουργεί ένα instance της κλάσης. Έπειτα βλέπουμε το Widget build το περιεχόμενο του οποίου γίνεται render στο UI. Επίσης μέσα στο TextStyle, στο fontSize γίνεται ένας έλεγχος. Ο τελεστής '?' σημαίνει αν το size είναι 0 τότε κάνε το size να είναι ίσον με την μεταβλητή Height12 της κλάσης Dimensions, ο τελεστής ':' σημαίνει αλλιώς κάνε το size να είναι αυτό που θα δώσει ο προγραμματιστής σαν παράμετρο σε ένα instance της κλάσης Small Text.

Big Text

Η βιβλιοθήκη Big Text είναι παρόμοια με την βιβλιοθήκη Small Text, η κύρια διαφορά είναι το μέγεθος της γραμματοσειράς. Επίσης όπως και η Small Text επαναχρησιμοποιείται καθ'όλη την διάρκεια δημιουργίας της εφαρμογής. Με αποτέλεσμα την μείωση περιττών επαναλήψεων κώδικα και χάσιμο χρόνου. Ο τρόπος με τον οποίο μπορούμε να προσθέσουμε την βιβλιοθήκη Big Text, είναι να την κάνουμε import σε οποιοδήποτε άλλο κομμάτι κώδικα θέλουμε να εφαρμοστεί, γράφοντας την ακόλουθη γραμμή κώδικα :

```
    import 'libraries path.../Big_Text.dart';
```

Στον πιο κάτω κώδικα απεικονίζεται η υλοποίηση της Big Text :

```
import 'package:flutter/cupertino.dart';
import 'package:meal_savers/utils/dimensions.dart';

class BigText extends StatelessWidget {
  Color? color;
  final String text;
  double size;
  TextOverflow overFlow;
  BigText(
      {super.key,
      this.color = const Color.fromARGB(255, 212, 209, 208),
      required this.text, this.size = 0,
      this.overFlow = TextOverflow.ellipsis});

  @override
  Widget build(BuildContext context) {
    return Text(text,
        maxLines: 1, overflow: overFlow,
        style: TextStyle(
```

```
            fontFamily: 'Roboto', fontWeight: FontWeight.w400,
            color: color,
            fontSize: size == 0 ? Dimensions.Height20 : size));
  }
}
```

Η βιβλιοθήκη Big Text, εισάγει την βιβλιοθήκη cupertino του flutter framework, και την βιβλιοθήκη dimensions. Στην συνέχεια έχουμε την κλάση BigText, η οποία κάνει επέκταση της κλάσης StatelessWidget. Μέσα σε αυτήν έχουμε την δήλωση των μεταβλητών: color, text, size και overFlow. Ο Constructor της κλάσης BigText αρχικοποιεί και δίνει τιμές στις πιο πάνω μεταβλητές. Η παράμετρος text είναι required επειδή δεν αρχικοποιείται στον Constructor, αλλά της δίνει τιμή ο προγραμματιστής όταν δημιουργεί ένα instance της κλάσης. Στην συνέχεια έχουμε το Widget build το περιεχόμενο του οποίου γίνεται render στο UI. Η παράμετρος maxLines της κλάσης Text μας αναφέρει τον αριθμό των γραμμών κειμένου που μπορούμε να γράψουμε με την συγκεκριμένη γραμματοσειρά. Τέλος το overflow εξαφανίζει το κείμενο που ξεπέρασε την παράμετρο maxLines.

App Text Field

Η βιβλιοθήκη App Text Field κατά κύριο λόγο έχει δημιουργηθεί για τις κλάσεις SignInPage και SignUpPage. Η συγκεκριμένη βιβλιοθήκη κάνει render στο UI ένα πεδίο στο οποίο ο χρήστης έχει την δυνατότητα να γράψει κείμενο και δίνει την δυνατότητα στον προγραμματιστή να δημιουργήσει προσαρμοσμένα Icons. Όπως και οι άλλες βιβλιοθήκες, και αυτή έχει βοηθήσει για την μείωση επαναλαμβανόμενου κώδικα, στην απλοποίηση της εφαρμογής και στην βελτίωση του χρόνου εργασίας. Ο τρόπος με τον οποίο μπορούμε να προσθέσουμε την βιβλιοθήκη App Text Field, είναι να την κάνουμε import σε οποιοδήποτε άλλο κομμάτι κώδικα θέλουμε να εφαρμοστεί, γράφοντας την ακόλουθη γραμμή κώδικα :

```
    import 'libraries path.../app_text_field.dart';
```

Στον πιο κάτω κώδικα απεικονίζεται η υλοποίηση της AppTextField :

```
import 'package:flutter/material.dart';
import '../utils/colors.dart';
import '../utils/dimensions.dart';

class AppTextField extends StatelessWidget {

  final String hintText;
  final IconData icon;
  bool isObscure;
  bool maxLines;
  AppTextField(
      {super.key, required this.hintText, required this.icon,
      this.isObscure = false, this.maxLines = false});
```

```
  @override
  Widget build(BuildContext context) {
    return Container(
      margin: Decoration code ...),
      decoration: BoxDecoration( Decoration code ...),
      child: TextField(
        maxLines: maxLines ? 3 : 1,
        obscureText: isObscure ? true : false,
        style: Decoration code ...,
          hintStyle: Decoration code ...,
          hintText: hintText,
           prefixIcon: Icon(icon,color: AppColors.starColor)
    focusedBorder: OutlineInputBorder( Decoration code ...),
    enabledBorder: OutlineInputBorder( Decoration code ...),
     border: OutlineInputBorder( Decoration code ...)));
    }
}
```

Η βιβλιοθήκη App Text Field, εισάγει την βιβλιοθήκη material του flutter framework, την οποία έχουμε αναφέρει στο κεφάλαιο 2.3.1'Build in Dart και Flutter Βιβλιοθήκες', και την βιβλιοθήκη Colors. Στην συνέχεια έχουμε την κλάση AppTextField, η οποία κάνει επέκταση της κλάσης StatelessWidget. Μέσα σε αυτήν έχουμε την δήλωση των μεταβλητών: hintText, icon, isObscure και maxLines. Ο Constructor της κλάσης AppTextField, δηλώνει τις παραμέτρους hintText, icon, και θέτει την κατάσταση false στις δύο boolean μεταβλητές isObscure και maxLines. Επιπλέον έχουμε το Widget build το οποίο κάνει render στο UI τα περιεχόμενα ενός Container (το Container είναι ένα γραφικό στοιχείο widget το οποίο συνδυάζει κοινή ζωγραφική, τοποθέτηση και ταξινόμηση άλλων widget). Το TextField το οποίο είναι παιδί του Container έχει την παράμετρο obscureText στην οποία χρησιμοποιούμε την boolean μεταβλητή isObscure, για να δώσουμε τιμή στην παράμετρο obscureText, αν δεν δώσουμε κάποια τιμή τότε θα ισχύει η τιμή που δώσαμε στον Constructor η οποία είναι false. Παράλληλα το TextField έχει και άλλες παραμέτρους όπως το hintText που είναι τύπου String, το οποίο παίρνουμε από τον προγραμματιστή όταν δίνει τις παραμέτρους σε ένα instance της κλάσης AppTextField. Αυτό το String πρέπει να είναι ένα ενδεικτικό κείμενο το οποίο θα αναφέρει στον χρήστη, το είδος της πληροφορίας που πρέπει να συμπληρώνει στο πεδίο. Μια άλλη παράμετρος του TextField που δεν έχουμε αναφέρει είναι η prefixIcon η οποία παίρνει αντικείμενα τύπου Icon, και ακριβώς για αυτόν το λόγο έχουμε δημιουργήσει την μεταβλητή icon τύπου IconData, για να μπορέσουμε να την δώσουμε σαν τιμή στην παράμετρο prefixIcon, φυσικά αυτήν την τιμή, την δίνει ο προγραμματιστής αργότερα όπως και του hintText.

Icons On Top

Η βιβλιοθήκη Icons On Top μας παρέχει ένα έτοιμο επαναχρησιμοποιήσιμο και προσαρμοσμένο Container, το οποίο θα κάνει render στο UI προσαρμοσμένα Icons. Η συγκεκριμένη κλάση ήταν ιδιαίτερα χρήσιμη για τον λόγο ότι εισάγεται σχεδόν σε όλες τις σελίδες της εφαρμογής. Όπως και οι άλλες βιβλιοθήκες, και αυτή έχει βοηθήσει για την μείωση επαναλαμβανόμενου κώδικα, στην απλοποίηση της εφαρμογής και στην βελτίωση του χρόνου εργασίας. Ο τρόπος με τον οποίο μπορούμε να προσθέσουμε την βιβλιοθήκη Icons On Top, είναι γράφοντας την ακόλουθη γραμμή κώδικα :

```
    import 'libraries path.../icons_on_top.dart';
```

Στον πιο κάτω κώδικα απεικονίζεται η υλοποίηση της IconsOnTop :

```
import 'package:flutter/material.dart';
import 'package:meal_savers/utils/dimensions.dart';
import '../utils/colors.dart';

class IconsOnTop extends StatelessWidget {
  final IconData icon;
  final Color backgroundColor;
  final Color iconColor;
  final double size;
  final double iconSize;

  const IconsOnTop(
      {super.key, required this.icon,
      this.backgroundColor = const Color(0xFFfcf4e4),
      this.iconColor = const Color(0xFF756d54),
      this.size = 40, this.iconSize = 15});

  @override
  Widget build(BuildContext context) {
    return Container(
      width: size, height: size,
      decoration: BoxDecoration(
          borderRadius: Decoration code ...,
          border: Decoration code ...,
          child: Icon(icon, color: iconColor, size: iconSize));
  }
}
```

Η βιβλιοθήκη Icons On Top, εισάγει την βιβλιοθήκη material του flutter framework, dimensions και Colors. Επιπλέον έχουμε την κλάση IconsOnTop, η οποία κάνει επέκταση της κλάσης StatelessWidget. Μέσα στο σώμα της οποίας έχουμε την δήλωση των μεταβλητών: icon, backgroundColor, iconColor,

size και iconSize. Ο Constructor της κλάσης IconsOnTop, αρχικοποιεί και δίνει τιμές στις πιο πάνω μεταβλητές, η παράμετρος icon είναι required επειδή δεν αρχικοποιείται στον Constructor, αλλά της δίνει τιμή ο προγραμματιστής όταν δημιουργεί ένα instance της κλάσης. Επιπρόσθετα έχουμε το Widget build το οποίο κάνει render στο UI τα περιεχόμενα ενός Container, το width και height του οποίου δίνονται από την μεταβλητή size η οποία αρχικοποιείται με την τιμή 40 από τον Constructor, αλλά μπορεί να αλλαχτεί αργότερα από τον προγραμματιστή όταν δίνει τις παραμέτρους σε ένα instance της κλάσης. Τέλος έχουμε την παράμετρο παιδί του Container, η οποία είναι ένα Icon που είναι ένα instance της κλάσης Icon η οποία ανήκει στην βιβλιοθήκη material του flutter. Αυτό το Icon έχει σαν παραμέτρους ένα icon, το χρώμα του icon, το μέγεθος του icon κ.τ.λ, εμείς σε αυτές τις παραμέτρους δίνουμε τιμή με τις μεταβλητές icon, iconColor και iconSize που έχουμε δημιουργήσει. Το αποτέλεσμα της βιβλιοθήκης είναι ότι ο προγραμματιστής που θα την εφαρμόσει στον κώδικα του θα έχει έτοιμο το σχεδιαστικό περιβάλλων της κλάσης αλλά θα μπορεί να αλλάξει τις παραμέτρους με βάση τις ανάγκες του.

Expendable Text

Η βιβλιοθήκη Expendable Text μας βοηθάει για την μείωση επαναλαμβανόμενου κώδικα, για απλοποίηση της εφαρμογής και στην βελτίωση του χρόνου εργασίας. Η Expendable Text δέχεται σαν παράμετρο ένα κείμενο και ελέγχει αν το κείμενο είναι μεγαλύτερο από ένα συγκεκριμένο μέγεθος, αν το κείμενο ξεπερνά το μέγεθος που ορίστηκε, τότε εξαφανίζει ένα μεγάλο ποσοστό του κειμένου και εμφανίζει το μήνυμα ”Show more”, αν ο χρήστης πατήσει σε αυτό το μήνυμα τότε θα εμφανιστεί και το υπόλοιπο κείμενο. Ο τρόπος με τον οποίο μπορούμε να προσθέσουμε την βιβλιοθήκη Expendable Text, είναι να την κάνουμε import σε οποιοδήποτε άλλο κομμάτι κώδικα θέλουμε να εφαρμοστεί, γράφοντας την ακόλουθη γραμμή κώδικα :

```
    import 'libraries path.../expendable_text.dart';
```

Στον πιο κάτω κώδικα απεικονίζεται η υλοποίηση της Expendable Text :

```
import 'package:flutter/material.dart';
import '../utils/colors.dart';
import '../utils/dimensions.dart';
import 'Small_Text.dart';

class ExpendableText extends StatefulWidget {
  final String text;
  const ExpendableText({super.key, required this.text});
  @override
  State<ExpendableText> createState() => _ExpendableTextState();
}
class _ExpendableTextState extends State<ExpendableText> {
  late String firstHalf;
  late String secondHalf;
  bool hiddenText = true;
  double textHeight = Dimensions.screenHeight / 5.63;
```

```
  @override
  void initState() {
    super.initState();
    if (widget.text.length > textHeight) {
      firstHalf = widget.text.substring(0, textHeight.toInt());
      secondHalf =widget.text.substring(textHeight.toInt() + 1,
      widget.text.length);
    } else {
      firstHalf = widget.text;
      secondHalf = "";
    }}
  @override
  Widget build(BuildContext context) {
    return Container(
      child: secondHalf.isEmpty
          ? SmallText(size: Dimensions.Height14, text: firstHalf)
          : Column( crossAxisAlignment: CrossAxisAlignment.start,
              children: [ SmallText(
text: hiddenText ? (firstHalf + "..."):(firstHalf + secondHalf)),
                InkWell( onTap: () { setState(() {
                     hiddenText = !hiddenText;
                   });},
                  child: Row(children: [SmallText(
                       text: "Show more" ),
                     Icon(
                       hiddenText
                           ? Icons.keyboard_arrow_down
                           : Icons.keyboard_arrow_up,
              )}
```

Η βιβλιοθήκη Expendable Text, εισάγει την βιβλιοθήκη material του flutter framework, την βιβλιοθήκη dimensions, Colors και SmallText. Στην συνέχεια έχουμε την κλάση ExpendableText, η οποία κάνει επέκταση την κλάση StatefulWidget (ένα Stateful Widget είναι δυναμικό: για παράδειγμα, μπορεί να αλλάξει την εμφάνισή του ως απόκριση σε συμβάντα που προκαλούνται από τις αλληλεπιδράσεις των χρηστών ή όταν λαμβάνει δεδομένα). Μέσα στο σώμα της κλάσης ExpendableText βλέπουμε την δήλωση της μεταβλητής: text η οποία είναι final και τύπου String, στην συνέχεια έχουμε τον Constructor της κλάσης ExpendableText. Επιπλέον στην ExpendableText έχουμε την abstract κλάση state η οποία κάνει επέκταση την StatefulWidget και είναι τύπου generic ExpendableText, η κλάση state υλοποιεί την μέθοδο createState() η οποία δέχεται σαν παράμετρο ένα instance υποκλάσεων της κλάσης state. Στην συνέχεια δηλώνουμε την private κλάση ExpendableTextState που επεκτείνει την κλάση State, μέσα σε αυτήν έχουμε την δήλωση των μεταβλητών: late (με αυτήν την λέξη-κλειδί δηλώνουμε μεταβλητές που θα αρχικοποιηθούν αργότερα), firstHalf, secondHalf, την boolean hiddenText όπου αρχικοποιείται σε 'ture', και την textHeight που είναι τύπου double και παίρνει την τιμή που προκύπτει από την διαίρεση

της μεταβλητής screenHeight της κλάσεις Dimensions και του αριθμού 5.63. Στην συνέχεια έχουμε την void μέθοδο initState(). Μέσα στην initState έχουμε μία if της οποίας η συνθήκη είναι : Αν το μέγεθος το οποίο παίρνει στην οθόνη η μεταβλητή text της κλάσης ExpendableText είναι μεγαλύτερο από την μεταβλητή textHeight τότε δώσε στην μεταβλητή firstHalf το κείμενο της text το οποίο ξεκινά από τον αριθμό 0 και τελειώνει στο textHeight, παράλληλα δώσε στην μεταβλητή secondHalf το κείμενο της text το οποίο ξεκινά από το textHeight+1 και τελειώνει όταν τελειώσει η μεταβλητή text. Αλλά αν η συνθήκη της if είναι false τότε δώσε στην μεταβλητή firstHalf την τιμή της μεταβλητής text, και στην secondHalf δώσε ένα άδειο String. Έπειτα έχουμε το Widget build το οποίο κάνει render στο UI τα περιεχόμενα ενός Container. Στην παράμετρο παιδί του Container έχουμε μία συνθήκη η οποία ελέγχει αν η μεταβλητή secondHalf είναι άδεια χρησιμοποιώντας την boolean build-in μεταβλητή της Dart 'isEmpty', τότε επέστρεψε μου ένα instance της κλάσης SmallText η οποία θα έχει σαν παραμέτρους το κείμενο από την μεταβλητή firstHalf με μέγεθος γραμματοσειράς το περιεχόμενο της μεταβλητής Height14 της κλάσης Dimensions. Αν ο έλεγχος της μεταβλητής isEmpty είναι false δηλαδή η secondHalf δεν είναι άδεια τότε επέστρεψε ένα Column widget, σε αυτό το widget έχουμε ένα instance της κλάσης SmallText το οποίο έχει μία συνθήκη στην παράμετρο text που ελέγχει αν η μεταβλητή hiddenText είναι true. Αν η μεταβλητή είναι 'true' τότε δώσε στην μεταβλητή text την τιμή της μεταβλητής firstHalf + '...', αν η hiddenText είναι false τότε δώσε στην text την τιμή των μεταβλητών firstHalf και secondHalf. Επιπλέον έχουμε την InkWell η οποία είναι κλάση της βιβλιοθήκης material και διαθέτει το onTap event το οποίο με την βοήθεια της μεθόδου setState μπορεί να αλλάξει την κατάσταση της μεταβλητής hiddenText από true σε false, το οποίο έχει σαν αποτέλεσμα ή την εμφάνιση όλου του κειμένου στην παράμετρο text ή την εμφάνιση κάποιου μέρους του κειμένου μαζί με τις τρεις τελείες.

Τέλος η κλάση InkWell έχει σαν παιδί ένα Row widget το οποίο είναι υπεύθυνο για το render μίας οριζόντιας γραμμής στοιχείων όπου στα πλαίσια αυτού του widget ονομάζονται 'Children', η Row έχει δύο στοιχεία, ένα instance της κλάσης SmallText όπου μας δείχνει το μήνυμα 'Show more' , και μία κλάση Icon η οποία υλοποιεί την συνθήκη που ελέγχει αν η μεταβλητή hiddenText είναι true τότε απεικονίζει ένα Icon και αν είναι false, απεικονίζει κάποιο άλλο Icon.

Κεφάλαιο 3ο: Υλοποίηση του MealSavers

Στα πλαίσια αυτού του κεφαλαίου, το οποίο πιστεύω πως είναι το πιο σημαντικό για την κατανόηση της υλοποίησης της εφαρμογής. Θα παρουσιαστεί αναλυτικά με ποιο τρόπο έχει γίνει η υλοποίηση της εφαρμογής, αρχίζοντας από την αρχιτεκτονική της. Στην συνέχεια θα εξηγήσουμε την υλοποίηση του frontend μέρους της εφαρμογής καθώς και όλες της κλάσεις οι οποίες έχουν παίξει καθοριστικό ρόλο για την λειτουργία της εφαρμογής. Επίσης θα εξηγήσουμε όλες τις κλάσεις που είχαν σημαντικό ρόλο για την δημιουργία του backend, και πώς έχει γίνει η υλοποίηση κάποιων σημαντικών λειτουργιών του, όπως το Routing, η δημιουργία των API, η δημιουργία των Endpoints και ο έλεγχος αυθεντικοποίησης. Τέλος θα δούμε την υλοποίηση της βάσης δεδομένων.

3.1 Αρχιτεκτονική Εφαρμογής

Η αρχιτεκτονική της εφαρμογής περιγράφει τα μοτίβα και τις τεχνικές που χρησιμοποιήθηκαν για το σχεδιασμό και την κατασκευή της εφαρμογής. Η αρχιτεκτονική μας παρέχει έναν οδικό χάρτη και βέλτιστες πρακτικές που μπορούμε να ακολουθήσουμε κατά τη δημιουργία της εφαρμογής, έτσι ώστε να καταλήξουμε σε μια καλά δομημένη εφαρμογή. Ως μέρος μιας αρχιτεκτονικής εφαρμογών, θα υπάρχουν υπηρεσίες τόσο front-end όσο και back-end. Η ανάπτυξη front-end αφορά την εμπειρία χρήστη της εφαρμογής, ενώ η ανάπτυξη back-end επικεντρώνεται στην παροχή πρόσβασης στα δεδομένα, τις υπηρεσίες και άλλα υπάρχοντα συστήματα που κάνουν την εφαρμογή να λειτουργεί.

Οι σύγχρονες αρχιτεκτονικές εφαρμογών όπως στοχεύουμε να είναι και η συγκεκριμένη αρχιτεκτονική της εφαρμογής μας, είναι πιο συχνά χαλαρά συνδεδεμένες, χρησιμοποιώντας μικροϋπηρεσίες και application programming interfaces (APIs), για τη σύνδεση υπηρεσιών, οι οποίες παρέχουν τη βάση για εγγενείς εφαρμογές στο cloud.

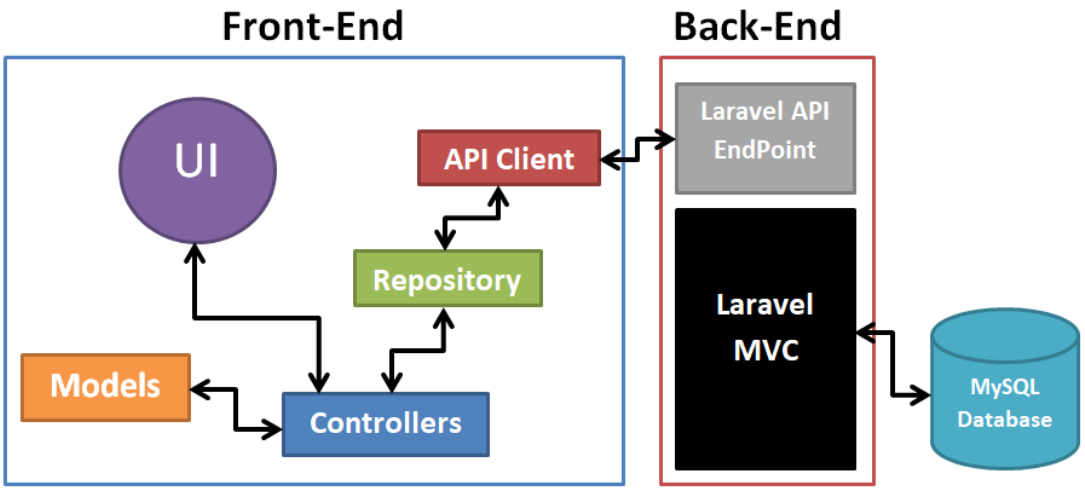
Στα υποκεφάλαια 3.1.1 και 3.1.2 θα εξηγήσουμε την υλοποίηση των αρχιτεκτονικών του Frontend και Backend αντίστοιχα. Αλλά πριν από αυτά θα ήθελα να ρίξουμε μια ματιά στην κλάση App Constants γιατί θεωρώ πως θα βοηθήσει στην καλύτερη κατανόηση των κλάσεων που την εισάγουν, στις οποίες θα αναφερθούμε στην συνέχεια αυτής της εργασίας.

Κλάση App Constants:

```
class AppConstants {
  static const String APP_NAME = "MealSavers";
  static const int APP_VERSION = 1;
  //PRODUCT ENDPOINTS
  static const String BASE_URL = "http://10.0.2.2:8000";
  static const String POPULAR_PRODUCT_URI = "/api/v1/products/popular";
  static const String RECOMMENDED_PRODUCT_URI = "/api/v1/products/recommended";
  static const String UPLOAD_URL = "/uploads/";
  //ORDERS ENDPOINTS
  static const String PLACE_ORDER_URI = "/api/v1/customer/order/place";
  //AUTH ENDPOINTS
  static const String REGISTRATION_URI = "/api/v1/auth/register";
  static const String LOGIN_URI = "/api/v1/auth/login";
  //USER ENDPOINTS
  static const String USER_INFO_URI = "/api/v1/customer/info";
  static const String TOKEN = "";
  static const String EMAIL = "";
  static const String PASSWORD = "";
  static const String CART_LIST = "Cart-List";
  static const String CART_HISTORY_LIST = "Cart-History-List";}
```

Η Κλάση App Constants περιέχει τις σταθερές της εφαρμογής, πράγμα το οποίο μας διευκολύνει σημαντικά, μειώνει τον χρόνο εργασίας και κάνει καλύτερη την οργάνωση της εφαρμογής.

3.1.1 Αρχιτεκτονική Front-End



Σχήμα 3.1: Αρχιτεκτονική Frontend.

Περιγραφή του Σχήματος 3.1 :

Αρχίζουμε με την κλάση API Client του frontend η οποία θα έχει το σώμα των μεθόδων GET και POST Request, που παίρνουν και στέλνουν δεδομένα στον server (στον server υλοποιείται ο κώδικας του backend και της βάση δεδομένων της εφαρμογής, το frontend είναι ο κώδικας που υλοποιείται στην συσκευή, στην οποία κατεβάζει ο χρήστης την εφαρμογή). Στην συνέχεια έχουμε το Repository που είναι απλά μία κλάση που καλεί τις μεθόδους GET και POST, που υλοποιούνται στην API κλάση. Η Κλάση Controller επεξεργάζεται τα δεδομένα που παίρνει από την αντίστοιχη κλάση Repository. Επίσης είναι η κλάση που θα καλείται από το UI για να παρουσιαστούν τα δεδομένα. Τέλος τα models κάνουν την μετατροπή των δεδομένων από dart (που χρησιμοποιούμε στο frontend), σε json (που χρησιμοποιεί ο server), και το αντίθετο.

Σε αυτό το υποκεφάλαιο θα εξηγήσουμε αναλυτικά και με παραδείγματα της εφαρμογής, όλα τα στάδια της αρχιτεκτονικής του Frontend η οποία απεικονίζεται στο Σχήμα 3.1. Αρχίζοντας από το API Client :

API Client:

Το API Client δεν είναι τίποτα άλλο παρά μία Dart κλάση όπου ο κύριος σκοπός της είναι να υλοποιεί το σώμα των μεθόδων GET και POST Request, οι οποίες παραλαμβάνουν και στέλνουν δεδομένα στον server.

Κλάση ApiClient :

```
import 'package:get/get.dart';
import 'package:shared_preferences/shared_preferences.dart';
import '../../utils/app_constants.dart';

class ApiClient extends GetConnect implements GetxService {
  late String token;
  final String appBaseUrl;
```

```
  late SharedPreferences sharedPreferences;
  late Map<String, String> _mainHeaders;

  ApiClient({required this.appBaseUrl, required this.sharedPreferences}) {
    timeout = Duration(seconds: 30);
    token = sharedPreferences.getString(AppConstants.TOKEN) ?? "";
    _mainHeaders = {
      'Content-type': 'application/json; charset=UTF-8',
      'Authorization': 'Bearer $token '
    };
  }

  Future<Response> getData(String uri, {Map<String, String>? headers}) async {
    try {
      Response response = await get(uri, headers: headers ?? _mainHeaders);
      return response;
    } catch (e) {
      return Response(statusCode: 1, statusText: e.toString());
    }
  }
  Future<Response> postData(String uri, dynamic body) async {
    try {
      Response response = await post(uri, body, headers: _mainHeaders);
      return response;
    } catch (e) {
      return Response(statusCode: 1, statusText: e.toString());
    }}
}
```

Η κλάση ApiClient, εισάγει το Getx package όπως βλέπουμε στην πρώτη γραμμή του κώδικα, το οποίο έχουμε αναφέρει στο κεφάλαιο 2.2.2'Flutter Getx Package', επίσης εισάγει το shared preferences package το οποίο έχουμε αναφέρει στο κεφάλαιο 2.3.1, και την κλάση App Constants. Στην συνέχεια έχουμε την κλάση ApiClient, η οποία κάνει επέκταση την κλάση GetConnect που είναι μία από τις κλάσης που μας παρέχει το Getx Package το οποίο είναι ένας εύκολος τρόπος υποβολής αιτημάτων και λήψης δεδομένων. Επιπλέον το GetConnect υλοποιεί τις μεθόδους GET, POST, PUT, DELETE, SOCKET οι οποίες είναι απαραίτητες για την επικοινωνία με το Rest API που δημιουργούμε στο Backend. Παράλληλα η κλάση ApiClient κάνει implement την GetxService που είναι επίσης από το Getx Package, και έχει σαν σκοπό να μας δώσει την δυνατότητα αποθήκευσης δεδομένων τοπικά στην συσκευή. Χρησιμοποιούμε την GetxService για να μπορέσουμε να αποθηκεύσουμε τοπικά το token το οποίο παίρνουμε από τον server, που είναι υπεύθυνο για την αυθεντικοποίηση των δεδομένων. Επιπλέον έχουμε τις μεταβλητές token, appBaseUrl, sharedPreferences, mainHeaders. Η κάτω παύλα στην mainHeaders κάνει την μεταβλητή Private, επίσης η mainHeaders είναι τύπου Map η οποία παίρνει δύο παραμέτρους 'key' και 'value, που στην συγκεκριμένη περίπτωση είναι και οι δύο String. Το περιεχόμενο της μεταβλητής mainHeaders είναι

η κεφαλίδα η οποία είναι απαραίτητη για να επιτευχθεί επικοινωνία με τον server, είτε POST, είτε GET request. Το περιεχόμενο της κεφαλίδας αναφέρει τον τύπο δεδομένων, το σύστημα κωδικοποίησης και το token αυθεντικοποιησης. Επιπρόσθετα έχουμε τον Constructor της κλάσης ApiClient που θέτει σαν παραμέτρους τις μεταβλητές appBaseUrl και sharedPreferences και δίνει τιμές στις πιο κάτω μεταβλητές :

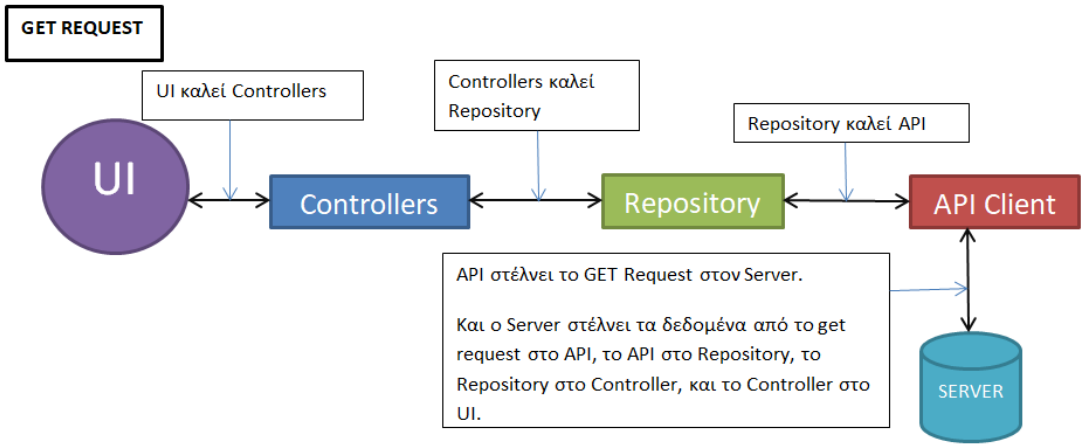
Στην μεταβλητή timeout δίνουμε την τιμή της κλάσης Duration στην παράμετρο seconds 30, πράγμα το οποίο σημαίνει πως όποια σύνδεση χρησιμοποιεί την μεταβλητή για παράμετρο χρόνου διάρκειας, τότε η διάρκεια της σύνδεσης θα είναι 30 δευτερόλεπτα.

Στην μεταβλητή token δίνουμε την τοπικά αποθηκευμένη τιμή της μεταβλητής TOKEN η οποία βρίσκεται στην κλάση AppConstants όπου έχουμε πρόσβαση μέσω της μεταβλητής sharedPreferences. Τα σύμβολα '??' είναι μία συνθήκη η οποία ελέγχει αν υπάρχουν τοπικά αποθηκευμένα δεδομένα στη μεταβλητή TOKEN, αν υπάρχουν τότε τα δίνει σαν τιμή στην μεταβλητή token, αλλά αν δεν υπάρχουν της δίνει την τιμή άδειο String.

Στην μεταβλητή mainHeaders η οποία είναι τύπου Map, δίνουμε δύο παραμέτρους key, το 'Content-type' και 'Authorization', για το Content-type δίνουμε value τον τύπο των δεδομένων που θα μεταφερθούν σε αυτή τη σύνδεση που είναι 'application/json' με σύστημα κωδικοποίησης 'charset=UTF-8', και για το Authorization δίνουμε σαν value την μεταβλητή token η οποία έχει τοπικά αποθηκευμένο τον κώδικα αυθεντικοποίησης.

getData

Γίνεται υλοποίηση της μεθόδου getData η οποία είναι η μέθοδος που υλοποιεί το GET request, δίνοντας μας την δυνατότητα να παίρνουμε δεδομένα από τον server. Η μέθοδος getData είναι τύπου Future που σημαίνει πως έχει δύο καταστάσεις, 'ολοκληρωμένη' και 'μη ολοκληρωμένη', όταν είναι ολοκληρωμένη επιστρέφει το περιεχόμενο της, όταν δεν είναι ολοκληρωμένη επιστρέφει μία εξαίρεση. Για αυτόν ακριβώς τον λόγο χρειάζεται και ο τροποποιητής async για να δείξει πως η εντολή είναι ασύγχρονη δηλαδή θα εκτελεστή όταν επιστρέψει αποτέλεσμα και όχι με την σειρά της. Επίσης η abstract κλάση Future είναι τύπου Response το οποίο παίρνουμε από την GetConnect. Στα πλαίσια αυτής της εργασίας δεν θα χρησιμοποιήσουμε το πακέτο http client αλλά το πακέτο Getx για να πάρουμε δεδομένα από τον server (φυσικά και το πακέτο Getx ενσωματώνει το http πρωτόκολλο). Η Getx επιστέφει αυτό το response στο οποίο είναι μέσα τα δεδομένα. Η μέθοδος getData έχει δύο παραμέτρους, το uri τύπου String και headers τύπου Map, η παράμετρος 'uri' θα παίρνει σαν τιμή ένα URL path, και όταν ενώνεται με την μεταβλητή baseUrl η οποία θα έχει το host URL του backend, θα δίνει την πλήρες διαδρομή. Ο τελεστής '?' μετά από το Map έχει γραφτεί για να μας αναφέρει ότι η παράμετρος headers μπορεί να είναι και άδεια, επίσης οι αγκύλες στις οποίες είναι μέσα η παράμετρος τύπου Map, δηλώνουν πως η παράμετρος headers είναι προαιρετική. Στην συνέχεια μέσα στην try statement κάνουμε την δήλωση της μεταβλητής response που είναι τύπου Response και της δίνουμε τα δεδομένα που παίρνουμε από τον server με την μέθοδο get της κλάσης GetConnect. Στην μέθοδος get βάζουμε σαν παραμέτρους τις παραμέτρους 'uri' και headers της κλάσης getData, και αν δεν δοθεί τιμή όταν κληθεί η μέθοδος getData στην παράμετρο headers, τότε θα πάρει την τιμή της μεταβλητής mainHeaders. Τέλος αν όλα πάνε καλά η μέθοδος getData επιστρέφει την μεταβλητή response που έχει μέσα της τα δεδομένα από τον server, τα οποία καταλήγουν στο UI ακολουθώντας την διαδρομή που απεικονίζει το Σχήμα 3.2.



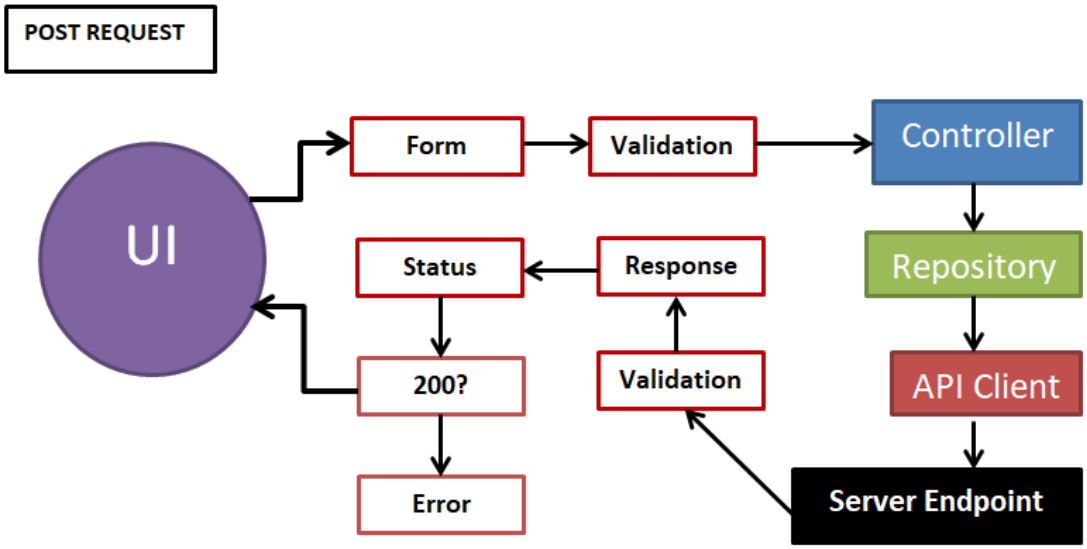
Σχήμα 3.2: Αρχιτεκτονική GET request.

postData

Στην συνέχεια γίνεται υλοποίηση της μεθόδου postData η οποία είναι η μέθοδος που υλοποιεί το POST request, δίνοντας μας την δυνατότητα να στέλνουμε δεδομένα από το frontEnd στον server. Η μέθοδος postData είναι τύπου Future, επίσης η abstract κλάση Future είναι τύπου Response το οποίο παίρνουμε από την GetConnect. Η μέθοδος postData έχει δύο παραμέτρους, το uri τύπου String και body τύπου dynamic δηλαδή μπορεί να πάρει τιμή οποιουδήποτε τύπου. Επιπρόσθετα μέσα στην try statement κάνουμε την δήλωση της μεταβλητής response που είναι τύπου Response και της δίνουμε τα δεδομένα τα οποία θέλουμε να στείλουμε στον server με την μέθοδο post της κλάσης GetConnect.

Στην μέθοδος post βάζουμε σαν παραμέτρους τις παραμέτρους 'uri', body και headers της μεθόδου postData, στην παράμετρο body θα βάζουμε τα δεδομένα που θέλουμε να στείλουμε στον server, και η παράμετρος headers θα έχει την κεφαλίδα. Αν η try statement αποτύχει, δηλαδή έχει γίνει κάποιο σφάλμα κατά την διάρκεια της όλης διαδικασίας τότε θα εκτελεστεί η catch statement που επιστρέφει ένα Exception το οποίο έχει status code 1 και ένα κείμενο το οποίο μας περιγράφει την εξαίρεση.

Τέλος αν όλα πάνε καλά η μέθοδος postData επιστρέφει την μεταβλητή response που έχει μέσα της την διαδρομή, κεφαλίδα και τα δεδομένα τα οποία θα σταλούν στον server. Ο Server με την σειρά του αν δεν γίνει κάποιο σφάλμα, στέλνει ένα response status code '200 OK' το οποίο σημαίνει ότι έχει παραλάβει τα δεδομένα. Η διαδρομή που ακολουθεί το POST request απεικονίζεται στο Σχήμα 3.3.



Σχήμα 3.3: Αρχιτεκτονική POST request.

Στην συνέχεια θα εξηγήσουμε πώς οι κλάσεις Repository καλούν τις μεθόδους getData και postData που υλοποιούνται στην κλάση ApiClient.

Repository

Το Repository είναι Dart κλάσεις όπου ο κύριος σκοπός τους είναι να καλούν και να δίνουν παραμέτρους στις μεθόδους GET και POST που υλοποιούνται στην κλάση ApiClient. Ακολούθως θα αναφέρουμε δύο παραδείγματα κλάσεων Repository που έχουν υλοποιηθεί στα πλαίσια της εφαρμογής, μία κλάση Repository η οποία καλεί την μέθοδο GET, και μία η οποία καλεί την μέθοδο POST.

Κλάση RecommendedProtuctRepo :

```
import 'package:get/get.dart';
import '../../utils/app_constants.dart';
import '../api/api_client.dart';

class RecommendedProtuctRepo extends GetxService {
  final ApiClient apiClient;

  RecommendedProtuctRepo({required this.apiClient});

  Future<Response> getRecommendedProductList() async {
    return await apiClient.getData(AppConstants.RECOMMENDED_PRODUCT_URI);
  }
}
```

Η κλάση RecommendedProtuctRepo είναι μία Repository κλάση που καλεί και δίνει παραμέτρους στην GET μέθοδο της κλάσης ApiClient. Η κλάση RecommendedProtuctRepo εισάγει το Getx package, την κλάση App Constants, και την κλάση ApiClient την οποία έχουμε αναλύσει στα πλαίσια αυτού του κεφαλαίου.

Επιπλέον έχουμε την κλάση RecommendedProtuctRepo η οποία κάνει επέκταση την GetxService που έχουμε ξαναδεί στην κλάση ApiClient. Μέσα στο σώμα της κλάσης RecommendedProtuctRepo έχουμε την δήλωση της μεταβλητής apiClient η οποία είναι final και τύπου ApiClient, επειδή η dart είναι αντικειμενοστραφής γλώσσα χειρίζεται όλες τις μεταβλητές σαν αντικείμενα, αυτό σημαίνει μέσω της μεταβλητής apiClient έχουμε πρόσβαση στις μεθόδους της κλάσης ApiClient και αυτός είναι ο λόγος δημιουργίας της συγκεκριμένης μεταβλητής. Επιπρόσθετα έχουμε τον Constructor ο οποίος αρχικοποιεί και θέτει αναγκαστική την απόδοση τιμής στην παράμετρο apiClient από οποιαδήποτε άλλη κλάση που υλοποιεί ένα instance της κλάσης RecommendedProtuctRepo. Στην συνέχεια υλοποιούμε την μέθοδο getRecommendedProductList η οποία είναι τύπου Future που σημαίνει πως έχει δύο καταστάσεις, 'ολοκληρωμένη' και 'μη ολοκληρωμένη', όταν είναι ολοκληρωμένη επιστρέφει το περιεχόμενο της, όταν δεν είναι ολοκληρωμένη επιστρέφει μία εξαίρεση, για αυτόν ακριβώς τον λόγο χρειάζεται και ο τροποποιητής async για να δείξει πως η εντολή είναι ασύγχρονη δηλαδή θα εκτελεστή όταν επιστρέψει αποτέλεσμα και όχι με την σειρά της. Επίσης η abstract κλάση Future είναι τύπου Response. Η μέθοδος getRecommendedProductList μας επιστρέφει τα json δεδομένα τα οποία παίρνει από το συγκεκριμένο API endpoint που δημιουργήσαμε στο backend, του οποίου μας δίνει την διαδρομή η μεταβλητή RECOMMENDED-PRODUCT-URI της κλάσης AppConstants, που είναι η παράμετρος 'uri' τύπου String της μεθόδου getData, στην οποία έχουμε πρόσβαση μέσο του αντικειμένου apiClient. Εν κατακλείδι η μέθοδος getRecommendedProductList έχει τα δεδομένα τα οποία παίρνει με τη συγκεκριμένη παράμετρο 'uri' από τον server, και στην συνέχεια αυτά τα δεδομένα τα παίρνουν οι controllers, τα επεξεργάζονται και τα απεικονίζουν στο UI.

Κλάση OrderRepo :

```
import 'package:meal_savers/data/api/api_client.dart';
import 'package:get/get.dart';
import 'package:meal_savers/models/place_order_model.dart';
import 'package:meal_savers/utils/app_constants.dart';

class OrderRepo {

  final ApiClient apiClient;

  OrderRepo({required this.apiClient});

  Future<Response> placeOrder(PlaceOrderBody placeOrder) async {
    return await apiClient.postData(
        AppConstants.PLACE_ORDER_URI, placeOrder.toJson());
  }
}
```

Η κλάση OrderRepo είναι μία Repository κλάση που καλεί και δίνει παραμέτρους στην POST μέθοδο της κλάσης ApiClient. Η κλάση OrderRepo εισάγει το Getx package, την κλάση App Constants, την κλάση ApiClient και την κλάση μοντέλου 'place order model' την οποία θα εξηγήσουμε αναλυτικά αργότερα στο κεφάλαιο (προς το παρόν το μόνο που χρειάζεται να ξέρουμε για αυτή την κλάση είναι πως υλοποιεί κάποιες μεθόδους οι οποίες μετατρέπουν json δεδομένα σε dart δεδομένα και το αντίθετο).

Επιπλέον έχουμε την κλάση OrderRepo, μέσα στο σώμα της οποίας κάνουμε την δήλωση της μεταβλητής apiClient η οποία είναι final και τύπου ApiClient, αυτό σημαίνει μέσω της μεταβλητής apiClient έχουμε πρόσβαση στις μεθόδους της κλάσης ApiClient. Στην συνέχεια έχουμε τον Constructor ο οποίος αρχικοποιεί και θέτει αναγκαστική την απόδοση τιμής στην παράμετρο apiClient από οποιαδήποτε άλλη κλάση που υλοποιεί ένα instance της κλάσης OrderRepo. Παράλληλα υλοποιούμε την μέθοδο placeOrder η οποία είναι τύπου Future. Επίσης η abstract κλάση Future είναι τύπου Response το οποίο παίρνουμε από το Getx package. Η μέθοδος placeOrder έχει την παράμετρο placeOrder που δέχεται τα δεδομένα που θα σταλούν στον server από τον αντίστοιχο controller, αυτά τα δεδομένα πρέπει να έχουν κάποια συγκεκριμένη μορφή η οποία καθορίζεται από την κλάση PlaceOrderBody που βρίσκεται μέσα στο μοντέλο 'place order model'. Η μέθοδος placeOrder χρησιμοποιώντας την μέθοδο postData της κλάσης ApiClient στην οποία έχουμε πρόσβαση μέσω της μεταβλητής apiClient, στέλνει τα δεδομένα της παραμέτρου placeOrder στον server. Η τοποθεσία που θα πάνε τα δεδομένα καθορίζεται από την παράμετρο 'uri' της μεθόδου postData η οποία έχει την τιμή της μεταβλητής PLACE ORDER URI που ανήκει στην κλάση AppConstants. Φυσικά στον server δεν μπορούμε να στείλουμε dart δεδομένα, για αυτόν το λόγο χρησιμοποιούμε την μέθοδο toJson() της κλάσης PlaceOrderBody στην οποία έχουμε πρόσβαση μέσω της μεταβλητής placeOrder που είναι τύπου PlaceOrderBody, η οποία μετατρέπει τα dart δεδομένα σε json δεδομένα.

models

Τα models είναι Dart κλάσεις οι οποίες υλοποιούν τις μεθόδους fromJson και toJson, που μετατρέπουν json δεδομένα σε dart και dart δεδομένα σε json αντίστοιχα, ο σκοπός των μοντέλων είναι σε περίπτωση GET request, να διαμορφώνουν τα δεδομένα από την μορφή που έχουν στην βάση δεδομένων σε dart κλάσεις. Αυτές οι dart κλάσεις θα επεξεργάζονται από τους controllers που ακολούθως θα καλούνται από το UI για να παρουσιάσουν τα δεδομένα. Επίσης σε περίπτωση POST request τα μοντέλα είναι αυτά που κάνουν την διαμόρφωση από dart κλάση σε μορφή η οποία θα γίνει αποδέκτη από την βάση δεδομένων.

Μοντέλο Products Model :

```
class Product {
  int? _totalSize; int? _typeId; int? _offset;
  late List<ProductModel> _products;

  List<ProductModel> get products => _products;
  Product(
      {required totalSize,
      required typeId, required offset, required products}) {
    this._totalSize = totalSize; this._typeId = typeId;
    this._offset = offset;this._products = products;
```

```
  }
  Product.fromJson(Map<String, dynamic> json) {
    _totalSize = json['total_size'];
    _typeId = json['type_id']; _offset = json['offset'];
    if (json['products'] != null) {
      _products = <ProductModel>[];
      json['products'].forEach((v) {
        _products.add(ProductModel.fromJson(v));
      });
    }}

 }

class ProductModel {
  int? id; String? name; String? description;
  int? price; String? img; int? quantity; String? location;
  String? createdAt; String? updatedAt; int? typeId;

  ProductModel(
      {this.id, this.name, this.description, this.price, this.img,
      this.quantity, this.location, this.createdAt, this.updatedAt,
      this.typeId});

  ProductModel.fromJson(Map<String, dynamic> json) {
    id = json['id']; name = json['name'];
    description = json['description']; price = json['price'];
    img = json['img']; quantity = json['quantity'];
    location = json['location']; createdAt = json['created_at'];
    updatedAt = json['updated_at']; typeId = json['type_id'];
  }
  Map<String, dynamic> toJson() {
    final Map<String, dynamic> data = new Map<String, dynamic>();
    data['id'] = this.id; data['name'] = this.name;
    data['description'] = this.description;
    data['price'] = this.price; data['img'] = this.img;
    data['quantity'] = this.quantity; data['location'] = this.location;
    data['created_at'] = this.createdAt; data['updated_at'] = this.updatedAt;
    data['type_id'] = this.typeId;
    return data;
  }
}
```

Το μοντέλο Products Model αποτελείται από δύο κλάσεις, τις κλάσεις Product και ProductModel. Στην κλάση ProductModel δηλώνουμε τις μεταβλητές: int id, name, description, price, img, quantity, location, createdAt, updatedAt και typeId, για όλες αυτές τις μεταβλητές υπάρχει μια αντίστοιχη στήλη στο table foods στην βάση δεδομένων της εφαρμογής. Επιπρόσθετα έχουμε τον Constructor της κλάσης ProductModel ο οποίος αρχικοποιεί τις μεταβλητές της κλάσης. Επιπλέον έχουμε την μέθοδο fromJson, η οποία έχει την παράμετρο json τύπου Map με τον τύπο της'key' παραμέτρου να είναι 'String', και της 'value' παραμέτρου να είναι dynamic. Η παράμετρος json της μεθόδου είναι απαραίτητο να είναι τύπου Map για τον λόγο ότι ο τύπος Map της Dart έχει ακριβώς την ίδια δομή με json που είναι επίσης 'Key' και 'value'. Με αυτόν τον τρόπο γίνεται δυνατή η μετατροπή δεδομένων json σε Map, και από Map σε json. Στο σώμα της μεθόδου βλέπουμε την καταχώριση τιμών στις μεταβλητές της κλάσης από τις αντίστοιχες στήλες τους στην βάση δεδομένων, με αυτόν τον τρόπο επιτυγχάνεται η μετατροπή των δεδομένων από json σε Map. Στην συνέχεια έχουμε την μέθοδο toJson η οποία είναι τύπου Map, στο σώμα της δημιουργούμε το αντικείμενο data της κλάσης Map του οποίου στην παράμετρο key μπαίνουν τα ονόματα των στηλών του table foods της βάσης δεδομένων, και στην παράμετρο value παίρνουν τις τιμές που τους δίνονται από τον αντίστοιχο controller. Η μέθοδος toJson επιστρέφει το αντικείμενο data το οποίο έχει τα δεδομένα που θέλουμε να στείλουμε στον server. Στην κλάση Products δηλώνουμε τις ιδιωτικές int μεταβλητές: totalSize, typeId και offset, επίσης δηλώνουμε την ιδιωτική late τύπου list μεταβλητή products η οποία παίρνει μία λίστα τύπου ProductModel. Επιπρόσθετα έχουμε την build-in μέθοδο get της Dart, η οποία κάνει την ίδια δουλεία με την getter και setter της java, που στην ουσία μας δίνει πρόσβαση στην ιδιωτική μεταβλητή products όταν την καλούμε από άλλες κλάσεις. Ο τελεστής '=>' δεν είναι τίποτα άλλο παρά ένας πιο γρήγορος τρόπος να γράψεις ' return'. Στην συνέχεια έχουμε τον Constructor της κλάσης Product ο οποίος αρχικοποιεί όλες τις μεταβλητές της κλάσης, αμέσως μετά έχουμε την μέθοδο fromJson της κλάσης Product η οποία έχει την παράμετρο json τύπου Map με τύπο της'key' παραμέτρου να είναι 'String' και της 'value' παραμέτρου να είναι dynamic . Στο σώμα της μεθόδου βλέπουμε την καταχώριση τιμών στις μεταβλητές της κλάσης από τις αντίστοιχες στήλες τους στην βάση δεδομένων, επίσης υλοποιείται μια statement if η οποία εκτελείται αν η μεταβλητή products δεν είναι null. Στο σώμα της if αρχικοποιούμε την μεταβλητή products σε άδεια λίστα και μετά με μία forEach loop παίρνουμε ένα ένα τα προϊόντα που έχουμε στον server και τα βάζουμε στην λίστα products.

Controllers

Οι Controllers είναι Dart κλάσεις στις οποίες γίνεται επεξεργασία των δεδομένων που παίρνουν από τις αντίστοιχες Repository κλάσεις. Επίσης είναι η κλάση που θα καλείται απευθείας από το UI για να παρουσιαστούν τα δεδομένα που ήρθαν από τον server.

Κλάση RecommendedProtuctController :

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:meal_savers/models/products_model.dart';
import '../data/repository/recommended_product_repo.dart';

class RecommendedProtuctController extends GetxController {
  final RecommendedProtuctRepo recommendedProtuctRepo;
```

```
  RecommendedProtuctController({required this.recommendedProtuctRepo});
  List<dynamic> _recommendedProductList = [];
  List<dynamic> get recommendedProductList => _recommendedProductList;
  bool _isLoaded = false;
  bool get isLoaded => _isLoaded;
  Future<void> getRecommendedProtuctList() async {
    Response response =
        await recommendedProtuctRepo.getRecommendedProductList();
    if (response.statusCode == 200) {
      print("Got Data ! ");
      _recommendedProductList = [];
        _recommendedProductList.addAll
        (Product.fromJson(response.body).products);
      _isLoaded = true;
      update();
    } else {
      print("could not Get Product");
    }
}
```

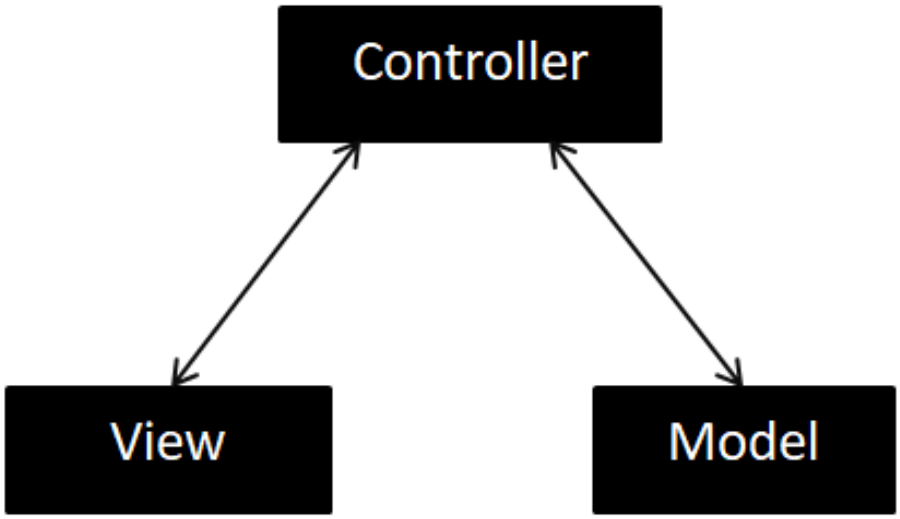
Η κλάση RecommendedProtuctController είναι μία Controller κλάση που εισάγει το Getx package, την βιβλιοθήκη material της flutter την κλάση App Constants, την κλάση ApiClient, την Repository κλάση RecommendedProtuctRepo και την κλάση μοντέλου Product Model την οποία μόλις έχουμε αναλύσει. Στην συνέχεια έχουμε την υλοποίηση της κλάσης η οποία επεκτείνει την abstract κλάση GetxController του Getx package. Η GetX Controller ελέγχει την κατάσταση του UI όταν κάνει wrap ένα μεμονωμένο widget, έτσι ώστε να ανακατασκευάζεται μόνο όταν υπάρχει αλλαγή στην κατάσταση του συγκεκριμένου widget. Επιπρόσθετα έχουμε την δήλωση της μεταβλητής recommendedProtuctRepo η οποία είναι final και τύπου RecommendedProtuctRepo, αυτό σημαίνει μέσω της μεταβλητής recommendedProtuctRepo έχουμε πρόσβαση στις μεθόδους της κλάσης RecommendedProtuctRepo. Επιπλέον έχουμε τον Constructor ο οποίος αρχικοποιεί και θέτει αναγκαστική την απόδοση τιμής στην παράμετρο recommendedProtuctRepo από οποιαδήποτε άλλη κλάση όπου υλοποιεί ένα instance της κλάσης RecommendedProtuctController. Παράλληλα δηλώνουμε και αρχικοποιούμε σε άδεια λίστα την private μεταβλητή recommendedProductList που είναι list τύπου dynamic (δηλαδή είναι λίστα οι οποία μπορεί να αποτελείται από ότι δίπατε int, float, String, κ.τ.λ). Επίσης έχουμε την build-in μέθοδο get της Dart, η οποία μας δίνει πρόσβαση στην ιδιωτική μεταβλητή recommendedProductList όταν την καλούμε από άλλες κλάσεις. Επίσης δηλώνουμε την boolean private μεταβλητή isLoaded που αρχικοποιείται με τιμή false, και από κάτω έχουμε την μέθοδο get της isLoaded. Στην συνέχεια έχουμε την μέθοδο getRecommendedProtuctList που είναι τύπου Future, επίσης η abstract κλάση Future είναι void, και σε αυτήν την μέθοδο δηλώνουμε την μεταβλητή response τύπου Response και της δίνουμε το αποτέλεσμα της μεθόδου getRecommendedProductList, που είναι μία λίστα με τα δεδομένα που πήραμε από τον server. Παράλληλα ελέγχουμε αν όλα πήγαν καλά κατά την διάρκεια της όλης διαδικασίας, και αν όλα είναι εντάξει τότε αρχικοποιούμε την μεταβλητή recommendedProductList σε άδεια λίστα. Στην συνέχεια με την build-in μέθοδο της flutter addAll και με την βοήθεια της μεθόδου fromJson του μοντέλου Products Model και της μεταβλητής products που

έχουμε αναφέρει στα models παίρνουμε τα json δεδομένα από τον server και τα μετατρέπουμε σε list από Maps. Αυτά τα δεδομένα τα δίνουμε στην μεταβλητή recommendedProductList η οποία χρησιμοποιείται από το UI για να απεικονίσει τα δεδομένα στην οθόνη. Επιπλέον αν όλα πήγαν καλά η μεταβλητή isLoaded γίνεται true και εκτελείται η μέθοδος update η οποία ενημερώνει το UI, εν έχει γίνει κάποιο σφάλμα τότε η μεταβλητή isLoaded θα παραμείνει false και θα έχουμε ένα μήνυμα σφάλματος.

Και έτσι φτάσαμε στο τέλος του υποκεφαλαίου της αρχιτεκτονικής του front end, όπως καταλάβατε για την πλήρη κατανόηση του συγκεκριμένου υποκεφαλαίου χρειάζονται κάποιες βασικές γνώσεις αντικειμενοστρεφών γλωσσών προγραμματισμού. Στο επόμενο υποκεφάλαιο θα αναφέρουμε την αρχιτεκτονική που χρησιμοποιήθηκε για την υλοποίηση του backend μέρους της εφαρμογής.

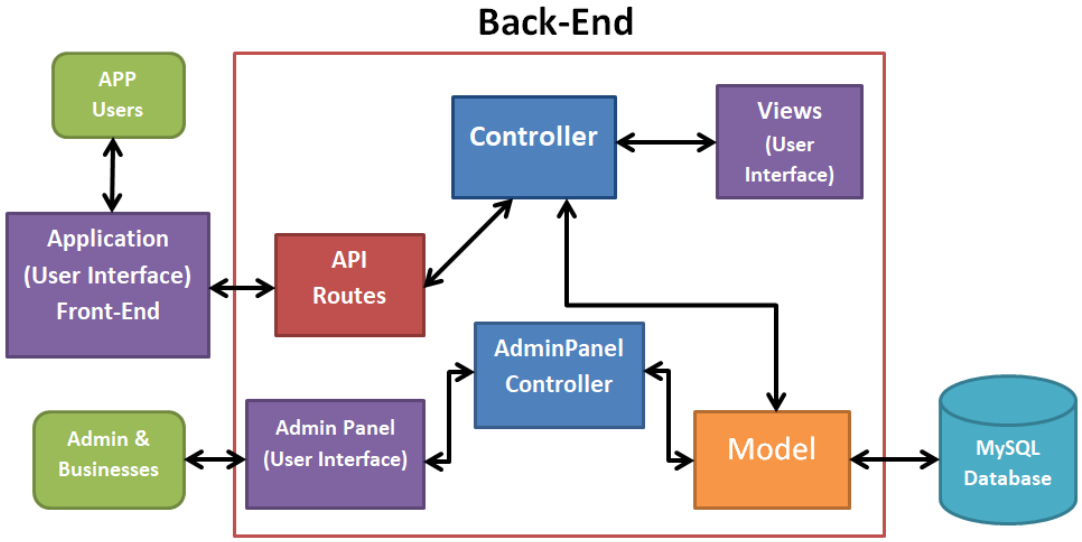
3.1.2 Αρχιτεκτονική Back-End

Η αρχιτεκτονική που χρησιμοποιεί η Laravel ονομάζεται MVC (Model-View-Controller), το οποίο είναι ένα αρχιτεκτονικό μοτίβο λογισμικού που χρησιμοποιείται συνήθως για την ανάπτυξη διεπαφών χρήστη που διαιρούν τη σχετική λογική του προγράμματος σε τρία διασυνδεδεμένα στοιχεία όπως βλέπουμε στο Σχήμα 3.4. Αυτό γίνεται για να διαχωριστούν οι εσωτερικές αναπαραστάσεις των πληροφοριών από τους τρόπους με τους οποίους οι πληροφορίες παρουσιάζονται και γίνονται αποδεκτές από τον χρήστη. Τα models είναι υπεύθυνα για το backend που περιέχει όλη τη λογική δεδομένων. Το view είναι η διεπαφή ή graphical user interface (GUI), και ο controller λειτουργεί ως σύνδεσμος μεταξύ του model και της view, λαμβάνει τα στοιχεία του χρήστη και αποφασίζει τι να κάνει με αυτά. Ο controller είναι η λογική της εφαρμογής που ελέγχει τον τρόπο εμφάνισης των δεδομένων.



Σχήμα 3.4: Αρχιτεκτονική MVC της Laravel

Στα πλαίσια αυτής της εφαρμογής χτίζουμε πάνω στο υπάρχων μοντέλο αρχιτεκτονικής MVC, και το παραμετροποιούμε για να εξυπηρετεί τις ανάγκες της συγκεκριμένης εφαρμογής. Στο Σχήμα 3.5 απεικονίζεται η αρχιτεκτονική του backend της εφαρμογής.



Σχήμα 3.5: Αρχιτεκτονική Backend της Εφαρμογής

Θα αρχίσουμε την εξήγηση του Σχήματος 3.5 με τα Views τα οποία έχουν html κώδικα και είναι το user interface (UI), που είναι μέρος του αρχικού MVC, το οποίο δημιουργείται αυτόματα όταν φτιάχνεις ένα νέο Laravel project, στα πλαίσια της συγκεκριμένης εφαρμογής δεν χρησιμοποιούμε τα Views, γιατί το frontend και UI με το οποίο έρχονται σε επαφή οι χρήστες, είναι ένα mobile application και όχι μία ιστοσελίδα. Επίσης οι διάφορες επιχειρήσεις οι οποίες χρησιμοποιούν την εφαρμογή για την πώληση των προϊόντων τους, κάνουν upload τα προϊόντα στο mobile application μέσω του Admin Panel UI και όχι των Views. Φυσικά για μελλοντικές επεκτάσεις δεν είναι απίθανη η χρήση των Views για την δημιουργία μιας ιστοσελίδας για την εφαρμογή.

Model

Ένα μοντέλο είναι μια PHP κλάση η οποία κάνει αναπαράσταση ενός πίνακα της βάσης δεδομένων, μέσω αυτού του μοντέλου μπορούμε να δηλώσουμε κύρια κλειδιά, ξένα κλειδιά και γενικότερα τις σχέσης μεταξύ των πινάκων της βάσης δεδομένων χωρίς SQL κώδικα. Επίσης ένα μοντέλο χρησιμοποιείται ως ένας τρόπος για την αναζήτηση δεδομένων προς και από ένα table εντός της βάσης δεδομένων. Η Laravel παρέχει έναν βασικό τρόπο για να το κάνουμε αυτό, χρησιμοποιώντας το Eloquent ORM (Το Eloquent είναι ένας σχεσιακός χάρτης αντικειμένων (object relational mapper ORM), που περιλαμβάνεται από προεπιλογή στο Laravel framework. Το ORM είναι ένα λογισμικό που διευκολύνει το χειρισμό των εγγραφών (records), της βάσης δεδομένων αναπαριστώντας δεδομένα ως αντικείμενα, λειτουργώντας ως ένα στρώμα αφαίρεσης πάνω από τη μηχανή βάσης δεδομένων που χρησιμοποιείται για την αποθήκευση των δεδομένων της εφαρμογής, με πιο απλά λόγια ένα Eloquent model μας παρέχει πολλαπλές και εύκολες μεθόδους που γράφουν και διαβάζουν στην βάση δεδομένων. Αυτές οι μέθοδοι γράφονται με php και μεταφράζονται σε SQL κώδικα). Κάθε πίνακας ενσωματώνει ένα μοντέλο για να αλληλεπιδρά μαζί του. Μπορούμε να δημιουργήσουμε μοντέλα με την εντολή 'php artisan make: model' που είναι ένα σύστημα PHP που βασίζεται στο MVC. Artisan είναι το όνομα της διεπαφής command-line που πε-

ριλαμβάνεται στο Laravel. Παρέχει μια σειρά από χρήσιμες εντολές για χρήση κατά την ανάπτυξη μίας laravel εφαρμογής.

Παράδειγμα μοντέλου :

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use App\Models\Order;

class OrderDetail extends Model
{
    use HasFactory;

    protected $casts = [
        'price' => 'float',
        'discount_on_food' => 'float',
        'total_add_on_price' => 'float',
        'tax_amount' => 'float',
        'food_id'=> 'integer',
        'order_id'=> 'integer',
        'quantity'=>'integer',
        'remaining'=>'integer',
    ];
    protected $primaryKey = 'id';

    public function order()
    {
        return $this->belongsTo(Order::class);
    }
    public function food()
    {
        return $this->belongsTo(Food::class,'food_id');
    }
}
```

Ο πιο πάνω κώδικας είναι η υλοποίηση του μοντέλου OrderDetail μέσω του οποίου μπορούμε να κάνουμε διάφορες επεξεργασίες στο αντίστοιχο table OrderDetail της βάση δεδομένων. Το namespace μας δείχνει την διαδρομή προς τον φάκελο στον οποίο βρίσκεται το συγκεκριμένο μοντέλο. Στην συνέχεια δημιουργούμε την κλάση OrderDetail που επεκτείνει το μοντέλο, και με αυτόν τον τρόπο κληρονομεί όλες τις μεθόδους που χρειάζεται για να αλληλεπιδρά με τον αντίστοιχο πίνακα στην βάση δεδομένων, η laravel ενώνει αυτόματα το μοντέλο OrderDetail με τον πίνακα της βάσης δεδομένων ο οποίος έχει το ίδιο όνομα, φυσικά υπάρχει και η επιλογή να κάνουμε overwrite αυτόν τον μηχανισμό με την ιδιότητα

’protected $table=’database table name’;’ και να αναφερθούμε με αυτό το μοντέλο σε όποιο πίνακα θέλουμε. Σε αυτό το σημείο, το μοντέλο είναι έτοιμο για να καταχωρήσει και να πάρει δεδομένα από τον πίνακα OrderDetail της βάσης δεδομένων. Αυτό το μοντέλο μπορεί να χρησιμοποιηθεί από τον αντίστοιχο Controller με την ιδιότητα use App\ Models \ OrderDetail στον οποίο μπορούμε να δημιουργήσουμε μία μεταβλητή και μέσω του μοντέλου OrderDetail που επεκτείνει το model που μας παρέχει την μέθοδο all(), μπορούμε να πάρουμε όλα τα δεδομένα που έχει ο πίνακας OrderDetail. Στην συνέχεια μπορούμε να δώσουμε αυτά τα δεδομένα σε μία μεταβλητή του αντίστοιχου Controller ο οποίος μέσω των API endpoints θα μπορέσει να τα στείλει στο frondend που θα παραληφθούν από την κλάση API Client την οποία έχουμε ανάλυση εις βάθους στο προηγούμενο υποκεφάλαιο. Αυτή η εντολή στην κλάση Controller είναι της μορφής ’$OrderDe = OrderDetail::all()’, όπου δίνουμε στην μεταβλητή ’OrderDe’ τα δεδομένα που βρίσκονται στον πίνακα OrderDetail. Επίσης στο συγκεκριμένο μοντέλο κάνουμε casts τις στήλες του πίνακα OrderDetail στους τύπους δεδομένων που επιθυμούμε. Στην συνέχεια δηλώνουμε την στήλη ’id’ ως κύριο κλειδί του πίνακα. Τέλος με τις μεθόδους order και food δημιουργούμαι σχέσης μεταξύ των πινάκων της βάσης δεδομένων.

Controllers

Αντί να ορίσουμε όλη τη λογική διαχείρισης αιτημάτων στα αρχεία διαδρομής (route files), θέλουμε να οργανώσουμε αυτήν τη συμπεριφορά χρησιμοποιώντας κλάσεις Controllers. Οι Controllers μπορούν να ομαδοποιήσουν τη σχετική λογική διαχείρισης αιτημάτων σε μία κλάση. Παράλληλα οι Controller είναι υπεύθυνη για τη διαχείριση των αιτημάτων των χρηστών και την ανάκτηση δεδομένων, αξιοποιώντας Μοντέλα. Στα πλαίσια της συγκεκριμένης εφαρμογής ο κύριος σκοπός των Controller είναι να παίρνουν μέσω των αντίστοιχων μοντέλων δεδομένα από την βάση δεδομένων, να μετατρέπουν αυτά τα δεδομένα σε μια κατάλληλη μορφή json response, στην οποία θα έχει πρόσβαση η κλάση API Client του frontend μέσω των API EndPoints. Οι Controllers αποθηκεύονται στον κατάλογο app/Http/Controllers, και μπορούμε να τους υλοποιήσουμε με την εντολή :

```
    php artisan make:controller
```

Παράδειγμα Controller :

```
<?php
namespace App\Http\Controllers\Api\V1;
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use App\Models\Food;

class ProductController extends Controller
{
       public function get_recommended_products(Request $request){
$list = Food::where('type_id', 3)->take(10)->orderBy('created_at','DESC')->get();

              foreach ($list as $item){
                 unset($item['selected_people']);
```

```
                unset($item['people']);
            }
             $data = [
                'total_size' => $list->count(),
                'type_id' => 3,
                'offset' => 0,
                'products' => $list
            ];
        return response()->json($data, 200);
    }
}
```

Ο πιο πάνω κώδικας είναι η υλοποίηση του Controller ο οποίος είναι υπεύθυνος για να διαχειρίζεται τα Products τα οποία κάνουν upload οι επιχειρήσεις από το Admin Panel στον πίνακα foods της βάσης δεδομένων. Ο Controller έχει πρόσβαση σε αυτά τα δεδομένα μέσω του μοντέλου Food. Το namespace μας δείχνει την διαδρομή προς τον φάκελο στον οποίο βρίσκεται ο συγκεκριμένος Controller, η κλάση Illuminate\Http\Request της Laravel παρέχει έναν αντικειμενοστραφή τρόπο αλληλεπίδρασης με το τρέχων αίτημα HTTP που χειρίζεται η εφαρμογή μας, καθώς και για ανάκτηση των δεδομένων εισόδου, των cookies και των αρχείων που υποβλήθηκαν με το αίτημα. Στην συνέχεια δημιουργούμε την κλάση ProductController που επεκτείνει τον Controller, ο οποίος είναι μία abstract κλάση της Laravel που μπορούμε να θεωρήσουμε σαν τον ”Base Controller”. Αυτή η κλάση έχει μερικές χρήσιμες μεθόδους τις οποίες μπορούμε να χρησιμοποιήσουμε. Στο σώμα της κλάσης έχουμε την μέθοδο getRecommendedProducts, στην οποία δημιουργούμε την μεταβλητή list, η οποία παίρνει τιμή μία λίστα δέκα προϊόντων σε φθίνουσα σειρά, που η στήλη ’typeId’ είναι ’3’ (το typeId 3 είναι η κατηγορία των Recommended Products), από τον πίνακα food της βάσης δεδομένων με την βοήθεια της μεθόδου get(), η οποία είναι μία από της μεθόδους που μας παρέχει το model το οποίο κάνει επέκταση το μοντέλο Food. Επιπλέον με την foreach loop, για κάθε ένα από τα Products που πήραμε από την βάση δεδομένων και τα βάλαμε στην λίστα list, φεύγουμε την στήλη ’selectedPeople’ και ’people’. Επίσης δημιουργούμε την μεταβλητή data η οποία είναι ένας συσχετιστικός πίνακας (Associative array είναι ένας αφηρημένος τύπος δεδομένων που μπορεί να κρατήσει δεδομένα σε ζεύγη (key, value). Μπορείτε να σκεφτείτε τους συσχετιστικούς πίνακες όπως μια λίστα αριθμών τηλεφώνου. Σε αυτήν τη λίστα, μπορείτε να αναζητήσετε το όνομα ενός ατόμου βρίσκοντας τον αριθμό τηλεφώνου του. Το όνομα είναι το value και ο αριθμός είναι το Key), σε αυτόν το πίνακα έχουμε το key ’totalSize’ που παίρνει value ’τον αριθμό των προϊόντων που βρίσκονται στην λίστα’, επίσης έχουμε το key ’products’ με value ’την λίστα με τα προϊόντα’. Τέλος αν όλα πάνε καλά η μέθοδος getRecommendedProducts επιστρέφει ένα json response, το οποίο έχει το περιεχόμενο της μεταβλητής data και status code 200. Με την ολοκλήρωση της εξήγησης του Controller αρχίζουμε να κατανοούμε πως λειτουργεί η όλη διαδικασία της μεταφοράς των δεδομένων από την βάση δεδομένων στο Application. Στην συνέχεια θα δούμε πως γίνεται η δρομολόγηση στο json response με την χρήση των API Routes, και την δημιουργία του EndPoint το οποίο παίρνουμε στο frontend.

API Routes

Route είναι ένας τρόπος δημιουργίας ενός URL request για την εφαρμογή μας. Αυτές οι διευθύνσεις URL δεν χρειάζεται να αντιστοιχίζονται σε συγκεκριμένα αρχεία σε έναν website και είναι τόσο αναγνώσιμες από τον άνθρωπο όσο και φιλικές προς το SEO (SEO Search Engine Optimization είναι η διαδικασία εγγραφής και επανεγγραφής του κώδικα σε έναν website με τρόπο που διευκολύνει τις μηχανές αναζήτησης (όπως το Google, το Yahoo και το Bing) να ανιχνεύσουν, να διαβάσουν και να ευρετηριάσουν το περιεχόμενο του website). Ποια είναι η διαφορά μεταξύ Routes και URL;

Για αρχή ένα Route χρησιμοποιεί μια ονομασμένη διαδρομή. Αυτό σημαίνει ότι η σύνδεση μεταξύ της φόρμας και του controller της, δεν υπαγορεύεται από το url που βλέπει ο χρήστης. Το επόμενο πλεονέκτημα είναι ότι με τα Routes μπορούμε να περάσουμε πρόσθετες παραμέτρους στο route helper και θα τις τοποθετήσει στη σωστή θέση της φόρμας. Παράλληλα το Routing είναι ο τρόπος με τον οποίο το API αντιστοιχίζει ένα URI με μια ενέργεια, για παράδειγμα στο συγκεκριμένο backend, ένα API Endpoint αντιστοιχίζει μία διαδρομή URI με την μέθοδο του controller την οποία θέλουμε να καλέσουμε με το συγκεκριμένο EndPoint. Τα Endpoints εκτελούν μια συγκεκριμένη λειτουργία, λαμβάνοντας κάποιο αριθμό παραμέτρων και επιστρέφουν δεδομένα στο API Client του frontEnd. Ένα Route είναι το ”όνομα” που χρησιμοποιείται για πρόσβαση στα EndPoints. Ένα Route μπορεί να έχει πολλαπλά EndPoints που σχετίζονται με αυτή. Τα API Routes αποθηκεύονται στο MealSaversBackend στον κατάλογο APP/ROUTES/API/V1.

Παράδειγμα API Route :

```
<?php
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
*/
Route::group(['namespace' => 'Api\V1'], function () {

    Route::get('recommended', 'ProductController@get_recommended_products');
});
```

Στο αρχείο API στον φάκελο Routes καταχωρούμε τις διαδρομές API για την εφαρμογή μας. Αυτές οι διαδρομές φορτώνονται από τον RouteServiceProvider μέσα σε μια ομάδα, η οποία εκχωρεί το ”api” middleware group (θα δούμε τι είναι το middleware στο υποκεφάλαιο ’Υλοποίηση του Backend’).

Η μέθοδος group της κλάσης Route μας επιτρέπει να οργανώσουμε διαδρομές που μοιράζονται χαρακτηριστικά, όπως διαδρομή, όνομα ή middleware. Η μέθοδος περιλαμβάνει δύο παραμέτρους, έναν πίνακα και μια callback μέθοδο. Με τον πίνακα δημιουργούμε υποφακέλους οι οποίοι προστίθενται στο path. Στην συγκεκριμένη περίπτωση η μεταβλητή namespace του RouteServiceProvider περιέχει την διαδρομή προς τους Controllers, δηλαδή App/Http/Controllers, στην οποία προστίθεται το Api/V1 και γίνεται App/Http/Controllers/Api/V1. Στο σώμα της callback μεθόδου έχουμε την μέθοδο Get της κλάσης Route η οποία χρησιμοποιείται για να ζητήσει δεδομένα από έναν καθορισμένο πόρο. Η μέθοδος Get έχει σαν παραμέτρους το EndPoint που σε αυτή την περίπτωση είναι το String 'recommended' και την μέθοδο του Controller από την οποία θα πάρει τα δεδομένα. Στο συγκεκριμένο παράδειγμα η μέθοδος από την οποία θα πάρουμε τα δεδομένα είναι η getRecommendedProducts του ProductController, την υλοποίηση της οποίας έχουμε αναφέρει στο παράδειγμα για τους Controllers. Επιπλέον το EndPoint recommended θα προστεθεί στο path το οποίο στην συνέχεια θα γίνει App/Http/Controllers/Api/V1/recommended. Βλέποντας την νέα διαδρομή κατανοούμε καλύτερα για πιο λόγο το EndPoint πήρε αυτό το όνομα. Η κλάση API Client του FrondEnd μπορεί να χρησιμοποιήσει αυτό το EndPoint, προσθέτοντας στο URI το BaseUrl , στην συγκεκριμένη περίπτωση επειδή χρησιμοποιούμε τον XAMPP ο οποίος είναι ένας localhost server, το BaseUrl θα είναι http://127.0.0.1:8000 και μετά την πρόσθεση του URI θα γίνει:

```
http://127.0.0.1:8000/App/Http/Controllers/Api/V1/recommended.
```

AdminPanel Controller

Οι AdminPanel Controllers είναι υπεύθυνοι για την μεταφορά και επεξεργασία δεδομένων που παίρνουν από τα μοντέλα. Επίσης οι AdminPanel Controllers στέλνουν και παραλαμβάνουν τα δεδομένα από το UI του Admin Panel. Παράλληλα είναι οι controllers οι οποίοι δημιουργούν το UI του Admin Panel στο οποίο έχουν πρόσβαση οι διάφορες επιχειρήσεις που θα συνεργάζονται με την εφαρμογή. Αυτές οι επιχειρήσεις θα έχουν ένα λογαριασμό στο Admin Panel τον οποίο θα τους δημιουργεί ο Admin. Μέσω αυτού το λογαριασμού οι επιχειρήσεις θα μπορούν να κάνουν upload ένα καλάθι με τα προϊόντα που τους έχουν απομείνει στο τέλος της εργάσιμης μέρας. Αυτή η διαδικασία γίνεται δυνατή με τους AdminPanel Controllers που παίρνουν τα δεδομένα από το Admin Panel και μέσω του κατάλληλου μοντέλου τα δεδομένα πηγαίνουν στους αντίστοιχους πίνακες της βάσης δεδομένων, στους οποίους έχουν πρόσβαση οι Controllers μέσω των αντίστοιχων μοντέλων.

Παράδειγμα AdminPanel Controller :

```
<?php

namespace App\Admin\Controllers;

use App\Models\FoodType;
use Encore\Admin\Controllers\AdminController;
use Encore\Admin\Form;
use Encore\Admin\Grid;
use Encore\Admin\Show;
use Encore\Admin\Layout\Content;
```

```
use Encore\Admin\Tree;

class FoodTypeController extends AdminController
{
    protected $title = 'Food Type';
    public function index(Content $content)
    {

            ->header('Food Type')
            ->body( K         );
            return $content
    }
    protected function form()
    {
        $form = new Form(new FoodType());
        $form->select('parent_id', __('Parent Category'));
        $form->text('title', __('Title'));
        $form->textarea('description', __('Description'));
        $form->number('order', __('Order'))->default(1);
        return $form;
    }
    protected function detail($id)
    {
        $show = new Show(FoodType::findOrFail($id));

        $show->field('id', __('Id'));
        $show->field('title', __('Title'));
        $show->field('description', __('Description'));
        $show->field('order', __('Order'));
        $show->field('created_at', __('Created_at'));
        $show->field('updated_at', __('Updated_at'));
        return $show;
    }

}
```

Ο πιο πάνω κώδικας είναι το περιεχόμενο της κλάσης FoodTypeController η οποία επεκτείνει την κλάση AdminController. Όταν κάναμε install το Admin Panel στο laravel project, αποκτήσαμε πρόσβαση σε μία πληθώρα από κλάσεις και μεθόδους οι οποίες κάνουν το render αντικειμένων στο UI του Admin Panel πολύ απλό. Συνεχίζουμε αναφέροντας την μέθοδο index η οποία έχει την παράμετρο content που είναι τύπου Content, η κλάση Content είναι μία από τις κλάσεις του Admin Panel η οποία μας προσφέρει ένα συγκεκριμένο layout. Στην συνέχεια βλέπουμε την μέθοδο 'from' η οποία δημιουργεί μία φόρμα της οποίας ο τύπος των πεδίων είναι : select, text, textarea και number. Τα συγκεκριμένα πεδία παίρνουν

δύο παραμέτρους, η πρώτη είναι το όνομα της στήλης του πίνακα FoodType της βάσης δεδομένων στην οποία αναφέρονται, και η δεύτερη είναι τα ονόματα που θα έχουν τα πεδία στο UI του Admin Panel. Ο σκοπός αυτής της φόρμας είναι να δώσει την δυνατότητα στον Admin να στείλει αυτά τα δεδομένα στην βάση δεδομένων μέσω του αντίστοιχου μοντέλου FoodType. Αυτό γίνεται εφικτό με την δημιουργία ενός αντικειμένου 'form' το οποίο παίρνει παραμέτρους τύπου μοντέλου FoodType. Τελειώνοντας υλοποιούμε την μέθοδο 'detail' η οποία παίρνει την παράμετρο 'id' που αναφέρεται στο κύριο κλειδί του πίνακα FoodType στην βάση δεδομένων. Στο σώμα της μεθόδου δημιουργούμε το αντικείμενο show το οποίο μέσω του FoodType μοντέλου, έχει πρόσβαση στην μέθοδο findOrFail, η οποία παίρνει σαν παράμετρο το κύριο κλειδί του πίνακα FoodType. Αν η μέθοδος findOrFail βρει το κύριο κλειδί τότε η μέθοδος 'detail' εμφανίζει στο UI του Admin Panel τα στοιχεία των στηλών του πίνακα τα οποία σχετίζονται με το συγκεκριμένο κύριο κλειδί. Με την εξήγηση του AdminPanel Controller φτάσαμε στο τέλος του υποκεφαλαίου 3.1 'Αρχιτεκτονική Εφαρμογής', που θεωρώ πως είναι το σημαντικότερο υποκεφάλαιο για την κατανόηση της εφαρμογής. Στο επόμενο υποκεφάλαιο θα εξηγήσουμε πως έχει γίνει η υλοποίηση των βασικών τομέων του Frontend, το οποίο θα μας βοηθήσει για την πλήρης κατανόηση όλων των υπηρεσιών της εφαρμογής.

3.2 Υλοποίηση του Frontend

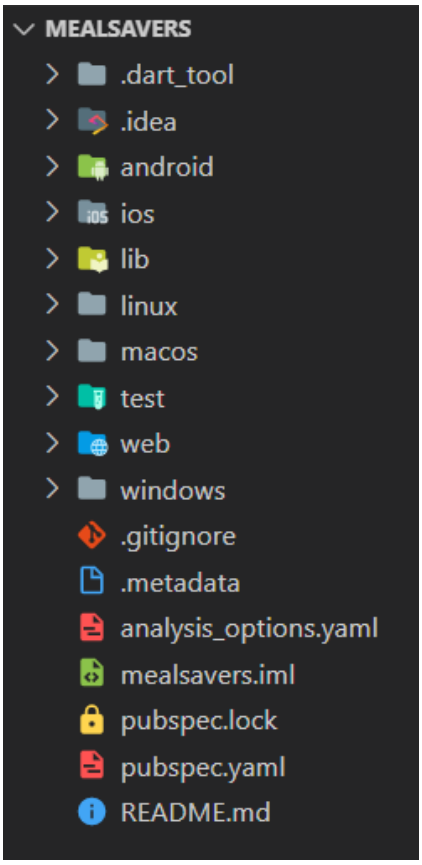
Στα πλαίσια του συγκεκριμένου υποκεφαλαίου θα εξηγήσουμε αναλυτικά πως έχουν υλοποιηθεί όλα τα βασικά στοιχεία από τα οποία αποτελείται το Frontend.

Αρχίζουμε την δημιουργία του Flutter project στο περιβάλλων του VS Code πηγαίνοντας στον φάκελο όπου θέλουμε να δημιουργήσουμε το project και εκτελώντας την εντολή :

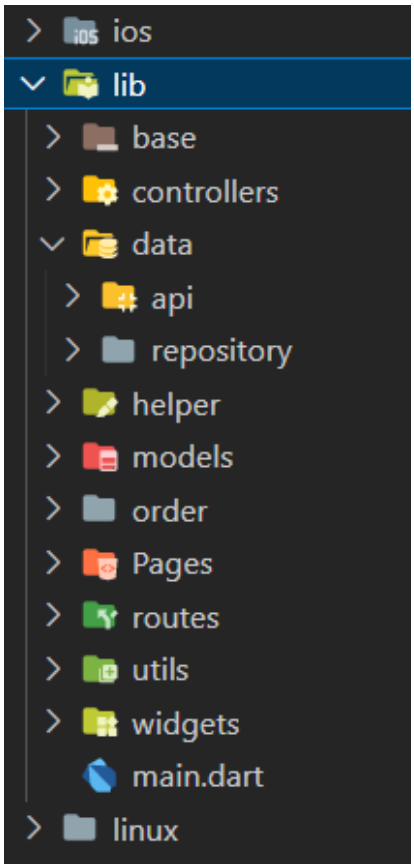
```
flutter create mealsavers
```

Το αποτέλεσμα εκτέλεσης της πιο πάνω εντολής είναι η δημιουργία ενός φακέλου με το όνομα mealsavers. Μέσα σε αυτόν δημιουργήθηκε ένα flutter Application με όλα τα απαραίτητα configurations και dependencies. Η δομή αποτελείται από 10 φακέλους και αρκετά αρχεία όπως βλέπουμε στο Σχήμα 3.6.

Όλη η δημιουργία του mobile application γίνεται μέσα στον φάκελο lib Σχήμα 3.7. Στον φάκελο lib δημιουργήσαμε τους φακέλους base, controllers, data ο οποίος περιέχει τους api και repository φακέλους, helper ο οποίος έχει τα dependencies, models, Pages, routes, utils, widgets, και το αρχείο main (main.dart). Ένα αρχείο που αξίζει να αναφερθεί είναι το 'pubspec.yaml', το οποίο καθορίζει τις εξαρτήσεις (dependencies), γραμματοσειρές ή αρχεία εικόνας που απαιτεί το project. Επίσης καθορίζει άλλες απαιτήσεις, όπως εξαρτήσεις από πακέτα ή συγκεκριμένους περιορισμούς στην έκδοση του Flutter SDK. Ο φάκελος '.idea' περιέχει όλες τις ρυθμίσεις που σχετίζονται με το IntelliJ IDEA Android studio (το Android Studio και το IntelliJ IDEA είναι και τα δύο εργαλεία ανάπτυξης που έχουν σχεδιαστεί για να βοηθούν τους προγραμματιστές να δημιουργούν εφαρμογές με ευκολία). Ο φάκελος 'android' και ο φάκελος 'ios' είναι οι φάκελοι που υπάρχουν για τη δημιουργία μιας εφαρμογής στις αντίστοιχες πλατφόρμες με τα αρχεία Dart που δημιουργούμε στον φάκελο lib να εκτελούνται σε αυτές.



Σχήμα 3.6: Flutter App structure



Σχήμα 3.7: Flutter App lib folder

Το επόμενο βήμα μετά την δημιουργία του Project είναι να κάνουμε install κάποια απαραίτητα πακέτα και εξαρτήσεις. Αυτά τα πακέτα καθορίζονται από το αρχείο 'pubspec.yaml', γράφοντας το όνομα του πακέτου και την έκδοση του κάτω από το πεδίο 'dependencies:' όπως βλέπουμε ποιο κάτω :

```
dependencies:
  flutter:
    sdk: flutter
  get: ^4.1.4
  shared_preferences: ^2.0.15
  intl: ^0.17.0
  webview_flutter: ^3.0.4
  cupertino_icons: ^1.0.2
```

Στα προηγούμενα κεφάλαια έχουμε αναφέρει κάποια από τα πακέτα που απεικονίζονται πιο πάνω, όπως το Getx Package και το shared preferences. Το intl package μας παρέχει ένα κοινό σημείο εισόδου για εργασίες που σχετίζονται με τη διεθνοποίηση, ένα instance της κλάσης Intl μπορεί να δημιουργηθεί για μια συγκεκριμένη τοπική ρύθμιση και να χρησιμοποιηθεί για τη δημιουργία μιας μορφής ημερομηνίας μέσω της μεθόδου anIntl.date(). Το webview flutter είναι ένα Flutter plugin που παρέχει ένα widget WebView σε Android και iOS. Αυτό το plugin χρησιμοποιείται για το render ιστοσελίδων σε συσκευές Android και iOS. Η βιβλιοθήκη cupertino icons μας παρέχει μια πληθώρα από Icons τα οποία μπορούμε να χρησιμοποιήσουμε για το Application. Οι εξαρτήσεις που προαναφέραμε μπορούν να εγκατασταθούν στο Application εκτελώντας την πιο κάτω εντολή στο terminal μέσα στον φάκελο του Meal Savers.

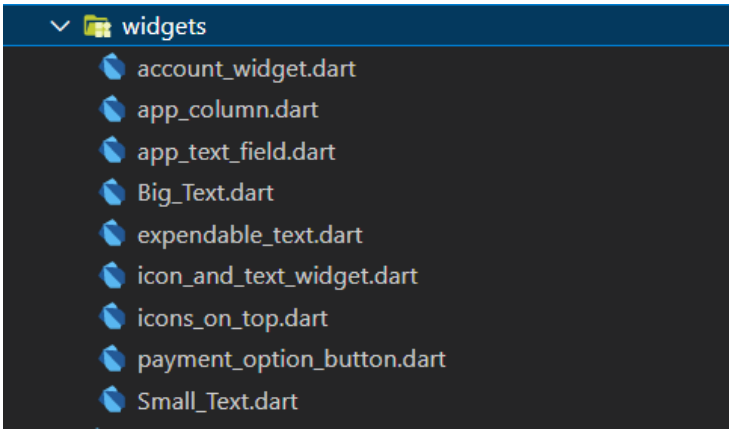
Εντολή :

```
flutter pub get 'dependencies name'
```

Στην συνέχεια θα εξηγήσουμε το περιεχόμενο και την λειτουργία τον υποφακέλων του lib folder, αρχίζοντας από τον φάκελο widgets.

3.2.1 Widgets

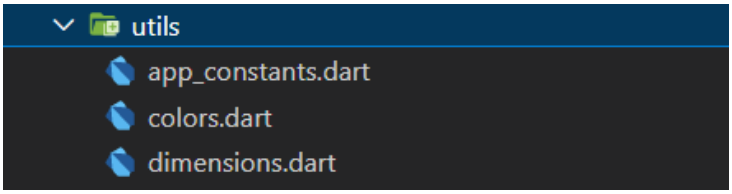
Στον Φάκελο Widgets δημιουργήθηκαν οι βιβλιοθήκες οι οποίες δεν είναι τίποτα άλλο παρά αρχεία Dart. Μέσα σε αυτά τα αρχεία δημιουργήσαμε κλάσεις που μας παρέχουν χρήσιμα Widgets τα οποία επαναχρησιμοποιούμε καθ'όλη την διάρκεια του σχεδιασμού του mobile Application. Όπως βλέπουμε στο Σχήμα 3.8, έχουμε αναφέρει και εξηγήσει δείχνοντας τον κώδικα από κάποιες βιβλιοθήκες του φακέλου Widgets στο υποκεφάλαιο 2.9' Βιβλιοθήκες'.



Σχήμα 3.8: Widgets Folder

3.2.2 Utils

Ο φάκελος Utils έχει την ίδια λειτουργία με τον φάκελο Widgets, η διαφορά τους είναι, πως ο φάκελος Utils, αντί να έχει αρχεία που παρέχουν χρήσιμα Widgets, έχει αρχεία τα οποία παρέχουν χρήσιμα εργαλεία που επαναχρησιμοποιούνται καθ'όλη την διάρκεια δημιουργίας του mobile Application. Τα αρχεία που περιέχει ο φάκελος Utils απεικονίζονται στο Σχήμα 3.9.

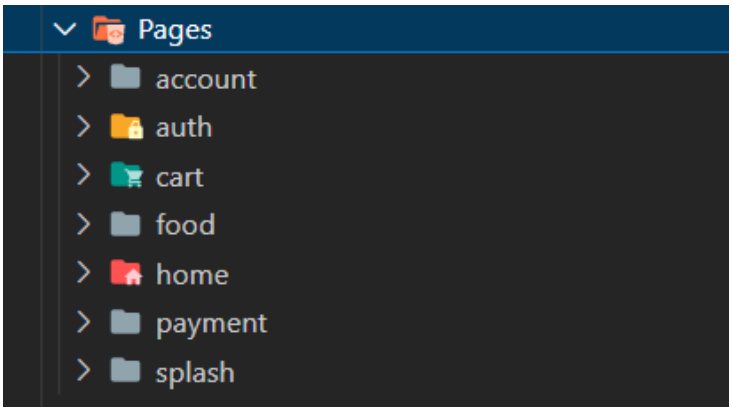


Σχήμα 3.9: Utils Folder

Έχουμε αναφέρει και εξηγήσει τα αρχεία colors και dimensions στο υποκεφάλαιο 2.9' Βιβλιοθήκες', και το αρχείο appConstants στο υποκεφάλαιο 3.1' Αρχιτεκτονική Εφαρμογής'.

Pages Folder

Ο φάκελος Pages έχει όλες τις σελίδες με τις οποίες μπορεί να έχει αλληλεπίδραση ο χρήστης του mobile Application. Όπως βλέπουμε στο Σχήμα 3.10, αυτές οι σελίδες χωρίζονται σε εφτά κατηγορίες: account, auth, cart, food, home, payment και splash.



Σχήμα 3.10: Pages Folder

3.2.3 Splash

Ο Splash Folder περιέχει ένα αρχείο Dart την Splash Page. Η Splash Page είναι η πρώτη σελίδα που απεικονίζεται κατά την εκκίνηση της εφαρμογής. Ακριβώς από κάτω θα δούμε τα βασικά κομμάτια κώδικα της κλάσης SplashScreen, και θα αναφέρουμε όλες τις διαδικασίες που λαμβάνουν χώρα κατά την διάρκεια προβολής της συγκεκριμένης σελίδας.

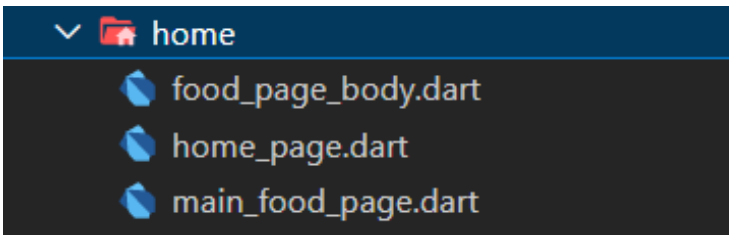
```
class SplashScreen extends StatefulWidget {
  const SplashScreen({super.key});

  @override
  State<SplashScreen> createState() => _SplashScreenState();
}
class _SplashScreenState extends State<SplashScreen> {

  Future<void> _loadResource() async {
    await Get.find<PopularProtuctController>().getPopularProtuctList();
    await Get.find<RecommendedProtuctController>().getRecommendedProtuctList();
  }
  @override
  void initState() {
    super.initState();
    _loadResource();
    Timer(const Duration(seconds: 3),
        () => Get.offNamed(RouteHelper.getInitial()));
  }
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    Decoration code...
    body: Decoration code...
    Image.asset(
           "assets/images/Meal_Savers2.gif",
           width: Dimensions.Height320,)
}}
```

Πριν την εξήγηση του Splash Page θα ήθελα να αναφέρω ότι ο πιο πάνω κώδικας δεν είναι ο πλήρες κώδικας της Splash Page, δεν αναφέρονται τα πακέτα και οι βιβλιοθήκες που χρησιμοποιούνται, και παραλείπονται πολλά κομμάτια κώδικα της κλάσης.

Η κλάση SplashScreen κάνει επέκταση την κλάση StatefulWidget, και στο σώμα της έχει την μέθοδο createState η οποία επιστρέφει την κλάση SplashScreenState την υλοποίηση της οποίας βλέπουμε από κάτω. Σε αυτή την κλάση γίνεται η υλοποίηση της μεθόδου loadResource η οποία κάνει load τις επιχειρήσεις και τα προϊόντα που προσφέρουν από τον server. Δηλαδή κατά την διάρκεια προβολής του Splash Page γίνεται παράλληλα και η φόρτωση των δεδομένων που θα παρουσιαστούν στο Home Page. Με την μέθοδο initState ορίζουμε τον χρόνο προβολής της Splash Page σε 3 δευτερόλεπτα, και με την λήξει αυτού του χρόνου θέλουμε να γίνει ανακατεύθυνση στο Home Page με την βοήθεια της μεθόδου getInitial της κλάσης RouteHelper τον κώδικα της οποία θα εξηγήσουμε όταν θα μιλήσουμε για το Routing του Frondend. Τέλος βλέπουμε το περιεχόμενο το οποίο θα γίνει προβολή στο UI. Αυτό το περιεχόμενο είναι κυρίως ένα gif αρχείο, το οποίο έχουμε τοπικά αποθηκευμένο στον φάκελο assets. Το Splash Page έχει την ίδια διάρκεια με τον χρόνο διάρκειας animation του gif, κατά την διάρκεια αυτού του χρόνου ξεκινά την φόρτωση των δεδομένων που θα παρουσιαστούν στο Home Page από τον server. Τελειώνοντας, όταν γίνει λήξει του χρόνου τότε ανακατευθύνει τον χρήστη στο Home Page.

3.2.4 Home



Σχήμα 3.11: Home Folder

Όπως βλέπουμε στο Σχήμα 3.11, ο φάκελος Home έχει τρία αρχεία, τα food-page-body, main-food-page και home-page. Στην συνέχεια θα αναφέρουμε κάποια κομμάτια κώδικα από τα τρία αρχεία, για να γίνει πιο εύκολη η κατανόηση της λειτουργίας του home Page.

```
class FoodPageBody extends StatefulWidget {
  Widget build(BuildContext context) {
  Decoration code...
       //Popular food list
       GetBuilder<PopularProtuctController>(builder: (popularProducts) {
         return popularProducts.isLoaded
             ? Container( child: ListView.separated(
                    scrollDirection: Axis.horizontal,
            itemCount: popularProducts.PopularProductList.length,
 separatorBuilder: (content, _) => itemBuilder: (content, index) {
                return GestureDetector(onTap: () { Get.toNamed(
                   RouteHelper.getPopularFood(index, "home"));
                   ...
         : CircularProgressIndicator();
       }),
       //Recommended food list
  GetBuilder<RecommendedProtuctController>(builder(recommendedProduct) {
         ....
}
```

Το πιο πάνω κομμάτι κώδικα ανήκει στην κλάση FoodPageBody η οποία έχει σαν κύριο σκοπό την υλοποίηση των ListView για τους τύπους προϊόντων Popular και Recommended. Το ListView δημιουργεί μια κυλιόμενη, γραμμική σειρά με widgets. Το GetBuilder του getx package χρησιμοποιείται για να κάνει τις προβολές του UI να αλληλεπιδρούν με τις μεταβλητές και τις μεθόδους των controllers. Με την βοήθεια του GetBuilder δημιουργούμε το 'popularProducts' που είναι ένα αντικείμενο του PopularProtuctController, με αυτόν τον τρόπο έχουμε πρόσβαση στις μεταβλητές και μεθόδους του συγκεκριμένου controller. Η GetBuilder επιστρέφει μία συνθήκη που ελέγχει αν η μεταβλητή isLoaded την οποία έχουμε αναφέρει στο (3.1.1 Αρχιτεκτονική Frontend Controllers), στην οποία έχουμε πρόσβαση μέσω του αντικειμένου popularProducts είναι 'true', τότε επέστρεψε μου ένα container αλλιώς επέστρεψε μου ένα Circular Progress Indicator. Το container περιέχει ένα οριζόντιο ListView του οποίου ο αριθμός των περιεχομένων εξαρτάται από τον αριθμό των επιχειρήσεων που έχουν διαθέσιμα προϊόντα. Αυτό μπορούμε να το επιτύχουμε παίρνοντας το μήκος της λίστας PopularProductList που έχουμε αναφέρει στο (3.1.1 Αρχιτεκτονική Frondend Controllers). Στην συνέχεια επιστρέφουμε ένα GestureDetector το οποίο είναι ένα widget που ανιχνεύει χειρονομίες. Ο GestureDetector χρησιμοποιείται σε περίπτωση όπου ο χρήστης πατήσει σε μία από τις επιχειρήσεις που παρουσιάζει το ListView. Με την βοήθεια του RouteHelper ο χρήστης πλοηγείται στο σωστό popularfooddetail page το οποίο είναι η σελίδα που μας παρουσιάζει τα διαθέσιμα προϊόντα των διαφόρων επιχειρήσεων. Τέλος η ίδια διαδικασία γίνεται και για τον Recommended τύπο προϊόντων.

Ο κύριος σκοπός του αρχείου main-food-page είναι καθαρά σχεδιαστικός, και να παρουσιάζει το περιεχόμενο της κλάσης FoodPageBody, όπως βλέπουμε στο πιο κάτω κομμάτι κώδικα.

```
class MainFoodPage extends StatefulWidget {
       ...
          ),
          //showing the body
          Expanded(
              child: SingleChildScrollView(
            child: FoodPageBody(),
}
```

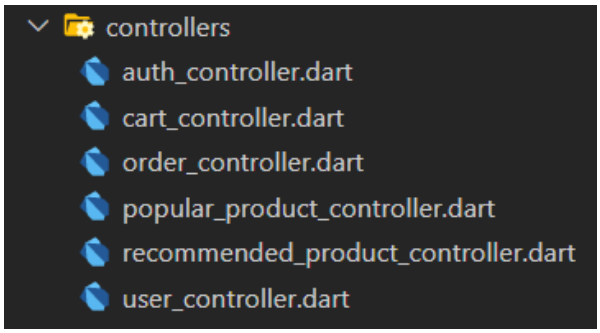
Όπως το MainFoodPage καλεί το FoodPageBody, έτσι και η κλάση HomePage καλεί το MainFoodPage. Κύριος σκοπός του αρχείου HomePage είναι η δημιουργία μίας κεντρικής σελίδας μέσω της οποίας ο χρήστης θα έχει την δυνατότητα να κινηθεί ελεύθερα και σε άλλες σελίδες της εφαρμογής. Αυτό μπορούμε να το πετύχουμε με την δημιουργία ενός Navigation Bar όπως βλέπουμε και στον πιο κάτω κομμάτι κώδικα της κλάσης HomePage.

```
class HomePage extends StatefulWidget {
  int _selectedIndex = 0;
  List pages = [ MainFoodPage(), AccountPage(),CartHistoryPage() ];
  void onTapNav(int index) {
    setState(() { _selectedIndex = index;}); }
  @override
  Widget build(BuildContext context) {
    bool _userLoggedIn = Get.find<AuthController>().userLoggedIn();
    if (_userLoggedIn) {
      Get.find<UserController>().getUserInfo();}
    return Scaffold(
      body: pages[_selectedIndex],
      bottomNavigationBar: CurvedNavigationBar( Decoration code...
          onTap: onTapNav,
          items: const [
            Icon( Decoration code... ),
            Icon( Decoration code... ),
            Icon( Decoration code... )
          ]));
  }
```

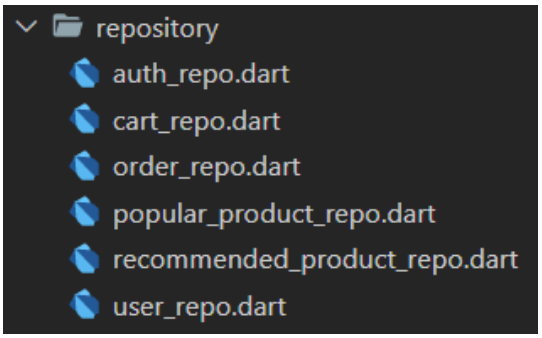
Στον πιο πάνω κώδικα παρουσιάζεται το βασικό κομμάτι της κλάσης HomePage. Πώς λειτουργεί το Navigation Bar ; Όταν ο χρήστης πατάει σε ένα από τα Icons του Navigation Bar καλείτε η μέθοδος onTapNav η οποία ανάλογα με το που πάτησε ο χρήστης παίρνει σαν παράμετρο το index το οποίο αντιστοιχεί με την σελίδα που βρίσκεται στο αντίστοιχο index της λίστας pages. Παράλληλα έχουμε και την boolean μεταβλητή userLoggedIn η οποία μέσω του find που είναι μια από τις μεθόδους του getx package παίρνει την τιμή που επιστρέφει η μέθοδος userLoggedIn του AuthController. Η userLoggedIn ελέγχει αν ο χρήστης είναι 'Logged in', και επιστρέφει true ή false. Αν ο χρήστης είναι Logged in και πλοηγηθεί στο AccountPage θα δει τα στοιχεία που έδωσε όταν έκανε εγγραφή με την βοήθεια της

μεθόδου getUserInfo της κλάσης UserController. Αν ο χρήστης δεν είναι Logged in και πλοηγηθεί στο AccountPage τότε θα οδηγηθεί στο sign in page.

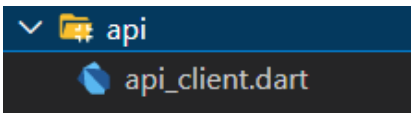
Controller-Repository-API Folders



Σχήμα 3.12: Controller Folder



Σχήμα 3.13: Repository Folder



Σχήμα 3.14: API Folder

Στα Σχήματα 3.12, 3.13, 3.14, παρουσιάζεται το περιεχόμενο των φακέλων Controller, Repository και API αντίστοιχα. Στο κεφάλαιο 3.1.1 με τίτλο Αρχιτεκτονική Frontend, έχουμε εξηγήσει αναλυτικά των κώδικα και την λογική κάποιον αρχείων που ανήκουν στους συγκεκριμένους φακέλους. Επίσης έχουμε εξηγήσει την αλληλεπίδραση μεταξύ controllers, repositorys και APIClient, καθώς και το αποτέλεσμα αυτής της συνεργασίας.

3.2.5 Authentication

Στον Φάκελο Authentication βρίσκονται τα Sign in και Sign up Pages. Το Sign up Page είναι η σελίδα στην οποία κάνει εγγραφεί ο χρήστης για αυθεντικοποίηση στοιχείων.

```
emailController = TextEditingController();
nameController = TextEditingController();
passwordController = TextEditingController();
passwordConfirmationController = TextEditingController();
phoneController = TextEditingController();
```

Τα παραπάνω αντικείμενα δημιουργούνται στην κλάση SignUpPage. Τα αντικείμενα είναι τύπου TextEditingController ο οποίος είναι ένας Controller που είναι υπεύθυνος για επεξεργασία κειμένου. Κάθε φορά που ο χρήστης τροποποιεί ένα πεδίο κειμένου με ένα σχετικό textEditingController, το πεδίο κειμένου ενημερώνει την τιμή του.

```
void registration(AuthController authController) {
 String email = emailController.text.trim();
 String name = nameController.text.trim();
 String password = passwordController.text.trim();
 String passwordConfirmation = passwordConfirmationController.text.trim();
```

Επιπλέον δημιουργούμε την μέθοδο registration η οποία παίρνει μία παράμετρο τύπου AuthController. Η μέθοδος registration είναι αυτή που κάνει τους διάφορους ελέγχους στα πεδία που βάζει τα στοιχεία του ο χρήστης. Επίσης στέλνει αυτά τα στοιχεία στον AuthController ο οποίος με την διαδικασία που έχουμε αναφέρει στο Κεφάλαιο 3.1.1 στέλνει τα στοιχεία στον server. Στον πιο πάνω κώδικα δημιουργούμε τις μεταβλητές οι οποίες θα παίρνουν τα στοιχεία που συμπληρώνει ο χρήστης στα αντίστοιχα πεδία της φόρμας. Η μέθοδος trim χρησιμοποιείται για να αφαιρέσουμε οποιαδήποτε κενά τα οποία μπορεί να έβαλε ο χρήστης κατά την διάρκεια εγγραφείς των στοιχείων. Επιπρόσθετα πάνω σε αυτές τις μεταβλητές θα γίνονται οι διάφοροι έλεγχοι όπως βλέπουμε στον πιο κάτω κώδικα.

```
if (name.isEmpty) {
    ShowMessage("....", title: "Username");
  } else if (!GetUtils.isEmail(email)) {
    ShowMessage(".....", title: "Valid Email address");
  } else if (password.length < 6) {
    ShowMessage(".....", title: "Password");
  }else {
    SignUpModel signUpModel = SignUpModel(
    name: name, phone: phone, email: email...);
    authController.registration(signUpModel).then((status) {
    ...
      if (status.isSuccess) {
        ShowMessage2("Your Account is ready", title: "Perfect");
        Get.offNamed(RouteHelper.getInitial());
      } else {
        ShowMessage(status.message);
      }}
```

Ο πιο πάνω κώδικας απεικονίζει τους ελέγχους που έλαβαν χώρα πριν την αποστολή των στοιχείων στον authController. Αν όλοι οι έλεγχοι ικανοποιηθούν, τότε η μέθοδος registration στέλνει τα στοιχεία των χρηστών στον authController. Αν ο authController με την διαδικασία που αναφέραμε στο Κεφάλαιο 3.1.1, μπορέσει να στείλει τα δεδομένα στον server. Η registration παράλληλα επιστρέφει το μήνυμα ”Your Account is ready” και ανακατευθύνει τον χρήστη στο Home Page.

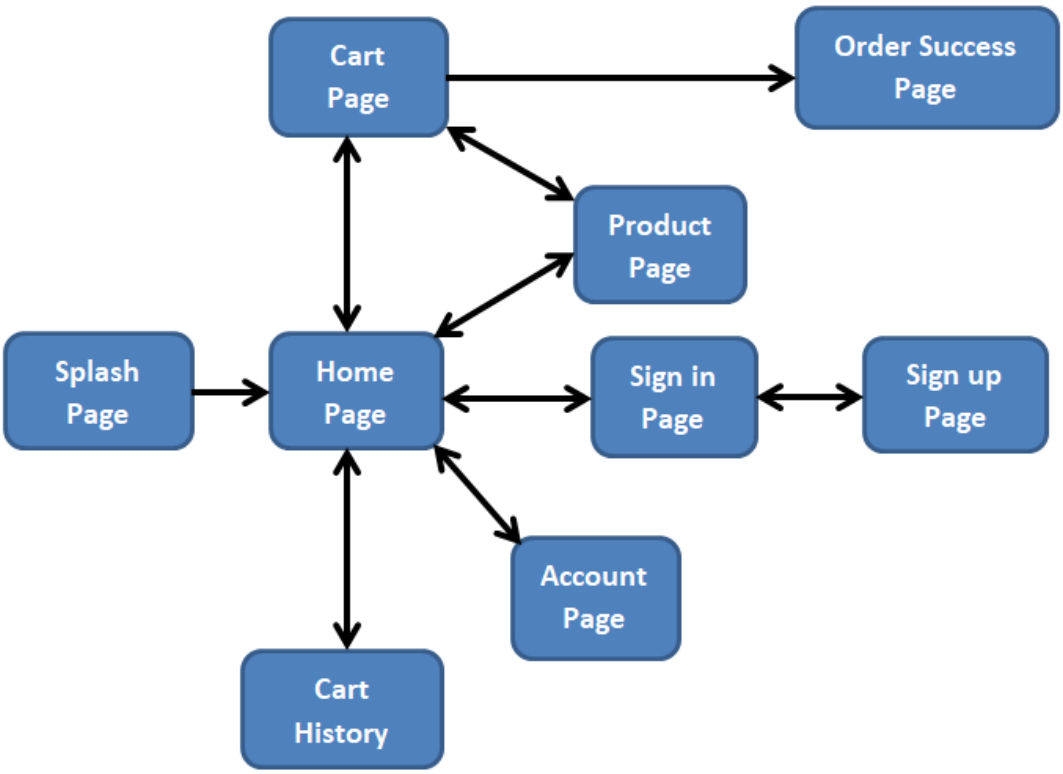
Στην συνέχεια το Sign in Page είναι η σελίδα με την οποία μπορεί ο χρήστης να κάνει σύνδεση στον λογαριασμό του. Αυτό γίνεται εφικτό με την υλοποίηση της μεθόδου login τον κώδικα της οποία βλέπουμε ακριβώς από κάτω :

```
void login(AuthController authController) {
  String email = emailController.text.trim();
  String password = passwordController.text.trim();
  if (email.isEmpty) {
    ShowMessage("...", title: "Email address");
  } else if (!GetUtils.isEmail(email)) {
    ShowMessage("...", title: "Valid Email address");
  } else if (password.isEmpty) {
```

```
    ShowMessage("...", title: "Password");
  } else {
    authController.login(email, password).then((status) {
      if (status.isSuccess) {
        Get.toNamed(RouteHelper.getInitial());
      } else {
        ShowMessage(status.message);
      }
```

Η μέθοδος login κάνει ελέγχους στα δεδομένα που συμπληρώνει ο χρήστης στην φόρα της σελίδας Sign in. Αν όλοι οι έλεγχοι ικανοποιηθούν τότε μέσω ενός αντικειμένου της κλάσης authController έχουμε πρόσβαση στην αντίστοιχη μέθοδο login της κλάσης authController η οποία δέχεται τις παραμέτρους email και password που συμπλήρωσε ο χρήστης. Στην συνέχεια μέσω της διαδικασίας που αναφέραμε στο Κεφάλαιο 3.1.1, στέλνουμε αυτές τις παραμέτρους μαζί με το token εξουσιοδότησης στον server, ο οποίος ελέγχει αν τα στοιχεία που έδωσε ο χρήστης αντιστοιχούν με τα στοιχεία που είναι ήδη αποθηκευμένα με το συγκεκριμένο token. Έπειτα ο server στέλνει τα δεδομένα του συγκεκριμένου χρήστη στο frontend. Στην συνέχεια ο authController ελέγχει αν πήραμε τα δεδομένα από τον server, αν τα δεδομένα ήρθαν τότε η μεταβλητή isSuccess γίνεται true και ανακατευθύνει τον χρήστη στο Home Page.

3.2.6 Routes



Σχήμα 3.15: Αρχιτεκτονική Routing

Το Σχήμα 3.15 απεικονίζει τις διαδρομές μεταξύ των σελίδων.

Στον Φάκελο Routes δημιουργήσαμε το αρχείο route-helper το οποίο υλοποιεί όλες τις διαδρομές του mobile Application. Πάμε να δούμε πως γίνεται αυτή η υλοποίηση :

```
class RouteHelper {
 static const String initial = "/";
 static const String splashScreen = "/splash-screen";
 static const String cartPage = "/cart-page";
 ...}
```

Στην κλάση RouteHelper έχουμε την δήλωση των String μεταβλητών οι οποίες παίρνουν τιμή τις διαδρομές. Για παράδειγμα η μεταβλητή initial θα αναφέρεται στο Home Page, η μεταβλητή splashScreen θα αναφέρεται στο splash Page κ.τ.λ.

```
static List<GetPage> routes = [
   GetPage(name: initial, page: () => HomePage()),
   GetPage(name: splashScreen, page: () => SplashScreen()),
   ... ]
```

Όπως βλέπουμε στον παραπάνω κώδικα δημιουργήσαμε την λίστα routes που είναι αντικείμενο της κλάσης List η οποία ανήκει στην βιβλιοθήκη 'dart:core', το παρεχόμενο αυτής της λίστας είναι τύπου GetPage. Η GetPage είναι μια κλάση η οποία ανήκει στον state management τομέα του Getx package. Επίσης ο Constructor της κλάσης GetPage έχει required δύο παραμέτρους, την παράμετρο τύπου String 'name', και την παράμετρο τύπου Widget Function() 'page'. Όπως καταλάβαμε και από τον πιο πάνω κώδικα στις παραμέτρους 'name' βάζουμε τις διαδρομές και στην παράμετρο 'page' βάζουμε την σελίδα στην οποία θέλουμε να οδηγούν οι συγκεκριμένες διαδρομές. Τώρα το μόνο που έχουμε να κάνουμε είναι να δημιουργήσουμε τις μεθόδους που θα μας επιστρέφουν τις διαδρομές. Σε αυτές τις μεθόδους θα έχουμε πρόσβαση από άλλες κλάσεις μέσω αντικειμένων της κλάσης RouteHelper. Στον πιο κάτω κώδικα παρουσιάζεται η υλοποίηση αυτών των μεθόδων :

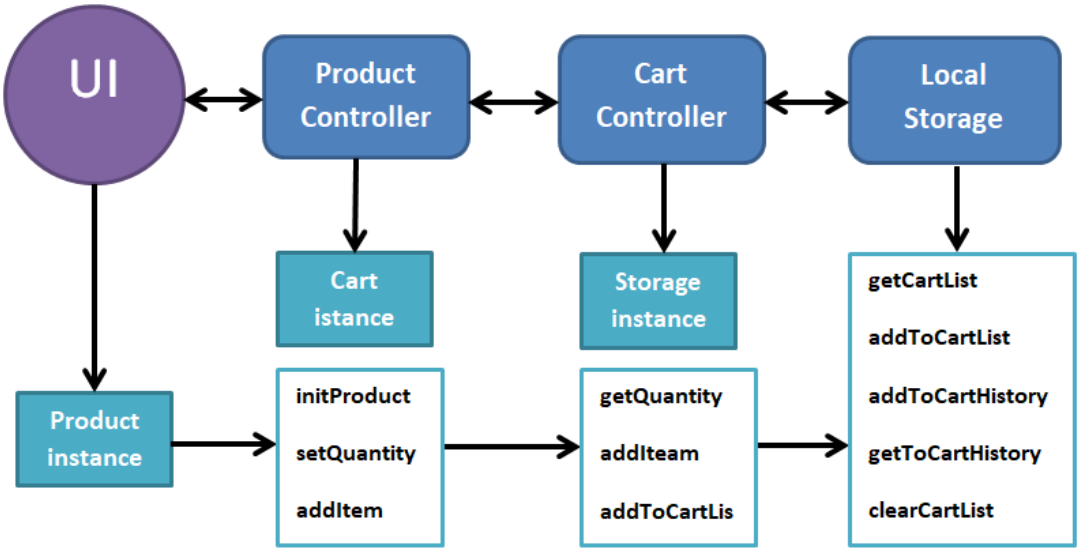
```
static String getsplashScreen() => '$splashScreen';
 static String getInitial() => '$initial';
   ...
```

Ο τελεστής '=>' δεν είναι τίποτα άλλο παρά 'return...'. Εν κατακλείδι καταλαβαίνουμε πόσο εύκολη έγινε η διαδικασία της δημιουργίας ενός Routing System χρησιμοποιώντας το state management του Getx Package.

3.2.7 Cart

Από το Σχήμα 3.16 καταλαβαίνουμε πώς κατασκευάζεται ένα καλάθι αγορών και πώς λειτουργεί με την χρήση των controllers. Το UI θα έχει ένα instance του product controller, ο product controller θα έχει ένα instance του cart controller, και ο cart controller θα έχει ένα instance του local storage, δηλαδή τύπου SharedPreferences.

Στον Φάκελο cart δημιουργήσαμε δύο αρχεία, το cart-history-page, και το cart-page. Το cart-history-page είναι η σελίδα η οποία απεικονίζει τις προηγούμενες παραγγελίες του χρήστη από την εφαρμογή.



Σχήμα 3.16: Αρχιτεκτονική του shopping Cart

Αυτό το επιτυγχάνει με την δημιουργία της μεταβλητής που βλέπουμε πιο κάτω:

```
var getCartHistoryList =
 Get.find<CartController>().getCartHistoryList().reversed.toList();
```

Με την μέθοδο find του Getx package έχουμε πρόσβαση στην μέθοδο getCartHistoryList του CartController. Η μέθοδος getCartHistoryList είναι μία μέθοδος τύπου list που περιέχει αντικείμενα τύπου CartModel. Επειδή η μέθοδος αποθηκεύει τις τελευταίες παραγγελίες στο τέλος της λίστας, και εμείς θέλουμε στην μεταβλητή getCartHistoryList να έχουμε την τελευταία παραγγελία στην αρχή της λίστας χρησιμοποιούμε την μέθοδο reversed.toList().

```
onTap: () {
  popularProduct.addItem(product);}
```

Το cart-page είναι η σελίδα στην οποία πηγαίνουν τα προϊόντα που σκοπεύει να παραγγείλει ο χρήστης και γίνονται οι παραγγελίες. Στον folder food έχουμε τα αρχεία PopularFoodDetail και RecommendedFoodDetail. Όταν ο χρήστης πατάει το κουμπί "Send to Cart" εκτελείται η μέθοδος addItem του CartController η οποία δέχεται σαν παράμετρο ένα product που είναι τύπου productModel και αυξάνει την μεταβλητή quantity του CartController. Αυτό το product καταλήγει στην μεταβλητή Iteams του CartController με την βοήθεια της μεθόδου addToCartList. Έπειτα έχουμε πρόσβαση στην μεταβλητή Iteams δημιουργώντας ένα instance του CartController στην κλάση cartPage.

```
GestureDetector(
  onTap: () {
  if (Get.find<AuthController>().userLoggedIn()) {
     var cart =Get.find<CartController>().getItems;
     var user =Get.find<UserController>().userModel;
     PlaceOrderBody placeOrder =PlaceOrderBody(cart: cart,
     user.userName, user.phone  ,.....);
      Get.find<OrderController>().placeOrder(placeOrder, _callback);
```

```
    } else {
      Get.toNamed(RouteHelper.getSignInPage());
  }}
```

Ο πιο πάνω κώδικας παρουσιάζει τον τρόπο με τον οποίο γίνεται το post request της παραγγελίας από το frontend στον server. Όταν ο χρήστης πατήσει το κουμπί 'Buy Now', εκτελείται ο πιο πάνω κώδικας. Αν ο χρήστης δεν είναι Logged In τότε ανακατευθύνεται στο SignInPage. Αν ο χρήστης είναι Logged In τότε δημιουργούμε την μεταβλητή cart και βάζουμε μέσα τα προϊόντα που είναι στην μεταβλητή Items του CartController τα οποία είναι τα προϊόντα που επέλεξε ο χρήστης να στείλει στο cartPage. Στην συνέχεια δημιουργούμε την μεταβλητή user στην οποία βάζουμε τα στοιχεία του χρήστη που βρίσκονταν στην μεταβλητή userModel του CartController. Επιπλέον δημιουργούμε το αντικείμενο placeOrder το οποίο είναι αντικείμενο της PlaceOrderBody που είναι κλάση του μοντέλου place-Order. Παράλληλα δίνουμε στην παράμετρο cart την μεταβλητή cart που μόλις δημιουργήσαμε. Επίσης με την μεταβλητή user συμπληρώνουμε και τις άλλες παραμέτρους όπως το όνομα του χρήστη, το τηλέφωνο κ.τ.λ. Στην συνέχεια βάζουμε το αντικείμενο placeOrder σαν παράμετρο στην μέθοδο placeOrder που είναι μία Future μέθοδος της κλάσης OrderController και παίρνει δύο παραμέτρους, μία παράμετρο τύπου PlaceOrderBody και μία callback μέθοδο. Η μέθοδος placeOrder είναι εκείνοι που στέλνει τα δεδομένα (την παραγγελία), στον server.

```
void _callback(bool isSuccess, String message, String orderID) {
   if (isSuccess) {
     Get.find<CartController>().clear();
     Get.find<CartController>().removeCartSharedPreference();
     Get.find<CartController>().addToHistory();
     Get.offNamed(RouteHelper.getOrderSuccessPage(orderID, "success"));
   } else {ShowMessage(message);
   }}
```

Ο πιο πάνω κώδικας μας παρουσιάζει την callback μέθοδο, η οποία είναι μια από τις παραμέτρους της μεθόδου placeOrder. Ο σκοπός της callback είναι : Αν η placeOrder έστειλε τα δεδομένα με επιτυχία στον server τότε καλεί την μέθοδο clear του CartController η οποία κάνει την μεταβλητή Iteams άδεια λίστα. Παράλληλα καλεί την μέθοδο removeCartSharedPreference του CartController, η οποία με την σειρά της καλεί την μέθοδο removeCartSharedPreference του CartRepo η οποία αδειάζει από την τοπική μνήμη τις μεταβλητές CART-LIST και CART-HISTORY-LIST της κλάσης AppConstants. Επίσης καλεί την μέθοδο addToHistory η οποία προσθέτει την συγκεκριμένη παραγγελία στο CartHistoryPage. Στην συνέχεια με την μέθοδο getOrderSuccessPage του RouteHelper ανακατευθύνει τον χρήστη στην OrderSuccessPage. Τέλος αν κάτι πάει στραβά τότε εμφανίζει ένα μήνυμα σφάλματος.

Και με την αναφορά της λειτουργίας του Cart φτάσαμε στο τέλος αυτού του σημαντικού υποκεφαλαίου στο οποίο έχουμε εξηγήσει πως έγινε η υλοποίηση του mobile Application. Στο επόμενο υποκεφάλαιο θα εξηγήσουμε πως έχει γίνει η υλοποίηση του Backend και της βάσης δεδομένων.

3.3 Υλοποίηση του Backend και Βάσης Δεδομένων

Οι σύγχρονες διαδικτυακές εφαρμογές συνήθως χωρίζονται σε δύο τύπους: Στατικές και δυναμικές. Στατικές εφαρμογές είναι εκείνες όπου το περιεχόμενο της εφαρμογής είναι προ δημιουργημένο για όλους, και οι χρήστες δεν μπορούν να το προσαρμόσουν. Από την άλλη πλευρά, στόχος αυτής της διπλωματικής εργασίας είναι η κατασκευή μιας προσαρμόσιμης και διαδραστικής εφαρμογής. Επομένως μια στατική εφαρμογή δεν ικανοποιεί τις απαιτήσεις της παρούσας διπλωματικής εργασίας. Ως εκ τούτου, ο τρόπος για να επιτευχθεί ο στόχος μας, είναι η κατασκευή μιας δυναμικής εφαρμογής. Το κύριο χαρακτηριστικό της δυναμικής εφαρμογές είναι η δυνατότητα εξατομίκευσης της εφαρμογής στον μεμονωμένο χρήστη. Για να γίνει αυτό πρέπει να αποθηκεύσουμε πληροφορίες που σχετίζονται με τον χρήστη και να πραγματοποιήσουμε μετασχηματισμούς δεδομένων εάν είναι απαραίτητο. Η τυπική προσέγγιση είναι η χρήση ενός backend.

Στα πλαίσια του συγκεκριμένου υποκεφαλαίου θα εξηγήσουμε αναλυτικά πως έχουν υλοποιηθεί όλα τα βασικά στοιχεία από τα οποία αποτελείται το Backend και η Βάση Δεδομένων.

Αρχίζουμε την δημιουργία του Laravel project στο περιβάλλων του VS Code, πηγαίνοντας στον φάκελο που θέλουμε να δημιουργήσουμε το project και εκτελούμε την εντολή :

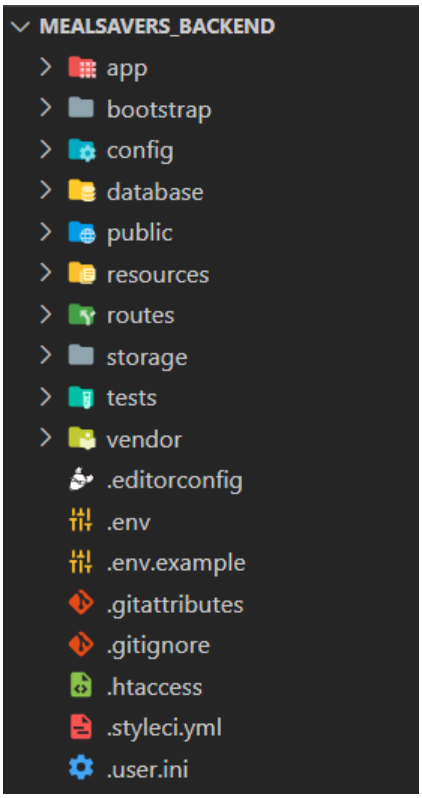
```
composer create-project laravel/laravel mealsavers-backend
```

Το αποτέλεσμα εκτέλεσης της πιο πάνω εντολής είναι η δημιουργία ενός φακέλου με το όνομα mealsavers-backend. Μέσα σε αυτόν δημιουργήθηκε ένα Laravel project με όλα τα απαραίτητα configurations και dependencies. Η δομή αποτελείται από 10 φακέλους και αρκετά αρχεία όπως βλέπουμε στο Σχήμα 3.17. Στο Σχήμα 3.18 βλέπουμε τα περιεχόμενα του φακέλου admin, που δημιουργήθηκαν όταν κάναμε εγκατάσταση το Admin Panel στο laravel project. Τα περιεχόμενα του φακέλου admin δημιουργούν το UI του admin Panel στο οποίο έχουν πρόσβαση οι επιχειρήσεις, και μπορούν μέσα από αυτό να ανεβάζουν προϊόντα στο Application και να βλέπουν τις παραγγελίες των χρηστών. Επίσης έχουμε αναφέρει την λειτουργία των admin panel Controllers και γενικότερα την λειτουργία του admin panel στο κεφάλαιο 3.1.2 Αρχιτεκτονική Backend.

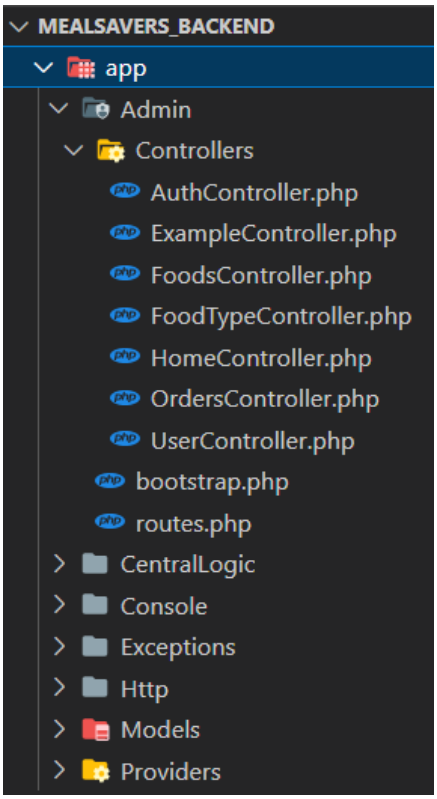
Ένα αρχείο που αξίζει να αναφερθεί είναι το αρχείο '.env' στο οποίο δηλώνουμε τις μεταβλητές περιβάλλοντος του έργου. Επίσης στο αρχείο '.env' κάνουμε την σύνδεση του backend και της Βάσης Δεδομένων συμπληρώνοντας τις μεταβλητές περιβάλλοντος οι οποίες αναφέρουν τις πληροφορίες της βάσης δεδομένων με την οποία θέλουμε να συνδεθούμε.

Ο πιο κάτω κώδικας απεικονίζει τις μεταβλητές περιβάλλοντος που είναι υπεύθυνες για την σύνδεση με την Βάση Δεδομένων :

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=mealsavers_backend
DB_USERNAME=root
DB_PASSWORD=********
```



Σχήμα 3.17: Laravel Folder structure



Σχήμα 3.18: Admin Folder

Επιπλέον στο αρχείο .env δηλώνουμε τις μεταβλητές περιβάλλοντος που έχουν να κάνουν με το ίδιο το έργο, όπως το όνομα και το URL.

Στον πιο κάτω κώδικα δηλώνουμε τις μεταβλητές περιβάλλοντος οι οποίες αναφέρονται στο έργο :

```
APP_NAME=MealSavers_Backend
APP_ENV=local
APP_KEY=base64:rcXGipMyDBKQXQDF8884GUq7t2VgwpD/dYh=
APP_DEBUG=true
APP_URL=http://127.0.0.1:8000/
```

3.3.1 Verification

Η επαλήθευση των χρηστών στο Backend γίνεται με την βοήθεια του Middleware. Το Middleware παρέχει έναν βολικό μηχανισμό για την επιθεώρηση και το φιλτράρισμα των αιτημάτων HTTP που εισέρχονται στο Backend. Για παράδειγμα, το Middleware λογισμικό, επαληθεύει ότι ο χρήστης της εφαρμογής έχει πιστοποιηθεί. Επιπρόσθετα το Middleware λειτουργεί ως γέφυρα μεταξύ ενός αιτήματος και μιας αντίδρασης. Δηλαδή επιβεβαιώνει εάν ο πελάτης της εφαρμογής έχει επαληθευτεί ή όχι. Εάν ο πελάτης επιβεβαιωθεί, ανακατευθύνεται στο Home Page, διαφορετικά ανακατευθύνεται στο Sign in Page.

Μετά την εγκατάσταση του middleware, δημιουργείται ο φάκελος Middleware μέσα στον φάκελο Http, ο οποίος περιέχει τα αρχεία που υλοποιούν τις κλάσεις που χρειαζόμαστε για την προστασία των διαδρομών και την δημιουργία του token.

```
Route::group(['prefix' => 'customer', 'middleware' => 'auth:api'], function () {
   Route::post('login', 'CustomerAuthController@login');
   ...
   Route::group(['prefix' => 'order'], function () {
          Route::post('place', 'OrderController@place_order');
```

Το πιο πάνω κομμάτι κώδικα βρίσκεται στο αρχείο api.php στον φάκελο api μέσα στον φάκελο routes. Στο συγκεκριμένο κομμάτι κώδικα βλέπουμε πως προστατεύουμε τα routes με την χρήση του middleware. Για να μπορέσουμε να έχουμε πρόσβαση σε αυτές τις διαδρομές πρέπει να έχουμε το token εξουσιοδότησης. Στο κεφάλαιο 3.1.2 έχουμε εξηγήσει αναλυτικά τι είναι το EndPoint, route, group, get και post, που βλέπουμε στον πιο πάνω κώδικα. Η δημιουργία του token γίνεται με την εντολή :

```
$token = $user->createToken('CustomerAuth')->accessToken;
```

Όπου η μεταβλητή user είναι η δημιουργία ενός χρήστη στην βάση δεδομένων μέσω του μοντέλου User όπως απεικονίζει ο πιο κάτω κώδικας.

Με τον συγκεκριμένο τρόπο δημιουργούμε ένα token για κάθε χρήστη της εφαρμογής.

```
$user = User::create([
          'f_name' => $request->f_name,
          'email' => $request->email,
          'phone' => $request->phone,
          'password' => bcrypt($request->password),
      ]);
```

Στον πιο κάτω κώδια βλέπουμε την μέθοδο login του UserController στην οποία γίνεται ο έλεγχος αυθεντικοποίησης τον στοιχείων που έδωσε ο χρήστης από το frontend.

```
public function login(Request $request)
   {
       $input = $request->only('email', 'password');
       $jwt_token = null;
       if (!$jwt_token = JWTAuth::attempt($input)) {
           return response()->json([
               'success' => false,
               'message' => 'Invalid Email or Password',
           ], 401);
       }
       $user = Auth::user();
       return response()->json([
           'success' => true,
           'token' => $jwt_token,
           'user' => $user
       ]);}
```

Αν το email και password είναι λάθος, τότε η μέθοδος login επιστρέφει ένα json reponse το οποίο περιέχει ένα associative array. Ο associative array επιστρέφει false στο Key 'success' και το μήνυμα 'Invalid Email or Password' στο key 'message' και το status code 401 'Unauthorized'. Στην περίπτωση που το email και password είναι σωστά τότε η μέθοδος login επιστρέφει ένα json reponse με το Key 'success' να είναι true , το key 'token' να έχει το token του συγκεκριμένου χρήστη, και στο key 'user' να έχει τα στοιχεία του χρήστη.

3.3.2 Υλοποίηση Βάσης Δεδομένων

Όπως έχουμε αναφέρει στο Κεφάλαιο 2.8, η βάση δεδομένων υλοποιείται στο phpMyAdmin με την βοήθεια του Laravel Migration. Το Laravel migration είναι ένα βασικό χαρακτηριστικό της Laravel που μας επιτρέπει να δημιουργήσουμε πίνακες στη βάση δεδομένων. Επίσης επιτρέπει την τροποποίηση και το μοίρασμα του σχήματος (schema), της βάσης δεδομένων. Επιπλέον με το migration μπορούμε να τροποποιήσουμε τους πίνακες προσθέτοντας μια νέα στήλη ή διαγράφοντας μια υπάρχουσα στήλη.

Τα πλεονεκτήματα του Laravel Migration είναι :

1. Web applications μπορούν να χτιστούν γρήγορα με αυτό.
2. Είναι εύκολο να το μάθεις.
3. Το documentation είναι εύκολα διαθέσιμο.
4. Πληροί σχεδόν κάθε κριτήριο που απαιτείται από τη σύγχρονη δημιουργία διαδικτυακών εφαρμογών.

Πως λειτουργεί το migration ; Σε ένα database migration, μετακινούμαι δεδομένα από βάσεις δεδομένων προέλευσης σε βάσεις δεδομένων προορισμού. Μετά την πλήρη μετεγκατάσταση των δεδομένων, διαγράφονται οι βάσεις δεδομένων προέλευσης και ανακατευθύνεται η πρόσβαση πελάτη στις βάσεις δεδομένων προορισμού.

Μπορούμε να δημιουργήσουμε Laravel Migrations εκτελώντας την εντολή :

```
php artisan make:migration 'migrations name'
```

Με την εκτέλεση της πιο πάνω εντολής, θα δημιουργηθεί ένα migration μέσα στον φάκελο migrations που βρίσκεται μέσα στον φάκελο database. Ο πιο κάτω κώδικα μας απεικονίζει ένα migration.

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
    class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
```

```
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        }
```

Με το συγκεκριμένο migration δημιουργούμε το table users με τις στήλες id, name, email, password κ.τ.λ, στην βάση δεδομένων. Στο Σχήμα 3.19 απεικονίζεται το αποτέλεσμα αυτού του Migration στην βάση δεδομένων phpMyAdmin.

```
SELECT * FROM `users`
```

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change
☐	2	f_name	varchar(255)	utf8mb4_unicode_ci		No	None			Change
☐	3	phone	varchar(20)	utf8mb4_unicode_ci		No	None			Change
☐	4	email	varchar(255)	utf8mb4_unicode_ci		No	None			Change
☐	5	status	tinyint(4)			No	1			Change
☐	6	email_verified_at	timestamp			Yes	NULL			Change
☐	7	password	varchar(255)	utf8mb4_unicode_ci		No	None			Change
☐	8	remember_token	varchar(100)	utf8mb4_unicode_ci		Yes	NULL			Change
☐	9	created_at	timestamp			Yes	NULL			Change
☐	10	updated_at	timestamp			Yes	NULL			Change

Σχήμα 3.19: Users Table Database

Όπως έχουμε εξηγήσει και στο Κεφάλαιο 3.1.2, οι σχέσεις μεταξύ πινάκων και η ανάθεση κύριων και ξένων κλειδιών, γίνεται στο μοντέλο, το οποίο είναι υπεύθυνο για την διαχείριση του συγκεκριμένου πίνακα στην βάση δεδομένων. Για παράδειγμα, το μοντέλο το οποίο είναι υπεύθυνο για τον πίνακα users, είναι το UserModel.

Στο Σχήμα 3.20 απεικονίζεται ο πίνακας OrderDetail της Βάσης Δεδομένων, ο οποίος αναφέρει τις παραγγελίες που κάνουν POST οι πελάτες από το mobile Application στην Βάση Δεδομένων.

```
SELECT * FROM `order_details`
```

id	order_id	food_id	price	food_details	quantity	remaining	tax_amount	created_at	updated_at	order_status
34	100004	2	7.00	{"id":2,"name":"Sigma Bakeries special","descripti...	1	0	0.76	2022-12-21 19:03:33	2022-12-21 17:03:33	1
35	100005	5	7.00	{"id":5,"name":"Sigma Bakeries Limassol","descript...	1	0	0.76	2022-12-22 22:07:18	2022-12-22 20:07:18	1
36	100006	7	10.00	{"id":7,"name":"Zorbas Special space_ship","descri...	1	0	0.76	2022-12-23 18:43:34	2022-12-23 16:43:34	1
37	100007	6	6.00	{"id":6,"name":"Sunfresh Bakery","description":"He...	1	0	0.76	2022-12-22 22:18:31	2022-12-22 20:18:31	1
38	100008	7	10.00	{"id":7,"name":"Zorbas Special space_ship","descri...	1	0	1.82	2022-12-23 14:02:38	2022-12-23 14:02:38	0
39	100009	5	7.00	{"id":5,"name":"Sigma Bakeries Limassol","descript...	1	0	1.27	2022-12-23 14:04:00	2022-12-23 14:04:00	0

Σχήμα 3.20: OrderDetail Table Database

Στο Σχήμα 3.21 απεικονίζεται ο πίνακας Foods της Βάσης Δεδομένων, ο οποίος αναφέρει το όνομα της επιχείρησης και τα προϊόντα τα οποία δεν έχουν πωληθεί μέχρι το τέλος της ημέρας. Αυτές είναι πληροφορίες που κάνει POST ο υπάλληλος ή ο ιδιοκτήτης της επιχείρησης από την ιστοσελίδα του admin Panel στην Βάση Δεδομένων.

```
SELECT * FROM `foods`
```

id	name	description	price	stars	quantity	remaining	img	location
1	Ermis bakery spacial	Μέσα στην σακούλα που θα παραλάβετε από τον φούρνο...	5	4	5	5	images/b37a1b6f523b1afbb91562ec268f9c5d.gif	Cyprus, Limassol
2	Sigma Bakeries special	Here in Sigma Bakeries we have the best offers, ou...	7	5	5	3	images/cdfa92cd09164e577bd979d43867bfb6.jpg	Limassol, Cyprus
3	Sigma Bakeries special offer	Here in Sigma Bakeries we have the best offers, ou...	7	3	5	4	images/7c4256070303bd142295fb38cd610a4a.jpg	Limassol, Cyprus
4	Regardo Bakery's box	Here in Regardo Bakeries we have the best offers, ...	4	4	5	3	images/39aafef9a78424e55a05e2b59785ae8f.jpg	Kato Polemidia ,Cyprus

Σχήμα 3.21: Foods Table Database

Με την αναφορά της Βάσης Δεδομένων φτάσαμε στο τέλος αυτού του σημαντικού κεφαλαίου. Σε αυτό το κεφάλαιο έχουμε εξηγήσει πως έγινε η υλοποίηση της εφαρμογής. Στο επόμενο κεφάλαιο θα δούμε τα δύο σενάρια χρήσης της εφαρμογής, τα οποία είναι η προοπτική του χρήστη της mobile Εφαρμογής, και η προοπτική της επιχείρησης στην ιστοσελίδα του Admin Panel.

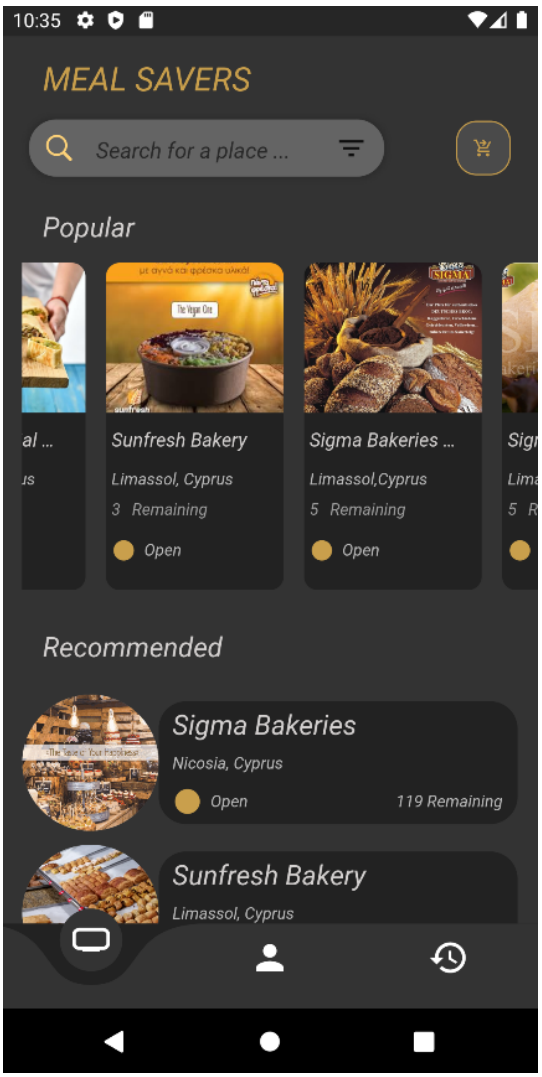
Κεφάλαιο 4ο: Η εφαρμογή MealSavers

Σε αυτό το κεφάλαιο θα προσομοιώσουμε τα δύο σενάρια χρήσης της εφαρμογής. Το πρώτο σενάριο θα είναι ένα σενάριο προοπτικής του πελάτη. Θα δούμε όλα τα βήματα που πρέπει να κάνει ο πελάτης στο mobile Application, όπως η επιλογή επιχείρησης και προϊόντος, η δημιουργία λογαριασμού, η είσοδος στον λογαριασμό και τέλος η παραγγελία προϊόντων. Το δεύτερο σενάριο θα είναι ένα σενάριο προοπτικής του ιδιοκτήτη καταστήματος. Θα δούμε όλα τα βήματα που πρέπει να κάνει ο ιδιοκτήτης καταστήματος στην ιστοσελίδα του Admin Panel, για να μπορέσει να ανεβάσει τα προϊόντα του στο mobile Application και να βλέπει τις παραγγελίες που δέχεται το κατάστημα.

4.1 Σενάριο χρήσης Πελάτη

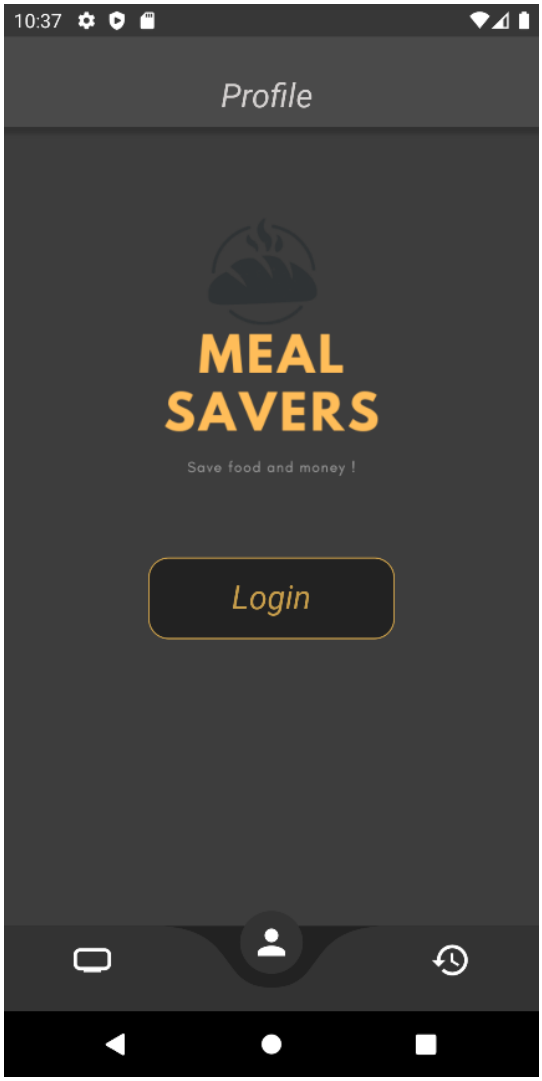
Εγκατάσταση εφαρμογής μέσω του Play Store για Andriod συσκευές ή εγκατάσταση εφαρμογής μέσω του App Store για iOS συσκευές. Η πρώτη σελίδα που θα παρουσιαστεί μετά την εγκατάσταση και την εκτέλεση της εφαρμογής είναι η Splash Page την οποία βλέπουμε στο Σχήμα 4.1.

Σχήμα 4.1: Splash Page

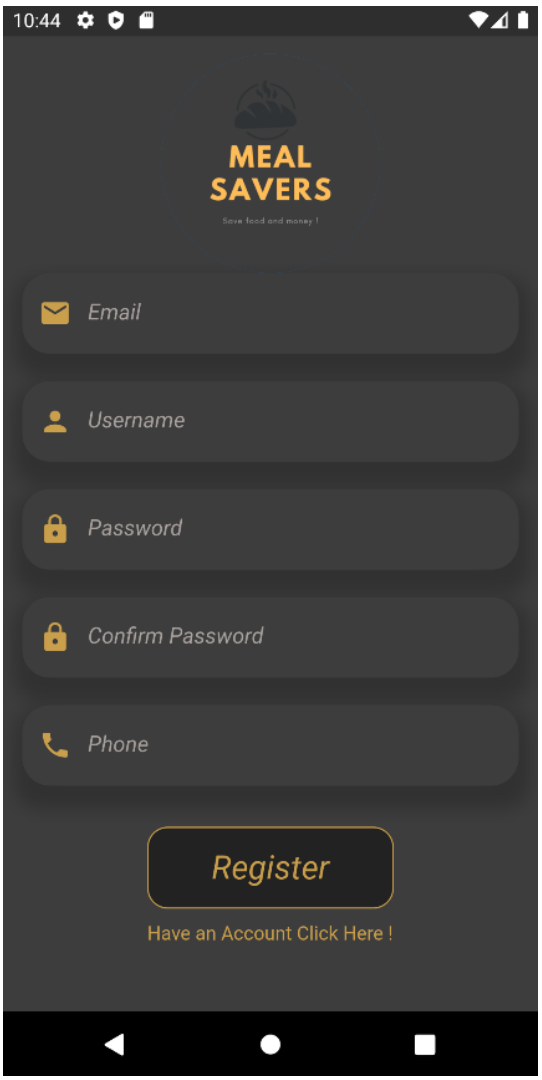


Σχήμα 4.2: Home Page

Αμέσως μετά την φόρτωση των δεδομένων από τον server ο χρήστης θα πλοηγηθεί στο Home Page το οποίο βλέπουμε στο Σχήμα 4.2. Από το navigation bar του Home Page ο χρήστης μπορεί να πλοηγηθεί στο Sign In Page Σχήμα 4.3. Από το Sign In Page μπορεί να πλοηγηθεί στο Sign Up Page Σχήμα 4.4, στο οποίο μπορεί να δημιουργήσει λογαριασμό τώρα ή θα γίνει ανακατεύθυνση του χρήστη στο Sign In Page όταν του ζητηθεί λογαριασμός στο μετέπειτα στάδιο της παραγγελίας. Παράλληλα από το Cart Icon ο χρήστης μπορεί να πλοηγηθεί στο καλάθι (Cart Page), και από το navigation bar μπορεί να πλοηγηθεί στο ιστορικό του (History Page), φυσικά καμία από τις δύο σελίδες δεν έχει περιεχόμενο στο παρόν στάδιο.

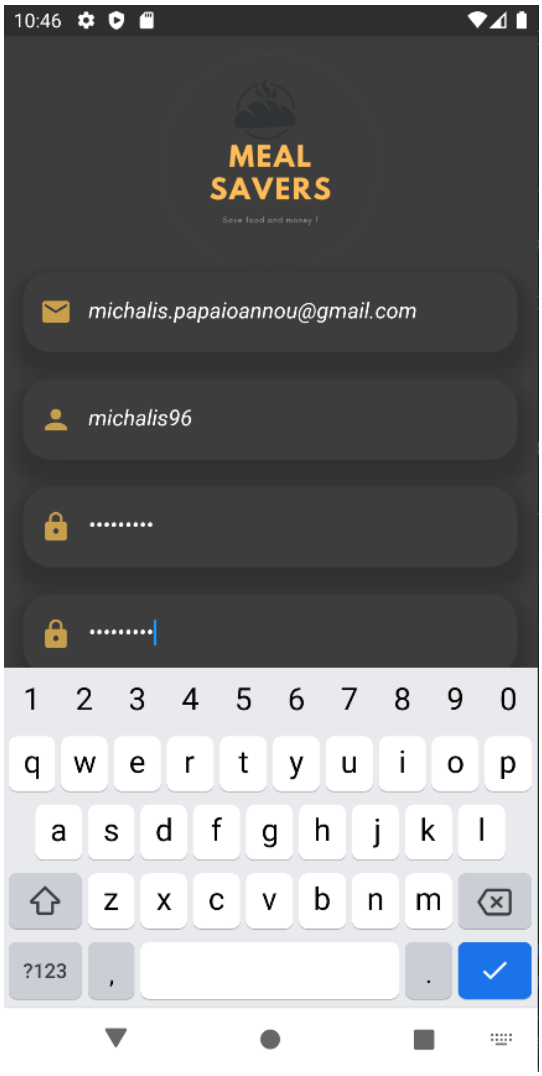


Σχήμα 4.3: Sign in Page

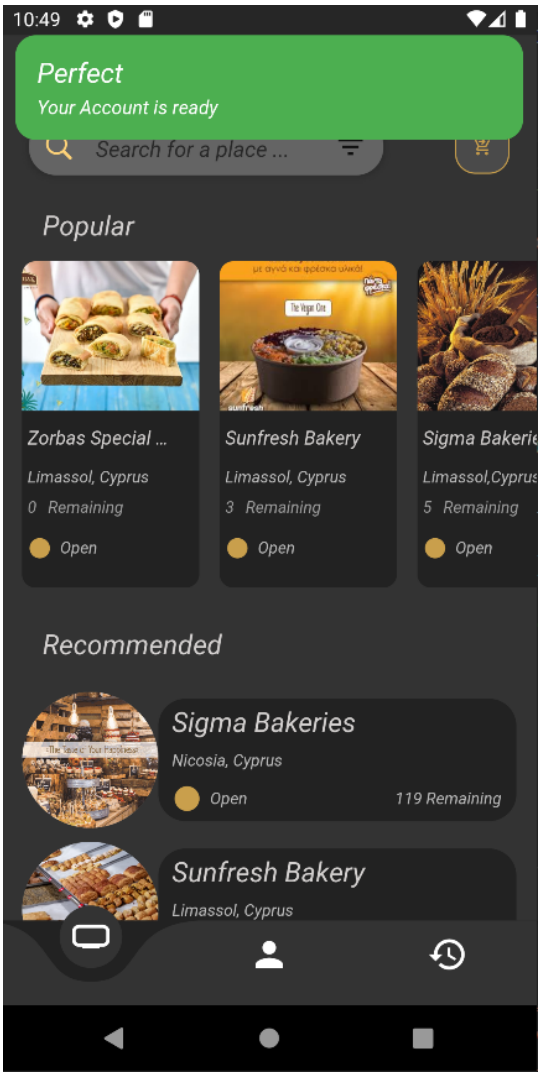


Σχήμα 4.4: Sign up Page

Όταν ο χρήστης συμπληρώσει όλα τα στοιχεία της φόρμας του Sign up Page και είναι έγκυρα, τότε θα συνδεθεί στον λογαριασμό του και θα πλοηγηθεί στο Home Page όπως βλέπουμε στα Σχήματα 4.5 και 4.6.

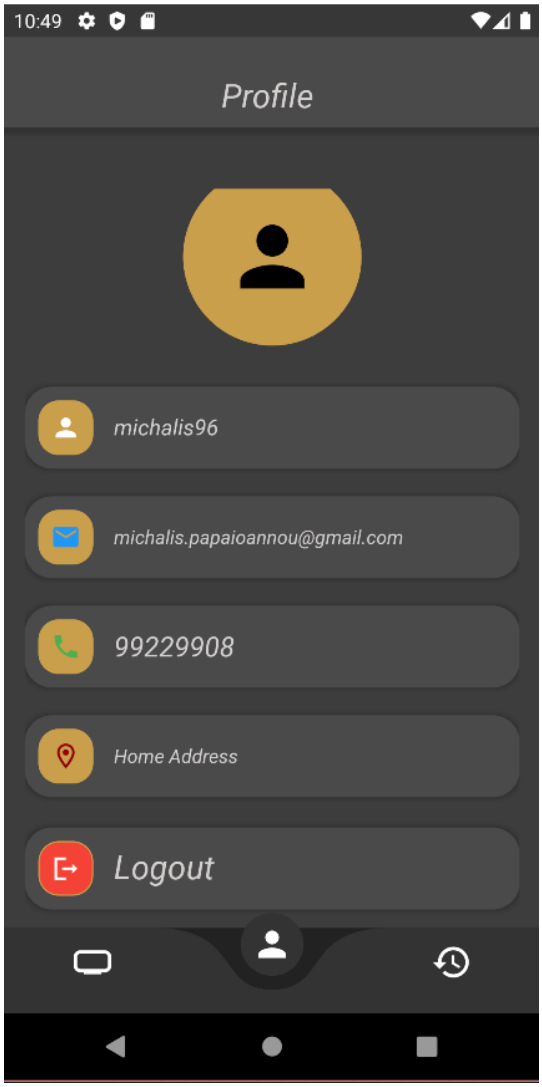


Σχήμα 4.5: Sign up Page

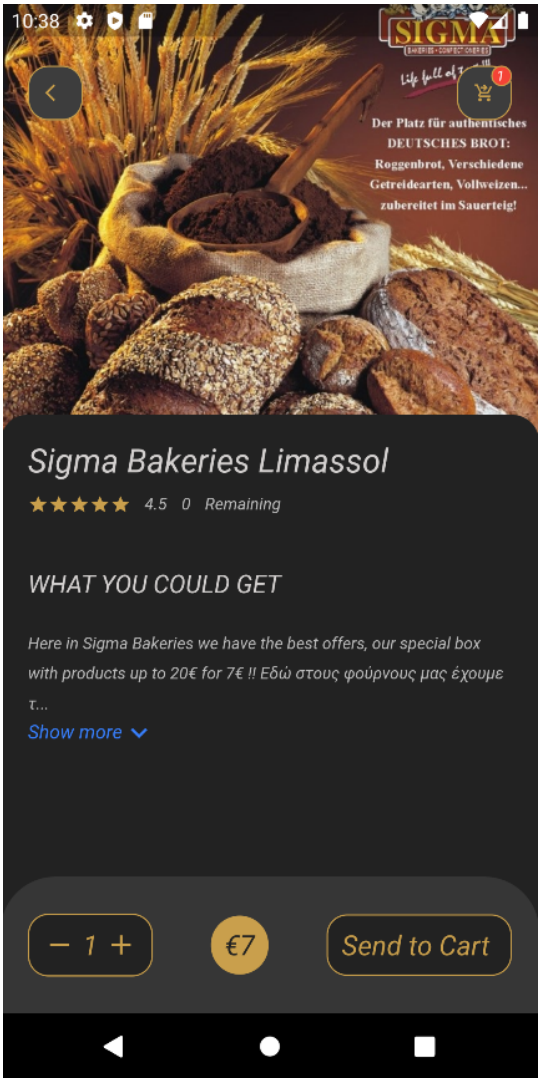


Σχήμα 4.6: Home Page

Σε αυτό το σημείο ο χρήστης είναι συνδεδεμένος στον λογαριασμό του και μπορεί από το navigation bar να πλοηγηθεί στο Profile Page Σχήμα 4.7, ή να επιλέξει κάποιον από τους διαθέσιμους φούρνους και να πλοηγηθεί στα προϊόντα που τους περίσσεψαν (Food Page), Σχήμα 4.8.

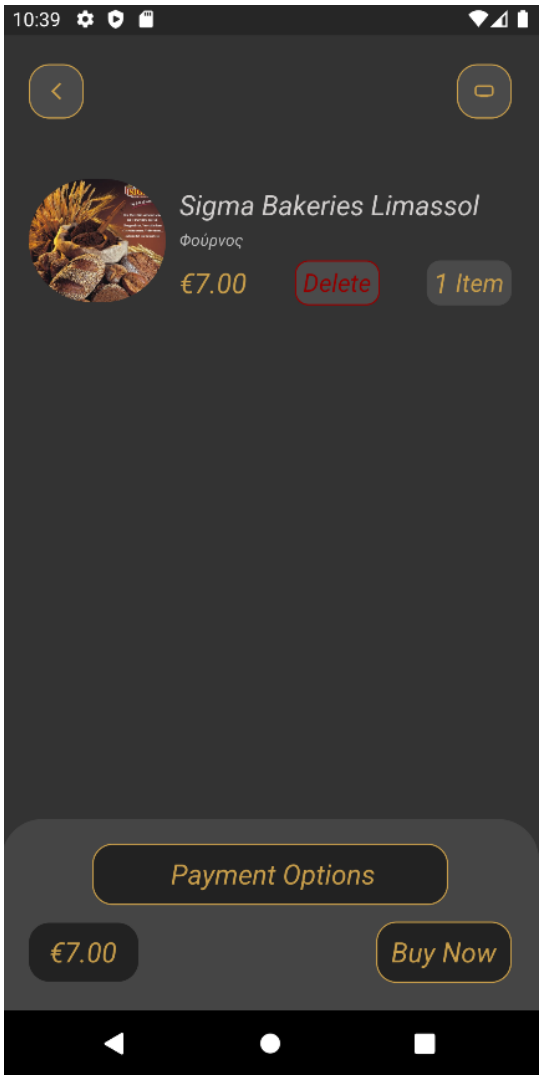


Σχήμα 4.7: Profile Page

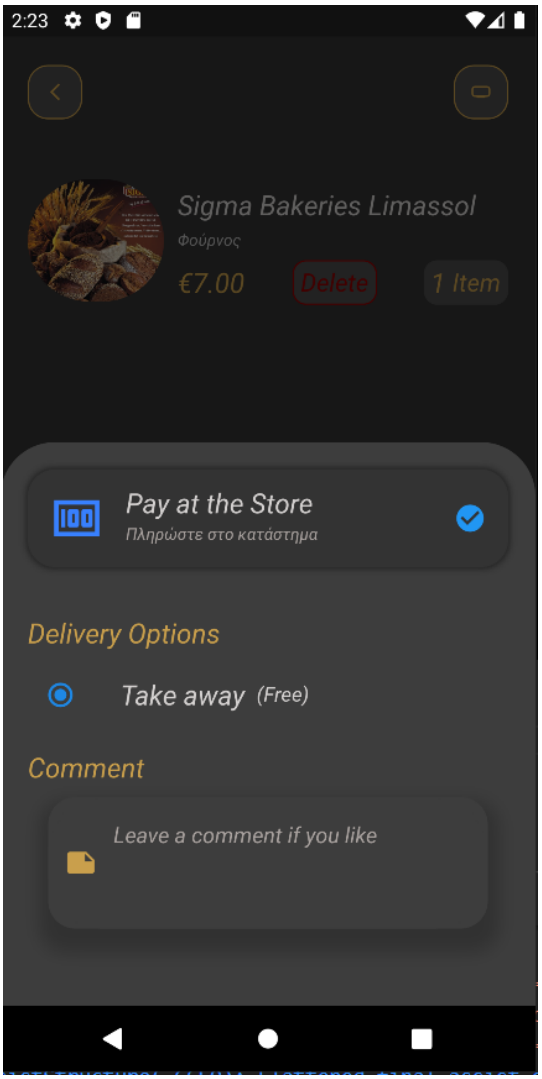


Σχήμα 4.8: Food Page

Στην σελίδα Food Page ο χρήστης μπορεί να διαβάσει στο description τι θα έχει το περιεχόμενο του πακέτου που σκέφτεται να παραγγείλει. Το πακέτο θα έχει κάποια φαγητά που δεν πωλήθηκαν μέχρι το τέλος της ημέρας από την επιχείρηση σε χαμηλότερες τιμές από την κανονική τους. Στην συνέχεια επιλέγουμε αν θέλουμε ένα ή δύο πακέτα, ανάλογα φυσικά και με το πόσα έχει ανεβάσει ο φούρνος, και πατάμε το κουμπί Send to Cart το οποίο βάζει τα προϊόντα στο καλάθι (Cart Page). Το Cart Page απεικονίζεται στο Σχήμα 4.9. Ο χρήστης μπορεί να πλοηγηθεί στο Cart Page πατώντας το Cart Icon.

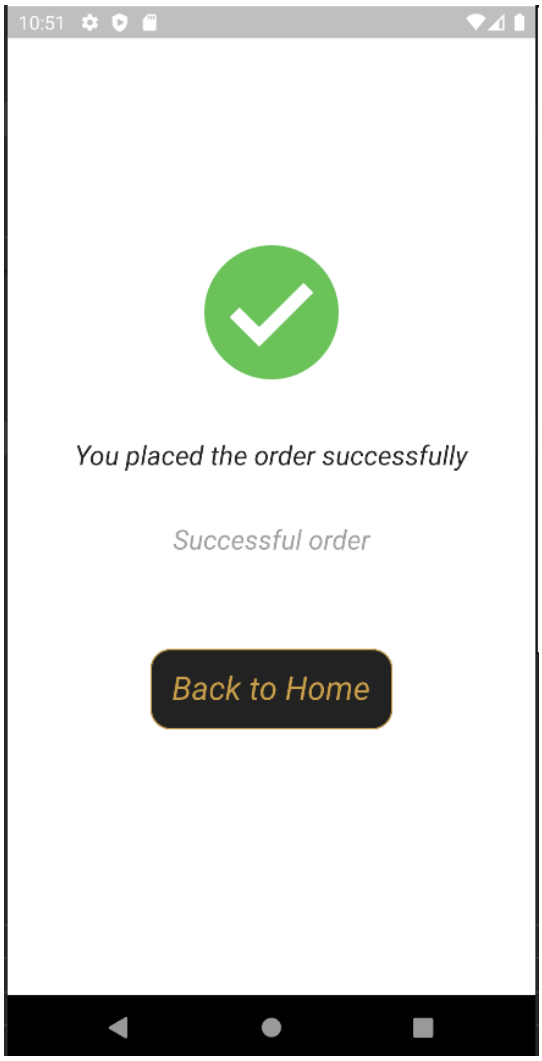


Σχήμα 4.9: Cart Page

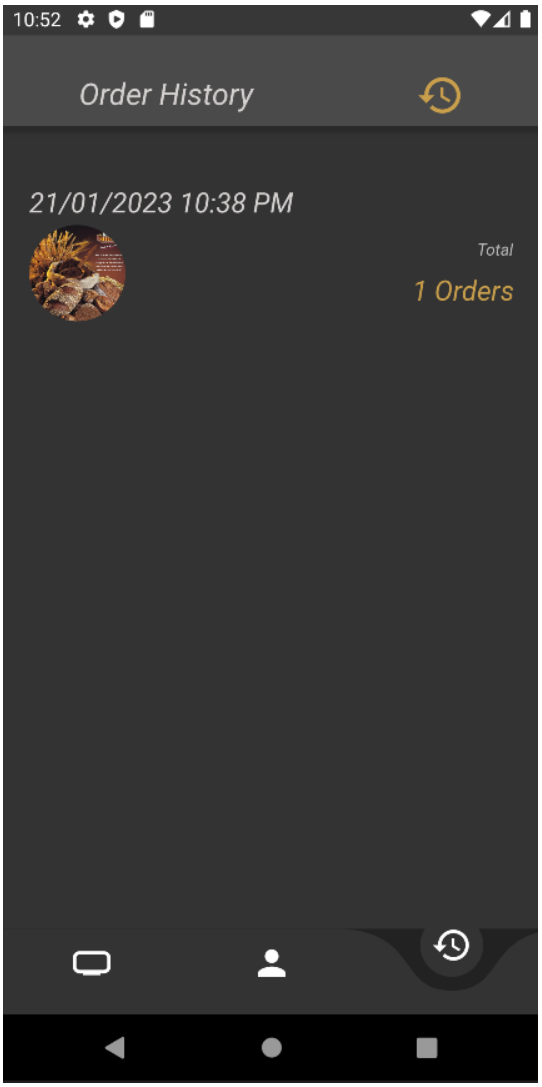


Σχήμα 4.10: Option Page

Από το Cart Page ο χρήστης μπορεί να διαγράψει το πακέτο που σκέφτεται να παραγγείλει, ή πατώντας πάνω στο εικονίδιο του πακέτου να πλοηγηθεί πίσω στην σελίδα του συγκεκριμένου πακέτου. Επίσης ο χρήστης πατώντας το κουμπί Payment Options, εμφανίζεται το pop-up που απεικονίζει στο Σχήμα 4.10, στο οποίο ο χρήστης μπορεί να αφήσει ένα Comment στον υπάλληλο του φούρνου σχετικά με την παραγγελία του. Στα πλαίσια της παρούσας διπλωματικής εργασίας δεν έχουμε δημιουργήσει άλλους τρόπους πληρωμής εκτός από πληρωμή στο κατάστημα, επίσης δεν έχουμε άλλη επιλογή εκτός Take away από το κατάστημα, αλλά ελπίζουμε να τα υλοποιήσουμε σε μελλοντικές επεκτάσεις της εφαρμογής. Τελειώνοντας αν ο χρήστης αποφάσισε να αγοράσει το πακέτο πατάει στο κουμπί 'Buy Now' το οποίο στέλνει την παραγγελία του στον φούρνο. Παράλληλα ο χρήστης πλοηγείται στο success Page το οποίο απεικονίζεται στο Σχήμα 4.11.



Σχήμα 4.11: Success Page



Σχήμα 4.12: Order History Page

Τέλος το Σχήμα 4.12 απεικονίζει το Order History Page στο οποίο καταχωρήθηκε η παραγγελία που ολοκλήρωσε ο χρήστης. Στο επόμενο υποκεφάλαιο θα δούμε το δεύτερο σενάριο που θα είναι ένα σενάριο προοπτικής του ιδιοκτήτη καταστήματος.

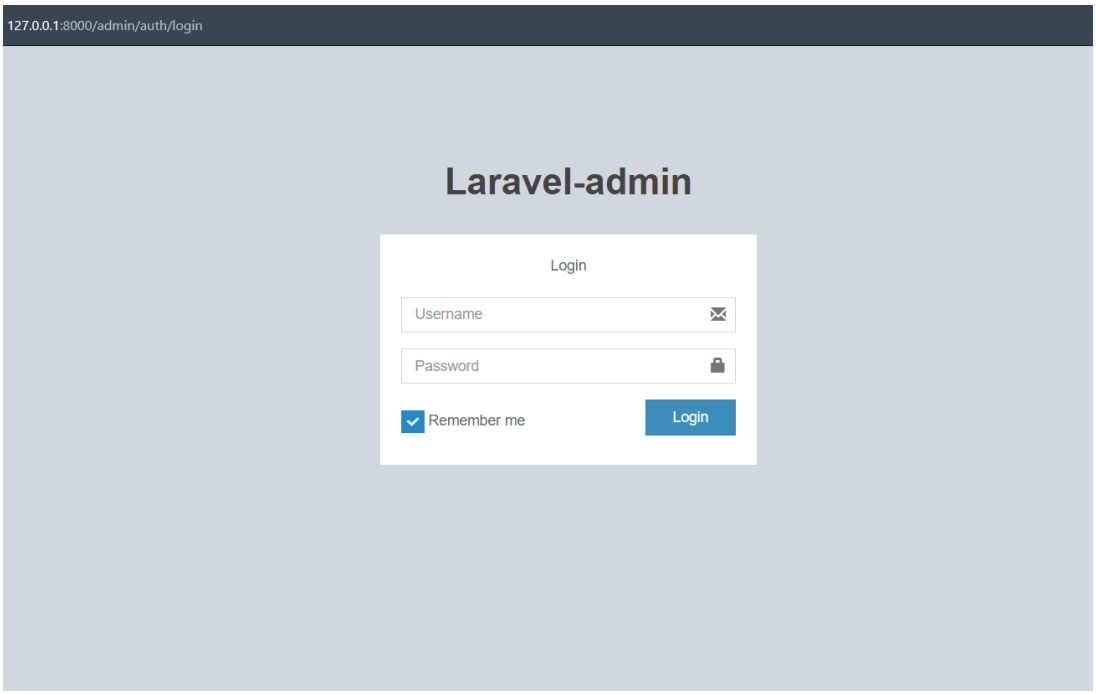
4.2 Σενάριο χρήσης ιδιοκτήτη καταστήματος

Ο ιδιοκτήτης του φούρνου θα πηγαίνει στην ιστοσελίδα που δημιουργήσαμε με την βοήθεια του Admin Panel με URL :

`http://127.0.0.1:8000/admin/auth/login`

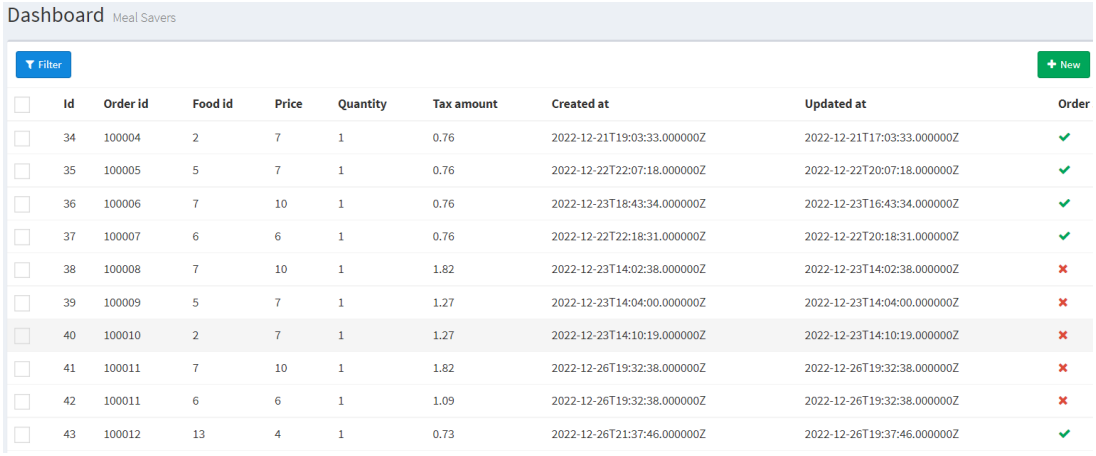
Φυσικά το base Url που είναι το ’ http://127.0.0.1:8000’ θα αντικατασταθεί με το domain name της ιστοσελίδας όταν ανεβάσουμε την εφαρμογή σε έναν Web server.

Το Σχήμα 4.13 απεικονίζει το Log in Page του Admin Panel στο οποίο θα κάνει σύνδεση ο ιδιοκτήτης του φούρνου με τον λογαριασμό που του έχει δημιουργήσει ο Admin.



Σχήμα 4.13: Backend login Page

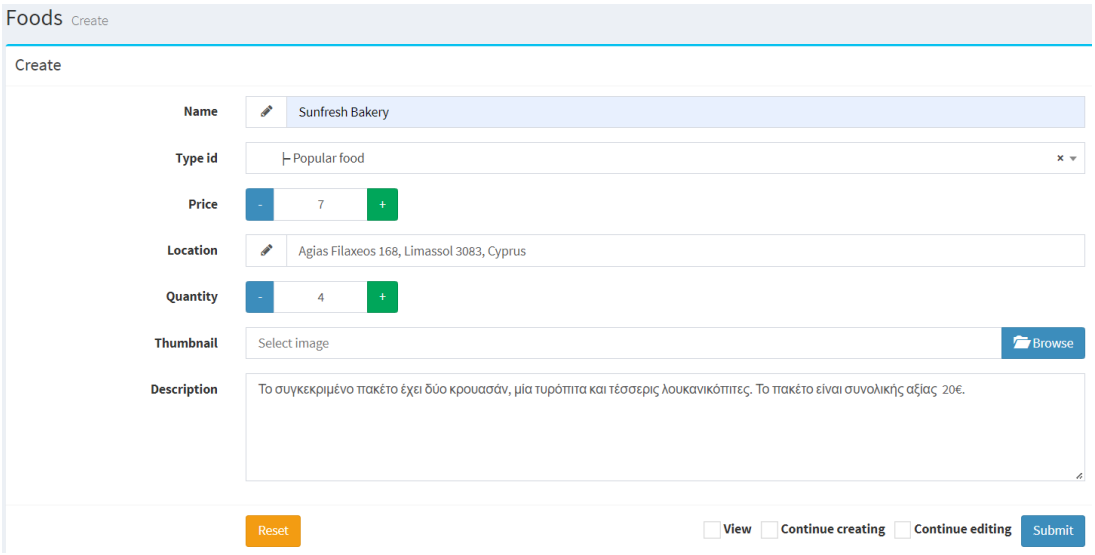
Ο Admin θα στέλνει στους ιδιοκτήτες των επιχειρήσεων, τους κωδικούς και τα username των λογαριασμών που θα δημιουργεί για τις επιχειρήσεις οι οποίες θα συνεργάζονται με την εφαρμογή. Οι συγκεκριμένοι λογαριασμοί δεν θα έχουν πρόσβαση σε όλο το admin Panel αλλά μόνο στους τομείς οι οποίοι είναι απαραίτητη.



Id	Order id	Food id	Price	Quantity	Tax amount	Created at	Updated at	Order
34	100004	2	7	1	0.76	2022-12-21T19:03:33.000000Z	2022-12-21T17:03:33.000000Z	✔
35	100005	5	7	1	0.76	2022-12-22T22:07:18.000000Z	2022-12-22T20:07:18.000000Z	✔
36	100006	7	10	1	0.76	2022-12-23T18:43:34.000000Z	2022-12-23T16:43:34.000000Z	✔
37	100007	6	6	1	0.76	2022-12-22T22:18:31.000000Z	2022-12-22T20:18:31.000000Z	✔
38	100008	7	10	1	1.82	2022-12-23T14:02:38.000000Z	2022-12-23T14:02:38.000000Z	✘
39	100009	5	7	1	1.27	2022-12-23T14:04:00.000000Z	2022-12-23T14:04:00.000000Z	✘
40	100010	2	7	1	1.27	2022-12-23T14:10:19.000000Z	2022-12-23T14:10:19.000000Z	✘
41	100011	7	10	1	1.82	2022-12-26T19:32:38.000000Z	2022-12-26T19:32:38.000000Z	✘
42	100011	6	6	1	1.09	2022-12-26T19:32:38.000000Z	2022-12-26T19:32:38.000000Z	✘
43	100012	13	4	1	0.73	2022-12-26T21:37:46.000000Z	2022-12-26T19:37:46.000000Z	✔

Σχήμα 4.14: Dashboard Page

Όταν ο ιδιοκτήτης της επιχειρήσεις πάρει τον κωδικό και το username από τον Admin. Μπορεί να κάνει Login. Όταν κάνει Login θα πλοηγηθεί στο Dashboard Page το οποίο παρουσιάζεται στο Σχήμα 4.14. Το dashboard αναφέρει πολλές πληροφορίες όπως : Το id της παραγγελίας , την τιμή, την ποσότητα, των φόρο ο οποίος υπολογίζεται ανάλογα με την τιμή της παραγγελίας. Τέλος το Dashboard αναφέρει την κατάσταση της παραγγελίας, δηλαδή αν έχει πληρωθεί ή όχι. Αν ο πελάτης έρθει παραλάβει και πληρώσει την παραγγελία τότε ο ιδιοκτήτης μπορεί να αλλάξει την κατάσταση της παραγγελίας από απλήρωτη σε πληρωμένη. Αν ο ιδιοκτήτης θέλει να ανεβάσει εναπομείναντα προϊόντα στο mobile Application, θα πρέπει από το Dashboard Page να πλοηγηθεί στο Food Page Σχήμα 4.15, από το οποίο θα μπορεί να ανεβάσει τα προϊόντα.



Σχήμα 4.15: Backend Food Page

Από το Food Page ο ιδιοκτήτης της επιχειρήσεις μπορεί να δημιουργήσει και να διαγράψει τα προϊόντα που θέλει στην εφαρμογή, για να δημιουργήσει τα προϊόντα, πρέπει να συμπληρώσει τα πεδία της φόρμας στο Food Page. Αυτά τα πεδία είναι : Το όνομα της επιχείρησης, την τιμή του πακέτου, την τοποθεσία της επιχείρησης, την ποσότητα των πακέτων που διαθέτει, και ένα description το οποίο θα αναφέρει τα προϊόντα που έχει μέσα το πακέτο. Στο τέλος ο ιδιοκτήτης θα πατάει το κουμπί Submit και θα στέλνονται τα δεδομένα στο mobile Application.

Με αυτά φτάσαμε στο τέλος του τέταρτου κεφαλαίου. Στο πέμπτο και τελευταίο κεφάλαιο θα αναφέρουμε τα συμπεράσματα της διπλωματικής εργασίας, αλλά και μελλοντικές επεκτάσεις της εφαρμογής.

Κεφάλαιο 5ο: Συμπεράσματα και Μελλοντικές επεκτάσεις

Στη συγκεκριμένη εργασία, έχουμε αναφέρει τα περιβαλλοντικά και οικονομικά προβλήματα της σπατάλης τροφίμων, καθώς και κάποιες εφαρμογές στο εξωτερικό οι οποίες βοηθούν στην καταπολέμηση αυτών των προβλημάτων. Στην συνέχεια έχουμε αναφέρει τις τεχνολογίες που χρησιμοποιήθηκαν για την δημιουργία της εφαρμογής όπως το Flutter Framework το οποίο έχει χρησιμοποιηθεί για την δημιουργία της mobile Εφαρμογής, και το laravel framework το οποίο μας βοήθησε στην δημιουργία του Backend και της Βάσης Δεδομένων. Επίσης στην συγκεκριμένη εργασία εστιάσαμε κυρίως στην υλοποίηση της εφαρμογής, η οποία δεν θα ήταν δυνατή χωρίς την χρήση του Getx Package το οποίο ήταν απαραίτητο για την υλοποίηση του Routing System , state management και στην επικοινωνία του Frontend με τον Server. Σε αυτό το σημείο μπορώ να πω με σιγουριά ότι το Getx Package ήταν το πιο χρήσιμο πακέτο σε όλη την διαδικασία δημιουργίας της εφαρμογής. Έπειτα στα πλαίσια της διπλωματικής εργασίας αναφέραμε τις αρχιτεκτονικές που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής, όπως μια παραλλαγή της αρχιτεκτονικής MVC η οποία χρησιμοποιήθηκε για την σχεδίαση και υλοποίηση του Backend, και την Controller-Repository-API αρχιτεκτονική που βοήθησε στην υλοποίηση του mobile Application. Παράλληλα δώσαμε έμφαση στην εξήγηση των συγκεκριμένων αρχιτεκτονικών όχι μόνο με διαγράμματα αλλά και με πραγματικά παραδείγματα κώδικα της εφαρμογής. Επίσης στο τέλος του δεύτερου και αρχές τρίτου κεφαλαίου έγινε εκτενής ανάλυση του κώδικα πολλών κλάσεων για να επιτευχθεί οικειότητα με την γλώσσα προγραμματισμού Dart, με συνέπεια την καλύτερη κατανόηση της υλοποίησης της εφαρμογής. Επιπρόσθετα είδαμε το σενάριο χρήσης της εφαρμογής από την προοπτική του πελάτη, και το σενάριο χρήσης της εφαρμογής από την προοπτική του ιδιοκτήτη της επιχείρησης. Παράλληλα έχουμε αναφέρει αναλυτικά όλα τα απαραίτητα βήματα που πρέπει να κάνει ο πελάτης και ο ιδιοκτήτης για να επιτευχθεί η λειτουργία του όλου συστήματος.

Όπως έχουμε αναφέρει στο τέταρτο κεφάλαιο, για να αυξήσουμε την πιθανότητα επιτυχίας του Meal Savers στην αγορά πρέπει να υλοποιήσουμε κάποιες μελλοντικές επεκτάσεις οι οποίες θα κάνουν πιο προσιτή την εφαρμογή στους χρήστες. Η πιο πιθανή επέκταση που μπορεί να γίνει στο μέλλον, είναι η δημιουργία ενός ψηφιακού τρόπου πληρωμής μέσω του mobile Application για τους χρήστες. Επίσης μία άλλη μελλοντική επέκταση που αξίζει αναφορά είναι η δημιουργία ενός Delivery συστήματος το οποίο θα έχει google maps integration. Τελειώνοντας θα ήθελα να αναφέρω παρά τις φιλοδοξίες για επιτυχία του Meal Savers. Ο πρωταρχικός λόγος κατασκευής της συγκεκριμένης εφαρμογής είναι για να βοηθήσουμε τους ανθρώπους της Κύπρου που δεν έχουν τις καλύτερες οικονομικές δυνατότητες, να μπορέσουν να αγοράσουν καλό φαγητό σε καλύτερες τιμές.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] W. contributors

[2] F. T. organization 2013.

[3] S. Lee, “How to run code in vs code,” 2021-08-17.

[4] S. Y. Ameen and D. Y. Mohammed, *Developing CrossPlatform Library Using Flutter*, vol. 7. Mar. 2022.

[5] M. U. of Applied Sciences 2019, *Hoang Ly State Management Analyses of the Flutter Application BSc. Thesis. Metropolia University of Applied Sciences*. 2019.

[6] F. C. flutter launcher icons Retrieved [Internet] Available https://pub.dev/packages/flutter launcher icons.

[7] M. F. ige, “https://birdeatsbug.com/blog/getx state management in flutter,” 06 2022.

[8] D. D. packages, plugins. [Internet]. Available https://docs.flutter.dev/development/packages, and plugins/developing packages.

[9] wikipedia, “https://en.wikipedia.org/wiki/dart(programming language).”

[10] Dart.dev, “https://dart.dev/overview.”

[11] Wikipedia, *Syntactic Sugar, https://en.wikipedia.org/wiki/Syntactic sugar*.

[12] Wikipedia, “Laravrl,https://en.wikipedia.org/wiki/laravel,”

[13] M. Surguy, “History of laravel php framework, eloquence emerging,” 07 2013.

[14] A. Admin-Panel, “https://laravel-admin.org/docs/en/quick-start.”

[15] R. Martin, *API Design for C++. Elsevier Science. p. 1. ISBN 9780123850041.* 2011.

[16] L. Kin, “Intro to apis: History of apis,” 10 2019.

[17] W. Laura, “Global cloud microservices market (2021 to 2026),” 08 2021.

[18] E. D. LLC, “Innovation acceleration headquarters scottsdale, az 85260,”

[19] EDUCBA, “https://www.educba.com/what is xampp/,” 2022.

[20] Wikipedia, “https://en.wikipedia.org/wiki/phpmyadmin.”

[21] official website, “https://www.postman.com.”

[22] Wikipedia, “https://en.wikipedia.org/wiki/postman(software).”