Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**ΕΦΑΡΜΟΓΗ ΓΙΑ ΤΗΝ ΕΝΙΣΧΥΣΗ ΤΟΠΙΚΗΣ ΟΙΚΟΝΟΜΙΑΣ ΜΕΣΩ ΗΛΕΚΤΡΟΝΙΚΟΥ ΕΜΠΟΡΙΟΥ** **COMMERCE APP FOR ENHANCING LOCAL ECONOMY**
Ονοματεπώνυμο Φοιτητή	**Ευστάθιος Τσάρας**
Πατρώνυμο	**Λεωνίδας**
Αριθμός Μητρώου	**ΜΠΣΠ15089**
Επιβλέπων	**Ευθύμιος Αλέπης, Αναπληρωτής Καθηγητής**

Ημερομηνία Παράδοσης **Δεκέμβριος 2021**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Ευθύμιος Αλέπης Αναπληρωτής Καθηγητής	Μαρία Βίρβου Καθηγήτρια	Κωνσταντίνος Πατσάκης Αναπληρωτής Καθηγητής

Περιεχόμενα

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στα πλαίσια του μεταπτυχιακού προγράμματος «Προηγμένα Συστήματα Πληροφορικής» και αφορά την υλοποίηση μιας ολοκληρωμένης λύσης ηλεκτρονικού εμπορίου. Σκοπός είναι η ενίσχυση τοπικών οικονομιών ανά πόλη – περιοχή μέσω ενός συστήματος ανταμοιβής και εξαργύρωσης πόντων για αγορές.

Η χρόνια ενασχόλησή μου με το front-end web development, μου έχει δώσει αρκετή εμπειρία και γνώση πάνω στο κομμάτι του ηλεκτρονικού εμπορίου, σπανίως όμως είχα την ευκαιρία να εμβαθύνω τεχνικά είτε στο κομμάτι του back-end ή σε mobile εφαρμογές. Αυτός είναι και ο λόγος που η εργασία αυτή εστιάζει περισσότερο στο τεχνικό μέρος από ότι στο επιχειρησιακό - οικονομικό μοντέλο.

Θα αναλυθούν τα δύο (2) εκ των τριών (3) μερών που απαρτίζουν συνολικά το σύστημα, αυτά είναι η εφαρμογή κινητού για τους πελάτες των καταστημάτων και το service api για την διαχείριση και αποθήκευση στην βάση δεδομένων. Το τρίτο μέρος που αναπτύσσεται στα πλαίσια συνεργαζόμενης διπλωματικής εργασίας και για το οποίο παρακάτω θα υπάρχουν απλές αναφορές, περιλαμβάνει μία ιστοσελίδα – back-office διαχείρισης των προσφορών, αγορών καθώς και προβολής χρήσιμων analytics για τους ιδιοκτήτες των καταστημάτων και τους εμπορικούς συλλόγους.

Σε αυτό το σημείο, θα ήθελα να ευχαριστήσω τον φίλο και συμφοιτητή Σωτήρη Μπούτρη για την σύλληψη της ιδέας, τις προτάσεις, την πολύ καλή συνεργασία που είχαμε αλλά και την βοήθειά του με την υλοποίηση του συμπληρωματικού μέρους που αναφέρθηκε προηγουμένως. Θα ήθελα επίσης να ευχαριστήσω τον καλό μου φίλο Αριστοφάνη Ζήση για την πολύτιμη βοήθειά του στην ανάπτυξη της mobile εφαρμογής και τις γνώσεις που μου μετέδωσε πάνω σε ένα αντικείμενο το οποίο θεωρούσα πάντα πολύ ενδιαφέρον.

Εισαγωγή

Ανάμεσα στους κλάδους που έχουν ευνοηθεί αρκετά από την τεχνολογία και συγκεκριμένα από την χρήση του διαδικτύου και των φορητών συσκευών, είναι αυτός της αγοράς. Πρώτη και μεγαλύτερη επανάσταση που ένωσε το διαδίκτυο με το εμπόριο αποτελεί το ηλεκτρονικό κατάστημα (e-shop), με τις εφαρμογές κινητών τηλεφώνων να ακολουθούν αργότερα. Έως τώρα είναι βέβαιο πως μία εφαρμογή κινητού από μόνη της δεν μπορεί να ανταγωνιστεί σε ταχύτητα και κόστος κατασκευής ένα e-shop, αλλά μπορεί να προσθέσει κάποια σημαντική λειτουργία που θα το κάνει να ξεχωρίζει από τα υπόλοιπα. Παραδείγματα χρήσης που βρίσκουμε συχνά είναι ειδικές εκπτώσεις μέσω εφαρμογής, τρισδιάστατη αναπαράσταση προϊόντων, ασφαλέστερη πληρωμή κτλ.

Σε μία εποχή λοιπόν που η τεχνολογία κυριαρχεί στις ζωές μας και συχνά αποτελεί μονόδρομο για την ανάπτυξη των επιχειρήσεων, οι περισσότεροι μπορούν να εντοπίσουν εύκολα τις θετικές πτυχές όπως την μη απαραίτητη φυσική παρουσία στο κατάστημα, τις ανταγωνιστικές τιμές και την κατ' οίκων αποστολή προϊόντων σε οποιοδήποτε μέρος.

Υπάρχουν όμως και αρνητικές πτυχές της κατάστασης αυτής, που πιθανώς να μην είναι τόσο ορατές, όπως η αδυναμία φυσικών καταστημάτων να σταθούν οικονομικά απέναντι σε ηλεκτρονικά καταστήματα που δουλεύουν π.χ. μόνο με αποθήκες. Η έλλειψη μιας ηλεκτρονικής επιλογής, για όσες επιχειρήσεις δεν μπορούν να την διαθέσουν, τις καθιστά εκτός συναγωνισμού καθώς η τάση για χρήση των υπηρεσιών αυτών όλο και αυξάνεται.

Αυτή η τάση όμως δημιουργεί και κοινωνικά προβλήματα όπως αυτό της αποξένωσης, ιδιαίτερα εντός μιας τοπικής κοινότητας/πόλης. Είναι δεδομένο πως κάποιος θα εμπιστευτεί περισσότερο ένα κατάστημα που γνωρίζει και τον γνωρίζουν και θέλουν να τον φροντίσουν όσο το δυνατό καλύτερα παρά να μπει σε μία διαδικασία αποστολών-επιστροφών, καθυστερήσεων κτλ. Οι σχέσεις που υπήρχαν κάποτε μέσα σε μία τοπική κοινότητα, η αναγνώριση, η εκτίμηση, η υποστήριξη, έχουν αρχίσει να μειώνονται δραματικά.

Η εφαρμογή που δημιουργήθηκε στα πλαίσια της παρούσας εργασίας, προσπαθεί να χρησιμοποιήσει το ηλεκτρονικό εμπόριο με τέτοιο τρόπο, ώστε να φέρει πιο κοντά τον κόσμο με τα καταστήματα της περιοχής του και να επεκτείνει τις αγοραστικές του δυνατότητες. Κάθε καταναλωτής επιβραβεύεται για τις αγορές του, με ένα σύστημα πόντων και εξαργύρωσή τους σε γενικές ή εξατομικευμένες προσφορές που παρέχονται από τα καταστήματα της περιοχής/πόλης.

Οι πελάτες (clients) όταν πραγματοποιήσουν αγορές από τα τοπικά καταστήματα επιβραβεύονται με πόντους για εξαργύρωση σε οποιοδήποτε κατάστημα της περιοχής. Η εξαργύρωση θα γίνεται μόνο σε προσφορές (offers) των καταστημάτων αυτών και θα είναι διαθέσιμες στην εφαρμογή των πελατών. Για να εξαργυρώσει ο πελάτης την προσφορά θα πρέπει να δηλώσει την πρόθεση για αγορά (claim) μέσω της οποίας θα παραχθεί ένας μοναδικός κωδικός (voucher). Οι πόντοι αυξάνονται με κάθε αγορά σε κατάστημα επί του τελικού κόστους, είτε με χρήση προσφοράς ή μη.

Οι ιδιοκτήτες καταστημάτων (shops) μέσω του διαχειριστικού που αναφέρθηκε πιο πάνω, θα μπορούν να καταχωρήσουν τις προσφορές τους για να προσελκύσουν πελάτες. Δεν πρόκειται για υπηρεσία ηλεκτρονικών παραγγελιών - αγορών και αποστολής τους, καθώς απαιτείται η φυσική παρουσία των πελατών στο κατάστημα για την ολοκλήρωση με QR Scan της συναλλαγής (transaction). Εφόσον πραγματοποιείται εξαργύρωση προσφοράς, πρέπει να ενημερωθεί το κατάστημα για τον μοναδικό κωδικό που αναφέρθηκε πιο πάνω και να γίνει η αντίστοιχη έκπτωση επί του κόστους.

Abstract

One of the sectors that have been mostly favored by the technology and more specifically by the usage of the internet and mobile apps, is that of commerce. The first and biggest revolution connecting the internet with commerce was the construction of e-shops, followed by mobile apps later on. It is certain till now that a mobile app cannot match the low cost and speed of construction/development that an e-shop would require. Nonetheless, it can add some extra functionality that will make it stand out from competition. There are some examples on this matter, such as special offers and vouchers, 3D product representation and safer payment methods.

Thus, in a time that technology rules in our lives and most often is mandatory for business growth, one can easily detect all the positive aspects like turning in-store physical presence into optional, making competitive prices and producing high availability in any place possible.

But there is a downside to this situation, like having physical stores struggling to stand against e-shops that may use only warehouses for their products and nothing more. Missing an e-commerce platform can make a business go out of competition in an ever-increasing trend for usage of such options.

This trend also leads to various social problems, such as social distancing, and is something to consider, especially in cases of small local societies and towns. It is a fact that anybody would prefer to trust a shop that knows first-hand and at the same time they know who their customer is and how to offer their services in the best way possible. The relations that used to exist in local societies and towns have already started to fade out.

The application that was developed within the context of this assignment, tries to use e-commerce in a way that will bring people together in local shops but also extend the shopping options and opportunities for everyone. Each customer is rewarded with points per purchase and can use them for claiming public or private offers.

When customers (or clients as in application) complete a purchase within a local shop, the rewarded points can be later used in any other shop within the city, town etc. These points are to be used only for offers which will be available for claiming in the mobile app. In order for a client to claim an offer, there will be a corresponding action inside the app. The points are calculated and updated over the final total cost of a purchase no matter if an offer was used or not.

Shop owners (or shops as in application) can use a back-office platform as mentioned earlier, in order to create their offers and attract customers. This is not an e-commerce service for ordering and sending the products, the client's physical presence is required inside the shop so that a QR scan is done for the transaction to be final and complete. If an offer is to be used, then the client should mention the voucher code that was received within the app, when claiming the offer.

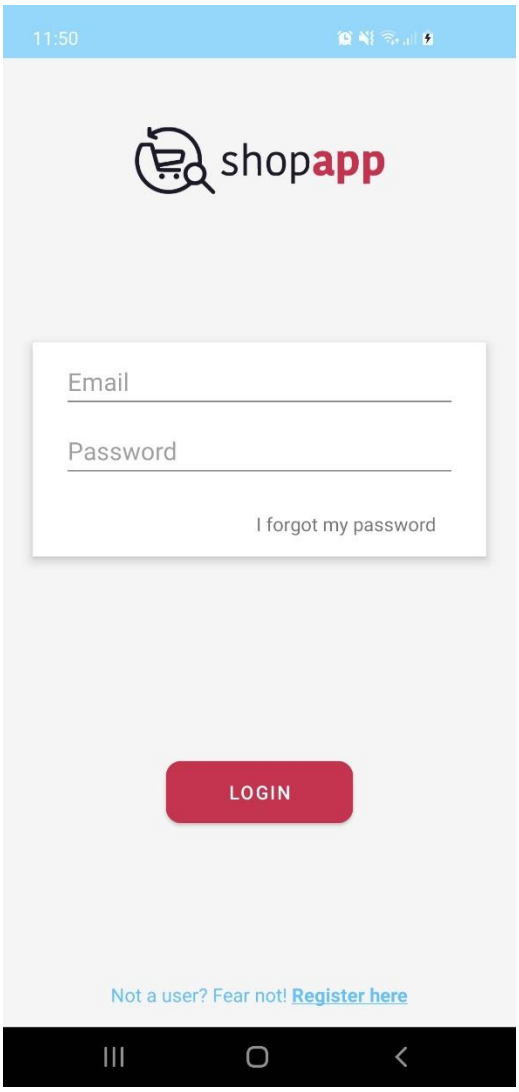
1. Παρουσίαση Εφαρμογής

1.1 Εισαγωγικές Οθόνες

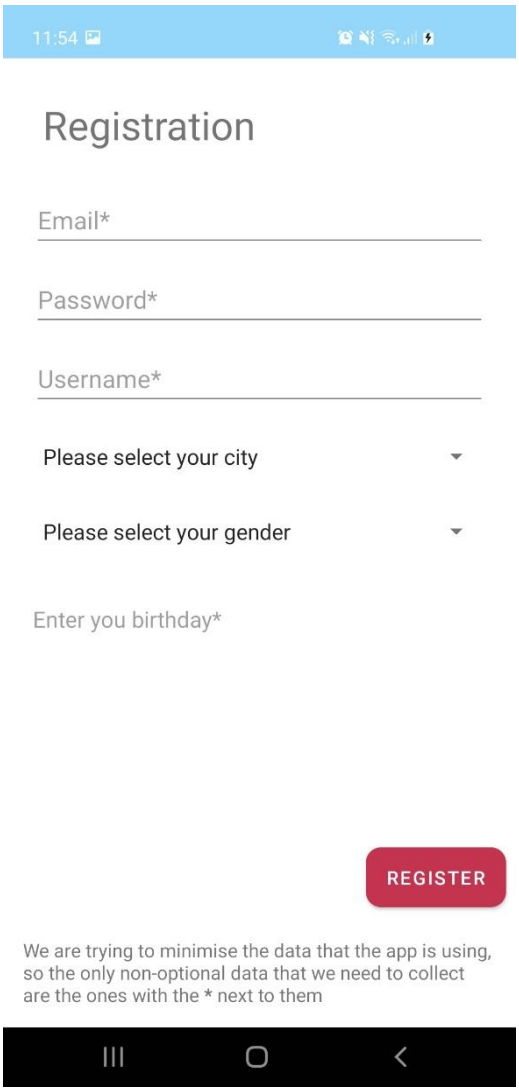
1.1.1 Σύνδεση – Εγγραφή Χρήστη

Όταν ένα χρήστης ανοίξει πρώτη φορά την εφαρμογή (ή έχει αποσυνδεθεί), θα δει την οθόνη στα αριστερά που τον προτρέπει να συνδεθεί δίνοντας το e-mail που έχει δηλώσει κατά την εγγραφή και τον κωδικό του. Από αυτή την οθόνη υπάρχουν ακόμη δύο επιλογές διαθέσιμες.

Η μία είναι η οθόνη δεξιά που τον παραπέμπει σε εγγραφή εφόσον δεν το έχει κάνει. Η άλλη επιλογή είναι να ανοίξει μία οθόνη στην οποία συμπληρώνει το e-mail του και να αιτηθεί να του σταλεί ένα link για ανανέωση του password, εφόσον το έχει ξεχάσει και αντιμετωπίζει αδυναμία σύνδεσης.



Εικόνα 1 - Είσοδος στην Εφαρμογή

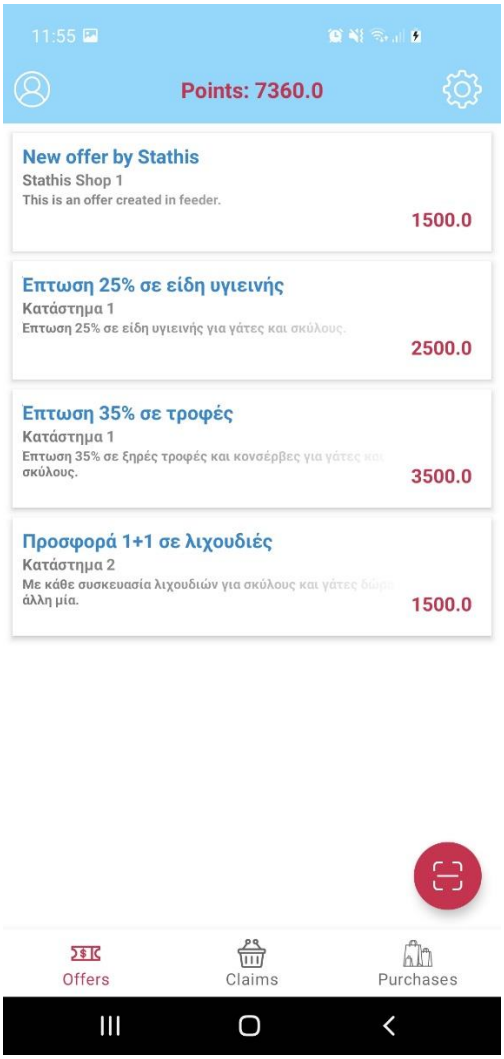


Εικόνα 2 - Εγγραφή νέου χρήστη

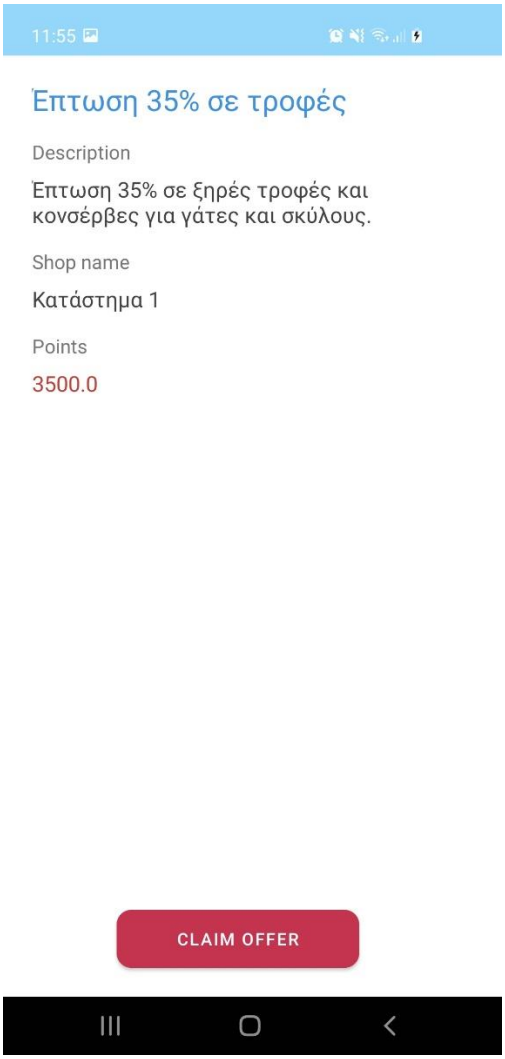
1.2 Κύριες Οθόνες Λειτουργίας

1.2.1 Λίστα Προσφορών – Άνοιγμα Προσφοράς

Η οθόνη της λίστα προσφορών είναι και η αρχική οθόνη κατά την είσοδο του χρήστη στην εφαρμογή. Στην πάνω μπάρα φαίνονται οι διαθέσιμοι πόντοι του χρήστη και τα 2 εικονίδια αριστερά και δεξιά ανοίγουν οθόνη προφίλ και ρυθμίσεων αντίστοιχα. Κατά την επιλογή μια προσφοράς, ανοίγει η οθόνη δεξιά με τις πληροφορίες της και μία επιλογή κάτω κάτω εφόσον ο χρήστης θέλει να αποκτήσει voucher για την προσφορά αυτή.



Εικόνα 3 - Λίστα προσφορών



Εικόνα 4 - Προβολή προσφοράς

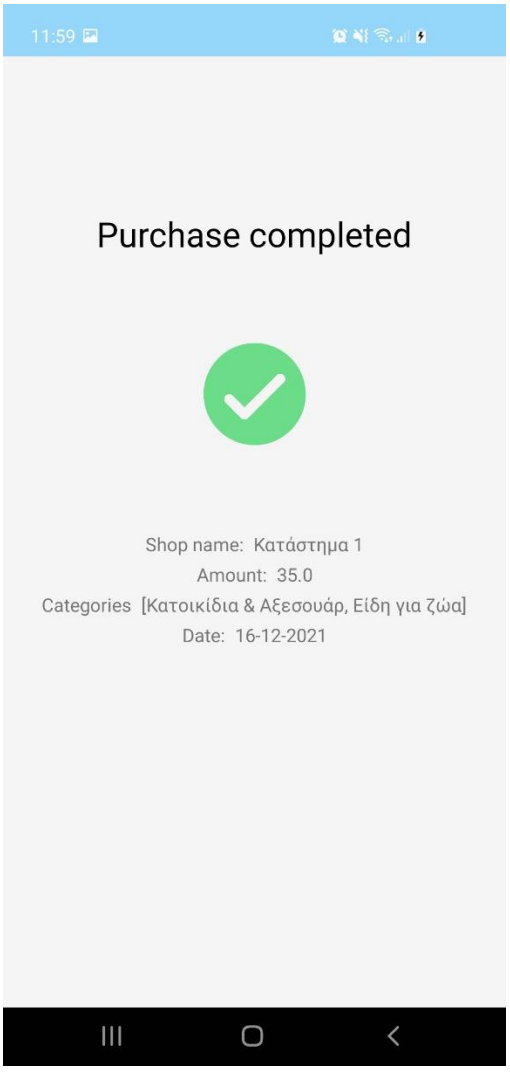
1.2.2 Λίστα Προσφορών με Εκδήλωση Ενδιαφέροντος

Εφόσον έχει εκδηλωθεί ενδιαφέρον για μια προσφορά με την αίτηση του voucher, υπάρχει η δυνατότητα να βρεθούν όλες οι προσφορές αυτές στην ενότητα “Claims”. Σε κάθε προσφορά θα φαίνεται κάτω από τον τίτλο της ο κωδικός που πρέπει να δοθεί στο κατάστημα για επιβεβαίωση.

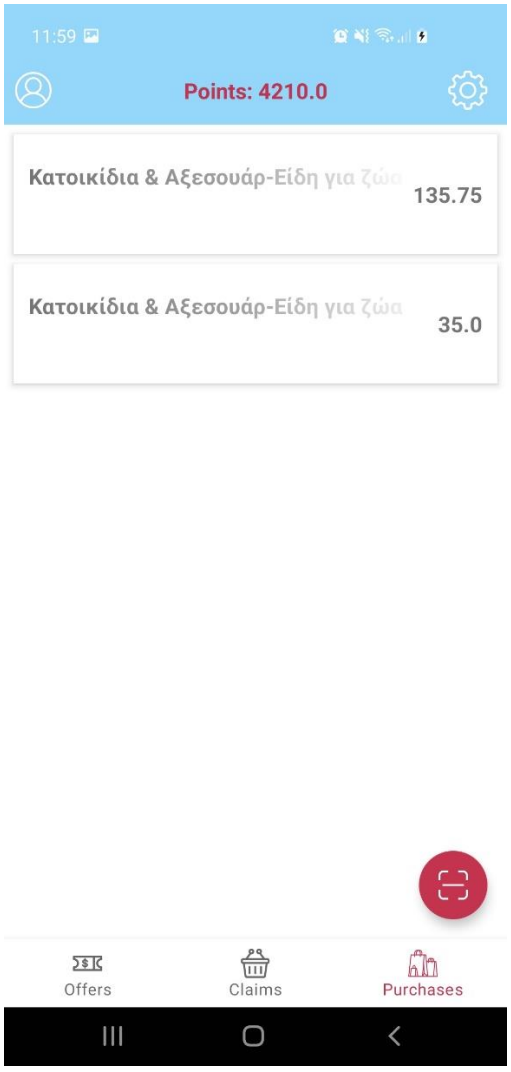
Εικόνα 5 - Λίστα προσφορών με λήψη κουπονιού

1.2.3 Ολοκλήρωση Αγοράς με QR scan και Λίστα Αγορών

Κατά την επιτυχή διαδικασία του QR scan θα εμφανιστεί η οθόνη αριστερά που θα αναφέρει τις πληροφορίες της συναλλαγής που ολοκληρώθηκε. Η λίστα με όλες τις συναλλαγές του χρήστη μπορεί να βρεθεί στην ενότητα “Purchases”.



Εικόνα 6 - Ολοκλήρωση αγοράς

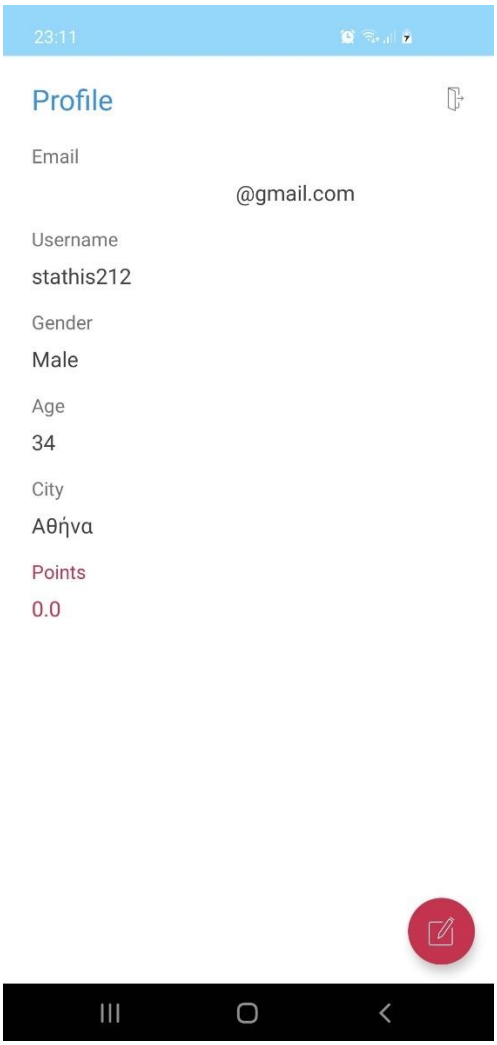


Εικόνα 7 - Προβολή λίστας αγορών

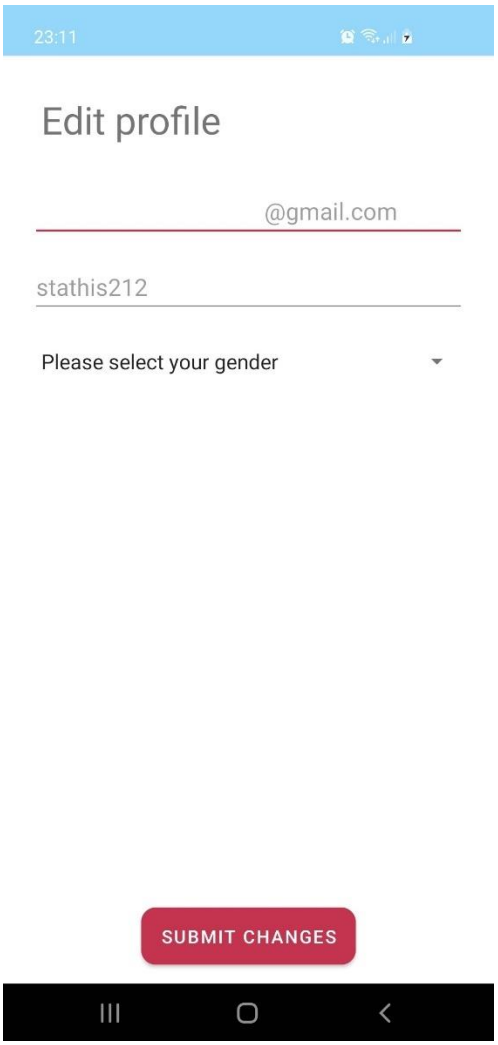
1.3 Λοιπές Οθόνες

1.3.1 Προφίλ και Επεξεργασία

Υπάρχει επίσης η δυνατότητα ένας χρήστης να δει τα στοιχεία του και να τα επεξεργαστεί.



Εικόνα 8 - Προβολή προφίλ χρήστη



Εικόνα 9 - Επεξεργασία προφίλ χρήστη

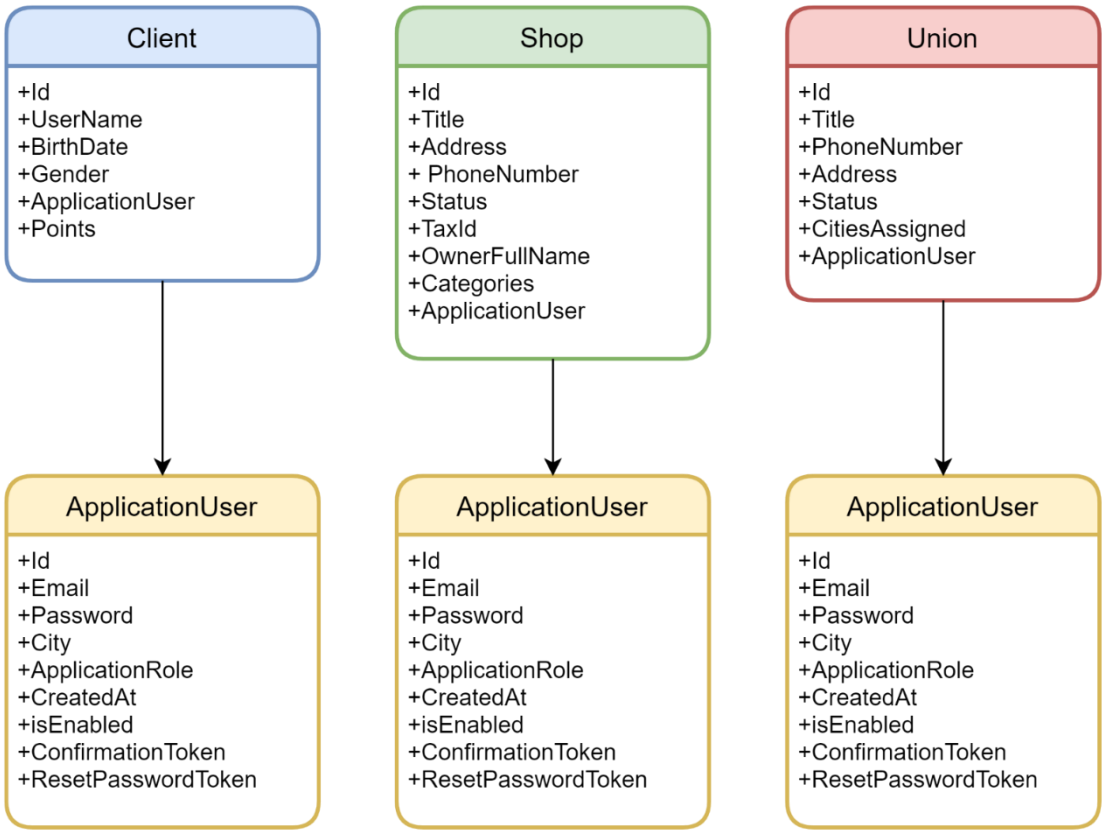
2. Ανάλυση Λειτουργίας

2.1 Ρόλοι Χρηστών

Client - Ο πελάτης του καταστήματος και χρήστης της mobile εφαρμογής
Shop - Ο ιδιοκτήτης του καταστήματος και χρήστης του διαχειριστικού σε ότι αφορά το κατάστημά του
Union - Εκπρόσωπος του εμπορικού συλλόγου και χρήστης του διαχειριστικού σε ότι αφορά τα καταστήματα που ανήκουν στον σύλλογο.
Admin - Διαχειριστής με πρόσβαση στα πάντα

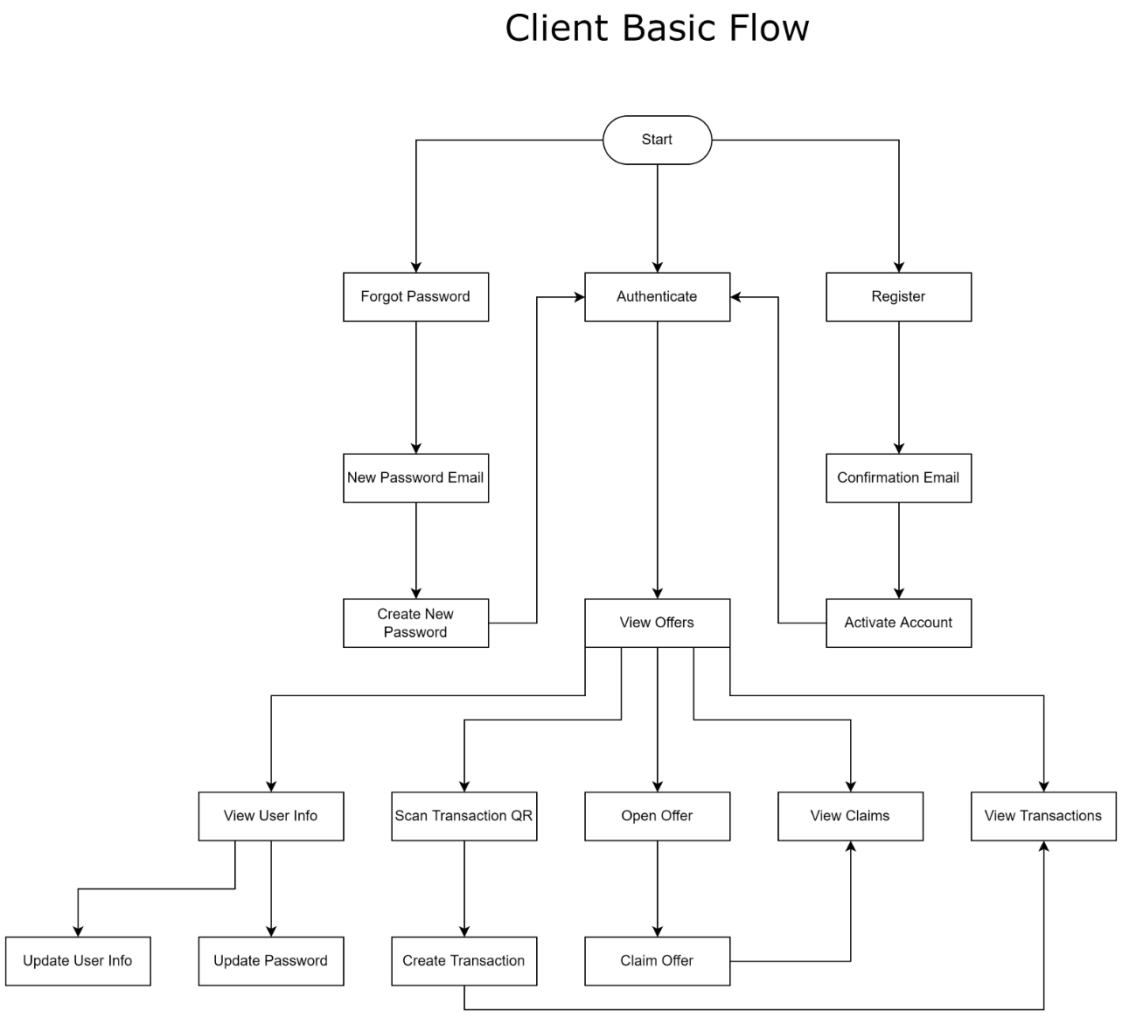
2.2 Δομή Χρηστών

Όπως φαίνεται και στην παρακάτω εικόνα, όλοι οι χρήστες της εφαρμογής αποτελούνται από δύο διαφορετικές οντότητες στην βάση που συνδέονται με σχέση 1 – 1. Κάθε Client, Shop και Union, έχουν ένα entity βάσει ρόλου με τα στοιχεία που αναλογούν στο κάθε ένα, αλλά παράλληλα έχουν και ένα ApplicationUser, δηλαδή ένα Entity το οποίο έχει κοινά στοιχεία για όλους και μπορεί να θεωρηθεί η βάση ενός χρήστη που ορίζει τα credentials, αν είναι επιβεβαιωμένος λογαριασμός καθώς και άλλες κοινές λειτουργίες.

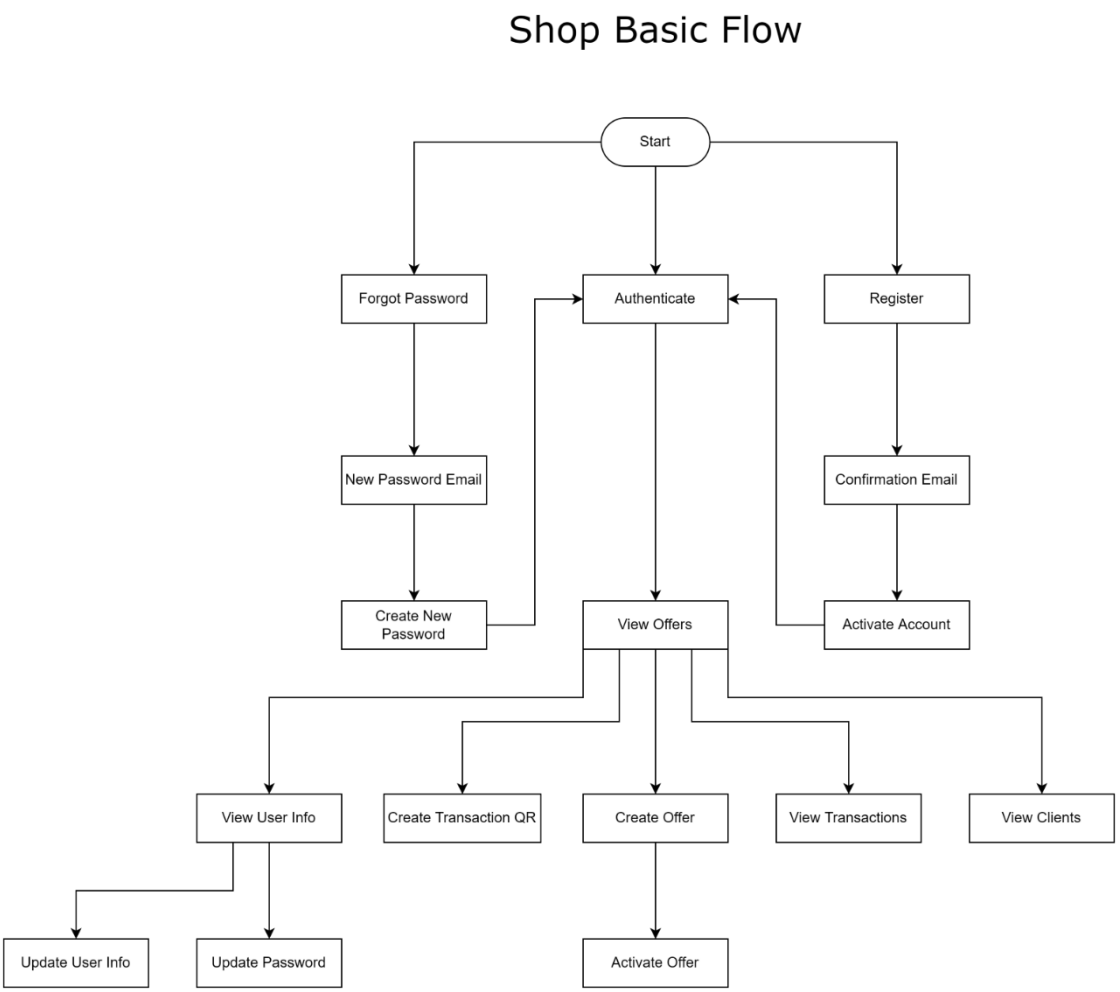


Εικόνα 10 - Στοιχεία χρηστών ανά ρόλο

2.3 Σενάρια και Ροή Χρήσης



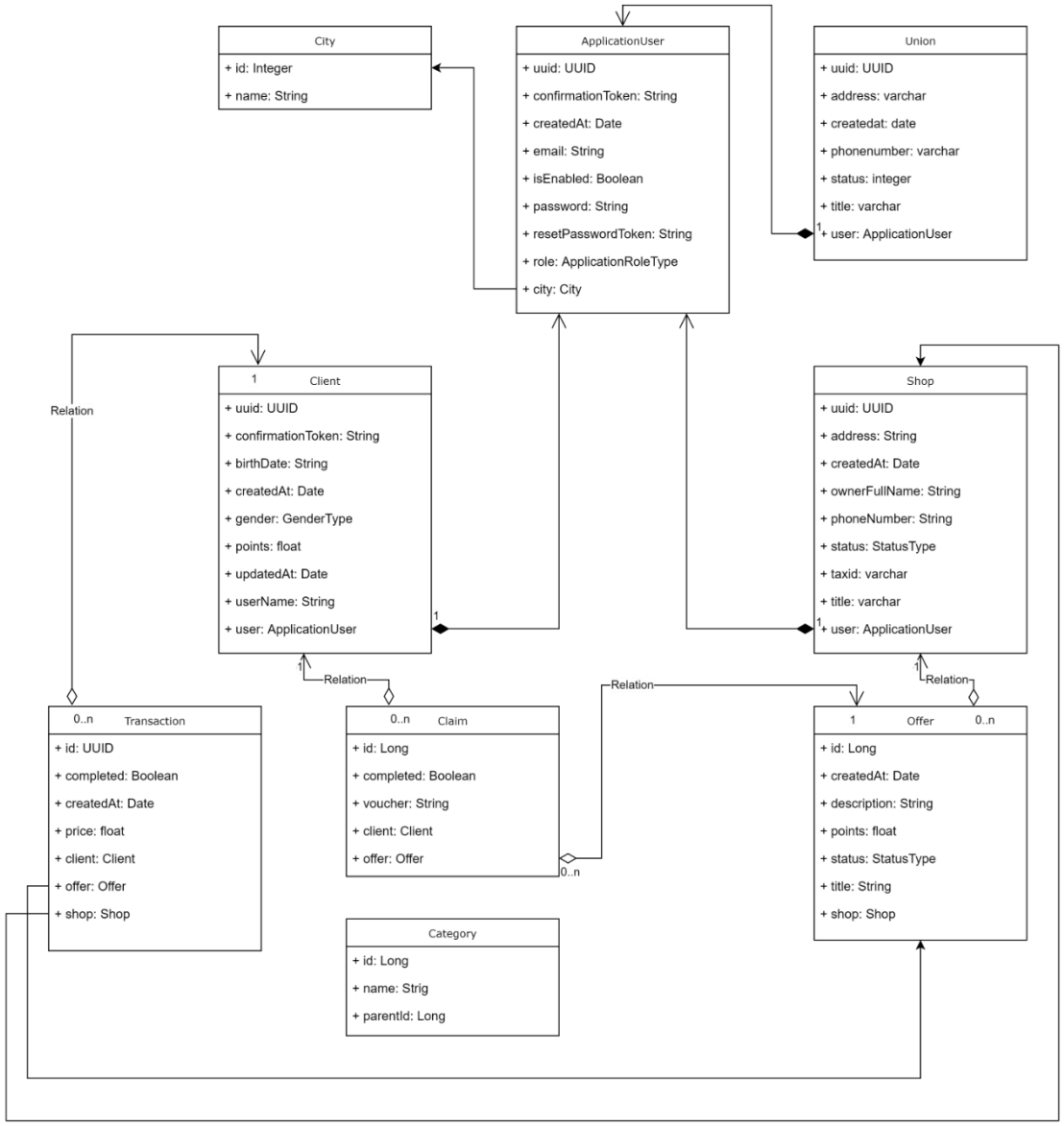
Εικόνα 11 - Ροή χρήσης για έναν πελάτη



Εικόνα 12 - Ροή χρήσης για ένα κατάστημα

3. Σχεδιασμός

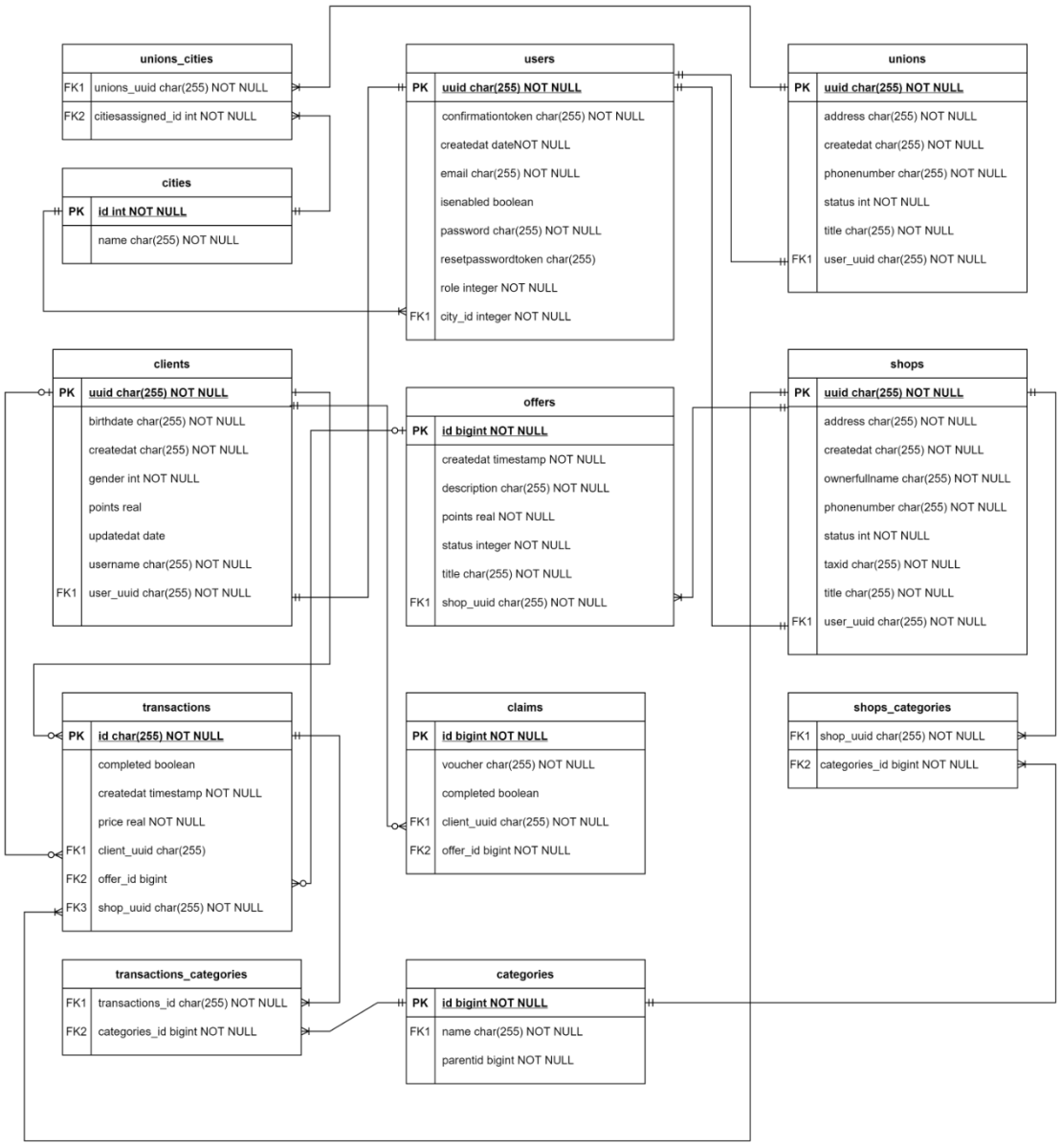
3.1 UML Διάγραμμα Κλάσεων



Εικόνα 13 - Διάγραμμα Κλάσεων UML

3.2 Σχεδιασμός Βάσης Δεδομένων

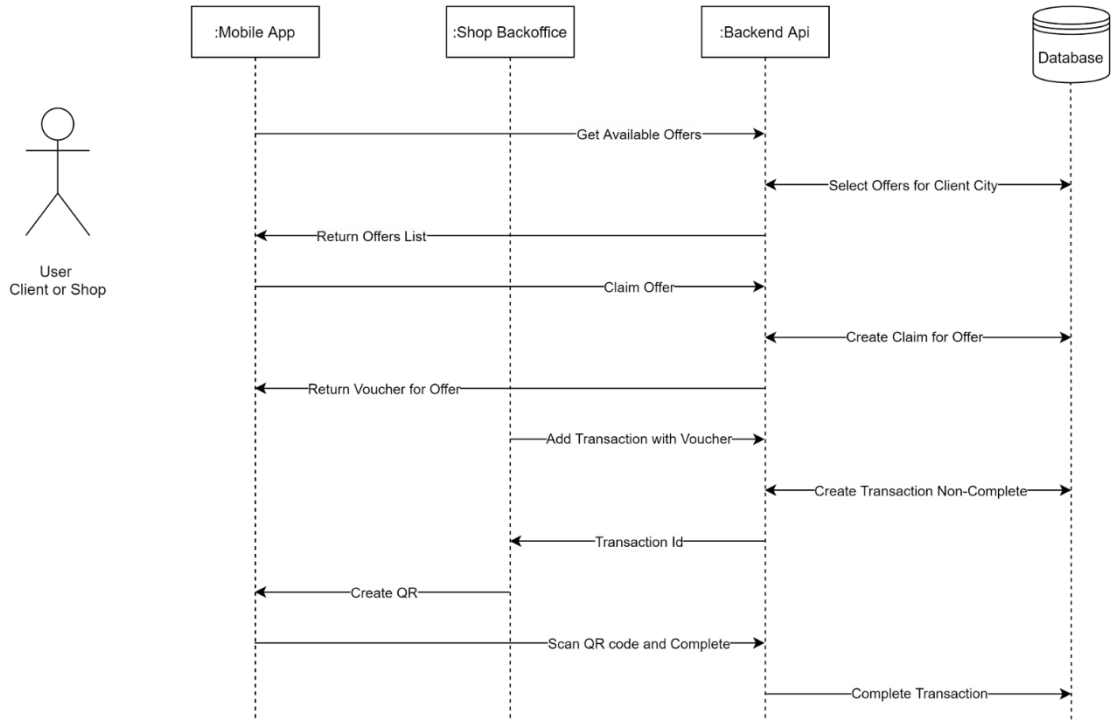
Να σημειωθεί πως οι πίνακες της βάσης δεδομένων δεν φτιάχνονται χειροκίνητα με την χρήση της PostgreSQL ή κάποιο σχετικό εργαλείο, αλλά χτίζονται αυτόματα με την βοήθεια του Hibernate ORM.



Εικόνα 14 - Σχεδιασμός βάσης δεδομένων

3.3 UML Διάγραμμα Ακολουθίας

Ο Client χρησιμοποιεί την εφαρμογή κινητού (mobile app) ενώ το Shop το διαχειριστικό (backoffice). Στην εικόνα παρακάτω φαίνεται η ροή με την οποία πρέπει να συμμετέχουν και οι δύο ώστε να πραγματοποιηθει μια συναλλαγή.



Εικόνα 15 - Διάγραμμα ακολουθίας UML

3.4 Σχεδιασμός REST Api

3.4.1 Βασικός Σχεδιασμός των Endpoints

Ως κοινή βάση ενός endpoint χρησιμοποιήθηκε το λεκτικό ‘api’. Έπειτα ακολουθεί το λεκτικό που αντιστοιχεί στο entity για το οποίο θα εκτελεστεί κάποια ενέργεια μετά την κλήση του. Τέλος ακολουθεί ένα λεκτικό που μπορεί να ορίζει μία συγκεκριμένη λειτουργία ή μία παράμετρος όπως το Id του entity πάνω στο οποίο θέλουμε να εκτελέσουμε κάποιες εντολές.

Έτσι καταλήγουμε να έχουμε τις εξής περιπτώσεις:

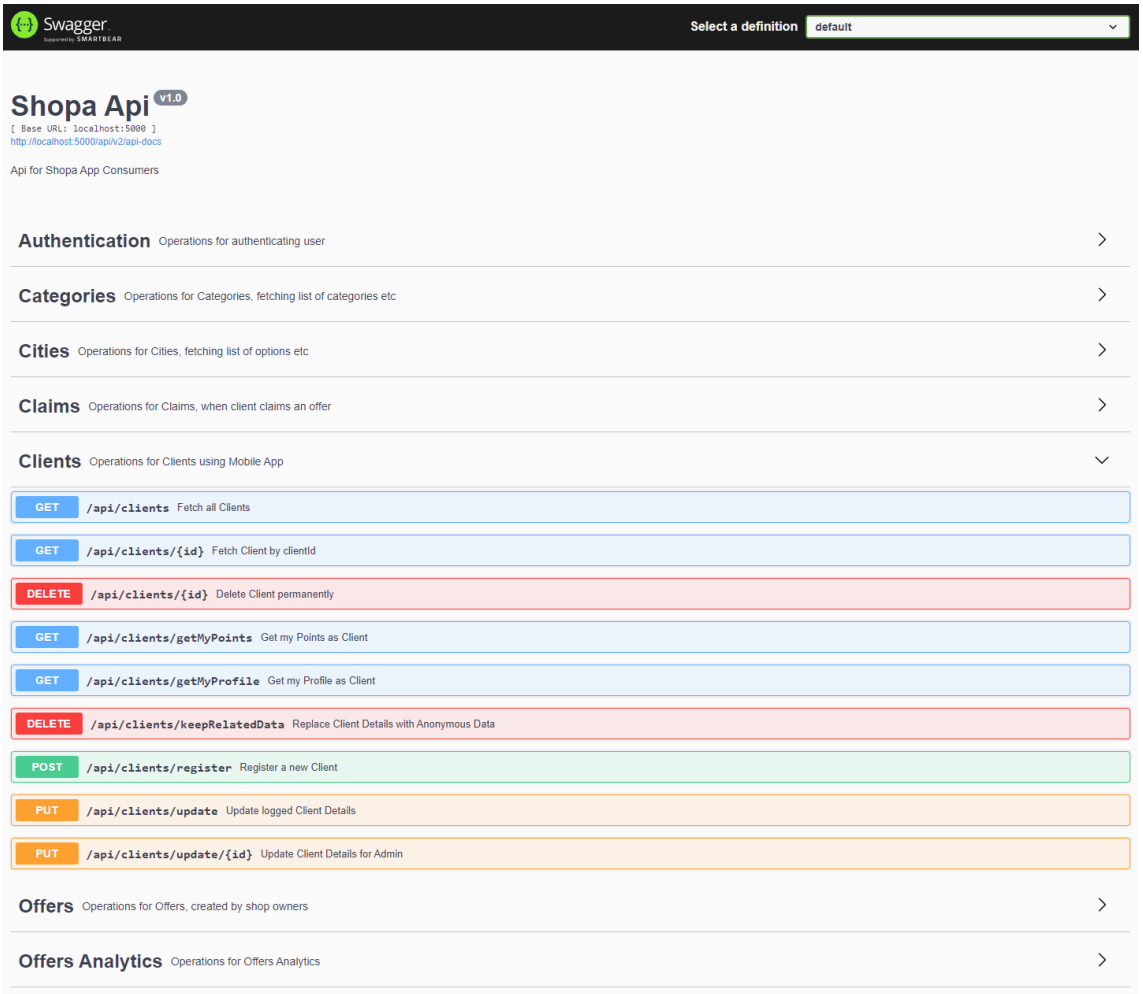
- {baseUrl}/api/{entityName}
- {baseUrl}/api/{entityName}/{id}
- {baseUrl}/api/{entityName}/{specificFunctionalityName}
- {baseUrl}/api/{entityName}/{specificFunctionalityName}/{id}

Αυτά μπορούν με πραγματικά παραδείγματα να είναι:

- http://localhost:5000/api/shops
- http://localhost:5000/api/shops/5
- http://localhost:5000/api/shops/register
- http://localhost:5000/api/shops/getShopsByCityId/2

3.4.2 Αναπαράσταση των Endpoints με Swagger

Στα πλαίσια της ανάπτυξης του api, προστέθηκε και ρυθμίστηκε το extension για το Swagger που παρέχει ένα καθαρό και δομημένο documentation. Πάνω σε αυτό μπορούμε μετέπειτα να δημιουργήσουμε ένα collection στο Postman για περαιτέρω δοκιμές της λειτουργικότητάς του, όπως απεικονίζεται και παρακάτω.

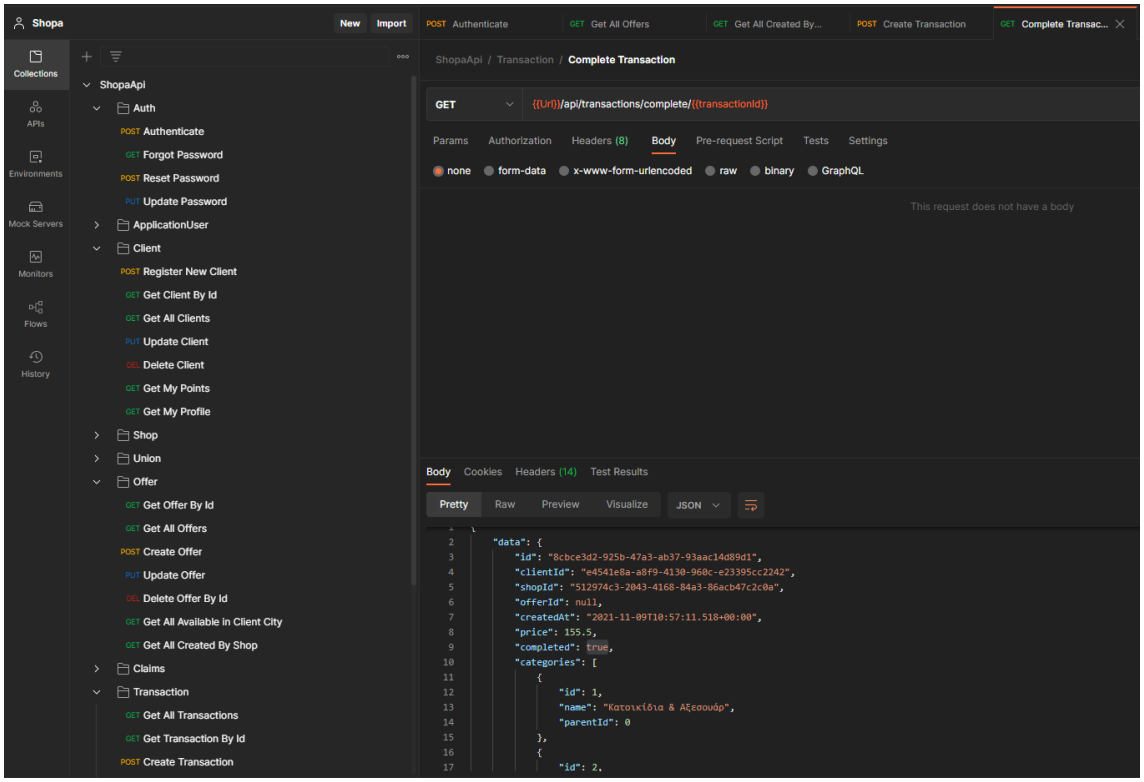


Εικόνα 16 - Swagger api documentation

3.4.3 Οργάνωση των Endpoints στο Postman για Testing

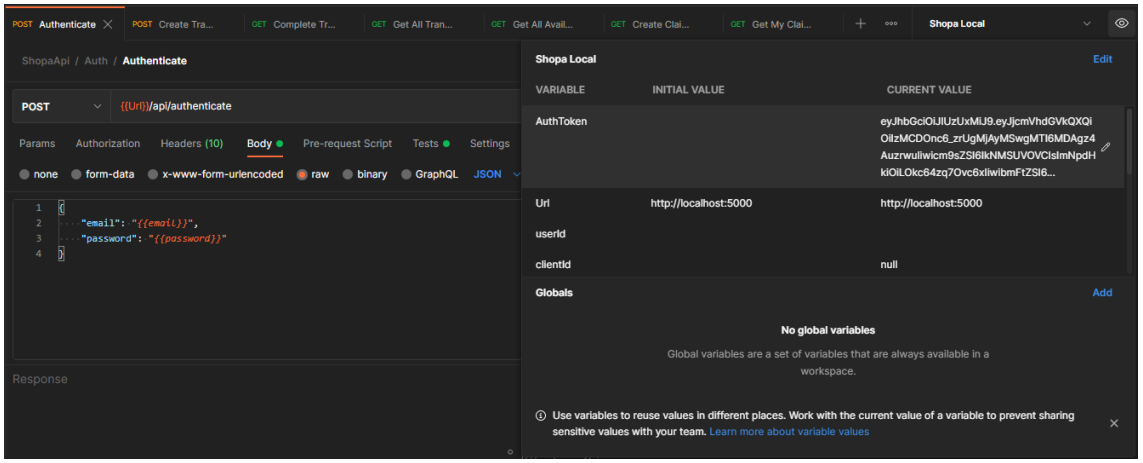
Το Postman είναι μία πλατφόρμα για την οργάνωση και χρήση των apis. Δημιουργώντας ένα collection για το Postman, παρέχεται η δυνατότητα να δοκιμαστούν άμεσα τα endpoints, εκτελώντας πιθανά σενάρια χρήσης, χωρίς να πραγματοποιούνται αυτά μέσα από κάποια εφαρμογή. Περισσότερες πληροφορίες βρίσκονται στο https://www.postman.com/

Στην παρακάτω εικόνα φαίνεται πως έγινε η οργάνωση του api στα πλαίσια αυτής της εργασίας καθώς και ένα δείγμα της μορφής των δεδομένων έπειτα από απάντηση του server.



Εικόνα 17 - Συλλογή se Postman

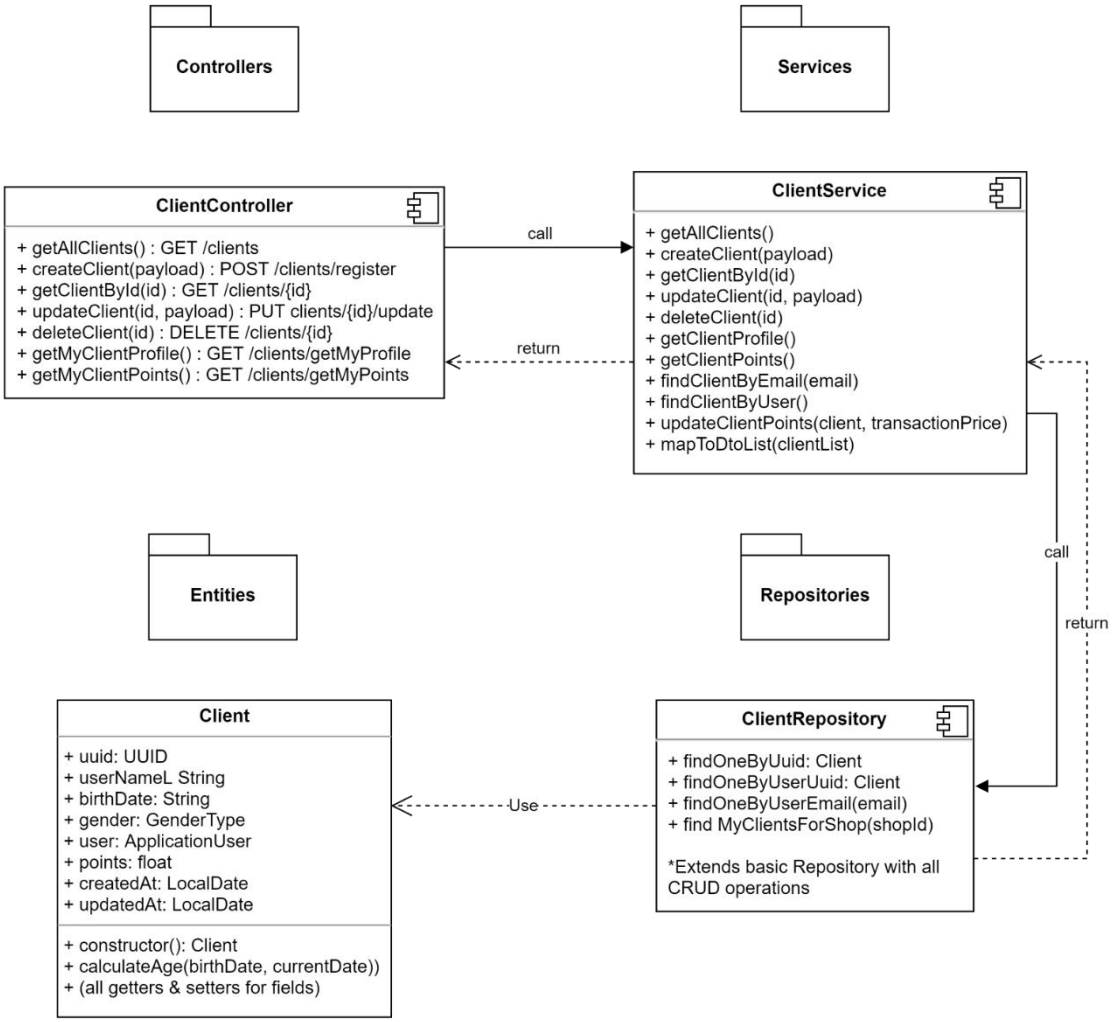
Εκτός από την συλλογή των endpoints, καλό είναι να δημιουργούνται και περιβάλλοντα με μεταβλητές ώστε να δίνονται διαφορετικές τιμές ανά περιβάλλον που πρέπει να δοκιμαστεί, για παράδειγμα local, development και production environment.



Εικόνα 18 - Περιβάλλοντα σε Postman

3.5 Παρουσίαση Υλοποίησης στο Back-End (παράδειγμα Client)

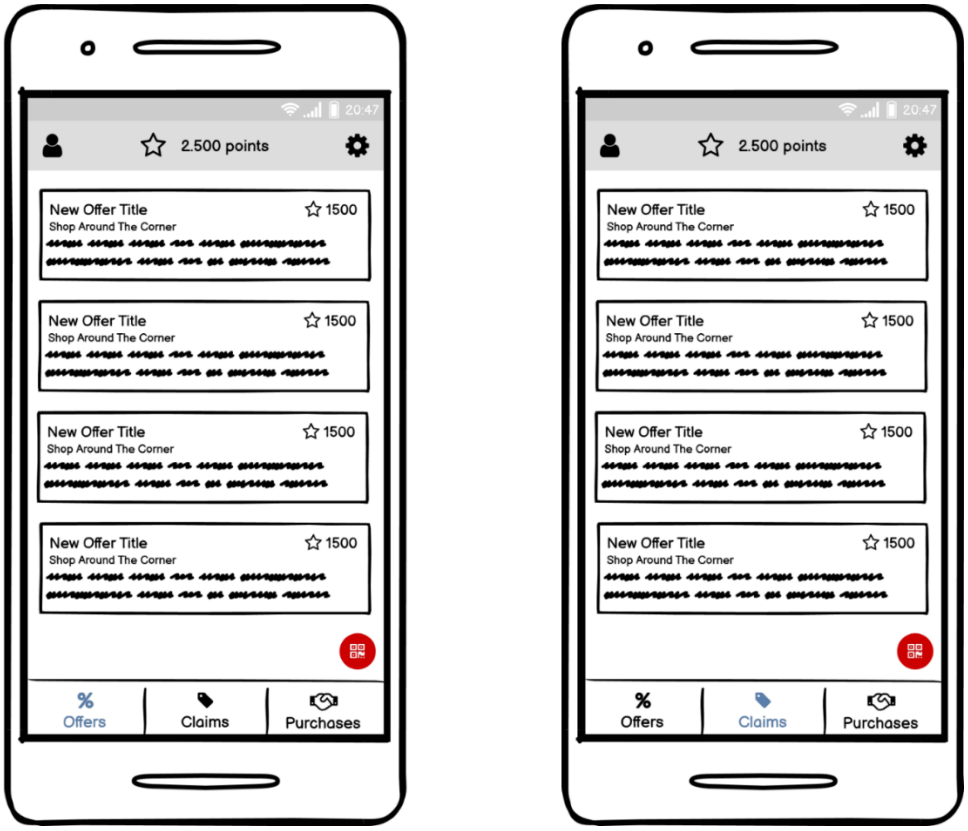
Για να πραγματοποιηθεί η υλοποίηση που θα αφορά μία οντότητα, για παράδειγμα τον Client (πελάτης καταστήματος), υπάρχουν 4 βασικοί πυλώνες, ένα Entity, ένα Repository, ένας Controller και ένα Service. Στην εικόνα φαίνεται η ροή με την οποία λειτουργεί μία κλήση που θα λάβει ο server σχετικά με τους Clients. Στην ενότητα 'Παραδείγματα Κώδικα' βρίσκεται μία ανάλυση του κάθε μέρους ξεχωριστά με ενδεικτικό κώδικα που έχει χρησιμοποιηθεί.



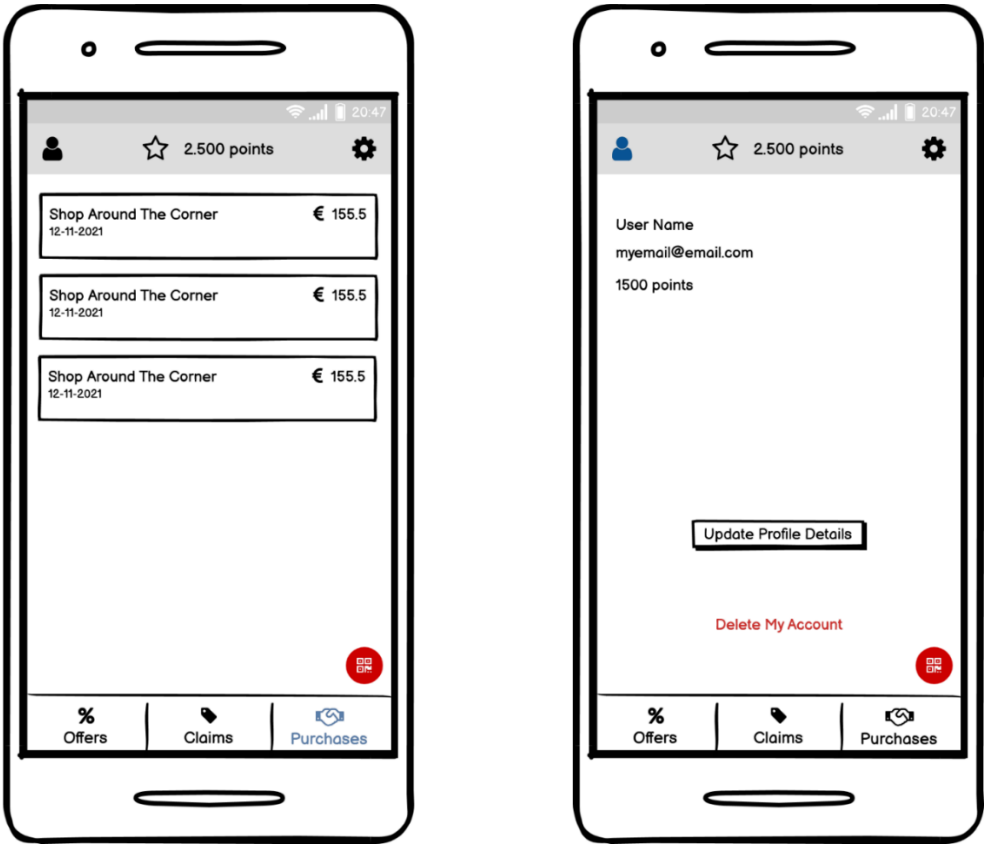
Εικόνα 19 - Ροή υλοποίησης σε back-end

3.6 Wireframes για Εφαρμογή Κινητού

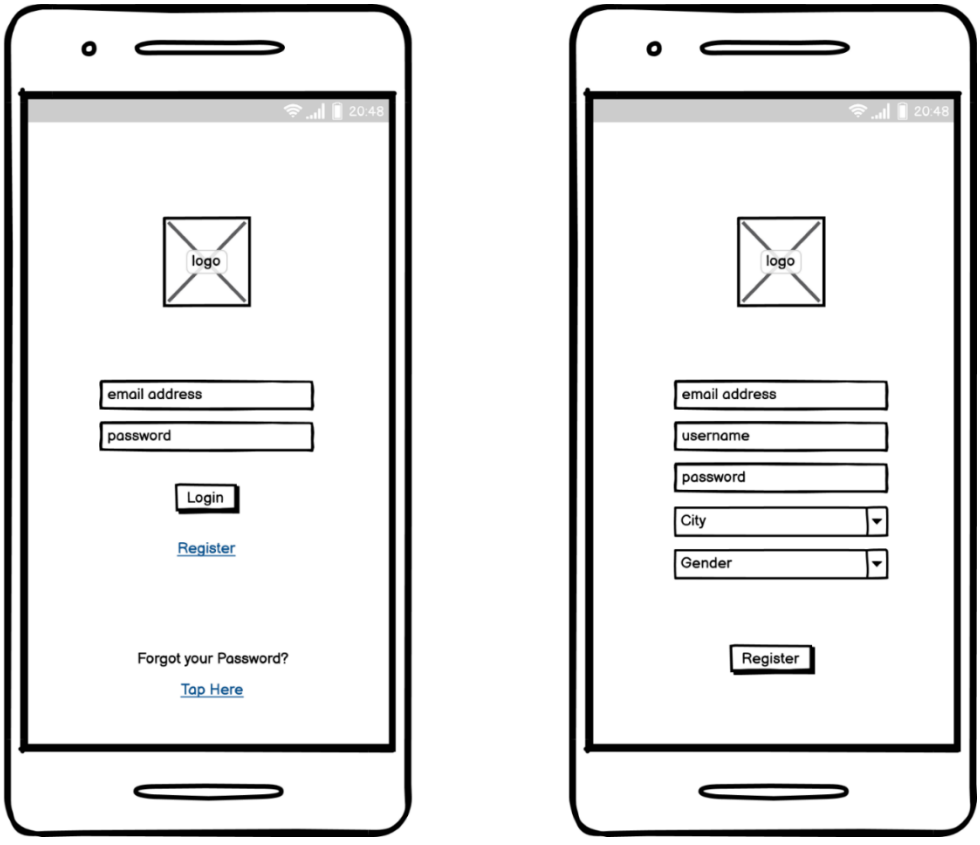
Κατά την φάση σχεδιασμού, χρησιμοποιήθηκε το εργαλείο Balsamiq για την δημιουργία μιας πρότυπης εμφάνισης της εφαρμογής και των εικόνων που απαιτείται να έχει. Αυτό βοήθησε κυρίως στην ανάπτυξη της mobile εφαρμογής αλλά και στην ανάπτυξη του api ώστε κάθε οθόνη να δείχνει τις ανάγκες της σε δεδομένα και ενέργειες από και προς το back-end. Πιο κάτω φαίνονται μερικές βασικές οθόνες της εφαρμογής όπως σχεδιάστηκαν αρχικά.



Εικόνα 20 - Mobile wireframes 1



Εικόνα 21 - Mobile wireframes 2



Εικόνα 22 - Mobile wireframes 3

3.7 Αρχιτεκτονική Ανάπτυξης Λογισμικού

3.7.1 Η Σωστή Αρχιτεκτονική

Η σωστή αρχιτεκτονική αποτελεί σημαντικό παράγοντα της συντήρησης και της επέκτασης της εφαρμογής.

Μας παρέχει ευκολία:

- Στην προσθήκη νέων χαρακτηριστικών
- Στις αλλαγές του υπάρχοντος περιεχομένου
- Στη διόρθωση των λαθών
- Στην ανάγνωση και κατανόηση του κώδικα
- Στην ομαδική δουλειά
- Στην συγχώνευση κώδικα

Είναι σημαντικός παράγοντας για καλύτερες και σωστότερες δοκιμές (testing).

Μας βοηθά στην επαναχρησιμοποίηση του κώδικά μας, καθώς το σωστό είναι να μην γράφουμε ίδιο κώδικα σε πολλαπλά σημεία, ούτε να χρησιμοποιούμε ξανά και ξανά εξαρτήσεις.

Έχει παρατηρηθεί πως στις ανάγκες της εκάστοτε εφαρμογής απαιτείται και μια διαφορετική προσέγγιση.

Παρ' όλα αυτά, υπάρχουν κάποιοι κανόνες στον σχεδιασμό του αντικειμενοστραφούς λογισμικού. Αυτοί οι κανόνες είναι γνωστοί με την ονομασία S.O.L.I.D. Principles of Object Oriented Software Design και παρουσιάστηκαν από τον Robert C. Martin (Uncle Bob), έναν πολύ γνωστό μηχανικό λογισμικού και εκ των ιδρυτών της agile software development μεθοδολογίας.

1. Single-Responsibility principle

Μία κλάση πρέπει να έχει μία μόνο δουλειά. Αυτό σημαίνει πως υπάρχει μόνο ένας λόγος για να αλλάξει.

2. Open-Closed principle

Δεν πρέπει να επιτρέπουμε σε τρίτα αντικείμενα να τροποποιούν οντότητες ενός αντικειμένου, αλλά πρέπει να δημιουργούμε τις συνθήκες για επέκτασή τους, έχοντάς τα ως σκελετό για υπο-αντικείμενα.

3. Liskov Substitution principle

Κάθε υποκλάση πρέπει να είναι υποκατάστατο για την γονική κλάση

4. Interface Segregation principle

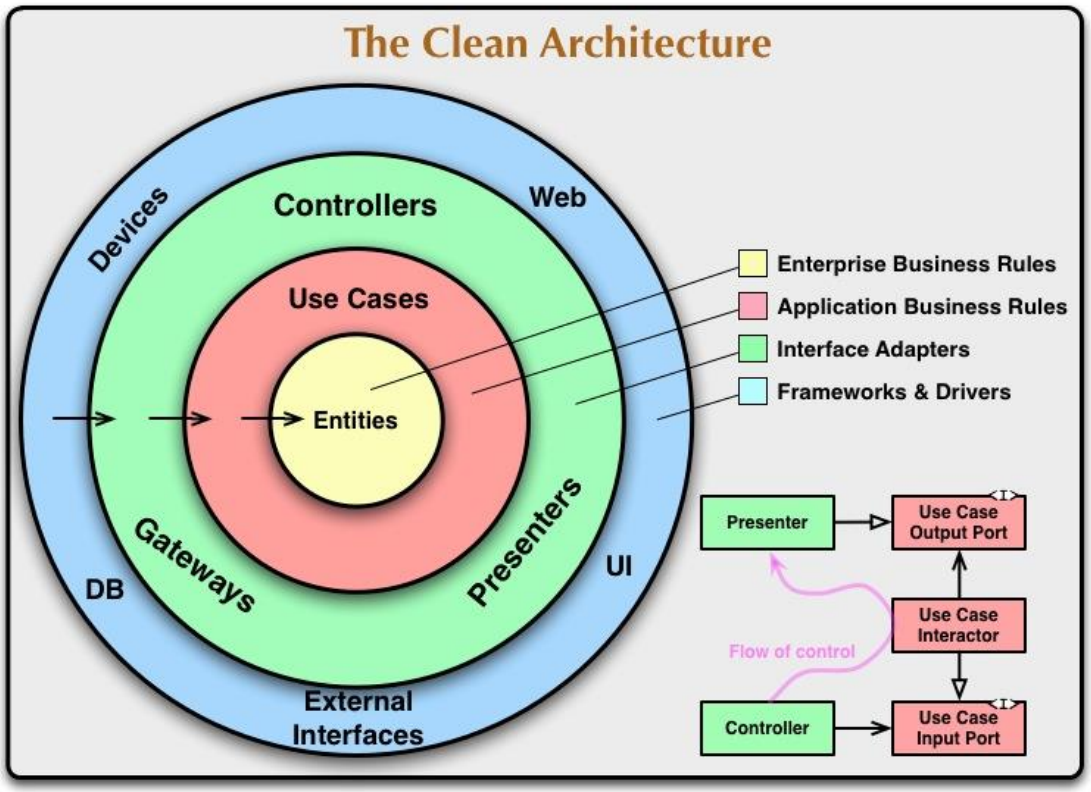
Δεν πρέπει να δημιουργούμε διεπαφές ή μεθόδους που δεν χρησιμοποιούνται.

5. Dependency Inversion Principle

Τα αντικείμενα δεν πρέπει να δημιουργούν νέα αντικείμενα από μόνα τους. Πρέπει να φτάνουν στα άλλα αντικείμενα με τη χρήση διεπαφών. Στην πραγματική ζωή δεν τα φτιάχνουμε όλα μόνοι μας, πηγαίνουμε στο supermarket και το αγοράζουμε.

3.7.2 Clean Architecture

Σύμφωνα με τον Robert C. Martin, η καθαρή αρχιτεκτονική ενός λογισμικού βασίζεται στην φιλοσοφία σχεδιασμού που διαχωρίζει τα κομμάτια του σε επίπεδα δαχτυλιδιών ή κύκλων όπως φαίνεται στην παρακάτω εικόνα:



Εικόνα 23 - Clean architecture

Ο βασικός κανόνας αυτής της αρχιτεκτονικής είναι ότι οι εξαρτήσεις κώδικα από κομμάτι σε κομμάτι, μπορούν να έχουν μόνο κατεύθυνση από τα έξω προς τα μέσα. Ο κώδικας των εσωτερικών επιπέδων δεν πρέπει να έχει επίγνωση των λειτουργιών που βρίσκονται στα ανώτερα επίπεδα.

3.7.4 MVI & MVVM Αρχιτεκτονικές για Android

Model-View-ViewModel (MVVM) είναι ένα μοτίβο αρχιτεκτονικής λογισμικού το οποίο διαχωρίζει την υλοποίηση του γραφικού μέρους (View) από την υλοποίηση της λογικής είτε του business είτε του back-end (Model) ώστε το πρώτο να είναι ανεξάρτητο από το τι θα χρησιμοποιηθεί για την υλοποίηση του δεύτερου. Το ViewModel είναι ένας ενδιάμεσος μετατροπέας των δεδομένων για την επικοινωνία από το Model προς το View.

Το MVVM είναι μια παραλλαγή του Presentation Model design pattern από τον Martin Fowler.

Model-View-Intent (MVI) είναι επίσης ένα μοτίβο αρχιτεκτονικής λογισμικού, το οποίο μας δίνει μια διαφορετική προσέγγιση χρησιμοποιώντας ένα state. Λύνει τα προβλήματα που προκαλούν τα διαφορετικά states που μπορεί να έχουν το model, view και viewmodel, ορίζοντας το Model ως ένα και μοναδικό Immutable Single Source of Truth όπου οποιαδήποτε αλλαγή πρέπει να περάσει από εκεί. Το View εξακολουθεί να έχει το ρόλο που αναφέρθηκε και στο MVVM, ενώ το Intent είναι η πρόθεση για δράση στην εφαρμογή είτε από τον χρήστη ή από την ίδια την εφαρμογή.

3.7.5 Τελική Μορφή Android Αρχιτεκτονικής

Όπως αναφέρθηκε στην αρχή, δεν υπάρχει σωστή αρχιτεκτονική καθώς αυτή αλλάζει ανάλογα με τις ανάγκες της εκάστοτε εφαρμογής και των ατόμων που την χρησιμοποιούν. Στην περίπτωση της εργασίας αυτής, αποφασίστηκε να χρησιμοποιηθούν οι κανόνες του Clean Architecture

χωρίζοντας την εφαρμογή σε 3 επίπεδα αλλά με μία αλλαγή στο ένα επίπεδο:

- Presentation UI (Υβρίδιο MVVM & MVI)
- Domain - Business
- Data - Networking Datasource

Η επικοινωνία των επιπέδων αυτών θα πραγματοποιηθεί με την χρήση των Coroutines που διευκολύνουν την ασύγχρονη λειτουργία και το Hilt για το Dependency Injection.

3.7.6 Ανάλυση Επιπέδων και Υβριδίου MVVM – MVI

Το επίπεδο Data, αναλαμβάνει ότι σχετίζεται με τα δεδομένα και την επικοινωνία που απαιτείται για την επεξεργασία τους μέσω του api.

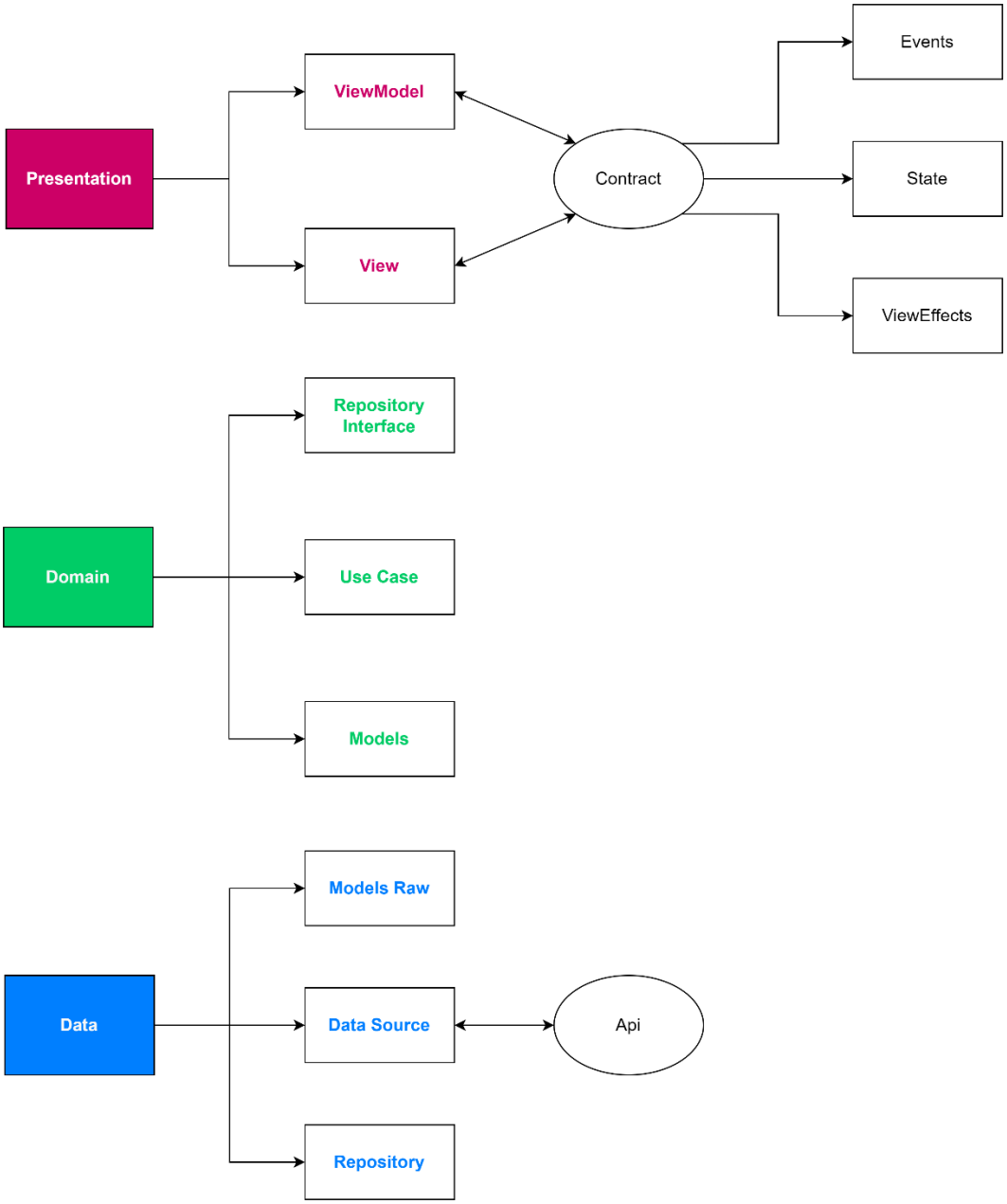
- Ως Data Source ορίζεται η πηγή από την οποία έρχονται τα δεδομένα και αποτελείται από interfaces με τις κλήσεις που γίνονται προς το back-end, τι url απαιτείται και σε τι μορφή είναι η απάντηση.
- Το Repository αναλαμβάνει την υλοποίηση των κλήσεων αυτών και κάνει extend το Repository Interface που είναι υλοποιημένο στο επίπεδο Domain.
- Η μορφή των δεδομένων που λαμβάνει η εφαρμογή από το api, ορίζεται σε Raw Models.

Το επίπεδο Domain αναλαμβάνει ότι σχετίζεται με το business της εφαρμογής.

- Τα Repository Interfaces περιέχουν abstract functions για τα απαραίτητα δεδομένα χρήσης. (Πάνω σε αυτό φτιάχνεται το Repository του Data επιπέδου)
- Τα Models είναι η μορφή των δεδομένων όπως τα καταλαβαίνει και τα χειρίζεται η εφαρφογή.
- Τα Use Cases με την βοήθεια των Repositories εκτελούν το business που απαιτείται ανά χρήση, π.χ. το τι πρέπει να γίνει όταν κάνει Authenticate ένας χρήστης ή όταν θέλει την λίστα των προσφορών.

Το επίπεδο Presentation, το οποίο έχει επιρροή από τον συνδυασμό των MVVM και MVI όπως αναφέρθηκε νωρίτερα, σχετίζεται με το τι είναι ορατό στην οθόνη της συσκευής και πως μεταφράζονται οι αλληλεπιδράσεις του χρήστη πάνω στην οθόνη.

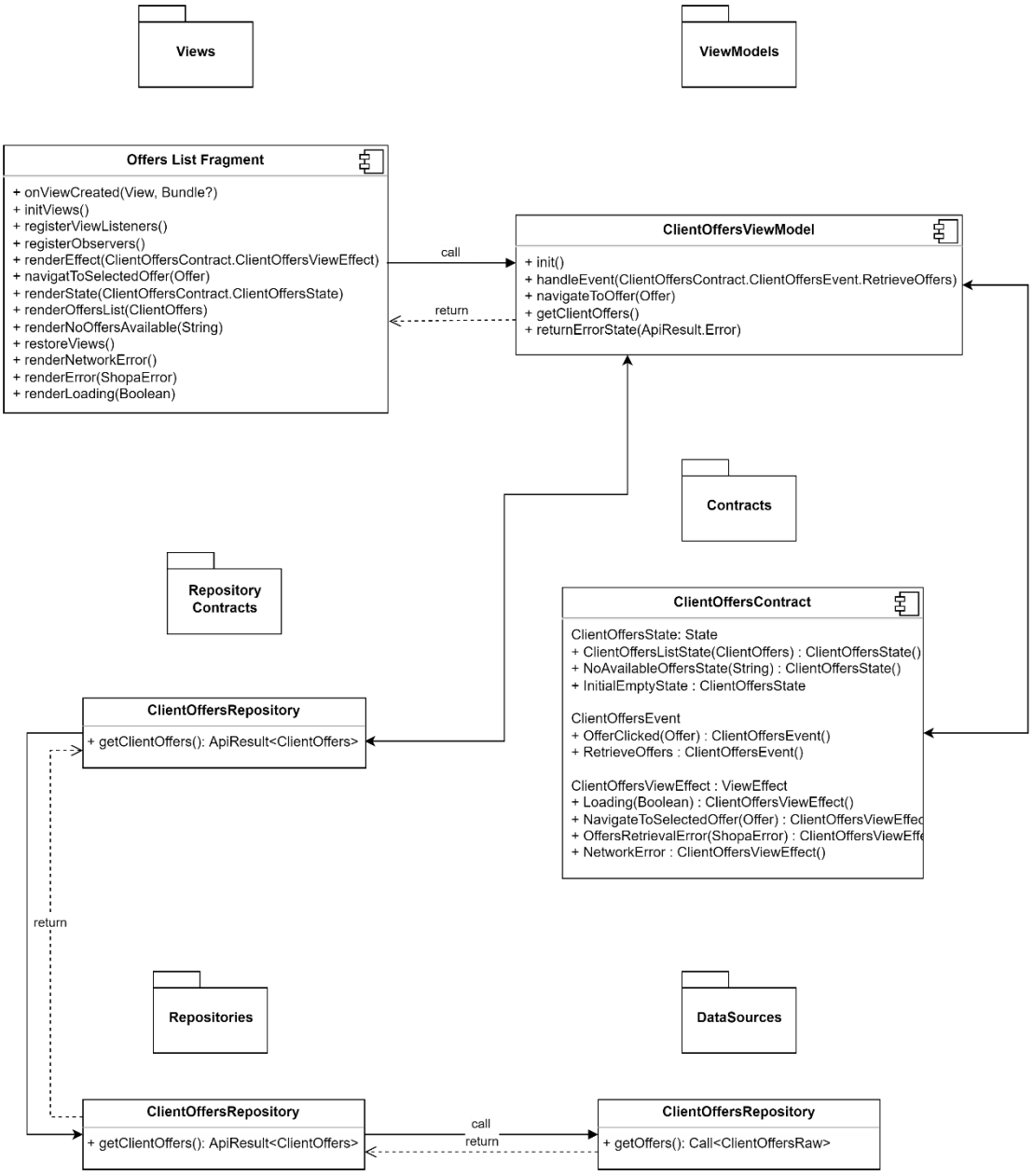
- Τα Views είναι είτε Activities ή Fragments που χτίζουν την οθόνη και για κάθε αλληλεπίδραση με αυτή καλούν το ViewModel.
- Το ViewModel είναι το μέρος που εκτελείται ότι πρέπει να ακολουθήσει την αλληλεπίδραση του χρήστη με την οθόνη.
- Τα Contracts μαθαίνουν τα Events από το ViewModel, καθορίζουν το State της εφαρμογής και εκτελούν τα ανά περίπτωση τα ViewEffects, τα οποία είναι one-time events.



Εικόνα 24 - MVVM & MVI αρχιτεκτονική

3.8 Παρουσίαση Υλοποίησης Οθόνης (παράδειγμα Client Offers List)

Για να πραγματοποιηθεί η υλοποίηση που θα αφορά ένα μία οθόνη, για παράδειγμα αυτή με το Client Offers List (λίστα με offers διαθέσιμα για τον Client), υπάρχουν 6 βασικοί πυλώνες, View, ViewModel, Contract, Repository Contract, Repository και DataSource. Παρακάτω στην εικόνα φαίνεται η σειρά με την οποία λειτουργούν μεταξύ τους όταν φορτώσει το fragment:



Εικόνα 25 - Ροή υλοποίησης σε Android

- Ο Χρήστης ανοίγει το Fragment και εκτελείται το OffersEvent
- Το ViewModel κάνει handle το Event καλώντας το OffersUseCase
- Το UseCase αυτό ζητά από το Repository τα Offers
- Το Api επιστρέφει πίσω τα raw data που γίνονται map σε model εφαρμογής
- Το UseCase λαμβάνει αυτά τα data και τα δίνει στο ViewModel
- Το ViewModel δημιουργεί ένα state για τα Offers αυτά και το επιστρέφει στο View
- Το View αναλαμβάνει να κάνει display το state στην οθόνη

4. Υλοποίηση

4.1 Βάση Δεδομένων

Για την βάση δεδομένων χρησιμοποιήθηκε η PostgreSQL η οποία είναι μία σχεσιακή βάση ανοικτού κώδικα. Η ανάπτυξή της διαρκεί πάνω από δύο δεκαετίες και είναι γνωστή για την αξιοπιστία και την ακεραιότητα των δεδομένων της. Τρέχει σε όλα τα βασικά λειτουργικά συστήματα και ακολουθεί το SQL Standard. Μερικά από τα πλεονεκτήματά της, όπως αναφέρονται σε επίσημη πηγή είναι :

- User-defined types
- Table inheritance
- Sophisticated locking mechanism
- Foreign key referential integrity
- Views, rules, subquery
- Nested transactions (savepoints)
- Multi-version concurrency control (MVCC)
- Asynchronous replication

Μερικές από τις εταιρείες που χρησιμοποιούν την postgres είναι η Apple, Cisco, Instagram.

4.2 Εγκατάσταση IDE

Για την ανάπτυξη του back-end χρησιμοποιήθηκε το IntelliJ της JetBrains, το οποίο θεωρείται το καλύτερο εργαλείο αυτή την στιγμή στην ανάπτυξη java εφαρμογών. Αυτό ενισχύεται και από το γεγονός πως το γνωστό σε όλους Android Studio για την ανάπτυξη mobile εφαρμογών, είναι στην πραγματικότητα μία ειδική έκδοση του IntelliJ, πράγμα που βοήθησε κατά την ανάπτυξη και της Android εφαρμογής για την εργασία αυτή. Αξίζει να σημειωθεί πως χρησιμοποιήθηκε η έκδοση Java 11 για το back-end σε Spring Boot και το Android 11 & Android 12 sdk για την mobile εφαρμογή.

4.3 Framework & Extensions

Το Spring Boot είναι ένα open-source λογισμικό, ανήκει στην εταιρεία Pivotal και χρησιμοποιεί ως γλώσσα προγραμματισμού την Java. Βασίζεται στο Spring Framework και εστιάζει περισσότερο στο Dependency Injection.

Έχει modular αρχιτεκτονική και με την χρήση εργαλείων που παρέχονται στο Spring Boot Starter (JPA, Web, Validation, JDBC κτλ) βοηθά στην ελαχιστοποίηση του κώδικα που απαιτείται καθιστώντας το ταυτόχρονα αξιόπιστο, γρήγορο και ικανό για οποιαδήποτε επέκταση αναγκών και λειτουργιών που χρειάζεται ένα api.

Περιλαμβάνει ενσωματωμένο web server (συνήθως τονTomcat) ώστε να μπορεί να λειτουργήσει απευθείας μετά την αρχικοποίησή της και σε οποιοδήποτε άλλο στάδιο χρειαστεί.

Στα πλεονεκτήματά του είναι:

- Η ευκολία κατανόησης
- Η αυξημένη παραγωγικότητα
- Η μείωση του χρόνου παραγωγής
- Η εύκολη παραγωγή λογισμικού έτοιμου για κατανάλωση
- Η δυναμική διαχείριση των REST endpoints

Το Spring Boot παρέχει τα λεγόμενα “Spring Boot Starters”, έτοιμα σετ από dependencies για την διευκόλυνση των developers ώστε να ελαχιστοποιηθεί ο χρόνος εγκατάστασης και ρύθμισης. Ένα παράδειγμα συχνής χρήσης είναι το spring-boot-starter-data-jpa το οποίο χρησιμοποιείται για σύνδεση με βάση δεδομένων.

4.4 Authentication

JSON Web Token (JWT) είναι μία κωδικοποιημένη αναπαράσταση κάποιων claims σε JSON μορφή και μπορούν δύο μέρη να τα μεταφέρουν μεταξύ τους. Τα claims είναι ψηφιακά υπογεγραμμένα από το μέρος που τα έχει εκδόσει, και το μέρος που τα λαμβάνει μπορεί αργότερα να χρησιμοποιήσει αυτή την ψηφιακή υπογραφή ώστε να αποδείξει πως είναι ο ιδιοκτήτης τους. Τα JWTs αποτελούνται από: header, payload, and signature. Κάθε μέρος χωρίζεται με μία τελεία και η μορφή του είναι: Header.Payload.Signature

Για παράδειγμα έχουμε:


```
eyJhbGciOiJIUzUxMiJ9.eyJjcmVhdGVkQXQiOi
IxMSDOnc6_zrUgMjAyMSwgMTI6MDAgz4AuzrwuI
iwicm9sZSI6IkNMSUVOVCIsImNpdHkiOiLOkc64
zq7Ovc6xIiwibmFtZSI6ImNsaWVudDIxMiIsImN
pdHlJZCI6MSwiYXV0aG9yaXRpZXMiOlt7ImF1dG
hvcml0eSI6IkNMSUVOVF9XUklURSJ9LHsiYXV0a
G9yaXR5IjoiT0ZGRVJfUkVBRCJ9LHsiYXV0aG9y
aXR5IjoiQ0xBSU1fV1JJVEUifSx7ImF1dGhvcml
0eSI6IkNMQUlNX1JFQUQifSx7ImF1dGhvcml0eS
I6IlRSQU5TQUNUSU9OX0NPTVBMRVRFIn0seyJhd
XRob3JpdHkiOiJUUkFOU0FDVElPTl9SRUFEIn0s
eyJhdXRob3JpdHkiOiJDTElFTlRfUkVBRCJ9LHs
iYXV0aG9yaXR5IjoiU0hPUF9SRUFEIn1dLCJibm
FibGVkIjp0cnVlLCJzdWIiOiJjbGllbnQxQGdtY
WlsLmNvbSIsImlhdCI6MTYzNjYzMTI5MywiZXhw
IjoxNjM2NjQ5MjkzfQ.L40msxDSlVPZndpFrYQ2
6Pf0qJ84CHN9eReRSCpWWszuE8ubEdaNqO-
GzmUgWMQh1yh_CmYUfaRSIAMzN4d3Fw
```


Εικόνα 26 - Παράδειγμα JWT token

Το οποίο μεταφράζεται ως:


```
{
  "createdAt": "11 Νοε 2021, 12:00 π.μ.",
  "role": "CLIENT",
  "city": "Αθήνα",
  "name": "client212",
  "cityId": 1,
  "authorities": [
    {
      "authority": "CLIENT_WRITE"
    },
    {
      "authority": "OFFER_READ"
    },
    {
      "authority": "CLAIM_WRITE"
    },
    {
      "authority": "CLAIM_READ"
    },
    {
      "authority": "TRANSACTION_COMPLETE"
    },
    {
      "authority": "TRANSACTION_READ"
    },
    {
      "authority": "CLIENT_READ"
    },
    {
      "authority": "SHOP_READ"
    }
  ],
  "enabled": true,
  "sub": "client1@gmail.com",
  "iat": 1636631293,
  "exp": 1636649293
}
```


Εικόνα 27 - Αποκωδικοποίηση JWT token

4.5 Παραδείγματα Κώδικα σε Back-End

4.5.1 Entity

Το Entity είναι μία κλάση POJO (Plain Old Java Object) η οποία αναπαριστά τον πίνακα στον οποίο για γίνουν persist τα δεδομένα στην σχεσιακή βάση τους. Κάθε instance του Entity αναπαριστά ένα row στον πίνακα. Προαιρετικά χρησιμοποιούνται κάποια annotations (π.χ. @NotNull) για επιπλέον ελέγχους και restrictions.

Κάθε τέτοια κλάση έχει έναν κενό constructor και έναν με παραμέτρους, ο οποίος χρησιμοποιείται όποτε η λειτουργικότητα πρέπει να φτάσει έως την βάση δεδομένων.

```java
@Entity
@Table(name = "clients")
public class Client {

    @Id
    @GenericGenerator(name = "uuid2", strategy = "uuid2")
    @GeneratedValue(strategy = GenerationType.IDENTITY, generator = "uuid2")
    @Column(updatable = false, nullable = false)
    @Type(type="org.hibernate.type.UUIDCharType")
    @JsonIgnore
    private UUID uuid;

    @NotNull
    private String userName;

    @NotNull
    private String birthDate;

    @NotNull
    private GenderType gender;

    @OneToOne(cascade = CascadeType.REMOVE, orphanRemoval = true)
    @JsonIgnore
    private ApplicationUser user;

    private float points;

    @CreatedDate
    @JsonFormat(pattern="dd-MM-yyyy")
    private LocalDate createdAt;

    @JsonFormat(pattern="dd-MM-yyyy")
    private LocalDate updatedAt;

    public Client() { }

    public Client(
            String userName,
            String birthDate,
            GenderType gender,
            ApplicationUser user
    ) {
```

Εικόνα 28 - Back-end Entity

4.5.2 Controller

Ο Controller είναι μία κλάση-υποδοχέας των endpoints που έχει το api. Για κάθε endpoint, ορίζουμε στο url το entity στο οποίο απευθύνεται ένας controller όταν λαμβάνει requests για τα operations που του αναλογούν. Π.χ. για το entity Client, το endpoint πρεπει να ξεκινά με /clients/*, κάτι το οποίο ορίζεται αρχικά στον controller και αντί για αστεράκι ορίζουμε όλα τα sub-endpoints που αντιστοιχούν σε κάποιο συγκεκριμένα functionality.

Αφού ο Controller λάβει ένα request, τότε καλεί ένα service, υλοποιημένο για να εξυπηρετεί το logic, του συγκεκριμένου controller. Πρόκειται για Rest Controller όπως φαίνεται και στο annotation, πράγμα που δείχνει ότι ασχολείται μόνο με την διαχείριση των Rest api calls και δεν πρέπει να θεωρηθεί controller από την MVC αρχιτεκτονική, όπου θα αναλάβει να επιστρέψει τα data σε ένα View ώστε να γίνει render.

```java
@Api(
        value="Clients Controller",
        description="Operations for Clients using Mobile App",
        tags = { "Clients" }
)
@RestController
@RequestMapping("/clients")
public class ClientController extends BaseController {

    private final ClientServiceImpl clientService;

    @Autowired
    public ClientController(ClientServiceImpl clientService) {

        this.clientService = clientService;
    }

    // Basic CRUD Operations _____________________

    @ApiOperation(value = "Fetch all Clients", notes = "This is for Admin to get all clients created", tags = { usageAdmin })
    @GetMapping
    @ResponseBody
    @PreAuthorize("hasAuthority('ALL_READ')")
    public ResponseEntity<Object> getAllClients() { return clientService.getAllClients(); }

    @ApiOperation(value = "Register a new Client", notes = "This is to be used from client form", tags = { usageMobile })
    @PostMapping("/register")
    @ResponseBody
    public ResponseEntity<Object> createClient(@Valid @RequestBody ClientCreatePayload clientCreatePayload) {
        return clientService.createClient(clientCreatePayload);
    }

    @ApiOperation(value = "Fetch Client by clientId", notes = "", tags = { usageMobile, usageShop })
    @GetMapping("/{id}")
    @PreAuthorize("hasAuthority('CLIENT_READ')")
    public ResponseEntity<Object> getClientById(@PathVariable("id") UUID id) { return clientService.getClientById(id); }

    @ApiOperation(value = "Update logged Client Details", notes = "", tags = { usageMobile })
    @PutMapping("/update")
    @PreAuthorize("hasAuthority('CLIENT_WRITE') or hasAuthority('ALL_WRITE')")
    public ResponseEntity<Object> updateClient(HttpServletRequest req, @Valid @RequestBody ClientUpdatePayload ClientUpdatePayload) {
        return clientService.updateClient(req, ClientUpdatePayload);
    }

    @ApiOperation(value = "Delete Client permanently", notes = "This is admin action for deleting ALL data related using anonymous clien
    @DeleteMapping("/{id}")
    @PreAuthorize("hasAuthority('CLIENT_WRITE') or hasAuthority('ALL_WRITE')")
    public ResponseEntity<Object> deleteClient(@PathVariable("id") UUID id) { return clientService.deleteClient(id); }

    // More Specific Operations _____________________

    @ApiOperation(value = "Get my Profile as Client", notes = "When logged in as Client", tags = { usageMobile })
    @GetMapping("/getMyProfile")
    @ResponseBody
    @PreAuthorize("hasAuthority('CLIENT_READ')")
    public ResponseEntity<Object> getMyClientProfile(HttpServletRequest req) throws NotFoundException {
        return clientService.getMyClientProfile(req);
    }
```

Εικόνα 29 - Back-end Controller

4.5.3 Service

Το Service είναι μια κλάση που αναλαμβάνει να εκτελέσει όλη την λειτουργικότητα πίσω από την κλήση ενός endpoint. Καλείται συνήθως από τον Controller και στον constructor ορίζονται όλα τα dependencies όπως π.χ. τα Repositories, Services και Helpers τα οποία χρησιμοποιεί στην πορεία ως μεσάζοντας του Controller και της βάσης δεδομένων.

Κάποιο Service μπορεί να καλέσει άλλο Service εφόσον χρειαστεί κάτι από ένα άλλο entity και πιθανώς να μην είναι στο scope του συγκεκριμένου service να το εκτελέσει. Αυτό γίνεται επίσης ώστε να μην χρειάζεται να γράφουμε τον ίδιο κώδικα σε πολλά σημεία.

Για τα services έχει επίσης χρησιμοποιηθεί ένα level of abstraction ώστε να δηλώνονται πιο καθαρά όλες οι functions που πρέπει να υλοποιηθούν.

```java
@Service
@Transactional
public class ClientServiceImpl implements ClientService {

    private final JwtTokenUtil jwtTokenUtil;

    private final ClientRepository clientRepository;

    private final ApplicationUserService userService;

    private final CityService cityService;

    @Autowired
    public ClientServiceImpl(
            JwtTokenUtil jwtTokenUtil,
            ClientRepository clientRepository,
            ApplicationUserService userService,
            CityService cityService) {
        this.jwtTokenUtil = jwtTokenUtil;
        this.clientRepository = clientRepository;
        this.userService = userService;
        this.cityService = cityService;
    }

    // Basic Crud Services _________________________

    public ResponseEntity<Object> getAllClients() {

        List<ClientDto> clientList = mapToDtoList(clientRepository.findAll());

        if (clientList.isEmpty()) {
            return ResponseHandler.generateResponse(CLIENT_LIST_EMPTY, clientList);
        }

        return ResponseHandler.generateResponse(CLIENT_LIST_FETCH, clientList);
    }

    public ResponseEntity<Object> getClientById(UUID clientId) throws NotFoundException {
        ClientDto clientDto = clientRepository
                .findById(clientId)
                .map(ClientDto::new)
                .orElseThrow(() -> new NotFoundException(CLIENT_DOES_NOT_EXIST.getMessage()));

        return ResponseHandler.generateResponse(CLIENT_FOUND, clientDto);
    }

    @Transactional(rollbackOn = BadRequestException.class)
    public ResponseEntity<Object> createClient(ClientCreatePayload clientDTO) throws AlreadyExistsException, BadRequestException {

        cityService.checkIfCityExists(clientDTO.getCityId());

        try {
            ApplicationUser user = userService.createUserWithRole(clientDTO, ApplicationRoleType.CLIENT);
```

Εικόνα 30 - Back-end Service

4.5.4 Repository

Το Repository είναι ένα interface που κάνει extend το JpaRepository του Spring JPA extension. Εκτελεί την επικοινωνία με την βάση μέσω ενός έξυπνου κώδικα που μεταφράζει το όνομα μίας function στο αντίστοιχο sql query. Π.χ. findOneByUserEmail(String email) θα μεταφραστεί

αυτόματα σε ‘SELECT client FROM Client JOIN client.user user WHERE user.email = :email’. Αν χρειαζόμαστε ένα πολύ σύνθετο query στην βάση το οποίο δεν είναι δυνατό να δηλωθεί ονομαστικά στο JPA, τότε μπορούμε να γράψουμε όλο το query με το annotation όπως φαίνεται στην εικόνα πιο κάτω.

```
public interface TransactionRepository extends JpaRepository<Transaction, Long> {
    Optional<Transaction> findById(UUID id);
    List<Transaction> findAllByClientUuid(UUID id);
    List<Transaction> findAllByClientUserEmail(String email);
    List<Transaction> findAllByClientUserCityId(Integer id);
    List<Transaction> findAllByShopUuidOrderByCreatedAtDesc(UUID id);

    @Query("SELECT DISTINCT t FROM Transaction t JOIN t.shop s WHERE " +
            "(:shopId is null OR CAST(s.uuid AS string) = :shopId) AND " +
            "(COALESCE(:completed, '') is null OR CAST(t.completed AS string) = :completed) ")
    List<Transaction> findAllByShopAndStatus(@Param("shopId") String shopId, @Param("completed") String completed);

    // Analytics ____________________________________________

    @Query("SELECT t FROM Transaction t JOIN t.shop s JOIN s.user u JOIN u.city c WHERE c.id in :ids")
    List<Transaction> findAllByCityIds(@Param("ids") List<Integer> cityIdsList);

    @Query("SELECT DISTINCT t FROM Transaction t JOIN t.shop s JOIN s.user u JOIN u.city c WHERE " +
            "(:shopId is null OR CAST(s.uuid AS string) = :shopId) AND " +
            "(:cityId is null OR c.id = :cityId) AND " +
            "(cast(:createdAfter AS timestamp) is null OR t.createdAt >= :createdAfter) AND " +
            "(cast(:createdBefore AS timestamp) is null OR t.createdAt <= :createdBefore) AND " +
            "(:categoryId is null OR :categoryId in t.categories) AND " +
            "t.completed = true")
    List<Transaction> findTransactionsByFilters(
            @Param("shopId") String shopId,
            @Param("cityId") Integer cityId,
            @Param("createdAfter") Date createdAfter,
            @Param("createdBefore") Date createdBefore,
            @Param("categoryId") Long categoryId
    );

    @Query(value = "SELECT DISTINCT t FROM Transaction t JOIN t.shop s JOIN s.user u JOIN u.city c WHERE " +
            "(t.offer is not null) AND " +
            "(:shopId is null OR CAST(s.uuid AS string) = :shopId) AND " +
            "(:cityId is null OR c.id = :cityId) AND " +
            "(cast(:createdAfter AS timestamp) is null OR t.createdAt >= :createdAfter) AND " +
            "(cast(:createdBefore AS timestamp) is null OR t.createdAt <= :createdBefore) AND " +
            "(:categoryId is null OR :categoryId in t.categories) AND " +
            "t.completed = true")
    List<Transaction> findTransactionsFromOffersByFilters(
            @Param("shopId") String shopId,
            @Param("cityId") Integer cityId,
            @Param("createdAfter") Date createdAfter,
            @Param("createdBefore") Date createdBefore,
            @Param("categoryId") Long categoryId
    );

    @Query(value = "SELECT DISTINCT t FROM Transaction t JOIN t.shop s JOIN s.user u JOIN u.city c WHERE " +
            "(t.offer is null) AND " +
            "(:shopId is null OR CAST(s.uuid AS string) = :shopId) AND " +
            "(:cityId is null OR c.id = :cityId) AND " +
            "(cast(:createdAfter AS timestamp) is null OR t.createdAt >= :createdAfter) AND " +
            "(cast(:createdBefore AS timestamp) is null OR t.createdAt <= :createdBefore) AND " +
            "(:categoryId is null OR :categoryId in t.categories) AND " +
            "t.completed = true")
```

Εικόνα 31 - Back-end Repository

4.5.5 Exception Handler

Για την διαχείριση των exceptions κατά την διεκπεραίωση μίας κλήσης στο api, έχει φτιαχτεί ένα Global Exception Handler, το οποίο «ακούει» για exceptions και αναλαμβάνει ανά κατηγορία να επιστρέψει ένα μήνυμα λάθους και έναν κωδικό HTTP. Αν για παράδειγμα το service επιστρέψει BadRequestException, τότε αυτομάτως θα τρέξει στο παρασκήνιο ο GlobalExceptionHandler για την BadRequestException class όπως έχει δηλωθεί και μέσω του annotation @ExceptionHandler().

```java
@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(ForbiddenException.class)
    public ResponseEntity<Object> handleForbiddenException(Exception ex, HttpServletRequest request) {
        return generateErrorResponse(ex.getMessage(), HttpStatus.FORBIDDEN);
    }

    @ExceptionHandler(BusinessException.class)
    public ResponseEntity<Object> handleBusinessExceptionException(Exception ex, HttpServletRequest request) {
        return generateErrorResponse(ex.getMessage(), HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(NotFoundException.class)
    public ResponseEntity<Object> handleResourceNotFoundException(NotFoundException ex) {
        return generateErrorResponse(ex.getMessage(), HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(NoDataFoundException.class)
    public ResponseEntity<Object> handleResourceNoDataFoundException(NoDataFoundException ex) {
        return generateErrorResponse(ex.getMessage(), HttpStatus.NO_CONTENT);
    }

    @ExceptionHandler(AlreadyExistsException.class)
    public ResponseEntity<Object> handleResourceAlreadyExistsException(AlreadyExistsException ex) {
        return generateErrorResponse(ex.getMessage(), HttpStatus.UNPROCESSABLE_ENTITY);
    }

    @ExceptionHandler(BadRequestException.class)
    public ResponseEntity<Object> handleBadRequestException(BadRequestException ex) {
        return generateErrorResponse(ex.getMessage(), HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(InternalServerException.class)
    public ResponseEntity<Object> handleInternalServerException(InternalServerException ex) {
        return generateErrorResponse(ex.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }

    @ExceptionHandler(ParseException.class)
    public ResponseEntity<Object> handleParseException(ParseException ex) {
        return generateErrorResponse(ex.getMessage(), HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(UserDisabledException.class)
    public ResponseEntity<Object> handleUserDisabledException(UserDisabledException ex) {
        return generateErrorResponse(ex.getMessage(), HttpStatus.PRECONDITION_FAILED);
    }

    @ExceptionHandler(WrongCredentialsException.class)
    public ResponseEntity<Object> handleWrongCredentialsException(WrongCredentialsException ex) {
        return generateErrorResponse(ex.getMessage(), HttpStatus.UNAUTHORIZED);
    }

    @ExceptionHandler(MethodNotAllowedException.class)
    public ResponseEntity<Object> handleWrongCredentialsException(MethodNotAllowedException ex) {
        return generateErrorResponse(ex.getMessage(), HttpStatus.METHOD_NOT_ALLOWED);
    }
```

Εικόνα 32 - Back-end exception handler

4.5.6 Constraints

Στο entity αλλά και στα μοντέλα που χρησιμοποιούνται ως dto (data transfer objects) φαίνονται κάποια annotations, τα οποία προσθέτουν κάποιους περιορισμούς στις τιμές των πεδίων. Ένα core annotation που χρησιμοποιείται αρκετά συχνά είναι το @NotNull, το οποίο όπως είναι κατανοητό και από το όνομα, δεν θα επιτρέψει να δοθεί μία τιμή null στο πεδίο. Υπάρχουν όμως και custom annotations που έχουν φτιαχτεί για να επεκτείνουν τις δυνατότητες της εφαρμογής. Παρακάτω ακολουθεί το παράδειγμα για το password:

```java
@Documented
@Constraint(validatedBy = PasswordConstraintValidator.class)
@Target({ TYPE, FIELD, ANNOTATION_TYPE })
@Retention(RUNTIME)
public @interface ValidPassword {

    String message() default "Invalid Password";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

}
```

Εικόνα 33 - Back-end password constraint interface

```java
public class PasswordConstraintValidator implements ConstraintValidator<ValidPassword, String> {

    @Override
    public void initialize(ValidPassword arg0) {
    }

    @Override
    public boolean isValid(String password, ConstraintValidatorContext context) {
        PasswordValidator validator = new PasswordValidator(Arrays.asList(
                new LengthRule(8, 30),
                new UppercaseCharacterRule( num: 1),
                new DigitCharacterRule( num: 1),
                new SpecialCharacterRule( num: 1),
                new WhitespaceRule()));

        RuleResult result = validator.validate(new PasswordData(password));
        if (result.isValid()) {
            return true;
        }
        context.disableDefaultConstraintViolation();
        context.buildConstraintViolationWithTemplate(
                Joiner.on(",").join(validator.getMessages(result)))
                .addConstraintViolation();
        return false;
    }
}
```

Εικόνα 34 - Back-end password constraint validator

```java
public class AuthPayload {

    @RequiredField(field = "Email")
    @Email
    private String email;

    @RequiredField(field = "Password")
    @ValidPassword
    @Size()
    private String password;
```

Εικόνα 35 - Back-end constraint annotation example

4.6 Παραδείγματα Κώδικα σε Android

4.6.1 Fragment (Client Offers List View)

Το View και κατά επέκταση το Fragment, όπως έχει δηλώσει και ο Josh Smith, είναι τα ρούχα που φορούν τα δεδομένα. Συγκεντρώνει τα οπτικά στοιχεία της διεπαφής (UI components) και αναλαμβάνει να χτίσει το layout και να καλέσει το ViewModel για οποιοδήποτε event χρειαστεί, από την αναπαράσταση μιας λίστας όπως οι προσφορές έως την κλήση για άνοιγμα ενός άλλου fragment που θα αφορά μία συγκεκριμένη προσφορά όταν ο χρήστης πατήσει πάνω της.

Τα fragments λύνουν ένα χρόνιο πρόβλημα που υπήρχε στις εφαρμογές Android ως προς την μεταφορά πληροφορίας από οθόνη σε οθόνη και την δημιουργία πολλαπλών references. Αυτό γινόταν επειδή για κάθε οθόνη χρειαζόταν ένα νέο Activity το οποίο δεν μπορούσε να μεταφέρει άμεσα ένα reference type (π.χ. object) σε ένα άλλο Activity.

Στην περίπτωση της εφαρμογής αυτής, χρησιμοποιείται ένα μοναδικό Main Activity στο context του οποίου ανήκουν όλα τα Fragments της εφαρμογής έχοντας έτσι όλα πρόσβαση στο reference. Αυτό σε πιο γενικά πλαίσια μας δίνει Modularity και Reusability έχοντας μικρότερα ξεχωριστά κομμάτια του User Interface. Ακολουθεί screenshot από τον κώδικα ενός fragment:

```
@ExperimentalCoroutinesApi
@InternalCoroutinesApi
@AndroidEntryPoint
class ClientOffersFragment : Fragment(R.layout.fragment_client_offers) {


    private val viewModel: ClientOffersViewModel by viewModels()
    private val adapter = ClientOffersAdapter { viewModel.onEvent(it) }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        initViews()
        registerObservers()
        registerViewListeners()
    }

    private fun initViews() {
        offersList.layoutManager = LinearLayoutManager(activity)
    }

    private fun registerViewListeners() {
        retryClientOffersRertieval.setOnClickListener { it: View!
            viewModel.onEvent(ClientOffersContract.ClientOffersEvent.RetrieveOffers)
        }
    }


    private fun registerObservers() {
        viewLifecycleOwner.lifecycleScope.launch { this: CoroutineScope
            viewModel.state().collect { it: ClientOffersContract.ClientOffersState
                renderState(it)
            }
        }
        viewLifecycleOwner.lifecycleScope.launch { this: CoroutineScope
            viewModel.effect().collect { it: ClientOffersContract.ClientOffersViewEffect
                renderEffect(it)
            }
        }
    }

    private fun renderEffect(effect: ClientOffersContract.ClientOffersViewEffect) {
        when (effect) {
            is ClientOffersContract.ClientOffersViewEffect.NetworkError -> renderNetworkError()
            is ClientOffersContract.ClientOffersViewEffect.OffersRetrievalError -> renderError(effect.error)
            is ClientOffersContract.ClientOffersViewEffect.Loading -> renderLoading(effect.loading)
            is ClientOffersContract.ClientOffersViewEffect.NavigateToSelectedOffer -> navigateToSelectedOffer(effect.offer)
        }
    }

    private fun navigateToSelectedOffer(offer: Offer) {
        val directions = MainAppNavigationDirections.actionFromClientOffersToOffer(offer)
        val navController = Navigation.findNavController(this.requireActivity(), R.id.nav_host_fragment)
        navController.navigate(directions)
    }
```

Εικόνα 36 - App Fragment

4.6.2 ViewModel (Client Offers List ViewModel)

Το ViewModel είναι αυτό που θα δώσει σε ένα View όλα τα απαραίτητα δεδομένα και λειτουργικότητα. Ορίζει την δομή και την συμπεριφορά της εφαρμογής, χωρίς αυτό ένα View είναι απλά μια εικόνα και ένα ViewModel μόνο του δεν έχει τίποτα να δείξει στον χρήστη. Εδώ πρέπει να γίνει και μία διευκρίνιση καθώς τα ViewModels θεωρούνται domain, αλλά σαν graphical domain και όχι όπως π.χ. τα Use Cases στο domain layer τα οποία αφορούν το business logic της εφαρμογής.

Στην περίπτωση της συγκεκριμένης εφαρμογής όμως, η οποία χρησιμοποιεί State management, αναλαμβάνει και ένα επιπλέον κομμάτι, αυτό του Intent από την MVI αρχιτεκτονική και όχι το traditional Android Intent. Δηλαδή η πρόθεση του χρήστη στο View, π.χ. Login, δηλώνει το “intention” για είσοδο του χρήστη στην εφαρμογή, τότε το ViewModel λαμβάνει το event αυτό από το View ασύγχρονα (Android Coroutines Framework).

Έπειτα αναλύεται το event και μεταφράζεται σε business specs, π.χ. στην περίπτωση αυτή μία κλήση στο back-end με τα στοιχεία εισόδου του χρήστη. Μόλις ληφθεί η απάντηση, το ViewModel σηματοδοτεί την αλλαγή του state και προώθησής του νέου πλέον state στο View ώστε να κάνει render τα data. Επίσης διαχειρίζεται τα View Effects, δηλαδή one-time events που είναι εμφανή στο View, π.χ. animations, errors, navigation κτλ.

```
@HiltViewModel
class ClientOffersViewModel @Inject constructor(val clientOffersUsecase: ClientOffersUsecase) :
    CoreViewModel<
            ClientOffersContract.ClientOffersState,
            ClientOffersContract.ClientOffersEvent,
            ClientOffersContract.ClientOffersViewEffect
            >(
        ClientOffersContract.ClientOffersState.InitialEmptyState
    ) {

    init {
        viewModelScope.launch { handleEvent(ClientOffersContract.ClientOffersEvent.RetrieveOffers) }
    }

    override suspend fun handleEvent(event: ClientOffersContract.ClientOffersEvent) {
        when (event) {
            is ClientOffersContract.ClientOffersEvent.RetrieveOffers -> getClientOffers()
            is ClientOffersContract.ClientOffersEvent.OfferClicked -> navigateToOffer(event.offer)
        }
    }

    private fun navigateToOffer(offer: Offer) {
        emitEffect { ClientOffersContract.ClientOffersViewEffect.NavigateToSelectedOffer(offer) }
    }

    private fun getClientOffers() {
        viewModelScope.launch { this: CoroutineScope
            emitEffect { ClientOffersContract.ClientOffersViewEffect.Loading( loading: true) }
            when (val response = clientOffersUsecase()) {
                is ApiResult.Success<ClientOffers> -> emitState { this: ClientOffersContract.ClientOffersState
                    ClientOffersContract.ClientOffersState.ClientOffersListState(response.data)
                }
                is ApiResult.Error -> emitEffect { returnErrorState(response) }
            }
            emitEffect { ClientOffersContract.ClientOffersViewEffect.Loading( loading: false) }
        }

    }

    private fun returnErrorState(error: ApiResult.Error): ClientOffersContract.ClientOffersViewEffect =
        when (error) {
            is ShopaError -> ClientOffersContract.ClientOffersViewEffect.OffersRetrievalError(error)
            is MalformedContractError -> ClientOffersContract.ClientOffersViewEffect.NetworkError
            else -> ClientOffersContract.ClientOffersViewEffect.NetworkError
        }
}
```

Εικόνα 37 - App ViewModel

4.6.3 Contract (Client Offers List Contract)

Η επικοινωνία View - ViewModel είναι uni-directional, και το Contract χρησιμοποιείται για την διεκπεραίωση αυτής της επικοινωνίας. Η ροή επικοινωνίας από το View προς το ViewModel χρησιμοποιεί τα Events. Η αντίστροφη ροή από το ViewModel προς το View χρησιμοποιεί τα States και ViewEffects.

```
class ClientOffersContract {

    sealed class ClientOffersState : State {
        data class ClientOffersListState(val offers: ClientOffers) : ClientOffersState()
        data class NoAvailableOffersState(val message: String) : ClientOffersState()
        object InitialEmptyState : ClientOffersState()
    }

    sealed class ClientOffersEvent : Event {
        data class OfferClicked(val offer: Offer) : ClientOffersEvent()
        object RetrieveOffers : ClientOffersEvent()
    }

    sealed class ClientOffersViewEffect : ViewEffect {
        data class Loading(val loading: Boolean) : ClientOffersViewEffect()
        data class NavigateToSelectedOffer(val offer: Offer) : ClientOffersViewEffect()
        data class OffersRetrievalError(val error: ShopaError) : ClientOffersViewEffect()
        object NetworkError : ClientOffersViewEffect()

    }

}
```

Εικόνα 38 - App Contract

4.6.4 UseCase (Client Offers List UseCase)

Τα Use Cases είναι κλάσεις που ανήκουν στο domain level και αφορούν το business logic της εφαρμογής. Κάθε ένα χρησιμοποιείται για συγκεκριμένο flow λειτουργικότητας και θεωρείται ένα abstraction της «ζήτησης» διαφόρων πραγμάτων. Μέσα από αυτό κινούνται δεδομένα από και προς τα entities της εφαρμογής. Είναι επίσης αυτό που αποφασίζει ποιο data source θα χρησιμοποιηθεί για την εκάστοτε ζήτηση.

```
class ClientOffersUsecase @Inject constructor(private val repository: ClientOffersRepositoryContract) {
    suspend operator fun invoke(): ApiResult<ClientOffers> = repository.getClientOffers()
}
```

Εικόνα 39 - App UseCase

4.6.5 Model (Client Offer Model)

Πρόκειται για την μορφή των μοντέλων όπως τα αναγνωρίζει η εφαρμογή, αφού έχει λάβει τα data από ένα data source σε raw model και τα έχει κάνει convert.

```
data class ClientOffers(val offers:List<Offer>) {
}
```

Εικόνα 40 - App List Model

```
@Parcelize
data class Offer(
    val offerId: Int,
    val title: String,
    val description: String,
    val points: String,
    val status: Boolean,
    val shopName: String,
    val shopId: String
): Parcelable
```

Εικόνα 41 - App Model

4.6.6 DI (Dependency Injection) Module

Ο λόγος που χρησιμοποιείται αυτή η κλάση, είναι για να δηλώσει στο Usecase ποιο implementation του Repository αντιστοιχεί στο Contract του Repository. Τα annotations που χρησιμοποιούνται για την λειτουργία της κλάσης, προέρχονται από το Hilt library.

```
@InstallIn(ViewModelComponent::class)
@Module
class ClientOffersRepositoryModule {

    @Provides
    fun provideClientOffersRepo(apiProvider: ApiProvider): ClientOffersRepositoryContract {
        return ClientOffersRepository(apiProvider)
    }

}
```

Εικόνα 42 - App Dependency Injection Module

4.6.7 Repository Contract (Client Offers List Repository Contract)

Πρόκειται για ένα abstraction του Repository που ορίζει το τι χρειάζεται να πάρει σαν δεδομένα και τι μοντέλο ακολουθούν αυτά.

```
interface ClientOffersRepositoryContract {
    suspend fun getClientOffers(): ApiResult<ClientOffers>
}
```

Εικόνα 43 - App Repository Contract

4.6.8 Raw Model (Client Offers Raw Model)

Αφορά την μορφή των δεδομένων όπως λαμβάνονται από ένα data source. Δεν είναι η μορφή που τα αναγνωρίζει η εφαρμογή, καθώς γίνεται ένα conversion σε Model για χρήση.

```
data class ClientOffersRaw(val data: List<OffersRaw>) {
    fun toClientOffers(): ClientOffers = ClientOffers(data.map { it.toOffer() })
}

data class OffersRaw(
    val id: Int,
    val title: String,
    val description: String,
    val points: Float,
    val status: String,
    val createdAt: String,
    val shop: String,
    val shopId: String
) {

    fun toOffer() = Offer(
        offerId = id,
        title = title,
        description = description,
        points = points.toString(),
        status = toOfferStatus(),
        shopName = shop,
        shopId = shopId
    )
```

Εικόνα 44 - App Raw Model

4.6.9 Repository (Client Offers List Repository)

Το Repository χρησιμοποιείται για την άντληση δεδομένων από ένα data source (π.χ. api, τοπική βάση δεδομένων κτλ). Είναι αυτό που επιλέγει data source ανά περίπτωση.

```
class ClientOffersRepository @Inject constructor(private val apiProvider: ApiProvider) : ClientOffersRepositoryContract {

    override suspend fun getClientOffers(): ApiResult<ClientOffers> =
        apiProvider.api(OffersDataSource::class.java).getOffers().toResult().validateMap { it.toClientOffers() }
}
```

Εικόνα 45 - App Repository

4.6.10 Datasource (Client Offers DataSource)

Η πηγή άντλησης δεδομένων για την εφαρμογή. Μπορεί να υπάρχουν πολλές πηγές για ταυτόχρονη χρήση. Στην συγκεκριμένη περίπτωση χρησιμοποιείται μόνο το api.

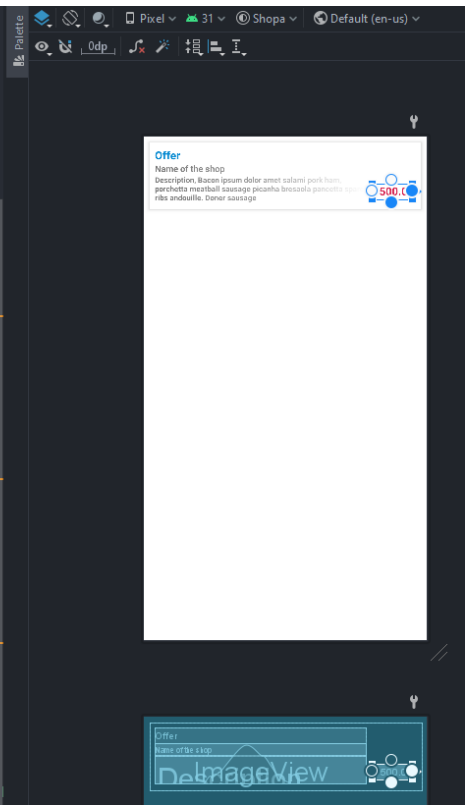
```
interface OffersDataSource {

    @GET( value: "offers/client")
    fun getOffers(): Call<ClientOffersRaw>
}
```

Εικόνα 46 - App DataSource

4.6.11 Layout (Client Offer Layout)

Πρόκειται για το template στο οποίο το View θα εμφανίσει τα δεδομένα που λαμβάνει από το ViewModel.

```
<TextView
    android:id="@+id/offerTitle"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:textColor="@color/light_blue_700"
    android:textSize="16dp"
    android:textStyle="bold"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toLeftOf="@+id/points"
    app:layout_constraintTop_toTopOf="parent"
    tools:text="Offer" />

<TextView
    android:id="@+id/shopName"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:lines="1"
    android:textSize="13dp"
    android:textStyle="bold"
    app:layout_constraintLeft_toLeftOf="@+id/offerTitle"
    app:layout_constraintRight_toRightOf="@+id/offerTitle"
    app:layout_constraintTop_toBottomOf="@+id/offerTitle"
    tools:text="Name of the shop" />

<TextView
    android:id="@+id/description"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:lines="3"
    android:textSize="11dp"
    android:textStyle="bold"
    app:layout_constraintLeft_toLeftOf="@+id/offerTitle"
    app:layout_constraintRight_toRightOf="@+id/offerTitle"
    app:layout_constraintTop_toBottomOf="@+id/shopName"
    tools:text="Description, Bacon ipsum dolor amet salami
```

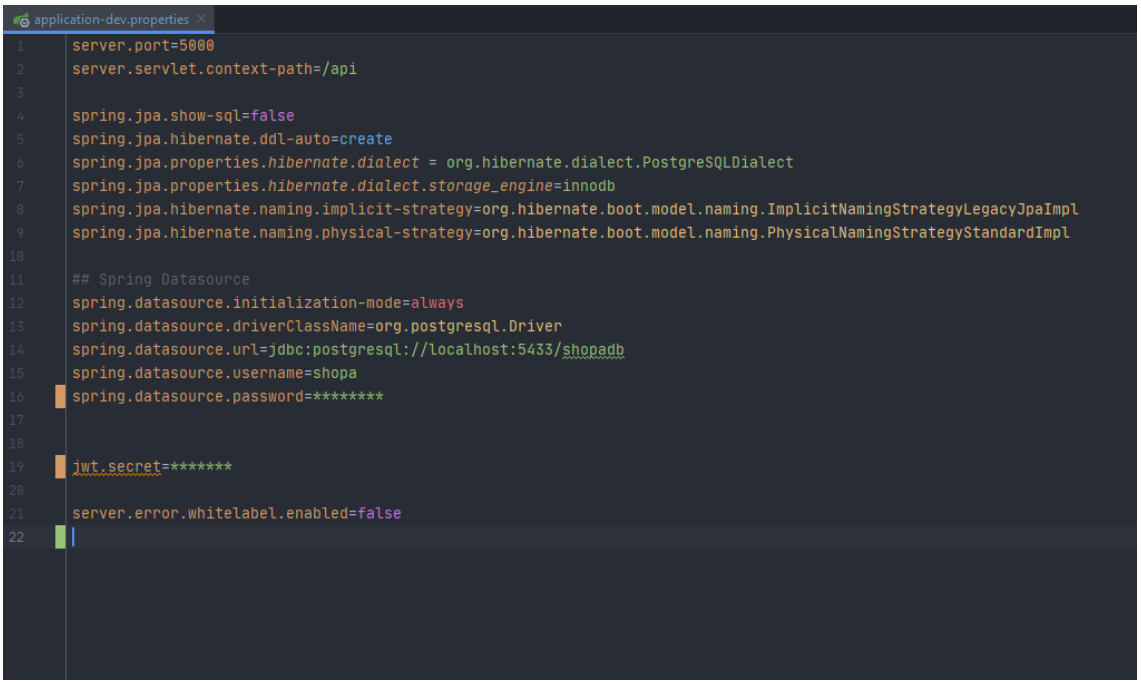


Εικόνα 47 - App Layout

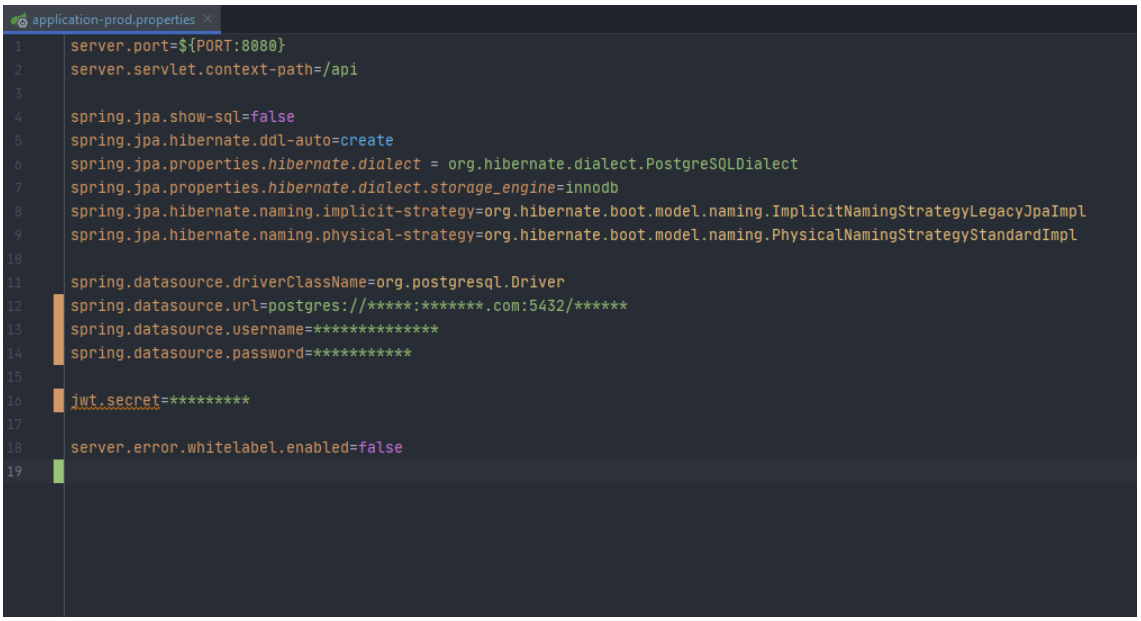
4.7 Ρυθμίσεις & Περιβάλλοντα

4.7.1 Περιβάλλοντα Ανάπτυξης στο Back-end

Στην ανάπτυξη λογισμικού απαιτούνται τουλάχιστον δύο περιβάλλοντα του κώδικα, το development και το production. Το κάθε ένα διαφοροποιείται σε βασικές ρυθμίσεις για το πως πρέπει να φερθεί ο κώδικας, π.χ. σε ποια βάση δεδομένων πρέπει να συνδεθεί, αν θα πρέπει να χτίζει νέα βάση κάθε φορά που γίνεται build ο κώδικα, το port που θα χρησιμοποιηθεί κτλ.



Εικόνα 48 - Back-end dev properties



Εικόνα 49 - Back-end production properties

4.7.2 Ρυθμίσεις ασφάλειας του Back-end (Security)

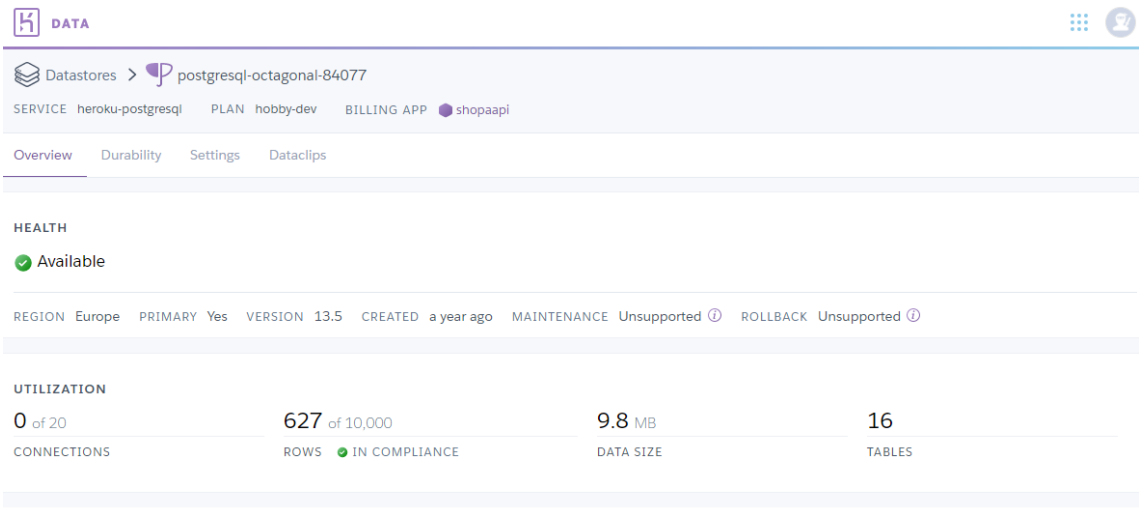
Κάποιες ρυθμίσεις βρίσκονται σε επίπεδο κώδικα όπως αυτές του security και κατ' επέκταση του access που μπορεί να έχει κάποιος στα endpoints χρησιμοποιώντας το jwt token, όπως αναφέρθηκε νωρίτερα. Στην εικόνα που ακολουθεί φαίνεται το αρχείο configuration που αφορά το security του api. Αρκετά βασικό θεωρείται να εξαιρεθούν κάποια endpoints από τον έλεγχο, όπως εκείνο του registration, γιατί πρέπει να είναι διαθέσιμα προς όλους χωρίς authentication.

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class ApplicationSecurityConfig extends WebSecurityConfigurerAdapter {

    private final JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

    private final ApplicationUserServiceImpl userDetailsService;

    private final JwtAuthenticationRequestFilter jwtAuthenticationRequestFilter;

    public ApplicationSecurityConfig(JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint, ApplicationUserServiceImpl
        this.jwtAuthenticationEntryPoint = jwtAuthenticationEntryPoint;
        this.userDetailsService = userDetailsService;
        this.jwtAuthenticationRequestFilter = jwtAuthenticationRequestFilter;
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    }

    @Bean
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        // We don't need CSRF for this example
        httpSecurity.cors().and().csrf().disable() HttpSecurity
                // dont authenticate this particular request
                .authorizeRequests() ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
                .antMatchers(AUTHENTICATE_URL).permitAll()
                .antMatchers(HttpMethod.POST, SIGN_UP_PROFILE_URL).permitAll()
                .antMatchers(HttpMethod.POST, SIGN_UP_SHOP_URL).permitAll()
                .antMatchers(HttpMethod.GET, CONFIRM_ACCOUNT).permitAll()
                .antMatchers(HttpMethod.GET, FORGOT_PASSWORD).permitAll()
                .antMatchers(HttpMethod.POST, RESET_PASSWORD).permitAll()
                .antMatchers(HttpMethod.GET, GET_CITIES).permitAll()
                .antMatchers(HttpMethod.GET, GET_CATEGORIES).permitAll()
                .antMatchers(HttpMethod.GET, GET_CATEGORIES).permitAll()
                .antMatchers(HttpMethod.GET, GET_CATEGORIES_BY_PARENT).permitAll()
                .antMatchers( ...antPatterns: "/v2/api-docs",
                        "/configuration/ui",
                        "/swagger-resources/**",
                        "/configuration/security",
                        "/swagger-ui/**",
                        "/webjars/**").permitAll()
                // all other requests need to be authenticated
                .anyRequest().authenticated()
```

Εικόνα 50 - Back-end security configuration

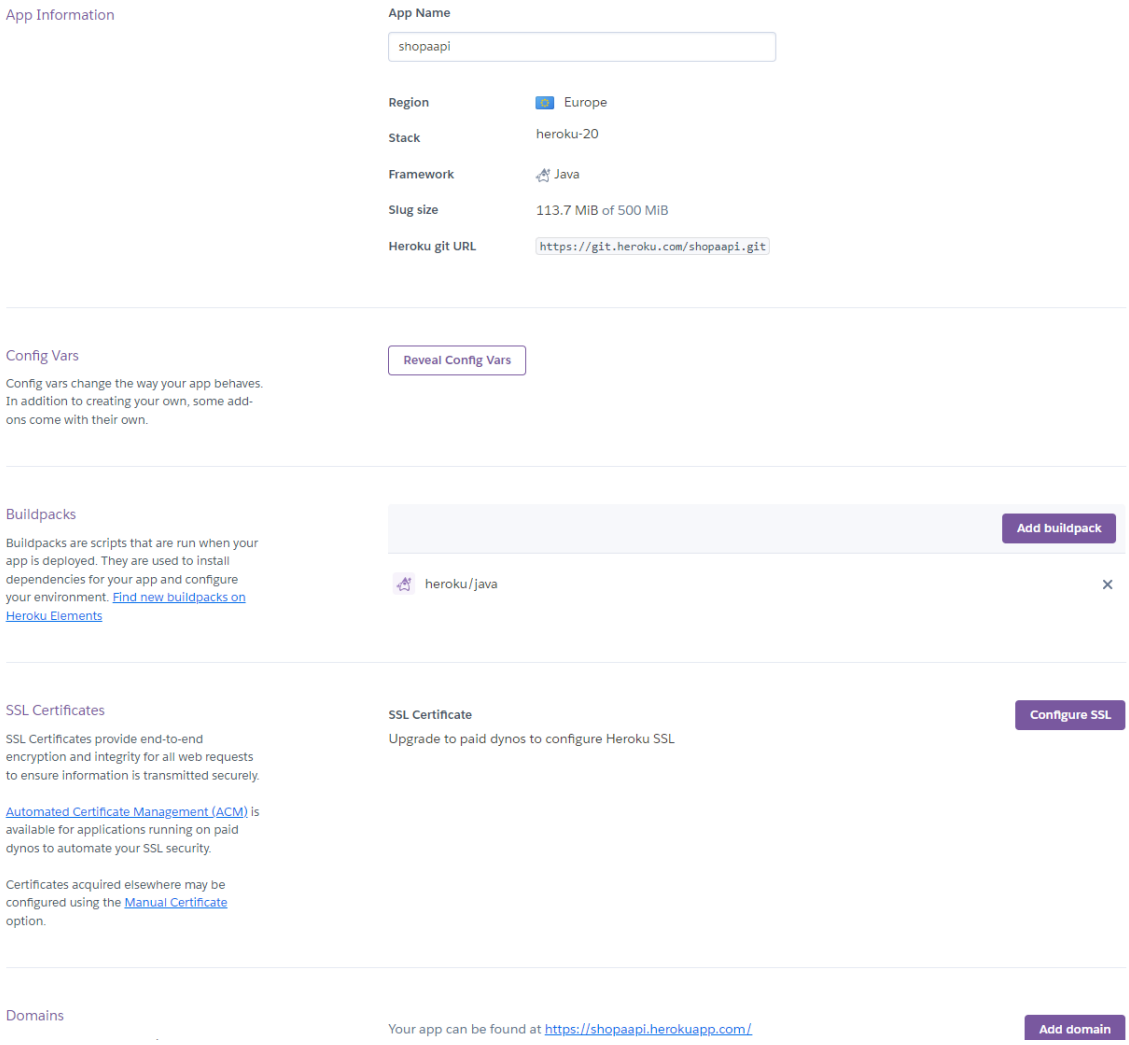
5. Deployment στο Heroku

Για την χρήση της εφαρμογής σε production environment, πρέπει να χρησιμοποιηθεί μία υπηρεσία PaaS (Platform as a Service) σε cloud και στην περίπτωση της συγκεκριμένης εφαρμογής επιλέχτηκε το Heroku. Αναπτύσσεται από το 2007 και αυτή την στιγμή υποστηρίζει τις Java, Node.js, Scala, Clojure, Python, PHP, Go και Ruby.
Για αρχή χρειάστηκε να δημιουργηθεί ένα Resource για την βάση δεδομένων σε Postgres όπως φαίνεται στην εικόνα παρακάτω:



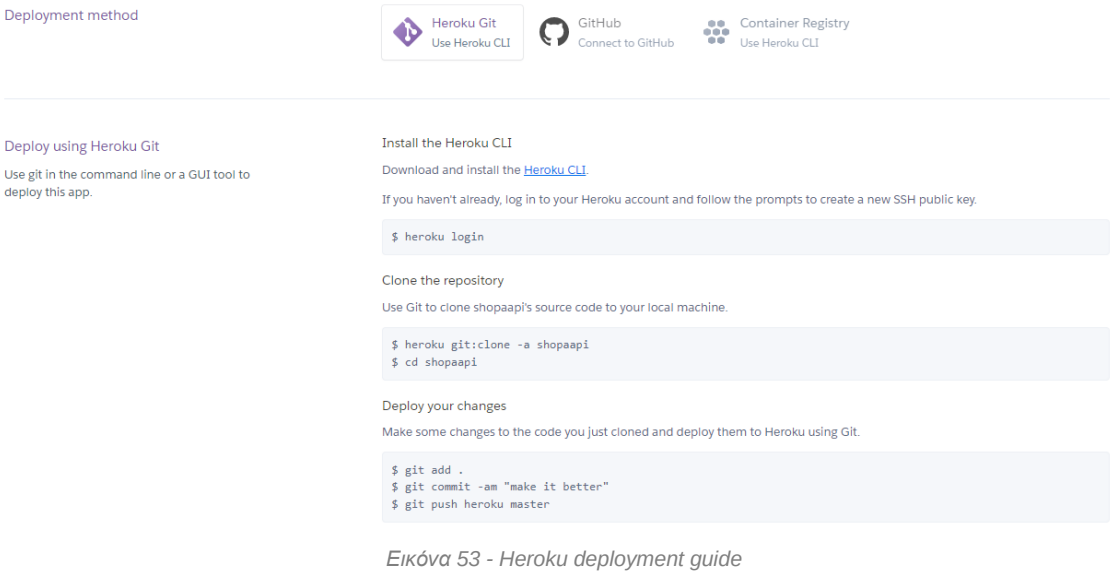
Εικόνα 51 - Heroku Postgres Resource

Έπειτα χρησιμοποιήθηκε ένα buildpack σε java ώστε να μπορεί να αναγνωρίζει σε τι γλώσσα είναι γραμμένο το api που κάνουμε deploy. Έχει δημιουργήσει ένα git resource στον server του Heroku και το χρησιμοποιούμε για να κάνουμε push ότι αλλαγές έχουμε στον κώδικα. Κάτω κάτω φαίνεται το domain στο οποίο μπορούμε να βρούμε και να χρησιμοποιήσουμε το api.



Εικόνα 52 - Heroku java buildpack

Τέλος μας δίνει έναν οδηγό για το πως μπορούμε να κάνουμε deploy κάθε φορά μέσω ενός terminal και εφόσον έχουμε εγκαταστήσει το Heroku CLI.



Εικόνα 53 - Heroku deployment guide

6. Μελλοντικές Επεκτάσεις

6.1 Επέκταση λειτουργιών στις προσφορές

Οι προσφορές θα μπορούσαν να αναπτυχθούν περαιτέρω, ειδικά ως προς την προσθήκη κατηγοριών, πράγμα που θα επέτρεπε και το φιλτράρισμά τους στην λίστα. Επίσης θα μπορούσε να προστεθεί στο προφίλ του χρήστη επιλογή κατηγοριών σύμφωνα με τα ενδιαφέροντά του, ώστε η λίστα να είναι πάντα προσαρμοσμένη σε αυτά και να μην βλέπει προσφορές σε κατηγορίες που πιθανώς να μην επιθυμεί. Τέλος, βάσει ιστορικού αγορών, θα μπορούσε να γίνει προώθηση προσφορών σε κατηγορίες και καταστήματα που προτιμά πιο συχνά.

6.2 GPS και χάρτες

Μία βασική επέκταση θα ήταν αυτή της προσθήκης χάρτη για κάθε κατάστημα και επιλογή ανοίγματος με κάποια τρίτη εφαρμογή GPS για κατεύθυνση του πελάτη προς αυτό. Μία άλλη επιλογή με βάση τον χάρτη θα ήταν η εμφάνιση προσφορών σε ακτίνα από την τρέχουσα τοποθεσία.

6.3 Επέκταση διαθέσιμων περιοχών

Αυτή την στιγμή δίνεται η δυνατότητα μόνο για μία επιλογή περιοχής στον καταναλωτή, αλλά αυτό θα μπορούσε να επεκταθεί σε δύο, πιθανώς και τρεις, καθώς είναι γνωστό πως δεν περνούμε τον χρόνο μας απαραίτητα σε περιοχές που μένουμε. Υπάρχουν και οι τόποι καταγωγής ή ακόμη και οι περιοχές στις οποίες εργαζόμαστε.

6.4 Προσθήκη φωτογραφιών

Θα βοηθούσε πολύ αν υπήρχαν φωτογραφίες των καταστημάτων αλλά πιθανώς και των προσφορών που φτιάχνονται για καλύτερη ενημέρωση των πελατών.

6.5 Ανάπτυξη συστήματος ανταμοιβής

Το σύστημα ανταμοιβής πόντων είναι πού απλό αυτή την στιγμή, υπολογίζοντας τους πόντους με βάση την τελική τιμή αγοράς επί τον αριθμό 10. Αυτό θα μπορούσε να γίνει πιο σύνθετο ανάλογα με επίπεδο ποσού, κατηγοριών, καταστήματος ή ακόμη και να υπάρξουν διαφορετικά συστήματα ανταμοιβής πέραν των πόντων.

7. Συμπεράσματα

Τα συστήματα ανταμοιβής πελατών έχουν δοκιμαστεί αρκετά χρόνια τώρα και είναι δεδομένη η αποτελεσματικότητά τους. Με τις κατάλληλες επεκτάσεις αλλά και την σωστή προώθηση των προβλημάτων που μπορεί να λύσει η συγκεκριμένη εφαρμογή, μπορεί να γίνει μία ελκυστική και οικονομική λύση για όλους, καταστήματα και πελάτες.

Σίγουρα δεν είναι εύκολο να ανατραπεί η τάση που κυριαρχεί στο εμπόριο αυτή την στιγμή, αλλά τα προβλήματα είναι εκεί και υπάρχει μία λύση προς την σωστή κατεύθυνση που δεν βάζει ως πρωταρχικό στόχο το κέρδος, αλλά τον άνθρωπο.

Εικόνες

Βιβλιογραφία και Πηγές

- Clean Architecture: A Craftsman's Guide to Software Structure and Design (Robert C. Martin Series)
- The Clean Coder: A Code of Conduct for Professional Programmers
- The Clean Architecture
  https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
- Learn Spring Boot by Baeldung
  https://www.baeldung.com/spring-boot
- Spring Boot Authorization Tutorial: Secure an API (Java)
  https://auth0.com/blog/spring-boot-authorization-tutorial-secure-an-api-java/
- Spring Security with JWT for REST API
  https://www.toptal.com/spring/spring-security-tutorial
- Build a Basic App with Spring Boot and JPA using PostgreSQL
  https://developer.okta.com/blog/2018/12/13/build-basic-app-spring-boot-jpa
- Spring Boot Tutorial | Full Course
  https://www.youtube.com/watch?v=9SGDpanrc8
- Spring Boot Tutorial | Spring Data JPA
  https://www.youtube.com/watch?v=8SGI_XS5OPw
- Spring Security Full Course
  https://www.youtube.com/watch?v=her_7pa0vrg

- Modern Android Architecture with MVI
  https://amsterdamstandard.com/en/post/modern-android-architecture-with-mvi-design-pattern

- MVI Architecture with Android
  https://medium.com/swlh/mvi-architecture-with-android-fcde123e3c4a

- Android Documentation for app developers
  https://developer.android.com/docs

- Get started with the Navigation Component
  https://developer.android.com/guide/navigation/navigation-getting-started?fbclid=IwAR2Dhz5rlWKc0xHK9bAwVhusWvZBKkIkMQ4boRveUZ9PuKpbwIE1TnbSlxc

- All about Hilt
  https://proandroiddev.com/all-about-hilt-a-dependency-injection-framework-869b9c2bcb09

- A pragmatic guide to Hilt with Kotlin
  https://medium.com/androiddevelopers/a-pragmatic-guide-to-hilt-with-kotlin-a76859c324a1