ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Σύστημα καταχώρησης αγγελιών ακινήτων»

Του Φοιτητή

Λάμαϊ Κριστέλ 154482

Επιβλέπων

Δρ. Κυριάκος Τσιακμάκης

Σεπτέμβριος 2025

Σύστημα καταχώρησης αγγελιών ακινήτων

Κωδικός: 21369

Φοιτητής: Λάμαϊ Κριστέλ

Εισηγητής: Δρ Κυριάκος Τσιακμάκης

Ημερομηνία ανάληψης Π.Ε. 15-10-2023

Ημερομηνία περάτωσης Π.Ε. 01-09-2025

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

Περίληψη

Η εργασία αυτή αφορά την ανάπτυξη μιας διαδικτυακής εφαρμογής δημοσίευσης ακινήτων. Αναλύεται μια ήδη υπάρχουσα εμπορική εφαρμογή που αφορά το αντικείμενο. Η εφαρμογή αποτελείται από δύο βασικούς κορμούς ανάπτυξης λογισμικού. Πρώτος η ανάπτυξη του API μας το οποίο λειτουργεί ως το BakEnd της εφαρμογής μας. Χρησιμοποιεί ως βασική τεχνολογία ανάπτυξης λογισμικού το .NET Framework. Υλοποιεί δύο βάσεις δεδομένων σε PostgreSQL με την βοήθεια του Entity Framework αλλά και του Redis. Ο δεύτερος είναι η Διεπαφή Χρήστη της ηλεκτρονικής μας εφαρμογής η αλλιώς το FrontEnd. Το FrontEnd ειναι ανεπτυγμενο σε Angular και υλοποιείται με την βοήθεια του Tailwind CSS και του Redux. Η εφαρμογή απαιτή στην επιχειρησιακή της λογική και την διαχείριση κάποιον συνδρομών σε κάποιους χρήστες. Για την εκπλήρωση αυτού του σκοπού έχει χρησιμοποιηθεί το Stripe. Για την δημοσίευση της εφαρμογής και την εκτέλεση της έχει χρησιμοποιηθεί το Docker. Όλες οι παραπάνω τεχνολογίες αναπτύσσονται με εξήγηση για το πως αλλά και για το γιατί χρησιμοποιήθηκαν. Επιπλέον περιέχεται και ο οδηγός χρήσης της εφαρμογής για κάθε χρήστη ξεχωριστά αλλά και αποσπάσματα κώδικα που βοηθούν τον αναγνώστη να καταλάβει την λογική υλοποίησης της εφαρμογής.

«Web Application for Real Estate Listings»

Abstract

This project concerns the development of a web application for real estate listings. An existing commercial application to this subject is analyzed, The application consists of two main software development components. The first is the development of the API, which serves as the back end of the application. It uses .NET Framework as its core development technology. It implements two databases in PostgreSQL with the help of Entity Framework, as well as Redis. The second component is the user interface of our web application, also known as the front end. The front end is developed in Angular and implemented using Tailwind CSS and Redux. The application also includes business logic for managing user subscriptions. Stripe is used to fulfill this functionality. For deploying and running the application is used Docker. All of the above technologies are presented with explanations of how and why they were used. Additionally, a user guide is included for each user type, align with code snippets that help the reader understand the implementation logic of the application.

Ευχαριστίες

Με βαθιά ευγνωμοσύνη θέλω να ευχαριστήσω τους γονείς μου για την στήριξη και τις αξίες που μου χάρισαν, καθώς και την γυναίκα μου για την υπομονή, την αγάπη και την αδιάκοπη ενθάρρυνση. Χωρίς αυτούς, η ολοκλήρωση αυτής της προσπάθειας δεν θα ήταν δυνατή.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εργασία

Η χρήση και δημιουργία διαδικτυακών εφαρμογών έχει αλλάξει ριζικά τον τρόπο με τον οποίο οι άνθρωποι αναζητούν τις πληροφορίες στο διαδίκτυο καθημερινά. Ένα τομέας που έχει επηρεαστεί πολύ είναι η εύρεση ακινήτων, όπου οι χρήστες επιθυμούν εύκολη και γρήγορη εύρεση των πληροφοριών που αναζητούν, δυνατότητες λεπτομερής αναζήτησης αλλά και ασφάλειας των συναλλαγών τους. Η πτυχιακή εργασία που παρουσιάζεται αφορά την ανάπτυξη μιας διαδικτυακής εφαρμογής για την εύρεση και δημοσίευση ακινήτων, η οποία στοχεύει να βελτιώσει την εμπειρία χρήστη, τόσο για ιδιώτες όσο και για επαγγελματίες. Η εφαρμογή παρέχει ένα ολοκληρωμένο περιβάλλον δημοσίευσης, αναζήτησης και διαχείρισης ακινήτων, ενώ ταυτόχρονα περιέχει και συνδρομητικές υπηρεσίες για μεσιτικά γραφεία.

1.2 Υλοποίηση

Η Υλοποίηση της εφαρμογής βασίστηκε σε σύγχρονες τεχνολογίες ανάπτυξης λογισμικού, εξασφαλίζοντας την λειτουργικότητα, την ασφάλεια και την επεκτασιμοτητα της. Στο επίπεδο του Backend χρησιμοποιήθηκε το .NET Core, σε συνδυασμό με την βάση δεδομένων PostgreSQL και για επεκτασιμότητα χρησιμοποιήθηκε και η βάση Redis ως βάση προσωρινής μνήμης. Η διαχείριση των βάσεων γίνεται με το Entity Framework. Για την υλοποίηση του Frontend χρησιμοποιήθηκε Angular, σε συνδυασμό με Tailwind CSS και Redux για καλύτερη οργάνωση της κατάστασης της εφαρμογής. Επιπλέον, ενσωματώθηκε το Stripe για τη διαχείριση συνδρομών. Επιπρόσθετα χρησιμοποιήθηκε το Docker ώστε η εφαρμογή να μπορεί να εκτελείται αξιόπιστα σε διαφορετικά περιβάλλοντα. Για την κατασκευή ακολουθήθηκε αρχιτεκτονική MVC, με χρήση JWT για την αυθεντικοποίηση και σαφή διαχωρισμό των ρόλων χρηστών. Στόχος ήταν η δημιουργία μιας ευέλικτης, αξιόπιστης και φιλικής προς τον χρήστη πλατφόρμας, έτοιμη να υποστηρίξει επεκτάσεις στο μέλλον.

Κεφάλαιο 2ο: Παρόμοια συστήματα

Στην αγορά κυκλοφορούν πολλά παρόμοια συστήματα με το δικό μας. Έχουν γίνει αρκετές προσπάθειες κατά καιρούς από διάφορους προγραμματιστές, επιχειρήσεις και εταιρείες για την υλοποίηση της βέλτιστης εφαρμογής εύρεσης ακινήτων. Πολλές από αυτές τις προσπάθειες είναι πετυχημένες, άλλες περισσότερο και άλλες λιγότερο.

Στην χώρα μας υπάρχουν πολλές ιστοσελίδες οι οποίες είναι δημοσιευμένες και χρησιμοποιούνται ευρέως. Παρακάτω θα αναλύσουμε την πιο διαδεδομένη ιστοσελίδα στην χώρα μας και αυτή που έχει εξαγοράσει άλλες μικρότερες παρόμοιες ιστοσελίδες στην χώρα.

2.1 Spitogatos

Η πιο ευρέως γνωστή ιστοσελίδα στην χώρα μας είναι ο spitogatos.gr. Ο spitogatos δημοσιεύθηκε τον Μάιο του 2006, αν και φαίνεται πως η διαδικασία υλοποίησης της πλατφόρμας ξεκίνησε το 2005 και χρειάστηκε περίπου 1 χρόνο για την ολοκλήρωση της πρώτης έκδοσης της πλατφόρμας. [1]

2.1.1 Αναβαθμίσεις

Ο spitogatos μετά την δημοσίευση της πρώτης έκδοσης τον Μάιο του 2006 έρχεται τα επόμενα χρόνια με σειρά αναβαθμίσεων για φτάσουμε στην σημερινή έκδοση που έχει εδραιωθεί στο χώρο. Οι τελευταίες και σημαντικότερες αναβαθμίσεις του στην ηλεκτρονική ιστοσελίδα είναι:

- 2019 Price Index (SPI): Δημιουργία του δείκτη τιμών ακινήτων, για να παρέχει μέσες τιμές ανά περιοχή για να ενημερώνεται καλύτερα ο χρήστης [2].
- 2022 Spitogatos Insights: Δημιουργήθηκε μια νέα μονάδα ευφυίας δεδομένων (data intelligence) για επιχειρηματικούς πελάτες (τράπεζες, συμβουλευτικές εταιρείες κ.α.). Αναλύσει την προσφορά / ζήτηση και του βασικούς δείκτες απόδοσης (KPI) δυναμικής αγοράς, βασισμένη σε big data αλλά και προηγμένη επεξεργασίας δεδομένων.

Τον Δεκέμβριο 2015 δημοσίευσε και την πρώτη του Εφαρμογή Κινητής Συσκευής (Mobile Application) για συσκευές Android [3], τον Ιανουάριο του 2016 δημοσίευσε την εφαρμογή συσκευές iOS [4]. Μέχρι σήμερα έχει καταφέρει να έχει περισσότερες από 500.000 εγκαταστήσεις σε συσκευές Android και iOS.

2.1.2 Τεχνολογίες υλοποίησης

Αν και είναι δύσκολο σε μια τέτοια μεγάλη εφαρμογή να βρούμε ποιες τεχνολογίες χρησιμοποιεί μια εφαρμογή αυτού του επιπέδου φαίνεται πως χρησιμοποιεί Vue.js για το FrontEnd της εφαρμογής και έναν συνδυασμό από PHP και Java για το Backend της. Χρησιμοποιεί επίσης για την διαχείριση της βάσης του κυρίως ElasticSearch μια μη σχεσιακή βάση δεδομένων. Επίσης φαίνεται να υπάρχουν πολλά προϊόντα της Amazon, όπως το S3 Bucket και το Amazon Cloudfont όποτε υποθέτουμε ότι ως εξυπηρετητή χρησιμοποιεί το AWS από την Amazon. [5]

Κεφάλαιο 3ο: Τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της διαδικτυακής εφαρμογής

Η κατασκευή της διαδικτυακής εφαρμογής βασίζεται στον συνδυασμού διαφόρων τεχνολογιών οι οποίες συνεργάζονται για να υλοποιήσουν τις βασικές λειτουργίες της εφαρμογής: περιβάλλον χρήστη, αυθεντικοποίηση και ταυτοποίηση, δημοσίευση ακινήτων, αγορά συνδρομής.

3.1 Angular

Για την υλοποίηση της διεπαφής χρήστη της διαδικτυακής εφαρμογής χρησιμοποιήθηκε Angular. Το Angular είναι ένα πλήρες Framework το οποίο χρησιμοποιείται για την ανάπτυξη διαδικτυακών εφαρμογών επιχειρησιακού επιπέδου. Είναι φτιαγμένο από την Google και προσφέρει ένα ολοκληρωμένο πακέτο για τη δημιουργία μονοσέλιδων εφαρμογών (SPA, single-page applications) και βαθμιαίες διαδικτυακές εφαρμογές (PWA, progressive web applications). Οι βασικοί λόγοι της επιλογής της συγκεκριμένης τεχνολογίας είναι: [6][7]

1. Είναι ένα πλήρες Framework και όχι μια απλή βιβλιοθήκη
   - Δεν χρειάζονται εξωτερικές βιβλιοθήκες για την διαχείριση της δρομολόγησης των σελίδων, φόρμες, HTTP αιτήματα ή διαχείριση καταστάσεων
   - Έχει καθαρή αρχιτεκτονική MVC / MVVM με αποτέλεσμα να κρατάει τον κώδικα οργανωμένο.
2. Typescript ανάπτυξη κώδικα
   - Η ανάπτυξη γίνεται σε Typescript η οποία προσφέρει αυστηρή τυποποίηση, καλύτερη αποσφαλμάτωση και ευκολότερη συντήρηση ειδικά όταν χρησιμοποιείτε σε μεγάλες ομάδες προγραμματισμού.
3. Διπλή κατεύθυνση δέσμευσης δεδομένων (Two-way data binding)
   - Η ενημέρωση της Διεπαφής Χρήστη (UI, User Interface) και του μοντέλου γίνεται αυτόματα.
4. Δομική ανάπτυξη
   - Οι εφαρμογές χωρίζονται σε ενότητες (modules) και μέρη (components), με αποτέλεσμα να διευκολύνεται η επεκτασιμότητα και η επαναχρησιμοποίηση του κώδικα.
5. Απόδοση
   - Το Angular υποστηρίζει εκ των προτέρων σύνταξη (AOT, Ahead-of-Time compilation) και Αργή Φόρτιση (Lazy Loading), βελτιώνοντας την ταχύτητα και μειώνοντας το αρχικό μέγεθος δέσμης (bundle size) φόρτωσης της εφαρμογής.
6. Σταθερότητα και Υποστήριξη
   - Υποστηρίζεται από τη Google και χρησιμοποιείται σε μεγάλα επιχειρησιακά έργα.
   - Μακροχρόνια υποστήριξη (LTS) για κάθε μεγάλη έκδοση.

7. Κοινότητα και Οικοσύστημα

   - Έχει πολύ μεγάλη κοινότητα με πολλά έτοιμα μέρη (components), βιβλιοθήκες και σεμινάρια.

Λόγοι οι οποίοι θα οδηγούσαν να μην χρησιμοποιηθεί αυτή η τεχνολογία θα ήταν αν θα έπρεπε να φτιαχτεί μια μικρή εφαρμογή με ένα γρήγορο τεχνολογικό πρότυπο ή αν θα προτιμούσαν λιγότερη σταθερότητα και περισσότερη ευελιξία, που αυτό συνήθως συμβαίνει σε μικρότερες και όχι τόσο απαιτητικές εφαρμογές.

3.1.1 Tailwind CSS

Το Tailwind CSS είναι ένα CSS Framework που μας επιτρέπει να δημιουργούμε γρήγορα και αποδοτικά διεπαφές χρήστη οι οποίες είναι ευέλικτες και προσαρμόζονται αυτόματα, χωρίς να χρειάζεται να αναπτύσσουμε ειδικό κώδικα CSS από την αρχή. Επιλέχθηκε η τεχνολογία γιατί: [8]

1. Ταχύτητα ανάπτυξης
   Διαθέτει έτοιμες κλάσεις (πχ. flex, p-4, bg-blue-500) που μας επιτρέπουν να αναπτύσσουμε πολύ πιο γρήγορα μορφοποιημένα HTML σχέδια.
2. Συνέπεια στην σχεδίαση
   Οι ίδιες κλάσεις χρησιμοποιούνται παντού, με αποτέλεσμα να έχουμε ένα ομοιόμορφο και συνεχές UI.
3. Προσαρμόζεται εύκολα
   Μέσω του tailwind.config.js μπορούμε να ορίσουμε χρώματα, γραμματοσειρές, διαστήματα και σημεία διακοπής με βάση το δικό μας σύστημα σχεδίασης.
4. Ενσωματωμένη responsive σχεδίαση
   Έχει αίτημα σημεία διακοπής (sm:, md:, lg:, xl:) για ανάπτυξης σε περιβάλλοντα κινητού.
5. Μικρότερο CSS αρχείο
   Με το PurgeCSS που περιλαμβάνει, αφαιρούνται οι κλάσεις που δεν χρησιμοποιούνται
6. Ευρεία κοινότητα
   Έχει μεγάλη κοινότητα και πλούσια οικοσυστήματα από διάφορα πρόσθετα (φόρμες, τυπογραφία, animations).

3.2.2 Redux

Το Redux είναι μια βιβλιοθήκη διαχείρισης κατάστασης (state management). Χρησιμοποιείται κυρίως από μεγάλες εφαρμογές όπου η διαχείριση της κατάστασης είναι δύσκολη. Οι λόγοι που χρησιμοποιείται είναι: [9]

1. Κεντρική διαχείριση κατάστασης
   Η κατάσταση της εφαρμογής αποθηκεύεται σε ένα κεντρικό σημείο
2. Σταθερότητα
   Η κατάσταση μπορεί να αλλάξει μόνο από actions και reducers, με αποτέλεσμα να έχουμε πιο προβλέψιμες αλλαγές κατάστασης.
3. Ευκολότερη αποσφαλμάτωση
   Χρησιμοποιώντας το εργαλείο Redux DevTools, μπορούμε να παρακολουθούμε κάθε αλλαγή κατάστασης αλλα και επιστροφή σε προηγούμενη κατάσταση.

4. Οργάνωση κώδικα
   Είναι ιδανικό για μεγάλες εφαρμογές με σύνθετη λογική καθώς διαχωρίζει το UI από τη λογική των δεδομένων.
5. Προβλεπόμενα αποτελέσματα
   Οι reducers είναι απλές συναρτήσεις, αυτό σημαίνει ότι κάθε ίδια ενέργεια έχει το ίδιο αποτέλεσμα
6. Μεγάλη κοινότητα και υποστήριξη
   Παρέχει ένα τεράστιο οικοσύστημα από διαμεσολαβητές για να έχουμε ασύγχρονες λειτουργίες.

Σε μια μικρότερη εφαρμογή δεν θα χρησιμοποιούσαμε το Redux, αλλά σε εφαρμογές αυτού του επιπέδου που η διαχείριση κατάστασης ειναι σημαντικη χρησιμοποιείται καθώς προσφέρει δομή και σταθερότητα.

3.2 ASP.NET Core ή .NET Core

Για την υλοποίηση του Backend της διαδικτυακής εφαρμογής χρησιμοποιήθηκε .NET Core το οποίο είναι ένα γρήγορο και πολλαπλών πλατφορμών framework της Microsoft το οποίο χρησιμοποιείται για την ανάπτυξη διαδικτυακών εφαρμογών και APIs. Είναι σχεδιασμένο να αποφέρει υψηλή απόδοση, ασφάλεια και υποδομές σύννεφου (cloud-native). Οι βασικοί λόγοι που οδήγησαν στην επιλογή της συγκεκριμένης τεχνολογίας είναι:[10][11]

1. Υψηλή απόδοση και ταχύτητα

   - Το .NET Core είναι από τα πιο γρήγορα διαδικτυακά frameworks (κορυφαίο στις λίστες μετρήσεων απόδοσης των Frameworks όπως το TechEmpower Benchmarks).

   - Υποστηρίζει ασύγχρονο προγραμματισμό και έχει βελτιστοποιηθεί για μεγάλη κίνηση.
2. Χρήση σε πολλές πλατφόρμες

   - Το Framework δεν περιορίζεται μόνο σε λειτουργικά συστήματα Windows, παρόλο που είναι προϊόν της Microsoft μπορεί να τρέξει σε Linux, macOS, Windows αλλά και σε Docker Containers.
3. Ασφάλεια επιχειρησιακού επιπέδου

   - Παρέχει ενσωματωμένη διαχείριση ταυτοποίησης (AuthN) και αυθεντικοποίησης (AuthZ) χρήστη.
4. Δομημένη ανάπτυξη & ευκολότερο scaling

   - Παρέχει μια δομημένη αρχιτεκτονική και έχει ενσωματωμένη Έγχυση Εξάρτησης Υπηρεσιών (Dependency Injection) ώστε να προσφέρει καλύτερη διαχείριση μεγάλων έργων.

   – Υποστηρίζει μικροϋπηρεσίες (microservices) και ανάπτυξη εφαρμογών σε επίπεδο σύννεφου (cloud-native apps).
5. Παρέχει ένα τεράστιο οικοσύστημα και υποστήριξη από την Microsoft

6. Επικοινωνία και χρήση βάσεων δεδομένων και APIs

   - Παρέχει άριστη συνεργασία με το Entity Framework Core για την διαχείριση σχεσιακών βάσεων δεδομένων

   - Εύκολη και πολύ γρήγορη δημιουργία RESTFul αλλά και GraphQL APIs.

7. Είναι ιδανικό για εφαρμογές επιχειρήσεων

   - Χρησιμοποιείται από κολοσσούς εταιρείες (π.χ. StackOverflow, GoDaddy, UPS), αποδεικνύοντας ότι είναι κατάλληλο για εφαρμογές με υψηλές απαιτήσεις και μεγάλης κλίμακας.

Δεν θα ήταν ιδανική η χρήση της συγκεκριμένης τεχνολογίας αν θα έχουμε να δημιουργήσουμε μια εξαιρετικά μικρή και απλή εφαρμογή ή αν προτιμούνται τεχνολογίες ανοιχτού κώδικα με υψηλή ευελιξία όπως η Node.js.

3.3 PostgreSql

Για την υλοποίηση της βάσης δεδομένων χρησιμοποιήθηκε η PostgreSQL, καθώς είναι μια από τις πιο διαδεδομένες και χρησιμοποιημένες ανοιχτού κώδικα βάσεις δεδομένων. Οι λόγοι χρήσης της συγκεκριμένης βάσης δεδομένων είναι:[12], [13]

1. Ισχυρή και αξιόπιστη βάση

   - Το PostgreSQL είναι πλήρως ACID compliant, διασφαλίζοντας μας την αξιοπιστία και την ακεραιότητα των δεδομένων.

   - Χρησιμοποιείται ευρέως σε περιβάλλοντα μεγάλων επιχειρήσεων και θεωρείται μία “έτοιμη-για-παραγωγή (production-ready) βάση για εφαρμογές μεγάλης κλίμακας.

2. Παρέχει προχωρημένα χαρακτηριστικά

   - Υποστηρίζει JSON & JSONB δεδομένα, τα οποία είναι ιδανικά για εφαρμογές που συνδυάζουν σχεσιακά και μη δεδομένα.

   - Παρέχει δυνατότητες όπως CTEs, Window Functions, Full-text search, δυνατότητες που δεν παρέχονται από άλλες δωρεάν βάσεις δεδομένων.

   - Παρέχει επιπλέων ενσωματωμένη υποστήριξη γεωχωρικών δεδομένων τα οποία χρησιμοποιούνται για GIS Apps (PostGIS).

3. Απόδοση & Κλιμάκωση

   - Παρέχει κλιμάκωση (Replication) και διαμερισμό (Partitioning) σε μεγάλα σύνολα δεδομένων (datasets).

   - Για βελτίωση της ταχύτητας μπορεί να τρέξει παράλληλα και πολύπλοκα ερωτήματα.

4. Επεκτασιμότητα & Παραμετροποίηση

   - Παρέχεται η δυνατότητα δημιουργίας συναρτήσεων, τύπων δεδομένων αλλά και

επεκτάσεων.

- Παρέχει και πολλές έτοιμες επεκτάσεις (π.χ. αναλύσεις, δεδομένα χρονοσειράς).

5. Είναι ανοιχτού κώδικά και δωρεάν
   - Δεν έχει άδειες χρήσης όπως για παράδειγμα η Oracle ή το MS SQL.
   - Έχει μία τεράστια κοινότητα, παρέχει συχνές ενημερώσεις και αξιόπιστη υποστήριξη.
6. Συμβατότητα
   - Είναι συμβατή με τεχνολογίες όπως .NET, Node.js, Python, Java, PHP και σχεδόν όλα τα framework που χρησιμοποιούνται στις μέρες μας.

Η PostgreSQL αν και παρέχει όλα τα παραπάνω και είναι μια άριστη δωρεάν επιλογή για κάθε εφαρμογή δεν θα ήταν ιδανικό να χρησιμοποιηθεί σε κάποια εφαρμογή με πάρα πολύ απλές ανάγκες, όπου σε αυτήν την περίπτωση θα μπορούσε να χρησιμοποιηθεί MySQL ή SQLite ή σε περίπτωση που χρειάζονται κάποια ειδικά επιχειρησιακά χαρακτηριστικά της Microsoft, τότε το MS SQL Server μπορεί να είναι πιο ιδανικό.

3.3.1 Entity Framework

Το Entity Framework (EF / EF Core) είναι ένας σχεσιακό σύστημα αντικειμένων (ORM, Object-Relational Mapper) που έχει αναπτυχθεί από τη Microsoft. Αναπτύχθηκε για .NET εφαρμογές ώστε να μας επιτρέπει να δουλεύουμε με βάσεις δεδομένων χρησιμοποιώντας αντικείμενα (C# objects) χωρίς να χρειάζεται να γράφουμε συνεχώς SQL ερωτήματα. Το EF μας προσφέρει: [14][15]

1. Ταχύτερη ανάπτυξη
   Με το EF μπορούμε να διαχειριστούμε τα δεδομένα της βάσης μας χρησιμοποιώντας απλό C# κώδικα (LINQ queries) χωρίς να χρειάζεται να αναπτύσσουμε πολύπλοκα SQL ερωτήματα.
2. Διάλεκτος SQL
   Δεν χρειάζεται να ανησυχούμε για την διαφορετικές διαλέκτους SQL που χρησιμοποιεί κάθε βάση. Το EF αναλαμβάνει τη μετάφραση για εμάς.
3. LINQ ερωτήματα (queries)
   Τα LINQ ερωτήματα είναι ερωτήματα σε C# (πχ. db.Users.Where(u => u.IsActive) με πλήρη υποστήριξη και έλεγχο.
4. Migrations
   Εύκολη διαχείριση αλλαγών στη βάση με migration κώδικα εύκολα μέσα από το .NET CLI.

Το EF είναι ιδανικό για .NET Διαδικτυακές εφαρμογές, γιατί απλοποιεί τη διαχείριση της βάσης, μειώνει τον κώδικα και εξασφαλίζει τη συμβατότητα με πολλές βάσεις δεδομένων.

3.4 Redis

Για την αποφόρτιση του συστήματος μας αλλά και για την βελτίωση της απόδοσης του δίπλα στην κεντρική βάση δεδομένων υλοποιήσαμε το Redis. Το Redis είναι μία μέσα στην μνήμη (in-memory)

βάση δεδομένων ή ένα σύστημα προσωρινής αποθήκευσης (caching) που χρησιμοποιείται ευρέως. Τα οφέλει του σε μία διαδικτυακή εφαρμογή είναι: [16][17]

1. Ταχύτητα

   - Όλα τα δεδομένα αποθηκεύονται στην RAM του μηχανήματος που φιλοξενεί την εφαρμογή, με αποτέλεσμα να παρέχει ταχύτατη ανάγνωση και εγγραφή δεδομένων στην μνήμη εξυπηρετόντας χιλιάδες εκατομμύρια αιτήματα ανά δευτερόλεπτο.
   - Είναι ιδανικό για "ζωντανές (real-time)" εφαρμογές (π.χ. τσατς, ειδοποιήσεις).

2. Σύστημα προσωρινής αποθήκευσης (caching)

   - Μπορεί να αποθηκεύει αποτελέσματα ερωτημάτων (queries), προσωρινά δεδομένα, δεδομένα συνεδριών (session data) ώστε να μειώνεται η φόρτιση στη βασική μας βάση (PostgreSQL).
   - Αυτό έχει ως αποτέλεσμα την σημαντική μείωση του χρόνου απόκρισης της εφαρμογής.

3. Διάφορες δομές δεδομένων

   - Το Redis δεν υποστηρίζει απλά αποθήκευση δεδομένων κλειδιού-περιεχομένου (key-value), αποθηκεύει λίστες (lists), σετ (sets), ταξινομημένα σετ (sorted sets), hashes, ρεύματα (streams) κατάλληλα να καλύψουν κάθε ανάγκη.

4. Κλιμάκωση και διαθεσιμότητα

   - Υποστηρίζει αναπαραγωγή (replication), ομαδοποίηση (clustering), επιμονή (persistence), στιγμιότυπα (snapshooting) και AOF άρα είναι κατάλληλο για χρήση σε παραγωγικό περιβάλλον (production environment).

5. Ενσωμάτωση

   - Παρέχει πάρα πολύ καλή και εύκολη ενσωμάτωση με όλα τα διαδικτυακά Frameworks (π.χ. .NET, Node.js, Python, Java, PHP) και έτοιμα ενδιάμεσα λογισμικά (middleware) για να περιορισμούς ποσοστών χρήσης (rate limiting).

3.5 Stripe

Καθώς η εφαρμογή απαιτεί από τους χρήστες που συνδέονται ως μεσιτικά γραφεία την εγγραφή τους σε μία συνδρομή υπήρχε η ανάγκη για την υλοποίηση της διαχείρισης της συνδρομητικής υπηρεσίας. Για τους λόγους που αναφέρονται παρακάτω επιλέχθηκε για αυτήν την δουλειά η διασύνδεση της εφαρμογής μας με τη Stripe, καθώς είναι μια από τις πιο διαδεδομένες λύσεις παγκοσμίως. [18]

1. Γρήγορη και εύκολη ενσωμάτωση

   - Διαθέτει έτοιμα Κιτ Ανάπτυξης Λογισμικού (SDK) και APIs, για όλα τα δημοφιλή Frameworks.

- Σε μόνο λίγες γραμμές κώδικα μπορεί να στηθεί μια σελίδα ταμείου ή ένα σύστημα διαχείρισης συνδρομών.

2. Ασφάλεια και συμμόρφωση

   - Η Stripe είναι συμμορφωμένη και πιστοποιημένη με τα πρότυπα PCI DSS, με αποτέλεσμα να αναλαμβάνει την ασφαλή διαχείριση των καρτών αλλά και των συναλλαγών.

   - Παρέχει έτοιμα εργαλεία για καταπολέμηση της απάτης των συναλλαγών αλλά και την προστασία από κακόβουλες συναλλαγές.

3. Υποστήριξη συνδρομών και επαναλαμβανόμενων πληρωμών

   - Ενσωματώνει την ολοκληρωμένη διαχείριση των συνδρομών.

   - Υλοποίηση Webhooks για την αυτόματη ενημέρωση του συστήματος για ανανεώσεις αλλά και ακυρώσεις των συνδρομών.

4. Δυνατότητα πολλαπλών μέσων πληρωμής

   - Δίνει την δυνατότητα στους χρήστες να πληρώσουν με κάθε δυνατό τρόπο πληρωμής (π.χ. πιστωτικές / χρεωστικές κάρτες, Apple Pay, Google Pay, SEPA κ.α.).

5. Υποστήριξη σε πάρα πολλές χώρες χώρες αλλά και διαφορετικά νομίσματα
6. Ενσωμάτωση πινάκων ελέγχου για παρακολούθηση στατιστικών

3.6 Docker

Για να τρέχει η εφαρμογή μας τοπικά χρησιμοποιήθηκε το Docker. Το docker είναι ένα εργαλείο “containerization”, δηλαδή να τρέχεις την εφαρμογή σου απομονωμένα, σε ένας ασφαλές και ελαφρύ περιβάλλον το οποίο περιλαμβάνει ότι χρειάζεται για να εκτελέσει μια εφαρμογή. Είναι καλό να χρησιμοποιείτε το Docker γιατι παρέχεται: [19]

1. Σταθερότητα στα περιβάλλοντα
   - Με απλά λόγια “αν δουλεύει στον υπολογιστή μου” θα δουλεύει και σε οποιονδήποτε άλλο υπολογιστή ή εξυπηρετητή (server).
   - Όλο το περιβάλλον που εκτελεί την εφαρμογή θα είναι ίδιο παντού.
2. Ευκολότερη ανάπτυξη & Συνεχής παράδοση (CI/CD)
   - Γρήγορη δημιουργία και διανομή έτοιμων “εικόνων δίσκου” για εγκατάσταση και εκκίνηση του λογισμικού στον εξυπηρετητή.
   - Ιδανικό για DevOps αγωγών (DevOps Pipelines) και αυτόματη εγκατάσταση και εκκίνηση λογισμικού.
3. Απομόνωση λογισμικού και ασφάλεια
   - Κάθε εφαρμογή τρέχει σε δικό της κοντέινερ στον εξυπηρετητή με αποτέλεσμα να μην επηρεάζει καμία άλλη εφαρμογή στον ίδιο εξυπηρετητή.
4. Κλιμάκωση και διαχείριση φορτίου
   - Προσφέρει εργαλεία όπως Docker Swarm ή Kubernetes μπορούμε να αυξομειώνουμε τα κοντέινερς ανάλογα με την κίνηση της εφαρμογής (scaling).

5. Ελαφριά και γρήγορα περιβάλλοντα
   - Τα κοντέινερ είναι πιο γρήγορα και ελαφριά από τις παραδοσιακές εικονικές μηχανές (VM, Virtual Machines).
6. Ευκολία κατά την δοκιμή και την ανατροπή (rollback)
   - Μπορούν με ενεργοποιήσουμε προσωρινά περιβάλλοντα για δοκιμή ή να κάνουμε επανεγκατάση εύκολα σε προηγούμενη έκδοση.

3.7 Visual Studio Code

Για την ανάπτυξη του κώδικα χρησιμοποιήθηκε το Visual Studio Code (VS Code) το οποίο είναι ένα από τα πιο δημοφιλή εργαλεία για την ανάπτυξη κώδικα και όχι τυχαία καθώς παρέχει: [20]

1. Υποστήριξη για πολλές γλώσσες & frameworks
   - HTML, CSS, JavaScript / TypeScript, C#, Python, PHP, Java, ακόμα frameworks όπως Angular, React, Vue, .NET κ.α.
   - Παρέχει επίσης επεκτάσεις για Node.js, Docker, Git, Βάσεις Δεδομένων κ.α.
2. Ελαφρύ και γρήγορο
   - Είναι αρκετά ελαφρύ ώστε να ανοίγει γρήγορα και να λειτουργεί ομαλά σε πολύ μεγάλα έργα, σε αντίθεση με άλλα αντίστοιχα εργαλεία όπως το Visual Studio ή το Eclipse που θέλουν χρόνο να ανοίξουν και να συγχρονιστούν.
3. Πλούσιο οικοσύστημα επεκτάσεων
   - Η αγορά (marketplace) του περιέχει χιλιάδες πρόσθετα για οποιαδήποτε λειτουργία, χρήση, προβολή η βοήθεια χρειάζεται ένα μεγάλο έργο.
4. Ενσωματωμένο Git και DevOps εργαλεία
   - Διαχείριση του Git απευθείας από το VS Code.
   - Σύνδεση με CI / CD αγωγούς
5. Αποσφαλμάτωση και ζωντανή προεπισκόπηση
   - Έχει ενσωματωμένο μηχανισμό για αποσφαλμάτωση και ζωντανό εξυπηρετητη (Live Server) για να βλέπουμε άμεσα τις αλλαγές που κάνουμε στην εφαρμογή μας.
6. Διαθέσιμο δωρεάν και σε όλες τις πλατφόρμες

3.8 Git και Github

Το Git και το Github είναι ένας βασικός συνδυασμός κατά την γραφή κώδικα για έλεγχο εκδόσεων και συνεργατική ανάπτυξη λογισμικού, ειδικά για διαδικτυακές εφαρμογές. [21]

3.8.1 Git

Το Git είναι ένα σύστημα ελέγχου εκδόσεων (version control system) που χρησιμοποιείται για να διαχειρίζεται τον πηγαίο κώδικα ενός έργου και παρέχει:

1. Ιστορικό αλλαγών (versioning)
   - Καταγράφονται και αποθηκεύονται όλες οι εκδόσεις του κώδικα, με δυνατότητα επιστροφής σε παλαιότερη έκδοση.
2. Ασφάλεια και πειραματισμός
   - Με τα branches μπορούμε να δοκιμάσουμε νέα χαρακτηριστικά (features) χωρίς να "χαλάμε" τον βασικό κώδικα της εφαρμογής μας.
3. Συνεργασία
   - Πολλοί προγραμματιστές μπορούν να δουλεύουν ταυτόχρονα.

3.8.2 Github

Το Github είναι μια διαδικτυακή πλατφόρμα που φιλοξενεί Git αποθετήρια (repositories) και προσφέρει εργαλεία για συνεργατική ανάπτυξη λογισμικού.

1. Απομακρυσμένο αποθετήριο (remote repository)
   - Ασφαλής αποθήκευση κώδικα σε περιβάλλον σύννεφου (cloud) με εύκολη πρόσβαση από παντού.
2. Συνεργατικά εργαλεία
   - Αιτήματα έλξης (pull requests), αξιολογήσεις κώδικα (code reviews), προβλήματα (issues) για μια ολοκληρωμένη ομαδική δουλειά.
3. CI / CD και αυτοματοποίηση
   - Με τισ δράσης του Github (Github Actions) παρέχεται αυτόματη κατασκευή (build), δοκιμή (test) και δημοσίευση (deploy) της εφαρμογής.
4. Δημοσιότητα και χαρτοφυλάκιο έργων
   - Είναι μια καλή επιλογή για έργα ανοιχτού-κώδικα για προβολή έργων.

Κεφάλαιο 4ο: Κατασκευή της διαδικτυακής εφαρμογής, αρχιτεκτονική, διαγράμματα χρήσης και τρόπος λειτουργίας

4.1 Κατασκευή Εφαρμογής

Η εφαρμογή για να κατασκευαστεί χρειάστηκε αρκετές τεχνολογίες και αρχιτεκτονικές. Χρησιμοποιήθηκαν δύο βασικές διαφορετικές τεχνολογίες για το BackEnd και το FrontEnd. Δύο διαφορετικές βάσης δεδομένων και αρκετές αρχιτεκτονικές. Καθώς η εφαρμογή αποτελείται από δύο ρόλους τον απλό χρήστη αλλά και το μεσιτικό γραφείο έπρεπε να δημιουργηθεί ένα σύστημα ρόλων ώστε να έχουμε την αναγνώριση του χρήστη για την χρήση της εφαρμογής. Στις παρακάτω ενότητες αναλύονται όλα τα σημαντικά σημεία για την δημιουργία της εφαρμογής μας.

4.1.1 Βήματα υλοποίησης

Για την υλοποίηση της εφαρμογής μας χρειάστηκε να χωρίσουμε την εφαρμογή μας σε 5 στάδια ώστε να πετύχουμε το βέλτιστο αποτέλεσμα. Τα στάδια υλοποίησης είναι:

1. Σχεδιασμός Βάσης Δεδομένων: Στο πρώτο στάδιο σχεδιάστηκε η μοντελοποίηση της βάσης δεδομένων μας, οι πίνακες που θα χρησιμοποιηθούν και ποια από ποια πεδία θα αποτελείται κάθε πίνακας.
2. Δημιουργία κλάσεων για το Entity Framework: Μετά τον σχεδιασμό της βάσης δημιουργήθηκαν οι απαραίτητες κλάσεις στην .NET για την μοντελοποίηση των πινάκων και πεδίων της βάσης, όπως και οι απαραίτητες κλάσεις που συνδέουν το Entity Framework με την βάση και την ενεργοποιούν.
3. Δημιουργία Γενικού Αποθετηρίου (Generic Repository): Επόμενη εργασία ήταν η δημιουργία των κλάσεων γενικού αποθετηρίου ώστε να μπορούμε να διαχειριστούμε τις απαραίτητες λειτουργίες επικοινωνίας με τις βάσεις μας.
4. Δημιουργία Microsoft Identity: Στην συνέχεια υλοποιήθηκαν οι απαραίτητες κλάσεις, συναρτήσεις και ελεγκτές για την διαχείριση της αυθεντικοποίησης και ταυτοποίησης των χρηστών στο API μας.
5. Δημιουργία ελεγκτών (controllers) διαχείρισης ακινήτων: Επόμενη εργασία κατα σειρά ήταν η δημιουργία όλων των απαραίτητων κλάσεων για την διαχείριση των ακινήτων, δημιουργία, επιλογή, ενημέρωση και διαγραφή.
6. Δημιουργία διεπαφής χρήστη: Στην συνέχεια αφήσαμε λίγο το API και εστιάσαμε στην υλοποίηση της ιστοσελίδας μας, δημιουργήθηκαν όλες οι απαραίτητες σελίδες και συναρτήσεις για την διαχείριση, προβολή και δημιουργία ακινήτων.
7. Δημιουργία προτύπου προδιαγραφών (Specification Pattern): Επόμενη εργασία ήταν η προσαρμογή του ελεγκτή (controller) GetProperties(), ώστε να λειτουργεί με βάση το πρότυπο προδιαγραφών αφού πρώτα δημιουργήθηκαν οι κλασεις υλοποίησης του.
8. Βελτίωση διαχείρισης ακινήτων στο API και FrontEnd: Στην συνέχεια έγινε η βελτίωση και τελική εργασία διαχείρισης ακινήτων στο API αλλά και στο FrontEnd ώστε να ολοκληρωθεί

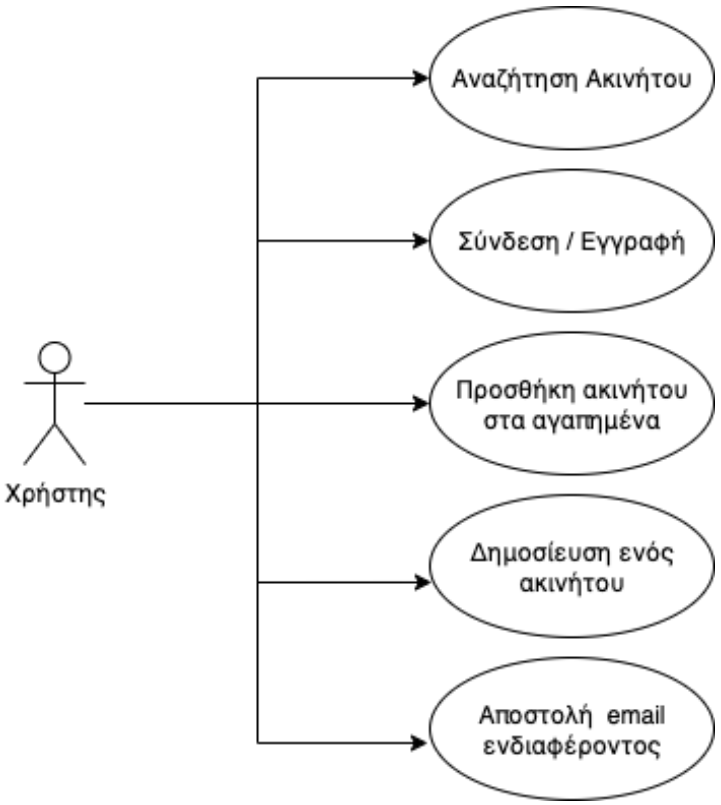
όλο το κομμάτι διαχείρισης ακινήτων και να μεταβούμε στην υλοποίηση διαχείρισης χρηστών στο FrontEnd.

9. Διαχείριση χρηστών στο FrontEnd: Επόμενη εργασία μας ήταν η δημιουργία όλων των απαραίτητων σελίδων στο FrontEnd αλλά και συναρτήσεων για την διαχείριση των χρηστών αλλά και ο περιορισμός προβολής σελίδων ή χρήσης συναρτήσεων με βάση τον ρόλο του χρήστη (πχ: σύνδεση, εγγραφή, επαναφορά κωδικού, δημοσίευση ακινήτου).
10. Ενσωματωση Stripe API: Επόμενη εργασία ήταν η δημιουργία της βιβλιοθήκης διαχείρισης συνδρομών με το Stripe API.
11. Σύνδεση βιβλιοθήκης Stripe με την βάση: Όπως αναγράφεται παρακάτω για την διαχείριση των συνδρομών μας χρειάστηκε να γίνουν κάποιες παραμετροποιήσεις στην βάση των χρηστών, επομένως η επόμενη εργασία ήταν αυτή η διασύνδεσης του Stripe API με την βάση μας.
12. Χρήση συνδρομών στο FrontEn: Μετά την ολοκλήρωση της υλοποίησης της Stripe βιβλιοθήκης η επόμενη εργασία ηταν η δημιουργία των σελίδων διαχείρισης και δημιουργίας συνδρομών.
13. Βελτιστοποίηση εφαρμογής: Στην συνέχεια έγινε έλεγχος και μικροδιορθώσεις σε FrontEnd και API για την ολοκλήρωση της εφαρμογής.
14. Δημιουργία αρχείου docker: Τελευταία εργασία ήταν η δημιουργία του Docker αρχείου ώστε να μπορέσει η εφαρμογή μας να εκτελεστεί σε κάποιον εξυπηρετητή και να είναι προσβάσιμη από το διαδίκτυο.

4.1.2 Απλός χρήστης

Ένας απλός χρήστης κατά την επίσκεψη του στην ιστοσελίδα μπορεί να πραγματοποιήσει αναζήτηση ακινήτων με βάση την τοποθεσία, περιοχή και άλλων προδιαγραφών που τον ενδιαφέρουν. Επιπλέον μπορεί να κάνει δημιουργία λογαριασμού ως απλός χρήστης. Ως απλός χρήστης μπορεί να αποθηκεύει ακίνητα στα αγαπημένα ακόμα και να κάνει δημοσίευση ενός ακινήτου.

Στην Εικόνα 4.1 φαίνεται το διάγραμμα χρήσης ενός απλού χρήστη στην ιστοσελίδα μας.

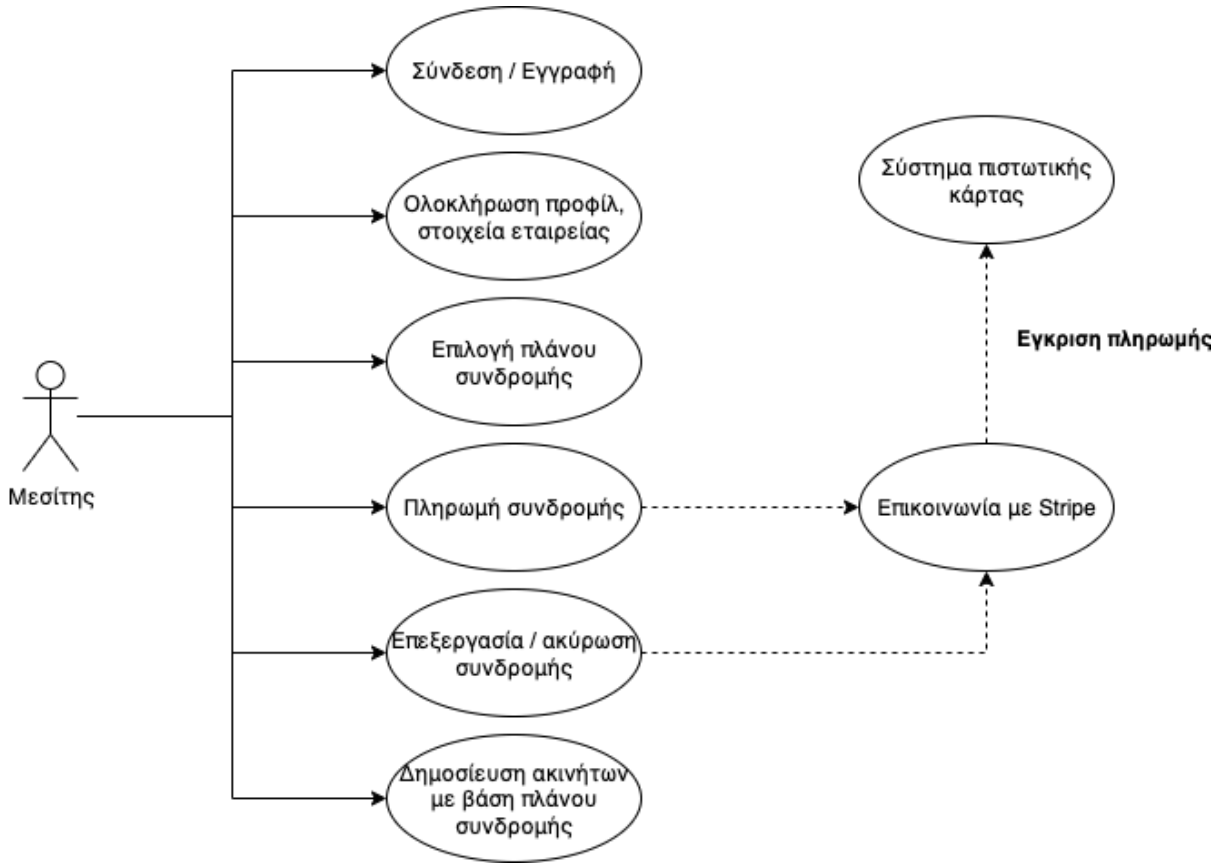


Εικόνα 4.1: Διάγραμμα χρήσης απλού χρήστη

4.1.3 Μεσιτικό γραφείο

Ένα μεσιτικό γραφείο εκτός από τις λειτουργίες του απλού χρήστη επιπλέον είναι υποχρεωμένος να κάνει μία συνδρομή στην ιστοσελίδα μας. Μπορεί να δημοσιεύει ακίνητα με βάση το πλάνο συνδρομής του, να διαχειρίζεται την συνδρομή του και να την ακυρώνει.

Στην Εικόνα 4.2 φαίνεται το διάγραμμα χρήσης ενός μεσιτικού γραφείου στην ιστοσελίδα μας.



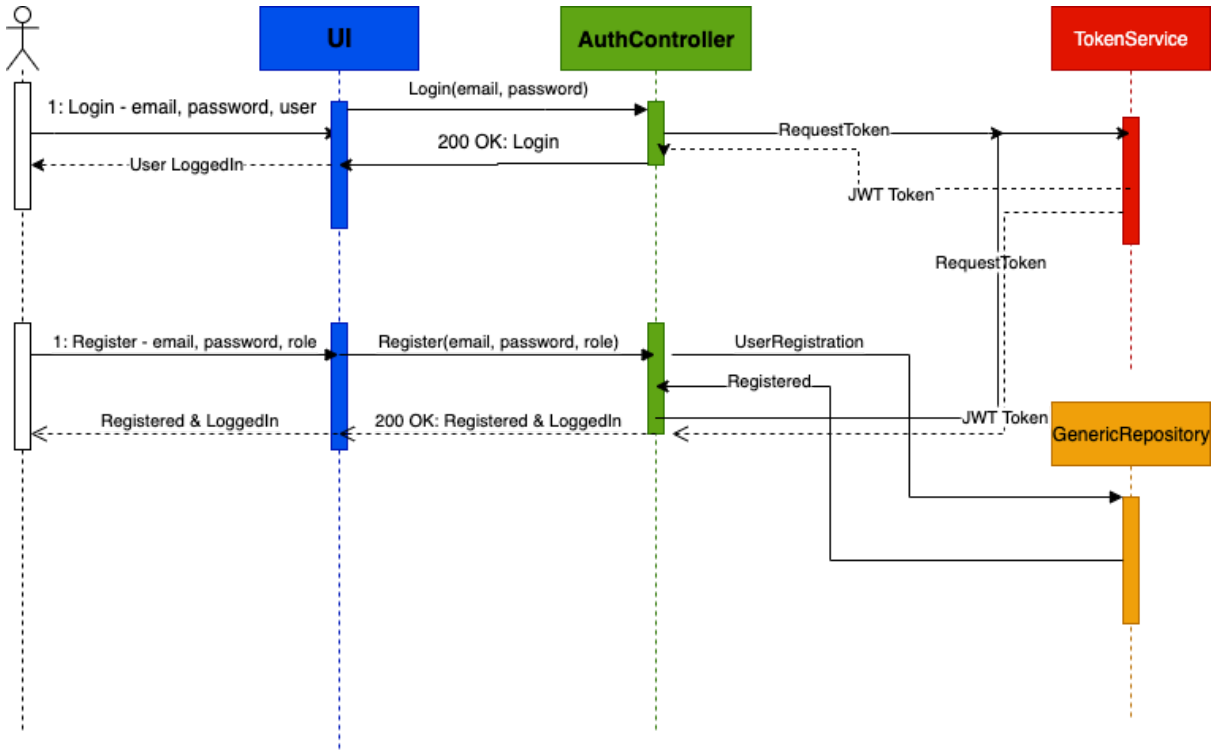
Εικόνα 4.2: Διάγραμμα χρήσης μεσίτη

4.1.4 Διαγράμματα ακολουθίας (Sequence Diagram)

Επειδή το έργο είναι αρκετά μεγάλο και περιέχει πολλές διαφορετικές κλάσεις υλοποίησης τα παρακάτω διαγράμματα ακολουθίας βοηθούν στην καλύτερη κατανόηση του κώδικα μας αλλά και της δομής της εφαρμογής μας.

4.1.4.1 Σύνδεση και Εγγραφή χρηστών

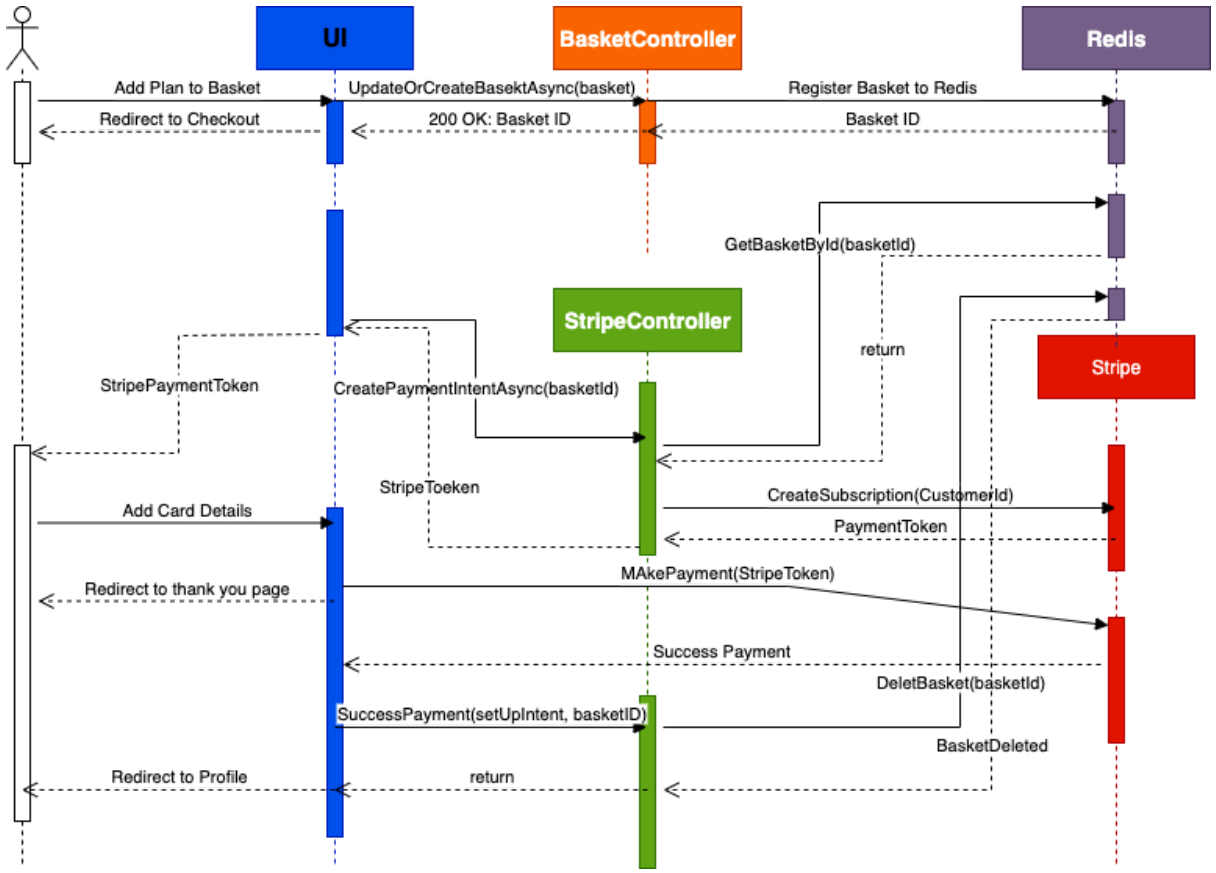
Στην Εικόνα 4.3 φαίνεται το διάγραμμα ακολουθίας Εγγραφής του χρήστη, και στην συνέχεια η σύνδεση του καθώς στην εφαρμογή μας ο χρήστης συνδέεται αυτόματα μετά την εγγραφή.



Εικόνα 4.3: Διάγραμμα ακολουθίας σύνδεσης και εγγραφής

4.1.4.2 Δημιουργία Συνδρομής

Μετά την σύνδεση του ως μεσίτης ο χρήστης πρέπει να επιλέξει ένα συνδρομητικό πακέτο. Το διάγραμμα που βρίσκεται στην Εικόνα 4.4 απεικονίζει το διάγραμμα ακολουθίας αγοράς συνδρομής.



Εικόνα 4.4: Διάγραμμα ακολουθίας αγοράς συνδρομής

4.2 Βάσεις Δεδομένων

Κατά την ανάλυση του συστήματος υπήρξε η ανάγκη για την δημιουργία δύο βάσεων δεδομένων, εκ των οποίων η μία κρατάει όλα τα δεδομένα των χρηστών και η άλλη όλα τα δεδομένα των ακινήτων της εφαρμογής μας. Ο κυριότερος λόγος της επιλογής δύο ξεχωριστών βάσεων δεδομένων ήταν κυρίως η ασφάλεια καθώς έχουμε διαχωρίσει τους χρήστες από ακίνητα τα οποία είναι και αυτά που είναι φανερά δημοσια στην ιστοσελίδα.

Για την κατασκευή των βάσεων έχει χρησιμοποιηθεί το Entity Framework όπως έχει προαναφερθεί οπότε για την κατασκευή τους δεν χρειάστηκαν να γραφτούν SQL ερωτήματα αλλά κώδικας σε επίπεδο κλάσεων με την χρήση .NET.

4.2.1 Βάση δεδομένων χρηστών

Η βάση δεδομένων για την διαχείριση των χρηστών, είτε αυτοί είναι απλοί χρήστες ή μεσιτικά γραφεία περιέχει εννέα πίνακες. Για την υλοποίηση της αυθεντικοποίησης και ταυτοποίησης των χρηστών στην εφαρμογή χρησιμοποιήθηκε η έτοιμη βιβλιοθήκη που παρέχεται από την .NET η Microsoft.Identity. Το Microsoft.Identity παρέχει απευθείας διαχείριση χρηστών και ρόλων χωρίς να χρειαστεί να αναπτύξουμε ένα σύστημα ταυτοποίησης και αυθεντικοποίησης χρηστών. [22]
Το Microsoft Identity κατά το Migration που κάνουμε με το Entity Framework δημιουργεί τους επτά από τους εννέα συνολικούς μας πίνακες, οι οποίοι είναι:

- AspNetRoleClaims με τα πεδία: Id, RoleId, ClaimType, ClaimValue
- AspNetRoles με τα πεδία: Id, Name, NormalizedName, ConcurencyStamp

- AspNetUserClaims με τα πεδία: Id, UserId, ClaimType, ClaimValue
- AspNetUserLogins με τα πεδία: LoginProvider, ProviderKey, ProviderDisplayName, UserId
- AspNetUserRoles με τα πεδία: UserId, RoleId
- AspNetUserTokens με τα πεδία: UserId, LoginProvider, Name, Value

Οι παραπάνω έξι πίνακες είναι οι βασικοί πίνακες που δημιουργούνται από το Microsoft Identity, για την ανάγκη της εφαρμογής μας τον βασικά πίνακα του χρήστη χρειάστηκε να τον καταπατήσουμε (override) και να προσθέσουμε κάτι έξτρα πεδία στην βάση μας που θα μας βοηθήσουν και θα χρειαστούμε για την κατασκευή της εφαρμογής μας.

- AspNetUsers με τα βασικά προκαθορισμένα πεδία: Id, UserName, NormalizedUserName, Email, EmailConfirmed, PasswordHash, SecurityStamp, ConcurrencyStamp, PhoneNumber, PhoneNumberConfirmed, TwoFactorEnabled, LockoutEnd, LockoutEnabled, AccessFailedCount
  - και τα έξτρα πεδία που χρειάστηκε να προσθέσουμε είναι: AgencyId, CompletedProfile, FullName, DateOfBirth, StripeCustomerId, ImagePath, ReceivePromotionalEmails

Κώδικας που δείχνει το μοντέλο του πίνακα AspNetUserts με τα πεδία που προσθέσαμε. Όλα τα προκαθορισμένα πεδία υλοποιούνται από την κλάση IdentityUder<int> όπου αντλή η βασική μας κλάση UserModel.

```
using Core.Model.Agencies; // Μοντέλο για τον πίνακα Agencies
using Microsoft.AspNetCore.Identity; // Βιβλιοθήκη Microsoft Identity

namespace Core.Models.Auth
{
	public class UserModel : IdentityUser<int>
	{
		public int? AgencyId { get; set; }
		public AgencyModel? AgencyModel { get; set; }
		public bool CompletedProfile { get; set; } = false;
		public string? FullName { get; set; }
		public DateOnly? DateOfBirth { get; set; }
		public string? StripeCustomerId { get; set; }
		public string? ImagePath { get; set; }
		public bool ReceivePromotionalEmails { get; set; } = false;
	}
}
```

Οι υπόλοιποι δύο πίνακες που δημιουργήσαμε χρησιμοποιήθηκαν για την καταγραφή των στοιχείων των μεσιτικών γραφείων και των συνδρομών των μεσιτικών γραφείων.

- Agencies με τα πεδία: Id, AgencyName, Country, Address, TinNumber, UserId, PlanId, City, ZipCode, CountryIsoCode
- Subscriptions με τα πεδία: Id, PlanId, SubscriptionId, UserEmail, FreeTrial, PaymentRetries, NextPayment

Ο παρακάτω κώδικας δείχνει την κλάση AppIdentityDbContext η οποία εκτελείται για την δημιουργία των πινάκων κατά την διαδικασία του Migration από το EntityFramework. Όλοι οι

βασικοί πίνακες του Microsoft Identity ορίζονται από την κλάση IdentitydbContext από την οποία αντλή η βασική μας κλάση AppIdentityDbContext.

```
using Core.Models.Agencies;
using Core.Models.Auth;
using Core.Models.Shop;
using Microsoft.AspNetCore.Identity; // Βιβλιοθήκη Microsoft Identity
// Βιβλιοθήκη Microsoft Identity που χρησιμοποιείται από το Entity Framework
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore; // Βιβλιοθήκη Entity Framework

namespace Infrastructure.Data.Identity
{
	public class AppIdentityDbContext : IdentityDbContext<UserModel, RoleModel, int, IdentityUserClaim<int>,
UserRoleModel, IdentityUserLogin<int>, IdentityRoleClaim<int>, IdentityUserToke<int>>
	{
		public AppIdentityDbContext(DbContextOptions<AppIdentityDbContext> options) : base(options) {}

		// Δημιουργία πρόσθετων πινάκων για την εφαρμογή μας
		public DbSet<AgencyModel> Agencies { get; set; }
		public DbSet<Subscription> Subscriptions { get; set; }

		// Μέθοδος που εκτελείται κατά την δημιουργία της βάσης για να ορίσουμε συσχετίσεις στις βάσης ή
συγκεκριμένους τύπους δεδομένων των κελιών
		protected override void OnModelCreating(ModelBuilder builder)
		{
			base.OnModelCreating(builder);
		}
	}
}
```

Στον παραπάνω κώδικα αναφερθήκαμε στην μέθοδο που εκτελείται κατά την δημιουργία της βάσης, η οποία χρησιμοποιείται για τον ορισμό τύπων δεδομένων και συσχέτισης των πεδίων των βάσεων. Επειδή συνήθως κατά την διαμόρφωση των χαρακτηριστικών των πεδίων των βάσεων έχουμε πολλά χαρακτηριστικά και συσχετίσεις απλά δηλώνουμε στην βασική μας κλάση (όπως παραπάνω με την μέθοδο OnModelCreating) πως υπάρχουν άλλες κλάσεις οι οποίες περιέχουν την συσχέτιση και αυτό γίνεται για λόγους ευανάγνωσης και πιο εύκολης διαχείρισης δημιουργούμε άλλες εξωτερικές κλάσης που έχουν την διαμόρφωση για κάθε πίνα ξεχωριστά όπως στον κώδικα παρακάτω.

```
using Core.Models.Auth;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkcore.Metadata.Builders; // Βιβλιοθήκη χρήσης της μεθόδου συσχέτισης

namespace Infrastructure.Data.Identity.Configuration
{
	public class UserConfigurationModel : IEntityTypeConfiguration<UserModel>
	{
		// Η συσχέτιση που θα κάνουμε είναι ότι ένας χρήστης μπορεί να έχει ένα μόνο μεσιτικό γραφείο
		public void Configure(EntityTypeBuilder<UserModel> builder)
		{
			builder.HasOne(x => x.AgencyModel)
			  .WithOne(a => a.User)
			  .OnDelete(DeleteBehavior.Cascade)
		}
	}
}
```

4.2.1 Βάση δεδομένων ακινήτων

Η βάση δεδομένων για τα ακίνητα περιέχει όλες τις πληροφορίες ενός ακινήτου, με τα χαρακτηριστικά τους, επιπλέων χαρακτηριστικά και ιδιότητες ακινήτων. Αποτελείται από 12 πίνακες οι οποίο συσχετίζονται μεταξύ τους. Οι πίνακες μας είναι:

- Properties με τα πεδία: Id, PropertyFor, SquareMeters, Price, OldPrice, Description, AccessFrom, DisabilitiesAccess, View, Positioning, Zone, AvailableFrom, DistanceFromSea, DistanceFromCity, DistanceFromCenter, DistanceFromAirport, UserEmail, PropertyCharacteristicsId, PropertyTypeId, PropertyCategoryId, DateCreated, LastUpdate, CityId, ComunityId, Alias, Address.
  - Ο πίνακας Properties περιέχει τα βασικά στοιχεία κάθε ακινήτου ανεξαρτήτως της κατηγορίας του.
- PropertyCategories με τα πεδία: Id, PropertyCategoryName, PropertyId, PropertyTypeId
  - Κατηγορίες ακινήτων (πχ. Σπίτι)
- PropertyTypes με τα πεδία: Id, TypeName
  - Τύπος ακινήτου ανά κατηγορία (πχ. Σπίτι → Διαμέρισμα)
- PropertyImages με τα πεδία: Id, PropertyImagePath, PropertyId
  - Φωτογραφίες και βίντεο ανά διαμέρισμα
- PropertyCharacteristics με τα πεδία: Id, ConstructionYear, RenovatedYear, RenovationType, AppartmentFloor, EnergyClass, Heating, TypeOfHeating, NumberOfRooms, Bathrooms, WCs, Kitchens, Balconies, Terraces, ParkingSpaces, ParkingType, LandSquareMeters, BuildingPermit, BuildingSquareMeters, StructureFactor, Shape, FrontageLength, ConstructionsMeters, CityPlan, PropertyId, PropertyAdditionalFeatureId
  - Χαρακτηριστικά ακινήτου (πχ. Ενεργειακή Κλάση, Θέσεις Στάθμευσης)
- PropertyAdditionalFeatures με τα πεδία: Id, Fournished, ElectricDevices, MaintenanceFees, DoorType, FloorType, Werehouse, PetsAllowed, Cameras, CableTv, FloorHeating, InternalStairs, Playroom, Elevator, SolarWaterHeater, Garden, Luxurius, Jacuzzi, SateliteAntenna, MotorisedShutters, TypeFrame, Neoclassical, Veranda, HousingStatus, Awnings, AlarmSystem, Pool, PropertyCharacteristicId.
  - Επιπρόσθετα δευτερεύοντα χαρακτηριστικά ενός ακινήτου.
- City με τα πεδία: Id, CityName, MunicipalityId
- Municipality: Id, MunicipalityName, DistrictId
- District: Id, District

Σε αυτήν την βάση έχουμε προσθέσει και τον πίνακα Favourites όπου χρησιμοποιείται για την αποθήκευση των ακινήτων στα αγαπημένα του εκάστοτε χρήστη.

- Favourites με τα πεδία: Id, PropertyId, UserEmail

Η βάση μας αυτή εκτός από τα χαρακτηριστικά των ακινήτων περιέχει και τα πλάνα συνδρομής της εφαρμογής μας, με τον πίνακα:

- Plans με τα πεδία: Id, PlanName, PropertiesToRegister, PricePerMonth, PricePerYear, SubscriptionType, Colour, StripePaymentId

Καθώς όπως έχει αναφερθεί οι βάσεις μας χρησιμοποιούν Entity Framework κάθε πίνακας αντικατοπτρίζεται από μία κλάση, έτσι για διευκόλυνση μας για να μην χρειάζεται σε κάθε κλάση να

ξαναγράψουμε την ιδιότητα Id που είναι ο αύξων μοναδικός μας αριθμός στην βάση δημιουργήσαμε μία βασική κλάση την οποία αντλούμε το Id σε κάθε άλλη κλάση μας όπως φαίνεται παρακάτω.

```
using Newtonsoft.Json; // Βιβλιοθήκη που χρησιμοποιείται για διαχείριση των κλάσεων σε συσχέτιση με Json προβολή

namespace Core.Models
{
	[JsonObject(MemberSerialization.OptIn)]
	public class BaseModel
	{
		[JsonProperty] // Ορίζουμε ότι σε κάποια κλήση API το ID θα περιέχεται στα δεδομένα μας
		public int Id { get; set; }
	}
}
```

```
using Newtonsoft.Json;

namespace Core.Models.Plans
{
	[JsonObject(MemberSerialization.OptIn)]
	public class PlanModel : BaseModel
	{
		[JsonProperty]
		public string? PlanName { get; set; }
		[JsonProperty]
		public int PropertiesToRegister { get; set; } = 0;
		[JsonProperty]
		public decimal PricePerMonth { get; set; } = 0;
		[JsonProperty]
		public decimal PricePerYear { get; set; } = 0;
		[JsonProperty]
		public PlanSubscription SubscriptionType { get; set; } = PlanSubscription.Monthly;
		[JsonProperty]
		public string? Colour { get; set; }
		[JsonProperty]
		public string? StripePaymentId { get; set; } = 0;
	}
}
```

Στον παρακάτω κώδικα αναγράφεται η κλάση DatabaseContext η οποία εκτελείται για την δημιουργία των πινάκων κατά την διαδικασία του Migration από το EntityFramework.

```
using System.Reflection;
using Core.Models.Plans;
using Core.Models.Properties;
using Core.Models.Municipalities;
using Microsoft.EntityFrameworkCore;

namespace Infrastructure.Data
{
	public class DatabaseContext : DbContext
	{
		public DatabaseContext(DbContextOptions<DatabaseContext> options) : base(options) {}

		public DbSet<PlanModel> Plans { get; set; }
		public DbSet<PropertyModel> Properties { get; set; }
		public DbSet<PropertyCharacteristicModel> PropertyCharacteristics { get; set; }
		public DbSet<PropertyAdditionalFeatureModel> PropertyAdditionalFeatures { get; set; }
		public DbSet<PropertyImageModel> PropertyImages { get; set; }
		public DbSet<PropertyTypeModel> PropertyTypes { get; set; }
		public DbSet<PropertyCategoryModel> PropertyCategories { get; set; }
		public DbSet<FavouriteModel> Favourites { get; set; }
		public DbSet<DistrictModel> Districts { get; set; }
```

```
        public DbSet<CityModel> Cities { get; set; }
        public DbSet<MunicipalityModel> Municipalities { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
        }
    }
}
```

Στον παρακάτω κώδικα παρουσιάζεται η συσχέτιση του πίνακά Properties με τους υπόλοιπους πίνακες σε εξωτερική κλάση η οποία χρησιμοποιείται κατά την δημιουργία της βάσης από το Entity Framework.

```
using Core.Models.Properties;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Infrastructure.Data.Configuration.Properties
{
    public void  Configure(EntityTypeBuilder<PropertyBuilder> builder)
    {
        // Εικόνες ακινήτων σχέση πολλά προς ένα
        builder.HasMany(p => p.PropertyImages)
            .WithOne(i => i.PropertyModel)
            .HasForeignKey(i => i.PropertyId)
            .OnDelete(DeleteBehavior.Cascade);

        // Χαρακτηριστικά ένα προς ένα
        builder.HasOne(p => p.PropertyCharacteristics)
            .WithOne(c => c.Property)
            .OnDelete(DeleteBehavior.Cascade);

        // Κατηγορίες ένα προς πολλά
        builder.HasOne(p => p.PropertyCategory)
            .WithMany()
            .HasForeignKey(p => p.PropertyCategoryId);

        // Πόλη ένα προς πολλά
        builder.HasOne(p => p.City)
            .WithMany()
            .HasForeignKey(p => p.CityId);

        // Δήμος ένα προς πολλά
        builder.HasOne(p => p.Municipality)
            .WithMany()
            .HasForeignKey(p => p.MunicipalityId);
    }
}
```

4.3 Αρχιτεκτονική και υλοποίηση API διαδικτυακής εφαρμογής

Το Backend της διαδικτυακής μας εφαρμογής κατασκευάστηκε σε αρχιτεκτονική Μοντέλο - Προβολή - Ελεγκτή (MVC, Model - View - Controller). Αποτελείται από το βασικό έργο (project) API και τις βιβλιοθήκες Infrastructure και Core. Για την επίτευξη της αυθεντικοποίησης και ταυτοποίησης του χρήστη χρησιμοποιείται η τεχνολογία JWT (Json Web Tokens.

Για την βασική Δημιουργία, Ανάγνωση, Ενημέρωση και Διαγραφή (CRUD, Create-Read-Update-Delete) έχει χρησιμοποιηθεί η αρχιτεκτονική Γενικού Αποθετηρίου (Generic Repository).

Κατά την ανάγνωση των δεδομένων από την βάση υπάρχει η ανάγκη να γίνεται αναζήτηση ή φιλτράρισμα των δεδομένων, για την επίτευξη αυτού του σκοπού έχει χρησιμοποιηθεί το πρότυπο προδιαγραφών.

Έχει δημιουργηθεί και μία τέταρτη βιβλιοθήκη η οποία εξυπηρετεί τον σκοπό της διαχείρισης των συνδρομών και του Stripe. Ο λόγος που έγινε ξεχωριστή βιβλιοθήκη η χρήση του API του Stripe είναι για να μπορεί η βιβλιοθήκη αυτή να επαναχρησιμοποιηθεί σε άλλα έργα.

4.3.1 MVC (Model - View - Controller)

Το MVC είναι ένα μοτίβο σχεδίασης λογισμικού που διαιρεί μια εφαρμογή σε τρία διασυνδεδεμένα μέρη. Αυτός ο διαχωρισμός βοηθά στην οργάνωση του κώδικα, τη βελτίωση της συντηρησιμότητας και στην διευκόλυνση των δοκιμών των εφαρμογών. [23]

4.3.1.1 Model - Core βιλιοθήκη

Στην εφαρμογή μας, τον ρόλο του μοντέλου έχει η βιβλιοθήκη Core, η οποία περιέχει όλα τα μοντέλα και τα πρότυπα που χρησιμοποιεί η εφαρμογή μας για την λειτουργία της. Η δομή φακέλων της βιβλιοθήκης μας περιέχει:

- DTOs: όπου περιέχει όλα τα μοντέλα που χρειαζόμαστε για την επιχειρηματική λογική της εφαρμογής μας.
- Interfaces: όπου περιέχει όλες τις διεπαφές που χρειαζόμαστε για την λειτουργία της εφαρμογής μας.
- Models: όπου περιέχει όλα τα μοντέλα κλάσεων που χρειαζόμαστε για την λειτουργία της εφαρμογής μας και της βάσης δεδομένων. Ο φάκελος Models χωρίζεται σε υποφακέλους ανάλογα με την κατηγορία των κλάσεων που έχουμε. (πχ. Properties, Auth).
- Templates: όπου περιέχει όλα τα πρότυπα των μηνυμάτων ηλεκτρονικού ταχυδρομείου που στέλνονται στους χρήστες για την λειτουργία της εφαρμογης. (πχ. Μήνυμα επιβεβαίωσης ηλεκτρονικού ταχυδρομείου, αλλαγή κωδικού πρόσβασης).

4.3.1.2 View - Infrastructure βιλιοθήκη

Η βιβλιοθήκη Infrastructure έχει τον ρόλο της προβολής στην εφαρμογή μας. Σε αυτήν την βιβλιοθήκη υλοποιούνται όλες οι απαραίτητες διαδικασίες για την προβολή των δεδομένων στον χρήστη. Η δομή των φακέλων της βιβλιοθήκης είναι:

- Repositories: όπου περιέχει τις απαραίτητες κλάσεις για την αντιστοίχιση των επιπέδων των δεδομένων μας και τις αντιστοιχίζει με την επιχειρηματική λογική.
- Data: Ο φάκελος data είναι και ο μεγαλύτερος φάκελος της βιβλιοθήκης μας ο οποίος χειρίζεται όλες τις κλάσεις δεδομένων της εφαρμογής μας και χωρίζεται σε υποφακέλους.
    - Configurations: στο κεφάλαιο 4.2 αναφερθήκαμε ότι κατά το Migration του Entity Framework έχουμε καποιες κλάσεις οι οποίες ορίζουν τις συσχετίσεις των πινάκων και τους τύπους δεδομένων των πεδίων, αυτός ο φάκελος περιέχει αυτές τις κλάσεις.

- Identity: Ο φάκελος αυτός περιέχει τις κλάσης που χρειάζονται για την δημιουργία της βάσης δεδομένων των χρηστών (δηλ: AppIdentityDbContext.cs)
- Migrations: Ο φάκελος Migrations δημιουργείται κατά την εκτέλεση του Migration από το Entity Framework. Το Entity Framework κατά την εκτέλεση του δημιουργεί κάποιες κλάσεις οι οποίες περιέχουν απαραίτητα δεδομένα για την βάση.
- SeedData: Ο φάκελος SeedData περιέχει κάποια json αρχεία τα οποία έχουν δημιουργηθεί από εμάς με βασικά δεδομένα για να γεμίζουμε την βάση μας καθώς τρέχουμε την εφαρμογή μας για να μπορούμε να κάνουμε δοκιμές.
- Περιέχει επίσης την κλάση την οποία χρειαζόμαστε για την δημιουργία της βάσης δεδομένων των ακινήτων (δηλ. DatabaseContext.cs)

4.3.1.3 Controller - API Βασικό έργο

Ο φάκελος API περιέχει όλο το βασικό μας έργο, το Program.cs και τους Controllers που χρειάζονται για την λειτουργία της εφαρμογής μας. Η δομή των φακέλων βασικού μας έργου είναι:

- Controllers: Περιέχει όλους τους ελεγκτές της εφαρμογής μας, δηλαδή τα τελικά σημεία διαχείρισης των αιτημάτων που κάνουν οι χρήστες από την εφαρμογή μας, χωρίζεται σε επιμέρους φακέλους όπου κατηγοριοποιεί τους ελεγκτές (πχ. Auth, Plans, Properties).
- Extensions: Ο φάκελος αυτός περιέχει κάποιες κλάσεις που χρησιμοποιούνται κατα την ενεργοποίηση της εφαρμογής. Οι κλασεις αυτές καλούνται από την βασική μέθοδό μας main, ο λόγος που έχουν δημιουργηθεί είναι για να είναι πιο καθαρός ο κώδικας μας στην μέθοδο main και πιο εύκολα διαχειρίσιμος.
- Helpers: Ο φάκελος αυτός περιέχει κάποιες κλάσεις που χρησιμοποιούνται σε κάποιες μεθόδους για την υλοποίηση κάποιον μικρών συναρτήσεων για την καλύτερη διαχείριση του κωδικα μας (πχ. AutoMapperProfile.cs)
- Services: Είναι ένας φάκελος που περιέχει κάποιες διεπαφές με τις κλάσεις υλοποίησης τους για συγκεκριμένες συναρτήσεις της εφαρμογής μας (πχ. TokenService.cs).

4.3.2 JWT (Json Web Token)

Το JWT είναι ένα ανοιχτό πρότυπο για την ασφαλή μετάδοση πληροφοριών μετά μερών εφαρμογών (πχ. FrontEnd - BackEnd) ως αντικείμενο JSON. Χρησιμοποιείται κυρίως για έλεγχο ταυτότητας, εξουσιοδότηση και ασφαλή κοινή χρήση πληροφοριών μεταξύ υπηρεσιών.[24] Στην εφαρμογή μας κατά την σύνδεση και ταυτοποίηση του χρήστη, δημιουργείται ένα cookie στον περιηγητη του χρήστη για 24 ώρες και ο χρήστης με αυτό το token που αποθηκεύεται στο cookie του περιηγητή του μπορεί να είναι συνδεδεμένος και να χρησιμοποιεί την εφαρμογή ως συνδεδεμένος χρήστης για 24 ώρες.

Στον παρακάτω κώδικα φαίνεται η κλάση που διαμορφώνει την χρήση του Microsoft Identity, ορισμό της χρήσης του JWT και πολιτική χρήσης μόνο από διαχειριστές.

```
using System.Text;
using Core.Models.Auth;
using Infrastructure.Data.Identity;
// Απαραίτητη βιβλιοθήκη για την χρήση του JWT
using Microsoft.AspNetCore.Authentication.JwtBearer;
```

```
using Microsoft.AspNetCore.Identity;
using Microsoft.IdentityModel.Tokens;

namespace API.Extensions
{
	public static IServiceCollection AddIdentityServiceExtensions(this IServiceCollection services, IConfiguration
config)
	{
		// Δήλωση Microsoft Identity
		services.AddIdentityCore<UserModel>(options =>
		{
			options.Password.RequireNonAlphanumeric = true;
			options.User.RequireUniqueEmail = true;
			options.Tokens.AuthenticatorIssuer = “JWT”;
			options.SignIn.RequiredConfirmedEmail = true;
		})
		.AddRoles<RoleModel>()
		.AddRoleManager<RoleManager<RoleModel>>()
		.AddSignInManager<SignInManager<UserModel>>()
		.AddRoleValidator<RoleValidator<RoleModel>>()
		.AddEntityFrameworkStores<AppIdentityDbContext>()
		.AddDefaultTokenProviders();

		// Ορισμού χρόνου λήξης του Token
		services.Configure<DataProtectionTokenProviderOptions>(opt =>
			opt.TokenLifespan = TimeSpan.FromHours(24));


		// Ορισμός της αυθεντικοποίησης χρήσης
		services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
			.AddJwtBearer(options =>
			{
				options.TokenValidationParameters = new TokenValidationParameters
				{
					ValidateIssuerSigningKey = true;
					IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF32.GetBytes(config[“Token:Key”]!)),
					ValidateIssuer = false,
					ValidateAudience = false
				};
			});

		// Ορισμός πολιτικής χρήσης μόνο από διαχειριστές
		services.AddAuthorization(options =>
		{
			options.AddPolicy(“RequireAdminRole”,
				policy => policy.RequireRole(“Admin”);
		});

		return services;
	}
}
```

4.3.3 Γενικό αποθετήριο (Generic Repository)

Το γενικό αποθετήριο (Generic Repository) είναι ένα μοτίβο σχεδίασης το οποίο χρησιμοποιείται για την δημιουργία ενός ευέλικτου και επαναχρησιμοποιούμενο μοτίβου πρόσβασης δεδομένων για την αλληλεπίδραση με την βάση δεδομένων. Χρησιμοποιεί ένα τυπικό δεδομένο για την βασική εκτέλεση τυπικών λειτουργιών των δεδομένων, σε διάφορους τύπους οντοτήτων χωρίς να χρειάζεται να γραφτεί συγκεκριμένος κώδικας για την κάθε οντότητα. [25]

Στον παρακάτω κώδικα αναλύεται η διεπαφή και η κλάση που υλοποιεί την διεπαφή για την υλοποίηση του γενικού αποθετηρίου.

Στον κωδικα μας στην αρχή είναι δηλωμένη η οντότητα T, η οποία αντιπροσωπεύει στη C# (.NET) μία γενική οντότητα την οποία κάθε φορά που δηλώνουμε την κλάση μας για χρήση της δίνουμε μια συγκεκριμένη οντότητα που θέλουμε να χρησιμοποιήσουμε για την συγκεκριμένη υλοποίηση.

Διεπαφή Γενικού Αποθετηρίου

```
using Core.Models;

namespace Core.Interfaces
{
	// Ορίζουμε ότι το T αναγνωρίζει το BaseModel ως μοναδικό αναγνωριστικό
	public interface IGenericRepository<T> where T: BaseModel
	{
		// Επιστροφή λίστας δεδομένων από την βάση
		Task<IReadOnlyList<T>> ListAllAsync();
		// Εύρεση δεδομένου στην βάση
		Task<T> GetByIdAsync(int id);
		// Εγγραφή στην βάση δεδομένων
		Task<T> AddAsync(T Base);
		// Ενημέρωση δεδομένων
		Task<T> UpdateAsync(T Base);
		// Διαγραφή δεδομένων
		Task<T> DeleteAsync(int id);
	}
}
```

Υλοποίηση Γενικού Αποθετηρίου

```
using Core.Interfaces;
using Core.Models;
using Infrastructure.Data;
using Microsoft.EntityFrameworkCore;

namespace Infrastructure.Repositories
{
	public class GenericRepository<T> : IGenericRepository<T> where T : BaseModel
	{
		// Χρήση της κλάσης DatabaseContext για επικοινωνία με την βάση
		private readonly DatabaseContext _context;
		public GenericRepository(DatabaseContext context)
		{
			_context = context;
		}

		public async Task<T> AddAsync(T Base)
		{
			_context.Set<T>.Add(Base);
			await _context.SaveChangesAsync();
			return Base;
		}

		public async Task<T> DeleteAsync(int id);
		{
			var entity = await _context.Set<T>().FindAsync(id);
			if (entity is null)
			{
				return entity!;
			}
			_context.Set<T>().Remove(entity);
```

```
                await _context.SaveChangesAsync();
                return entity;
            }

            public async Task<T> GetByIdAsync(int id)
            {
                return await _context.Set<T>().FindAsync(id);
            }

            public async Task<IReadOnlyList<T>> ListAllAsync()
            {
                return await _context.Set<T>().AsNoTracking().ToListAsync();
            }

            public async Task<T> UpdateAsync(T Base)
            {
                if (await TExists(Base.Id))
                {
                    _context.Entry(Base).State = EntityState.Modified;
                    await _context.SaveChangesAsync();
                    return Base;
                }
                return null;
            }

            private async Task<bool> TExists(int id)
            {
                return await _context.Set<T>().AnyAsync(c => c.Id == id);
            }
        }
}
```

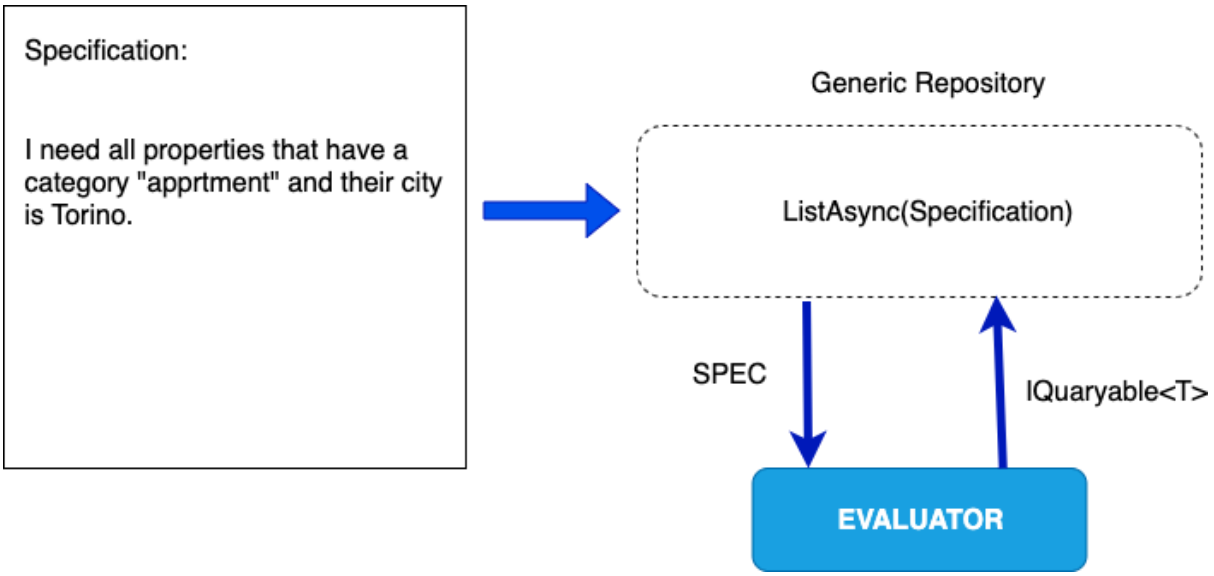
Χρήση Γενικού Αποθετηρίου σε ελεγκτή του προγράμματος μας

```
using Core.Interfaces;
using Core.Models.Plans;
using Infrastructure.Data;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace API.Controllers.Plans
{
    public class PlansController : BaseApiController
    {
        // Χρησιμοποιείται για την καταγραφή του συστήματος μας
        private readonly ILogger<PlansController> _logger;
        // Χρήση γενικού αποθετηρίου για το μοντέλο PlanModel
        private readonly IGenericRepository<PlanModel> _planRepo;

        public PlansContoller(ILogger<PlansController> logger, IGenericRepository<PlanModel> planRepo)
        {
            _planRepo = planRepo;
            _logger = logger;
        }

        [HttpGet] // HTTP Μέθοδος που θα χρειαστεί για το τελικό σημείο του API
        public async Task<ActionResult<IReadOnlyList<PlanModel>>> GetPlansAsync()
        {
            var plans = await _planRepo.ListAllAsync();
            return Ok(plans); // Επιστρέφει την λίστα με HTTP κωδικό 200
        }

        [HttpGet("{id}")] // HTTP μέθοδος με παράμετρο το Id από την εγγραφή στη βάση
        public async Task<ActionResult<PlanModel> GetPlanByIdAsync(int id)
        {
```

```
                        // Με το await λέμε στον κωδικα μας να περιμένει το αποτέλεσμα πριν προχωρήσει παρακάτω
                        var plan = await _planRepo.GetByIdAsync(id);
                        if (plan is null)
                                return NotFound(); // HTTP κωδικός 404

                        return Ok(plan);
                }

                [HttpPost]
                // Με το FromBody ορίζουμε την μέθοδο μας από πιο σημείο του HTTP αιτήματος θα διαβάσει τα
απαραίτητα στοιχεία
                public async Task<ActionResult<PlanModel>> AddPlanAsync([FromBody] PlanModel planModel)
                {
                        var plan = await _planRepo.AddAsync(planModel);
                        return Ok(plan);
                }

                [HttpPut]
                public async Task<ActionResult<PlanModel>> UpdatePlanAsync([FromBody] PlanModel planModel)
                {
                        var plan = await _planRepo.UpdateAsync(planModel);
                        if (plan is null)
                                return NotFound();

                        return Ok(plan);
                }

                [HttpDelete("{id}")]
                public async Task<ActionResult> DeletePlanAsync(int id)
                {
                        var plan = await _planRepo.DeleteAsync(id);
                        if (plan is null)
                                return NotFound();

                        // HTTP κωδικός 200 με μήνυμα για το FrontEnd
                        return Ok($"Plan {plan.PlanName} successfully deleted");
                }
        }
}
```

4.3.4 Πρότυπο Προδιαγραφών (Specification Pattern)

Το πρότυπο προδιαγραφών στην C# (Specification Pattern) είναι ένα πρότυπο σχεδιασμού συμπεριφοράς το οποίο ενσωματώνει μία συνθήκη ή έναν συγκεκριμένο επιχειρηματικό κανόνα σε ένα επαναχρησιμοποιήσιμο αντικείμενο το οποίο ονομάζεται προδιαγραφή (εξού και το όνομα πρότυπο προδιαγραφών). Αυτή η προδιαγραφή στην συνέχεια μπορεί να χρησιμοποιηθεί ώστε να εξετάσουμε αν ένα αντικείμενο ή οντότητα ικανοποιεί την συνθήκη της προδιαγραφής. [26] Στην Εικόνα 4.5 φαίνεται μια σχηματική αναπαράσταση από το πρότυπο προδιαγραφών.



Εικόνα 4.5: Specification Pattern σχηματική επεξήγηση

Στον κώδικα μας το πρότυπο προδιαγραφών χρησιμοποιείτε κατά το φιλτράρισμα των ακινήτων σε μία αναζήτηση, το οποίο αναλύεται στων κώδικα παρακάτω.

Διεπαφή προτύπου προδιαγραφών

```
using System.Linq.Expressions;

namespace Core.Interfaces
{
	public interface ISpecification<T>
	{
		Expression<Func<T, bool>> Criteria { get; }
		List <Expression<Func<T, object>>> Includes { get; }
		Expression<Func<T, object>> OrderBy { get; }
		Expression<Func<T, object>> OrderByDescending { get; }
	}
}
```

Υλοποίηση διεπαφής προτύπου προδιαγραφών

```
using Core.Interfaces;
using Core.Models;
using Microsoft.EntityFrameworkCore;

namespace Infrastructure.Repositories
{
	// TEntity ορισμός γενικού μοντέλου
	public class SpecificationEvaluator<TEntity> where TEntity : BaseModel
	{
		public static IQuerable<TEntity> GetQuert(IQuerable<TEntity> inputQuery,
			ISpecification<TEntity> spec)
		{
			var query = inputQuery;

			if (spec.Criteria !=  null)
			{
				query = query.Where(spec.Criteria; // πχ. Τύπος ακινήτου = Διαμέρισμα
			}
```

```
                if (spec.OrderBy != nul)
                {
                        query = query.OrderBy(spec.OrderBy);
                }

                if (spec.OrderByDescending != null)
                {
                        query = query.OrderByDescending(spec.OrderByDescending);
                }

                query = spec.Includes.Aggregate(query, (current, include) => current.Include(include));

                return query;
        }
    }
}
```

Γενική υλοποίηση διεπαφής

```
using System.Linq.Expressions;
using Core.Interfaces;

namespace Infrastructure.Repositories
{
    public class BaseSpecification<T> : ISpecification<T>
    {
        public class BaseSpecification() {}

        public class BaseSpecification(Expression<Func<T, bool>> criteria)
        {
            Criteria = criteria;
        }

        public Expression<Func<T, bool>> Criteria { get;}

        public List<Expression<Func<T, object>>> Includes { get; } = new List<Expression<Func<T,
object>>>();
        public Expression<Func<T, object>> OrderBy { get; private set; }

        public Expression<Func<T, object>> OrderByDescending { get; private set; }

        protected void AddInclude(Expression<Func<T, object>> includeExpression)
        {
            Includes.Add(includeExpression);
        }

        protected void AddOrderBy(Expression<Func<T, object>> orderByExpression)
        {
            OrderBy = orderByExpression;
        }

        protected void AddOrderByDescending(Expression<Func<T, object>> orderByDescExpression)
        {
            OrderByDescending = orderByDescExpression;
        }
    }
}
```

Μετά την δημιουργία του βασικού μας κώδικα για την χρήση του προτύπου και τις κυριότερες μεθόδους που θα χρειαστούμε στην εφαρμογή μας, στο παρακάτω κώδικα δημιουργούμε μια υλοποίηση στην συγκεκριμένη οντότητα των αινήτων μας.

Δημιουργούμε την διεπαφή ακινήτου μας

```
using Core.Models;

namespace Core.Interfaces
{
	public interface IPropertyRepository<T> where T : BaseModel
	{
		Task<IReadOnlyList<T>> ListAsync(ISpecification<T> spec);
	}
}
```

Στην συνέχεια υλοποιούμε την διεπαφή

```
using Core.Interfaces;
using Core.Models;
using Infrastructure.Data;
using Microsoft.EntityFrameworkCore;
namespace Infrastructure.Repositories.PropertiesRepositories
{
	public class  PropertyRepository<T> : IPropertyRepository<T>  where T : BaseModel
	{
		private readonly DatabaseContext _context;
		public PropertyRepository(DatabaseContext context)
		{
			_context = context;
		}

		public async Task<IReadOnlyList<T>> ListAsync(ISpecification<T> spec)
		{
			// Χρήση προδιαγραφής πριν την επιστροφή των δεδομένων
			return await ApplySpecification(spec).ToListAsync();
		}

		// Δημιουργία προδιαγραφής
		private IQueryable<T> ApplySpecification(ISpecification<T> spec)
		{
			return SpecificationEvaluator<T>.GetQuery(_context.Set<T>().AsQueryable(), spec);
		}
	}
}
```

Στην συνέχεια ορίζουμε την υλοποίηση της συγκεκριμένης προδιαγραφής, δηλαδή τα στοιχεία που θα ελέγχει η προδιαγραφή που δημιουργήσαμε για τα ακίνητά μας.

```
using Core.Models.Properties;

namespace Infrastructure.Repositories.PropertiesRepositories
{
	public class PropertySpecifications : BaseSpecification<PropertyModel>
	{
		public PropertySpecifications(string? sort, int? categoryId, int? cityId, int? type)
		   : base (x =>
			(!categoryId.HasValue || x.PropertyCategoryId == categoryId) &&
			(!cityId.HasValue || x.CityId = cityId) &&
			(!type.HasValue || x.PropertyFor == (PropertyForEnum)type)
```

```
                )
        {
                AddInclude(x => x.PropertyCharacteristics!);
                AddInclude(x => x.PropertyImages!);
                AddOrderBy(x => x.LastUpdate!);

                if  (!string.IsNullOrEmpty(sort))
                {
                        switch (sort)
                        {
                                case “dateDesc”:
                                        AddOrderByDeescending(x => x.LastUpdate);
                                        break;
                                case “dateAsc”:
                                        AddOrderBy(x => x.LastUpdate);
                                        break;
                                case “priceDesc”:
                                        AddOrderByDeescending(x => x.Price);
                                        break;
                                case “priceAsc”:
                                        AddOrderBy(x => x.Price);
                                        break;
                                default:
                                        AddOrderBy(x => x.LastUpdate);
                                        break;
                        }
                }
        }
    }
}
```

Χρήση της προδιαγραφής

```
using AutoMapper;
using Core.Interfaces;
using Microsoft.AspNetCore.Mvc;
using Infrastructure.Repositories.PropertyRepositories;

namespace.API.Controllers.Properties
{
        public class PropertyController : BaseApiController
        {
                private readonly ILogger<PropertyController> _logger;
                private readonly IPropertyRepository<PropertyModel> _propertyRepo;
                private readonly IMapper _mapper;
                public PropertyController(ILogger<PropertyController> logger, IPropertyRepository propertyRepo,
IMapper mapper)
                {
                        _propertyRepo = property.Repo;
                        _logger = logger;
                        _mapper = mapper;
                }


                [AllowAnonymous] // Χρήση τελικού σημείου από όλους
                [HttpGet]
                public async Task<ActionResult<IReadOnlyList<PropertyToReturnDto>>>
GetPropertiesListAsync([FromQuery] GetPropertiesDto params)
                {
                        var spec = new PropertySpecifications(params.sort, params,categoryId, params.cityId,,
params.type);
                        var properties = await _propertyRepo.ListAsync(spec);
```

```
                    _logger.LogInformation($"List properties based on spec: {spec}");

                    return Ok(_mapper.Map<IReadOnlyList<PropertyToReturnDto>>(properties));
                }
            }
}
```

4.3.5 Ενσωμάτωση Stripe

Όπως προαναφέρθηκε για την διαχείριση των συνδρομών και των πληρωμών στην εφαρμογή μας έχει χρησιμοποιηθεί η Stripe. Η χρήση του Stripe API έχει γίνει σε ξεχωριστή βιβλιοθήκη ώστε να μπορεί να επαναχρησιμοποιηθεί και σε άλλες εφαρμογές. Η βιβλιοθήκη περιέχει ολόκληρη την υλοποίηση του API της Stripe και μια κλάση η οποία λειτουργεί ως υπηρεσία χρήσης για την μετατροπή του API και την χρήση του με βάση την λογική που χρειάζεται η εφαρμογή μας για την χρήση της.

Στον παρακάτω κώδικα αναγράφεται η κλαση που έχουμε δημιουργήσει ως υπηρεσία για την χρήση του Stripe με την λογική που χρειαζόμαστε για την χρήση της εφαρμογής μας.

```
using Stripe; // Βιβλιοθήκη Stripe

namespace StripeIntegration;

public partial class StripeService
{
   public async Task<IEnumerable<Subscription>> GetSubscriptionsAsync(string customerId,
      StripeListedSubscriptionStatus? status = null, StripePagination? pagination = null)
   {
      try
      {
         return await new SubscriptionService().ListAsync(new SubscriptionListOptions
         {
            Customer = customerId,
            Status = status?.ToStripeStatusCode(),
            Limit = pagination?.Limit,
            StartingAfter = pagination?.StartingAfterId,
         });
      }
      catch (StripeException e)
      {
         throw new StripeIntegrationException(e);
      }
   }

   // Δημιουργία μιας νέας συνδρομής
   public async Task<StripeIntentToken> CreateSubscriptionAsync(string customerId, SubscriptionSetupOptions options)
   {
      StripeIntegrationException.ThrowIfNullOrWhitespace(options.PriceId, nameof(options.PriceId));

      if (options.Unique)
      {
         var existingSubscriptions = await GetSubscriptionsAsync(customerId);

         if (existingSubscriptions.Any())
         {
            StripeIntegrationException.ThrowCustomerAlreadyHasASubscriptionException();
         }
      }

      await ClearCustomerSetupIntents(customerId);

      var subscriptionOptions = new SubscriptionCreateOptions
      {
         Customer = customerId,
```

```
        Items =
        [
            new SubscriptionItemOptions
            {
                Price = options.PriceId,
            }
        ],
        TrialSettings = new SubscriptionTrialSettingsOptions
        {
            EndBehavior = new SubscriptionTrialSettingsEndBehaviorOptions
            {
                MissingPaymentMethod = "cancel"
            },
        },
        TrialPeriodDays = options.TrialDays,
        PaymentSettings = new SubscriptionPaymentSettingsOptions
        {
            SaveDefaultPaymentMethod = "on_subscription",
        },
        PaymentBehavior = "default_incomplete",
        DefaultPaymentMethod = options.PaymentMethodId,
    };

    subscriptionOptions.AddExpand("latest_invoice.payment_intent");
    subscriptionOptions.AddExpand("pending_setup_intent");

    var subscriptionService = new SubscriptionService();
    try
    {
        var subscription = await subscriptionService.CreateAsync(subscriptionOptions);

        if (subscription.PendingSetupIntent != null)
        {
            if (!string.IsNullOrWhiteSpace(options.PaymentMethodId))
            {
                await new SetupIntentService().ConfirmAsync(subscription.PendingSetupIntent.Id);

                return new StripeConfirmedSetupIntentId(subscription.PendingSetupIntent.Id);
            }

            return new StripeSetupClientSecret(subscription.PendingSetupIntent.ClientSecret);
        }

        return new StripePaymentClientSecret(subscription.LatestInvoice.PaymentIntent.ClientSecret);
    }
    catch (StripeException e)
    {
        throw new StripeIntegrationException(e);
    }
}


// Ακύρωση συνδρομής χρήστη
public async Task CancelSubscriptionAsync(string subscriptionId)
{
    var subscriptionService = new SubscriptionService();
    try
    {
        await subscriptionService.CancelAsync(subscriptionId);
    }
    catch (StripeException e)
    {
        throw new StripeIntegrationException(e);
    }
}
```

```
  // Ανανέωση μεθόδου πληρωμής συνδρομής
   public async Task<string> UpdateSubscriptionPaymentMethodAsync(string subscriptionId, string paymentMethodId)
   {
      var subscriptionService = new SubscriptionService();
      var subscription = await subscriptionService.GetAsync(subscriptionId);

      if (subscription == null)
      {
         StripeIntegrationException.ThrowInvalidSubscriptionIdException(subscriptionId);
      }

      var paymentMethodService = new PaymentMethodService();
      var paymentMethod = await paymentMethodService.GetAsync(paymentMethodId);

      if (paymentMethod == null)
      {
         StripeIntegrationException.ThrowInvalidPaymentMethodIdException(paymentMethodId);
      }

      await subscriptionService.UpdateAsync(subscriptionId, new SubscriptionUpdateOptions
      {
         DefaultPaymentMethod = paymentMethodId,
      });

      return subscription!.DefaultPaymentMethodId;
   }
}
```

4.4 Αρχιτεκτονική και υλοποίηση Front End εφαρμογής

Για την δημιουργία του Front End χρησιμοποιήθηκε η αρχιτεκτονική σχεδίαση micro-front ends, η οποία είναι βασισμένη στην διάσπαση του front-end μιας εφαρμογής σε μικρότερες επαναχρησιμοποιούμενες μικρο εφαρμογές. Ο διαχωρισμός τους γίνεται με την δημιουργία ενοτήτων (modules). Έχουν δημιουργηθεί κάποιες συναρτήσεις για την υλοποίηση της αυθεντικοποίησης και ταυτοποίησης του χρήστη με το API της εφαρμογής μας.

Για να πετύχουμε καλύτερη εμπειρία χρήστη και στην εφαρμογή μας έχουμε χρησιμοποιήσει την τεχνολογία redux, η οποία μας βοηθά να φορτώνουμε τα δεδομένα της βάσης μας παράλληλα με την φόρτωση της σελίδας μας, για να επιταχύνουμε την απόκριση της ιστοσελίδας.

4.4.1 Micro-FrontEnds

Όπως προαναφέρθηκε για να πετύχουμε την αρχιτεκτονική των micro-front end έχουμε διαχωρίσει τις εφαρμογές μας σε μικρότερες δομές.
Η δομή φακέλων της εφαρμογής μας για το FrontEnd περιέχει τρεις φακέλους που είναι χωρισμένοι ως ενότητες. Οι φακελοι είναι:

- Core: ο οποίος περιέχει βασικά μέρη (components) τα οποία χρησιμοποιούνται σε όλη την εφαρμογή μας (πχ. headers, footer, cookies)
- Modules: όπου περιέχει όλες τις ενότητες για την λειτουργία της εφαρμογής μας (πχ. properties-list, home, single-property)
- Shared: έχουμε δημιουργήσει τον φάκελο shared καθώς στην εφαρμογή μας χρειαζόμαστε κάποια εικονίδια για την καλύτερη εμπειρία χρήστη αυτά τα εικονίδια είναι κάποια SVG εικονίδια τα οποία για να μπορέσουμε να τα τροποποιούμε δυναμικά σε μέγεθος και χρώμα ανάλογα με τις ανάγκες μας, τα έχουμε κάνει σαν ξεχωριστά μέρη (components) στον φάκελο shared και τα επαναχρησιμοποιούμε όπου χρειαστεί.

Ο διαχωρισμός των βασικών front end της εφαρμογής μας γίνεται στην δομή Modules όπου είναι αυτή που περιέχει όλα τα μικρότερα front ends που έχουμε δημιουργήσει σε υπό φακέλους και η δομή των υποφακέλων είναι:

- auth: ο οποίος φάκελος περιέχει όλα τα τελικά σημεία τα οποία διαχειρίζεται ο χρήστης κατά το σύστημα ταυτοποίησης και αυθεντικοποίησης του. Ο φάκελος περιέχει επίσης τους υποφακέλους:
  - forgot-password: σελίδα που ο χρήστης έχει ξεχάσει τον κωδικό του
  - login: σελίδα σύνδεσης χρήστη
  - register: σελίδα εγγραφής χρήστη
  - reset-password: σελίδα επαναφοράς κωδικού
- pages: ο οποίος φάκελος περιέχει όλες τις στατικές σελίδες της εφαρμογής μας και χωρίζεται στους υποφακέλους:
  - business: περιέχει την σελίδα που έχει τα απαραίτητα στοιχεία για την εγγραφή κάποιου μεσιτικού γραφείου στην ιστοσελίδα μας.
  - contact: σελίδα επικοινωνίας
  - cookies-policy: σελίδα πολιτικής cookies
  - home: αρχική σελίδα
  - not-found: σελίδα 404
  - privacy-policy: σελίδα πολιτικής απορρήτου
  - terms-of-use: σελίδα όρων χρήσης
- profile: περιέχει τις απαραίτητες σελίδες για το προφίλ των χρηστών και χωρίζεται στους υποφακέλους:
  - complete-profile: μετά την εγγραφή του χρήστη έχουμε μια δεύτερη φόρμα στην οποία ζητάμε περισσότερες πληροφορίες για τον χρήστη
  - personal-settings: η σελίδα στην οποία ο χρήστης βλέπει τις ρυθμίσεις του προφίλ του και τα στοιχεία του τα οποία μπορεί και να επεξεργαστεί
  - profile: όπου περιέχει το προφίλ του χρήστη
  - subscriptions: όπου μπορεί να δει και να διαχειριστεί τις συνδρομές του
- properties: περιέχει όλες τις απαραίτητες σελίδες για τα ακίνητα και χωρίζεται στους υποφακέλους:
  - add-property: σελίδα προσθήκης ακινήτου
  - favourites: μέρος (component) διαχείρισης αγαπημένων
  - properties-list: σελίδα προβολής λίστας ακινήτων
  - property: σελίδα ακινήτου
- shop: περιέχει όλες τις απαραίτητες σελίδες για την διαχείριση των συνδρομών στην εφαρμογή μας και έχε ως υποφακέλους τους:
  - cart: σελίδα καλαθιού
  - checkout: σελίδα πληρωμής συνδρομής
  - plans: σελίδα προβολής πλάνων συνδρομών

Τα micro-frontEnds δηλώνονται και φορτώνται ως ενότητες (modules) στην σελίδα που ορίζουμε την δρομολόγηση των σελίδων μας. Στον παρακάτω κώδικα φαίνεται η δήλωση των ενοτήτων.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
```

```
import { CoreModule } from './core/core.module';
import { StoreModule } from '@ngrx/store';
import { metaReducers, reducers } from './reducers';
import { StoreDevtoolsModule } from '@ngrx/store-devtools';
import { EffectsModule } from '@ngrx/effects';
import { HttpClientModule } from '@angular/common/http';
import { RouterModule, Routes } from '@angular/router';
import { environment } from 'src/environments/environment';
import { EntityDataModule } from '@ngrx/data';
import { RouterState, StoreRouterConnectingModule } from '@ngrx/router-store';
import { AuthModule } from './modules/auth/auth.module';
import { NotFoundComponent } from './modules/pages/not-found/not-found.component';
import { AuthGuard } from './modules/auth/services/auth.guard';

// Δήλωση ενοτήτων στην δρομολόγηση της ιστοσελίδας μας
const routes: Routes = [
  {
    path: '',
    loadChildren: () => import('./modules/pages/pages.module').then(m => m.PagesModule)
  },
  {
    path: 'auth',
    loadChildren: () => import('./modules/auth/auth.module').then(m => m.AuthModule)
  },
  {
    path: 'profile',
    loadChildren: () => import('./modules/profile/profile.module').then(m => m.ProfileModule),
    canActivate: [AuthGuard]
  },
  {
    path: 'properties',
    loadChildren: () => import('./modules/properties/properties.module').then(m => m.PropertiesModule)
  },
  {
    path: 'shop',
    loadChildren: () => import('./modules/shop/shop.module').then(m => m.ShopModule)
  },
  // 404
  {
    path: '**',
    pathMatch: 'full',
    component: NotFoundComponent
  }
];

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    CoreModule,
    HttpClientModule,
    RouterModule.forRoot(routes, {}),
    AuthModule.forRoot(),
    StoreModule.forRoot(reducers, {
      metaReducers,
      runtimeChecks: {
        strictActionImmutability: true,
        strictStateImmutability: true,
        strictActionSerializability: true,
        strictStateSerializability: true,
      }
    }),
    StoreDevtoolsModule.instrument({
      maxAge: 25,
      logOnly: !environment.production
```

```
    }),
    EffectsModule.forRoot([]),
    EntityDataModule.forRoot({}),
    StoreRouterConnectingModule.forRoot({
      stateKey: 'router',
      routerState: RouterState.Minimal
    })
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Στον παρακάτω κώδικα φαίνεται η δρομολόγηση που υπάρχει σε κάθε ενότητα για τις υποσελίδες της ενότητας μας.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { PlansComponent } from './plans/plans.component';
import { CartComponent } from './cart/cart.component';
import { CheckoutComponent } from './checkout/checkout.component';
import { OrderOverviewComponent } from './order-overview/order-overview.component';
import { RouterModule, Routes } from '@angular/router';
import { IconsModule } from 'src/app/shared/icons/icons.module';
import { EntityDataService, EntityDefinitionService, EntityMetadataMap } from '@ngrx/data';
import { PlanEntityService } from './services/plan-entity.service';
import { PlansDataService } from './services/plans-data.service';
import { PlansResolver } from './services/plans.resolver';
import { ThankYouComponent } from './thank-you/thank-you.component';

// Δήλωση υποσελίδων ενότητας
const routes: Routes = [
  {
    path: 'plans',
    component: PlansComponent,
    resolve: {
      plan: PlansResolver
    }
  },
  {
    path: 'cart',
    component: CartComponent
  },
  {
    path: 'checkout',
    component: CheckoutComponent
  },
  {
    path: 'review',
    component: OrderOverviewComponent
  },
  {
    path: 'thank-you',
    component: ThankYouComponent
  }
]

// Entity metadata
const entityMetadata: EntityMetadataMap = {
  Plan: {}
};

@NgModule({
  declarations: [
    PlansComponent,
```

```
    CartComponent,
    CheckoutComponent,
    OrderOverviewComponent,
    ThankYouComponent,
  ],
  imports: [
    CommonModule,
    RouterModule.forChild(routes),
    IconsModule
  ],
  providers: [
    PlanEntityService,
    PlansDataService,
    PlansResolver
  ]
})
export class ShopModule {

  constructor(
    private eds: EntityDefinitionService,
    private entityDataService: EntityDataService,
    private plansService: PlansDataService
  ) {
    eds.registerMetadataMap(entityMetadata);
    entityDataService.registerService('Plan', plansService);
  }
}
```

4.4.3 Redux

Όπως έχει αναφερθεί το redux είναι μια βιβλιοθήκη διαχείρισης κατάστασης της εφαρμογής μας και χρησιμοποιείται κυρίως για την αποθήκευση δεδομένων σε ένα κεντρικό σημείο της εφαρμογής μας ώστε να είναι εύκολα προσβάσιμα από κάθε μέρος (component) της εφαρμογής μας.
Στον παρακάτω κώδικα εξηγείται η διαδικασία που χρησιμοποιείται το Redux κατά την χρήση του component όπου διαχειριζόμαστε τα πλάνα της εφαρμογής μας, ώστε να μπορούμε τα πλάνα να τα έχουμε διαθέσιμα και στα τελικά σημεία δημιουργίας της συνδρομής μας.

Στον παρακάτω κώδικα φαίνεται η κλάση PlanEntityService η οποία ορίζει την οντότητα (Entity) των πλάνων για τις συνδρομές μας.

```
// Βιβλιοθήκη ngrx η οποία χρησιμοποείται για το Reduz
import { EntityCollectionServiceBase, EntityCollectionServiceElementsFactory } from "@ngrx/data";
import { Injectable } from "@angular/core";
import { Plan } from "../models/plan.model";

@Injectable()
export class PlanEntityService extends EntityCollectionServiceBase<Plan> {

  // Δήλωση οντότητας Plan
   constructor(serviceElementsFactory: EntityCollectionServiceElementsFactory) {
      super('Plan', serviceElementsFactory);
   }
}
```

Στον παρακάτω κώδικα δηλώνουμε στο Redux από που θα αντλεί τα δεδομένα για να δημιουργήσει την οντότητα Plan.

```
import { Injectable } from "@angular/core";
import { DefaultDataService, HttpUrlGenerator } from "@ngrx/data";
import { HttpClient } from "@angular/common/http";
import { Observable } from "rxjs";
import { environment } from "src/environments/environment";
```

```
import { Plan } from "../models/plan.model";

@Injectable()
export class PlansDataService extends DefaultDataService<Plan> {


   constructor(http: HttpClient, httpUrlGenerator: HttpUrlGenerator) {
      super('Plan', http, httpUrlGenerator);
   }

   // HTTP  αίτημα στο API μας για να πάρουμε την λίστα με τα πλάνα συνδρομών
   override getAll(): Observable<Plan[]> {
      return this.http.get<Plan[]>(`${environment.apiUrl}/Plans`);
   }
}
```

Στην παρακάτω κλάση έχουμε ένα Resolver ο οποίος κατά την εκτέλεση του δημιουργεί την οντότητα Plan. Πριν την δημιουργία της οντότητας ελέγχεται αν έχει ξαναδημιουργηθεί η όχι στην συνεδρεία του χρήστη.

```
import { Injectable } from "@angular/core";
import { ActivatedRouteSnapshot, Resolve, RouterStateSnapshot } from "@angular/router";
import { Observable, filter, first, tap } from "rxjs";
import { PlanEntityService } from "./plan-entity.service";

@Injectable()
export class PlansResolver implements Resolve<boolean> {
   constructor(private planService: PlanEntityService) { }

   resolve(route: ActivatedRouteSnapshot,
      state: RouterStateSnapshot): Observable<boolean> {
      return this.planService.loaded$
         .pipe(
            tap(loaded => {
               if (!loaded) {
                  // αν δεν έχει φορτωθεί η οντότητα έτοιμα στην κλάση να φορτωθεί
                  this.planService.getAll();
               }
            }),
            filter(loaded => !!loaded),
            first()
         );
   }
}
```

Στον παρακάτω απόσπασμα κώδικα (είναι ίδιος με την προηγούμενη ενότητα) φαίνεται η χρήση του Resolver που δημιουργήσαμε κατα την φόρτωση του component όπου περιέχει τα πλάνα συνδρομών μας.

```
…
const routes: Routes = [
  {
    path: 'plans',
    component: PlansComponent,
    resolve: {
      plan: PlansResolver // Εκτελούμε την κλάση resolve
    }
  }

// Δήλωση Redux και resolver στην κλάση μας
```

```
// Entity metadata
const entityMetadata: EntityMetadataMap = {
  Plan: {}
};

@NgModule({
…
providers: [
    PlanEntityService,
    PlansDataService,
    PlansResolver
  ]
})

// Εκτέλεση Redux στην ενότητα Shop
export class ShopModule {

  constructor(
    private eds: EntityDefinitionService,
    private entityDataService: EntityDataService,
    private plansService: PlansDataService
  ) {
    eds.registerMetadataMap(entityMetadata);
    entityDataService.registerService('Plan', plansService);
  }
}
```

4.4.4 Αυθεντικοποίηση χρήστη στο Front End

Κατά την αυθεντικοποίηση του χρήστη στο Front End χρησιμοποιούμε πάλι το Redux καθώς είναι ένα σύστημα διαχείρισης κατάστασης και διαχειριζόμαστε την κατάσταση του χρήστη αν είναι συνδεδεμένος ή όχι.

Στα παρακάτω αποσπάσματα κώδικα φαίνεται η δήλωση των τύπων και των δράσεων των ενεργειών για την κατάσταση του.

```
// αρχείο action-types.ts
import * as AuthActions from './auth.actions';

export { AuthActions }

// αρχείο auth.actions.ts
import { createAction, props } from "@ngrx/store";
import { User } from "../models/user.model";

export const login = createAction(
    "[Login Page] User Login",
    props<{ user: User }>()
)

export const logout = createAction(
    "[Top Menu] Logout"
)

// αρχείο auth.selectors.ts, δήλωση κατάστασης χρήστη
import { createFeatureSelector, createSelector } from "@ngrx/store";
import { AuthState } from "../reducers"

export const selectAuthState =
    createFeatureSelector<AuthState>("auth");

export const isLoggedIn = createSelector(
    selectAuthState,
    auth => !!auth.user
);
```

```
export const isLoggedOut = createSelector(
    isLoggedIn,
    loggedIn => !loggedIn
);

// αρχείο index.ts όπου ορίζει τον reducer για την κατάσταση
import { createReducer, on } from "@ngrx/store";
import { User } from "../models/user.model";
import { AuthActions } from "../services/action-types";

export interface AuthState {
    user: User | undefined;
}

export const initialAuthState: AuthState = {
    user: undefined
}

export const authReducer = createReducer(
    initialAuthState,

    // login
    on(AuthActions.login, (state, action) => {
        return {
            user: action.user
        }
    }),

    on(AuthActions.logout, (state, action) => {
        return {
            user: undefined
        }
    })
)
```

Αφού έχουμε ορίσει ότι χρειαζόμαστε για την κατάσταση, δημιουργούμε την κλάση που διαχειρίζεται την σύνδεση και την αποσύνδεση του χρήστη.

```
import { Injectable } from "@angular/core";
import { Router } from "@angular/router";
import { Actions, createEffect, ofType } from "@ngrx/effects";
import { AuthActions } from "./action-types";
import { tap } from "rxjs";

@Injectable()
export class AuthEffects {

    // login
    login$ = createEffect(() =>
        this.actions$
            .pipe(
                ofType(AuthActions.login),
                tap(action => localStorage.setItem('user', JSON.stringify(action.user))) // αποθήκευση token χρήστη
            ),
        { dispatch: false }
    );

    // logout
    logout$ = createEffect(() =>
        this.actions$
            .pipe(
                ofType(AuthActions.logout),
```

```
            tap(action => {
              localStorage.removeItem('user'); // διαγραφή token χρήστη
              this.router.navigateByUrl('/');
            })
          ),
      { dispatch: false }
    );

    constructor(private actions$: Actions, private router: Router) { }
}
```

Επιπλέον έχουμε δημιουργήσει και την κλάση AuthGuard στην οποία ορίζουμε τι μπορεί να επισκεφθει ο χρήστης όταν είναι συνδεδεμένος.

```
import { Injectable } from "@angular/core";
import { ActivatedRouteSnapshot, CanActivate, CanActivateFn, Router, RouterStateSnapshot, UrlTree } from
"@angular/router";
import { Store, select } from "@ngrx/store";
import { Observable, tap } from "rxjs";
import { AppState } from "src/app/reducers";
import { isLoggedIn } from "./auth.selectors";

@Injectable()
export class AuthGuard {

    constructor(
        private store: Store<AppState>,
        private router: Router
    ) {

    }

    canActivate(
        route: ActivatedRouteSnapshot,
        state: RouterStateSnapshot): Observable<boolean> {

        // Ελέγχει αν ο χρήστης είναι συνδεδεμένος, αν είναι τότε του επιτρέπει να δει την σελίδα αλλιώς τον στέλνει στην
σελίδα σύνδεσης
        return this.store
            .pipe(
                select(isLoggedIn),
                tap(loggedIn => {
                    if (!loggedIn) {
                        this.router.navigateByUrl('/auth/login');
                    }
                })
            )
    }
}
```

Στο παρακάτω απόσπασμα κώδικα φαίνεται η ενεργοποίηση του AuthGuard σε κάποια σελίδα.

```
const routes: Routes = [
{
    path: 'profile',
    loadChildren: () => import('./modules/profile/profile.module').then(m => m.ProfileModule),
    canActivate: [AuthGuard] // αν δεν είναι συνδεδεμένος ο χρήστης δεν μπορεί να μπει στην σελίδα του προφίλ
  }
]
```

4.4.5 Ενσωμάτωση Stripe στο Front End

Η ενσωμάτωση του Stripe για την διαχείριση πληρωμών στο FrontEnd χρησιμοποιούμε μια βιβλιοθήκη της stripe για χρήση σε angular. Το Stripe χρησιμοποιεί iframes για την δημιουργία της φόρμας διαχείρισης καρτών.
Στο παρακάτω απόσπασμα κώδικα φαίνεται η χρήση της βιβλιοθήκης του Stripe.

```
import { HttpClient } from '@angular/common/http';
import { inject, Injectable } from '@angular/core';
import { loadStripe, Stripe, StripeElements, StripePaymentElement } from '@stripe/stripe-js';
import { User } from 'src/app/modules/auth/models/user.model';
import { Basket } from 'src/app/modules/shop/models/basket.model';
import { BasketService } from 'src/app/modules/shop/services/basket.service';
import { environment } from 'src/environments/environment';

@Injectable({
  providedIn: 'root'
})
export class StripeService {
  private http = inject(HttpClient);
  private basketService = inject(BasketService);
  private stripe: Promise<Stripe | null>;
  private elements?: StripeElements;
  private paymentElement?: StripePaymentElement;
  basket?: Basket;

  constructor() {
    this.stripe = loadStripe(environment.stripeKey);
  }

    getStripeInstance() {
        return this.stripe;
  }

  // Δημιουργία iframe φόρμας καρτών
  async initializeElements() {
    if (!this.elements) {
      const stripe = await this.getStripeInstance();
      this.basket = await this.getCurrentBasketSource();
      if (stripe) {
        // console.log(basket);
          this.elements = stripe.elements({
            locale: "it-IT",
            mode: "subscription",
            amount: this.basket.plan.pricePerMonth,
            currency: "eur",
            appearance: {
              labels: "floating"
            }
          })
      } else {
        throw new Error("Stripe has not been loaded")
      }
    }
    return this.elements;
  }

  // συνάρτηση εκτέλεσης πληρωμής
  async confirmPayment() {
    const stripe = await this.getStripeInstance();
    const customer = await this.createCustomer();
    // console.log("Customer", customer);
    const basket = await this.createOrUpdatePaymentIntent();
```

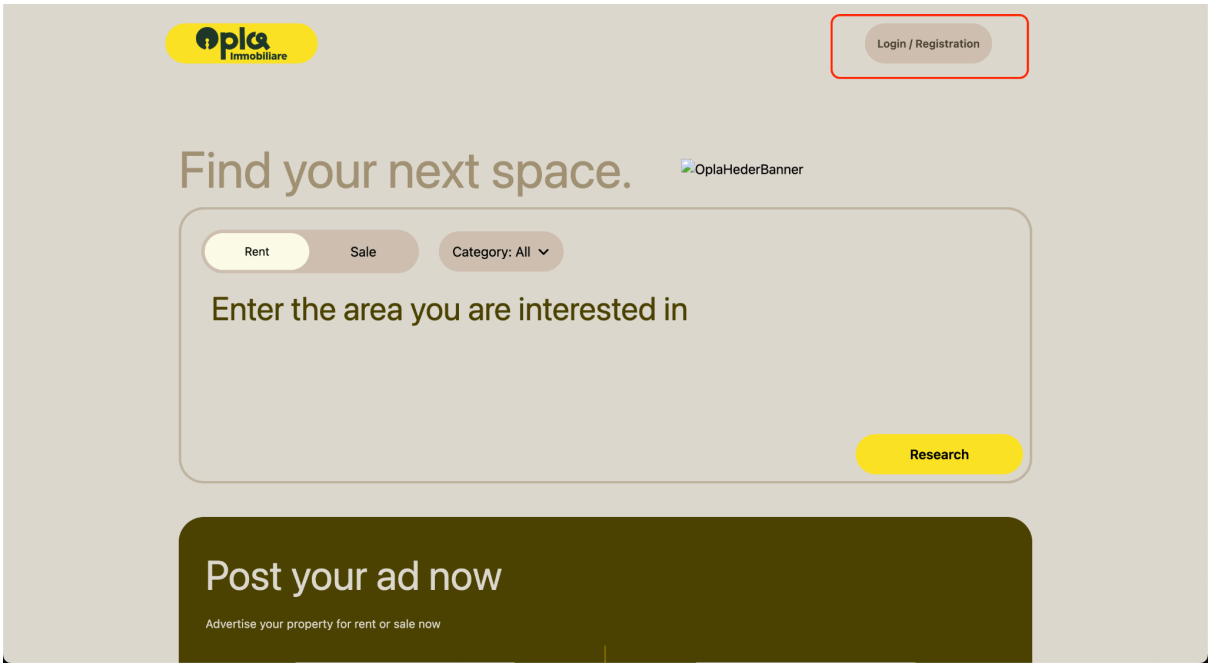
```
  if (stripe) {
    await this.elements?.submit();
    await stripe.confirmSetup({
      clientSecret: <string>basket!.stripeToken,
      elements: this.elements,
      confirmParams: {
        return_url: `${environment.clientUri}'/shop/thank-you'`
      },
    })
  }
 }
}
```

4.5 Χρήση Εφαρμογής

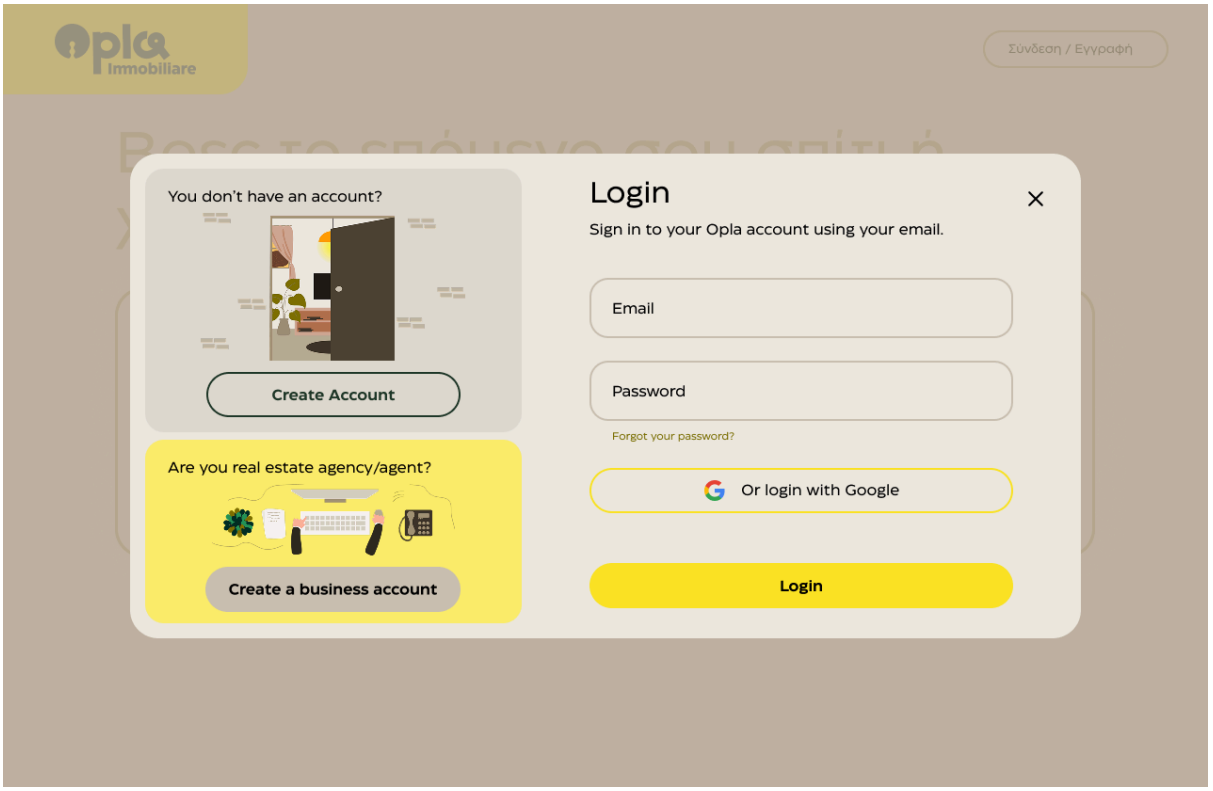
Παρακάτω περιγράφεται βήμα - βήμα η διαδικασία χρήσης και περιήγησης της ιστοσελίδας

4.5.1 Σύνδεση - Εγγραφή - Ανάκτηση Κωδικού

Ένα χρήστης κατά την επίσκεψή του στην ιστοσελίδα βλέπει την αρχική σελίδα που φαίνεται στην Εικόνα 4.6. Πατώντας πάνω δεξιά στο κουμπί Σύνδεση/Εγγραφή του εμφανίζεται ένα αναδυόμενο παράθυρο (Εικόνα 4.7) το οποίο περιέχει τα την φόρμα σύνδεσης και τα κουμπιά για εγγραφή ως απλός χρήστης ή μεσιτικό γραφείο ή να κάνει επαναφορά κωδικού πρόσβασης.



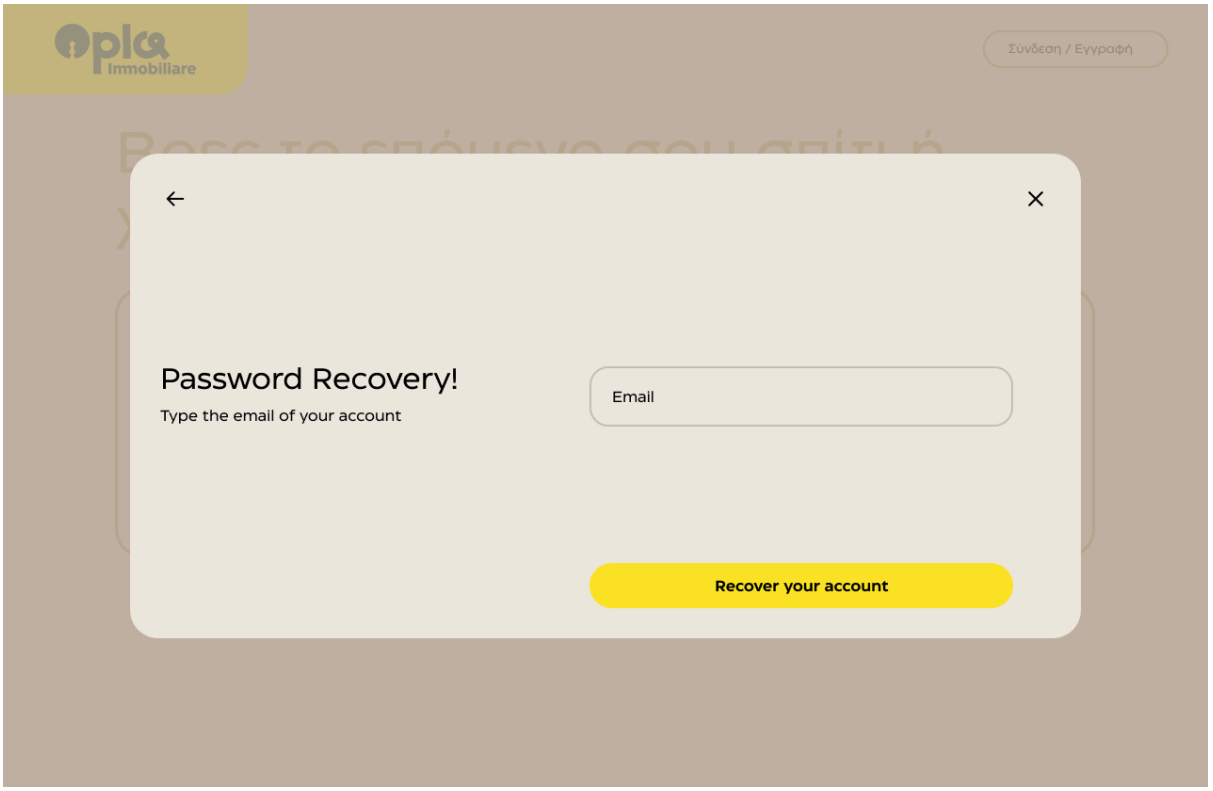
Εικόνα 4.6: Αρχική σελίδα ιστοσελίδας



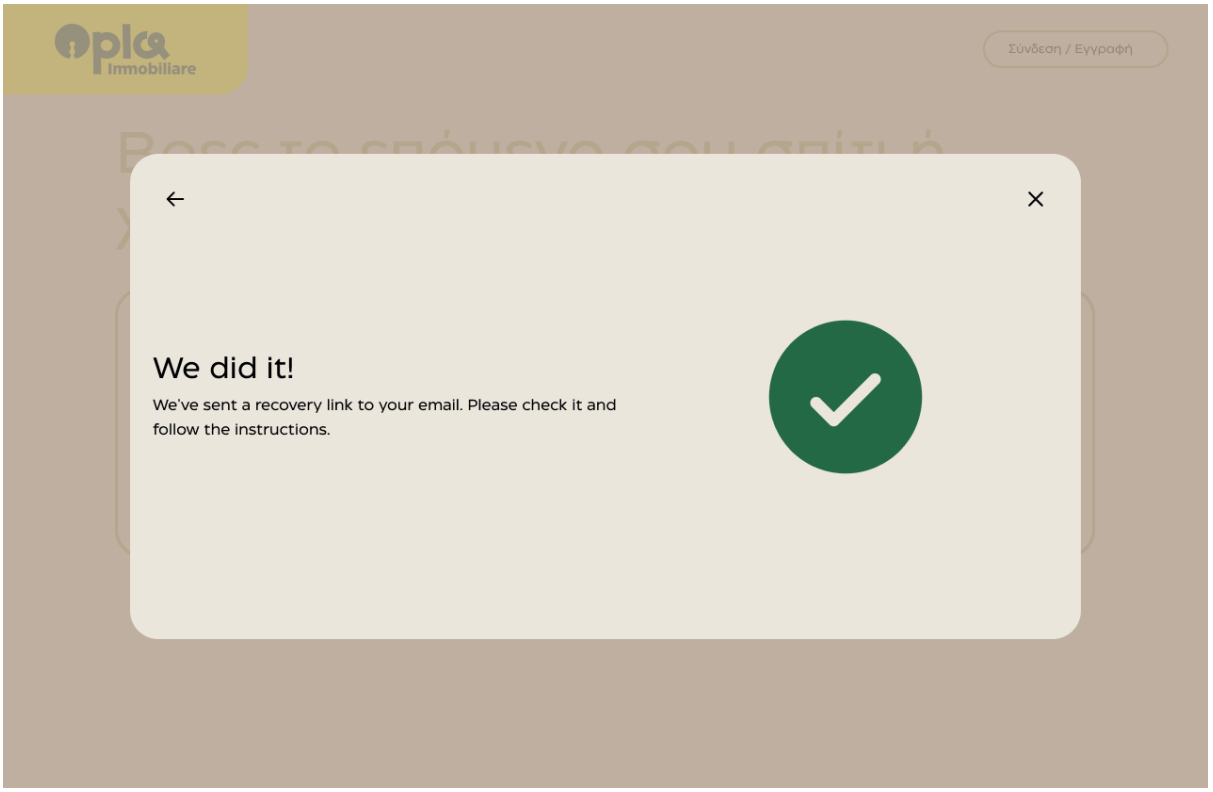
Εικόνα 4.7: Αναδυόμενο παράθυρο που ανοίγει πατώντας το κουμπί σύνδεσης

4.5.1.1 Επαναφορά κωδικού

Πατώντας ο χρήστης το κουμπί "Ξέχασες τον κωδικό σου" κάτω από πεδίο κωδικός το αναδυόμενο παράθυρο αλλάζει μορφή και εμφανίζει στον χρήστη μια φόρμα (Εικόνα 4.8). Αν ο χρήστης βρεθεί κατά την αναζήτηση στην βάση τότε εμφανίζεται ότι ο χρήστης θα πρέπει να δει το μέιλ του και να ακολουθήσει την διαδικασία επαναφοράς κωδικού (Εικόνα 4.9).

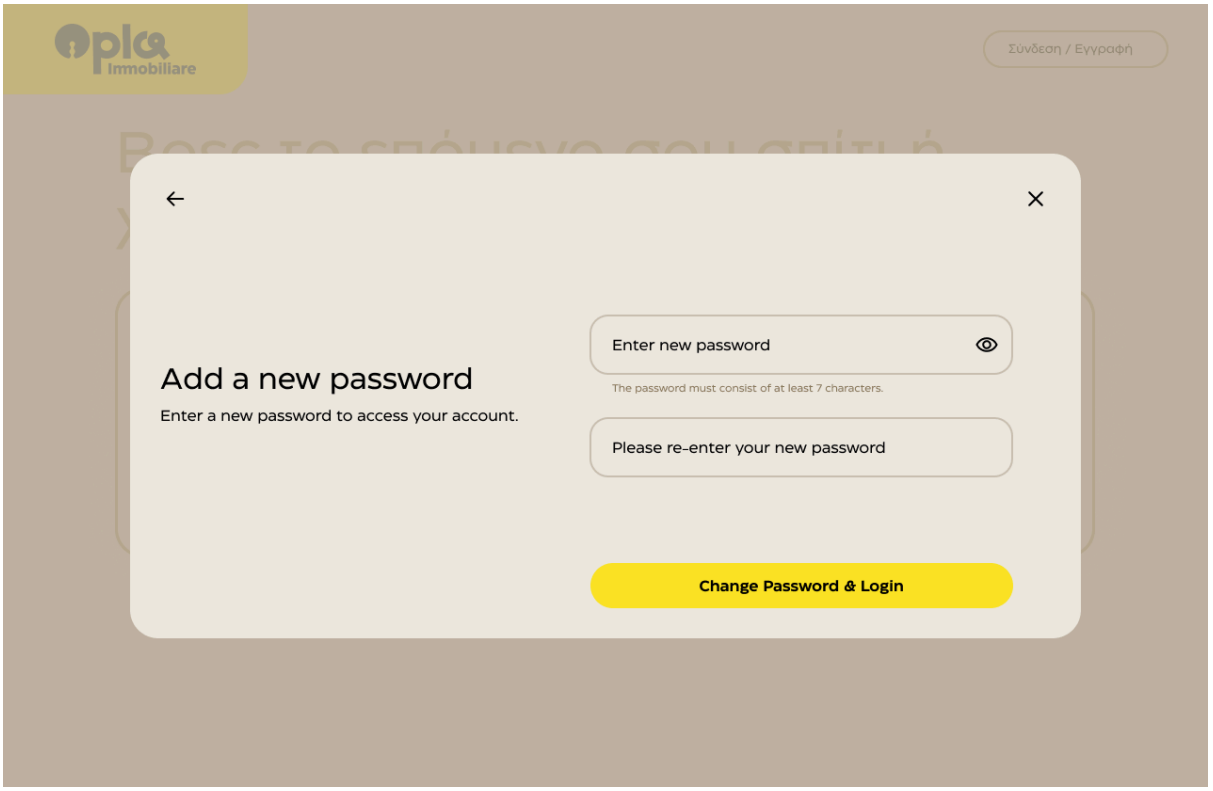


Εικόνα 4.8: Φόρμα email για την επαναφορά κωδικού



Εικόνα 4.9: Επιτυχής αποστολή email επαναφοράς κωδικού

Ακολουθώντας την διαδικασία επαναφοράς κωδικού από το email του ο χρήστης ανακατευθύνεται στην σελίδα δημιουργίας νέου κωδικού σύνδεσης για τον λογαριασμό του, Εικόνα 4.10.

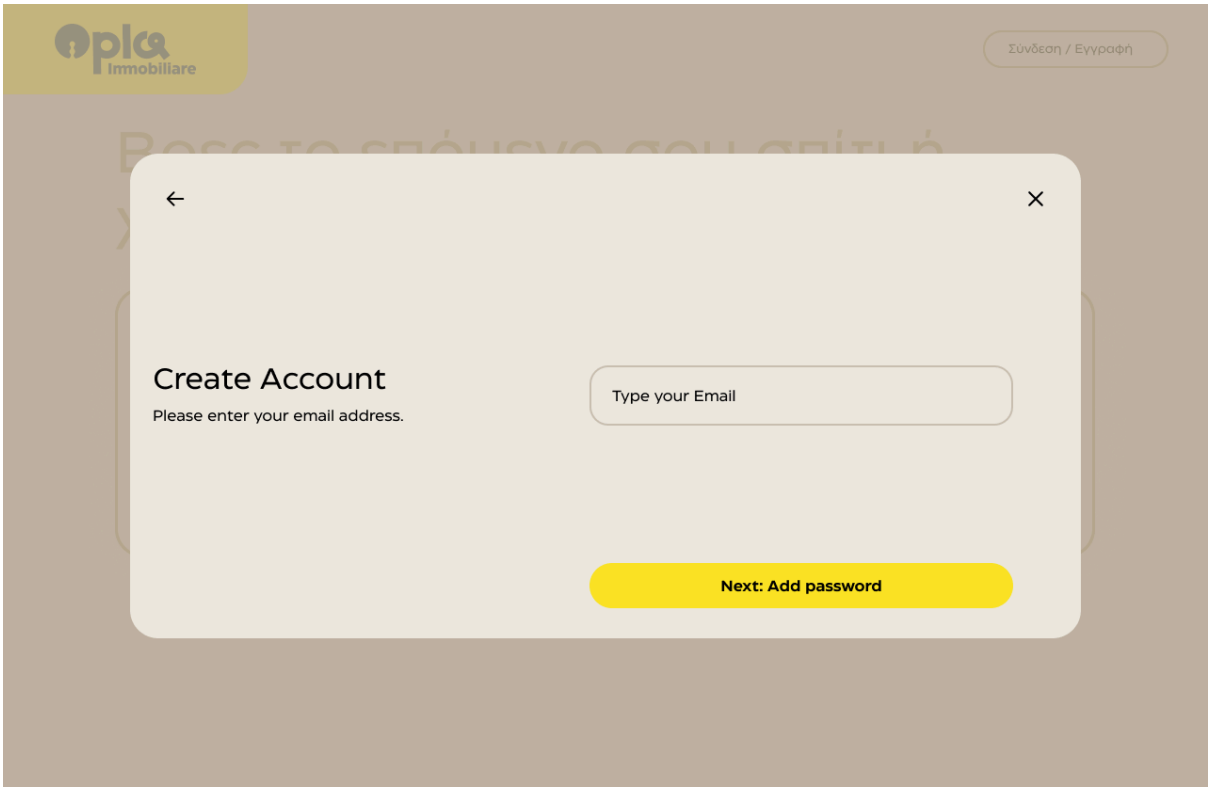


Εικόνα 4.10: Επιτυχής αποστολή email επαναφοράς κωδικού

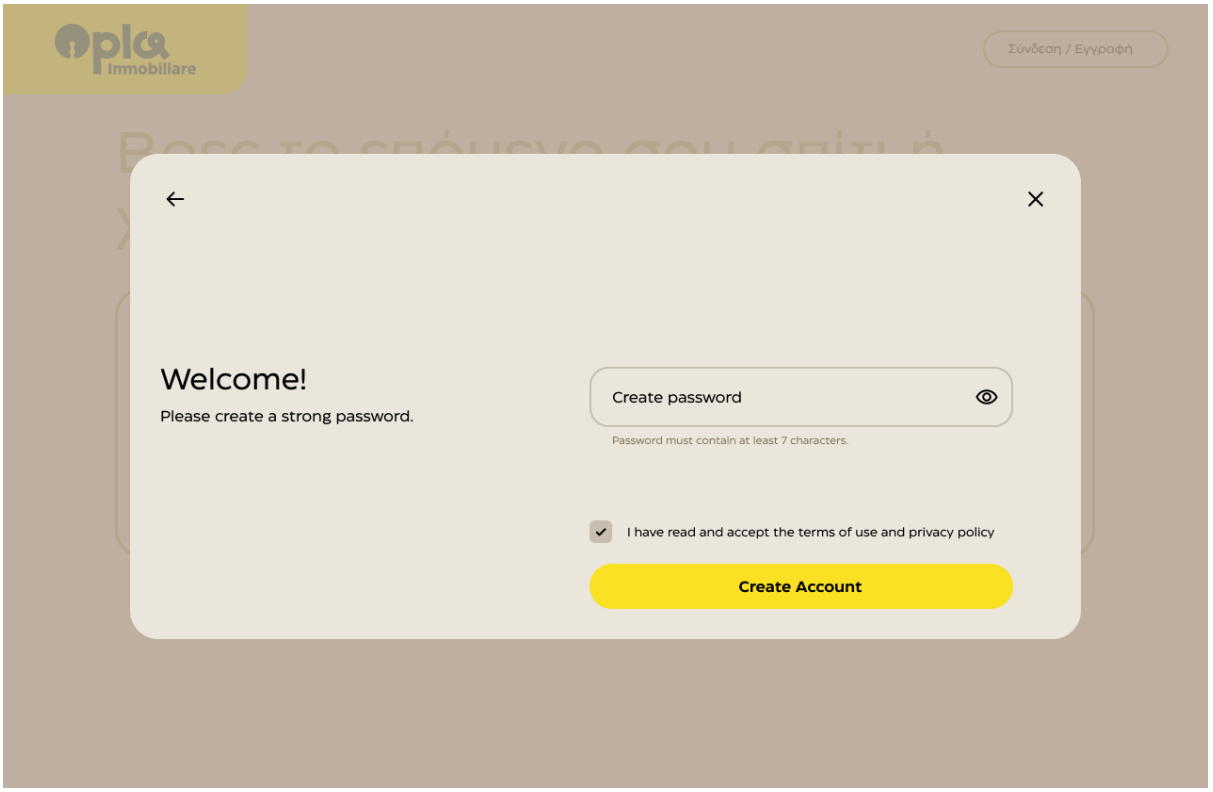
Μετά την επιτυχή αλλαγή κωδικού ο χρήστης συνδέεται αυτόματα και ανακατευθύνεται στην σελίδα του προφίλ του.

4.5.1.2 Εγγραφή απλού χρήστη

Ο χρήστης για την εγγραφή του από το αναδυόμενο παράθυρο πατάει “Δημιουργία Λογαριασμού” και εμφανίζεται η φόρμα εισαγωγής email (Εικόνα 4.11). Πατώντας “Επόμενο: Δημιουργία Κωδικού” εμφανίζεται στον χρήστη η φόρμα εισαγωγής κωδικού πρόσβασης (Εικόνα 4.12).

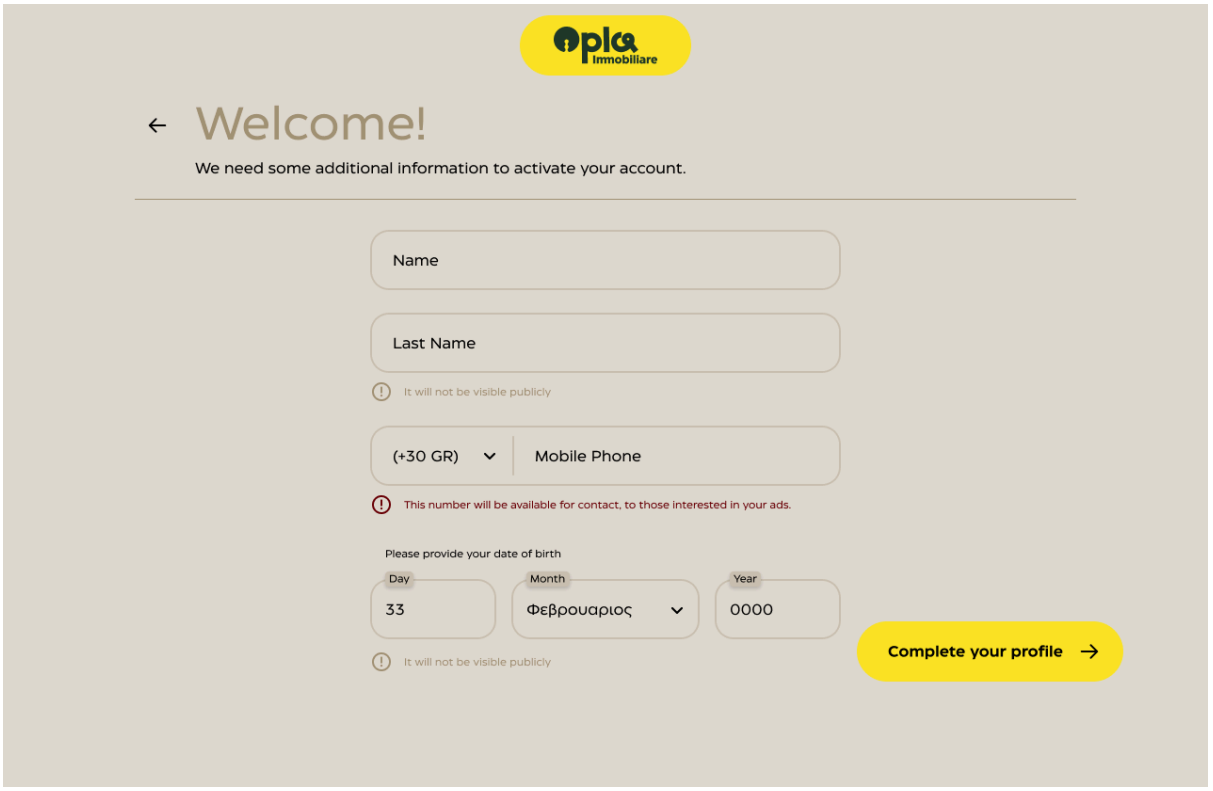


Εικόνα 4.11: Φόρμα εισαγωγής email εγγραφής



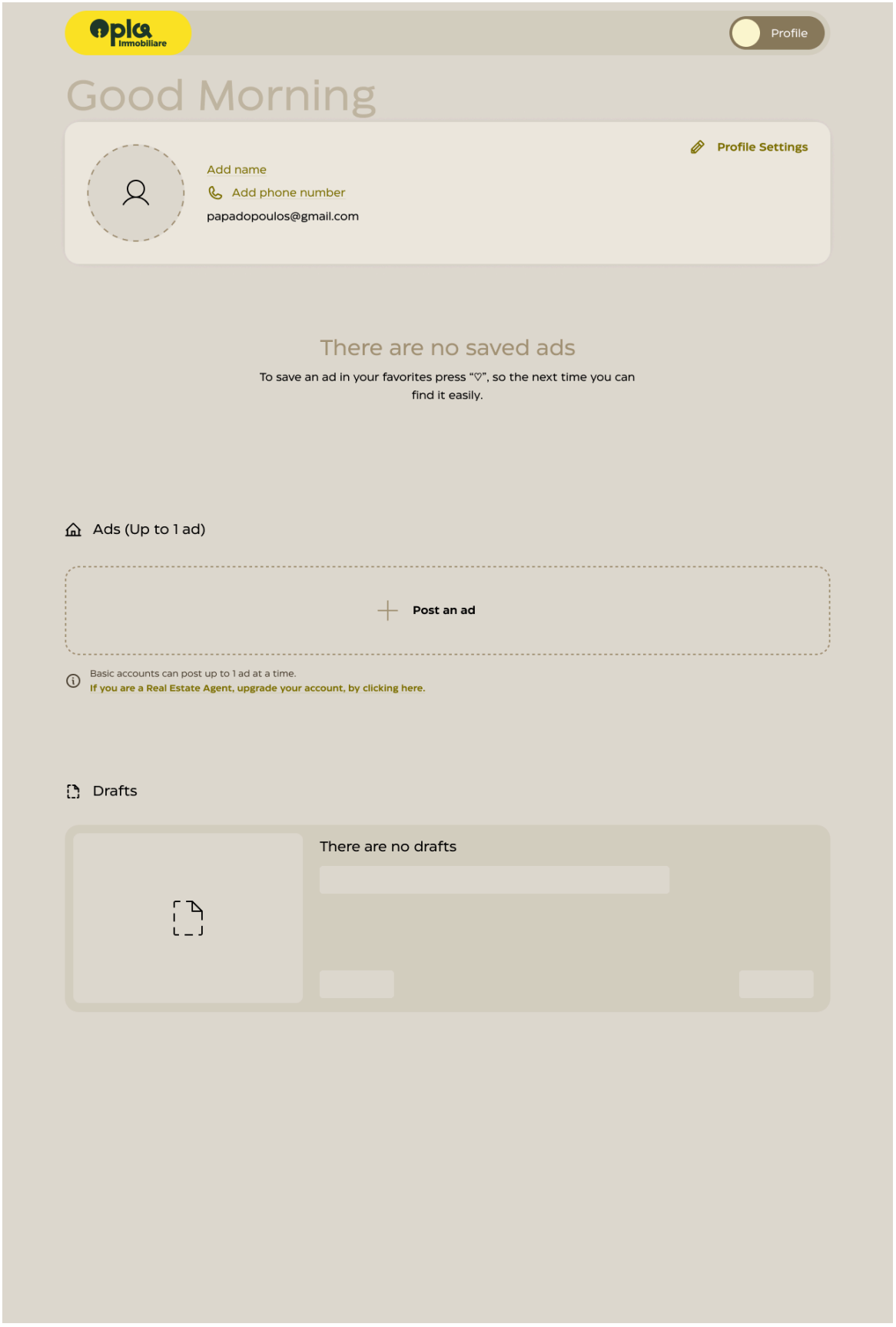
Εικόνα 4.12: Φόρμα εισαγωγής κωδικού εγγραφής

Μετά την επιτυχή εγγραφή του χρήστη, ο χρήστης συνδέεται αυτόματα και ανακατευθύνεται στην σελίδα ολοκλήρωσης του προφίλ του η οποία είναι υποχρεωτική για να μπορέσει να χρησιμοποιήσει την ιστοσελίδα μας ως συνδεδεμένος χρήστης. Του ζητείται να συμπληρωσει κάποια περισσότερα στοιχεία επικοινωνίας για αυτόν (Εικόνα 4.13).



Εικόνα 4.13: Σελίδα ολοκλήρωσης προφίλ χρήστη.

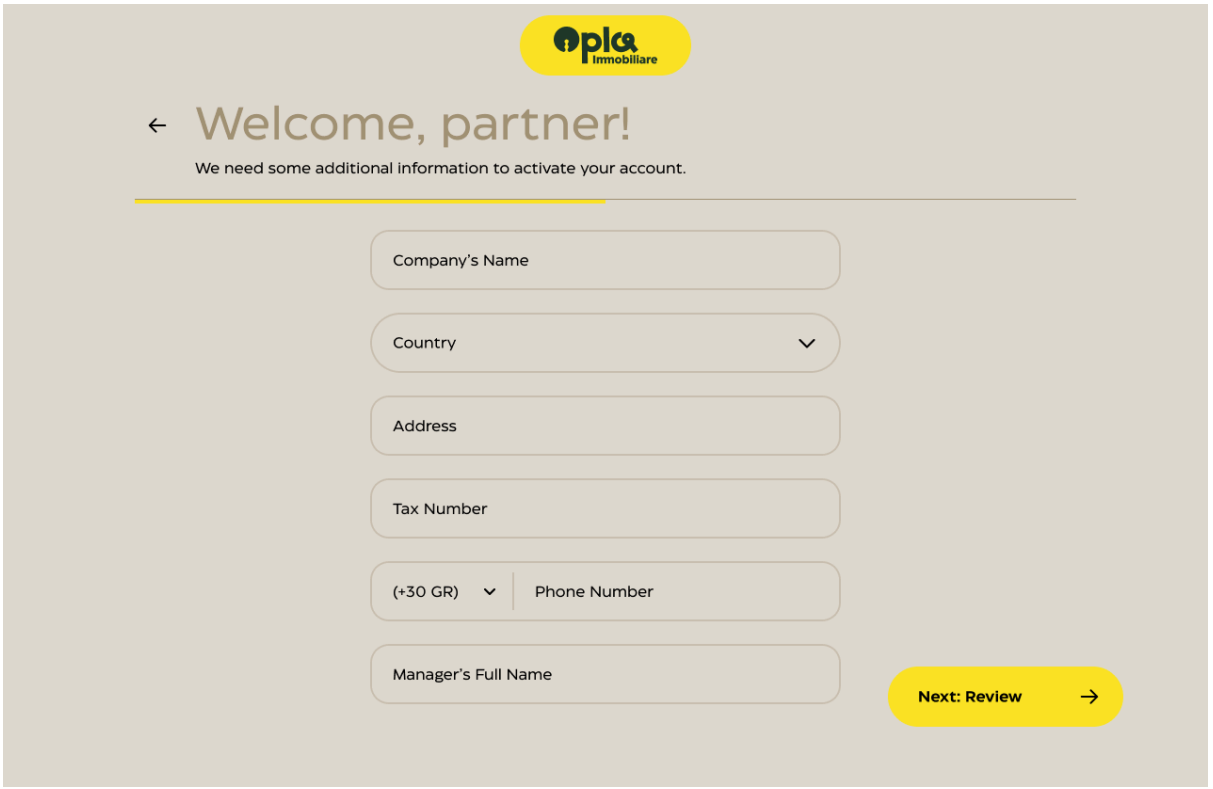
Μετά την επιτυχή συμπλήρωση της φόρμα εγγραφής ο χρήστης μεταφέρεται στην σελίδα του προφίλ του (Εικόνα 4.14).



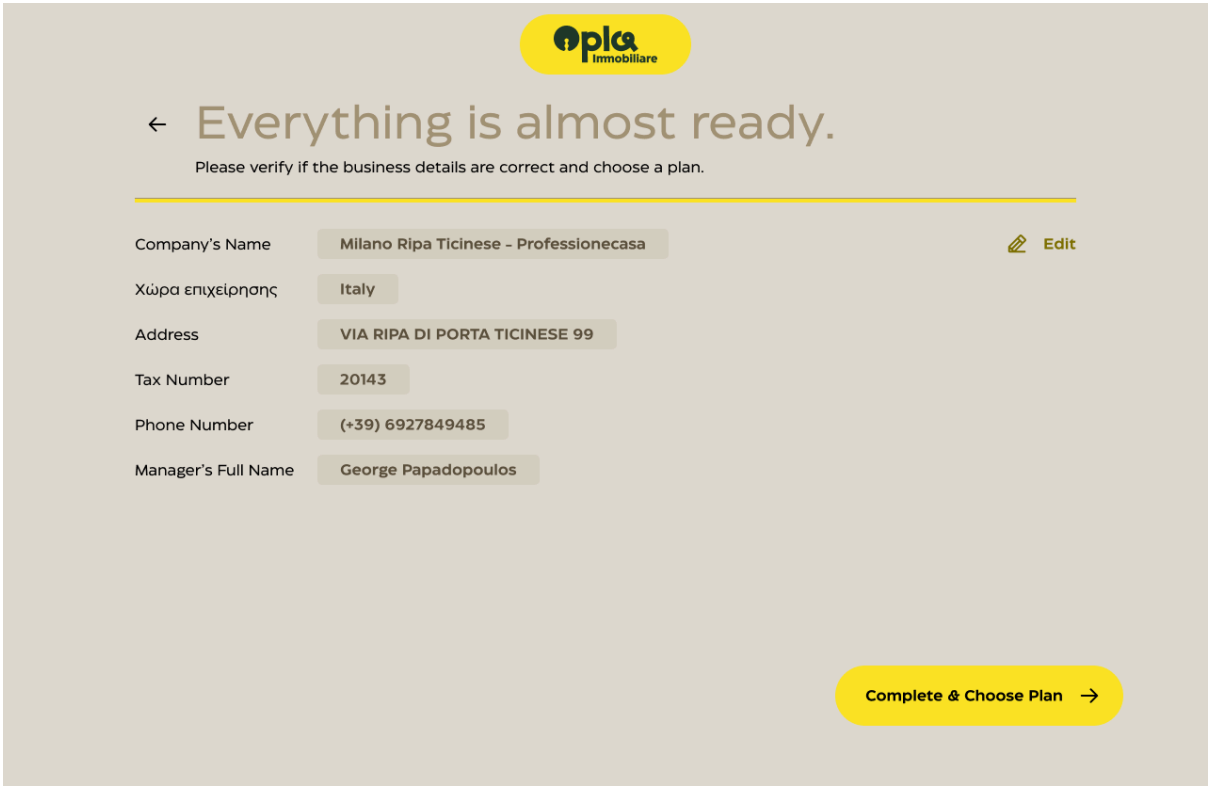
Εικόνα 4.14: Σελίδα προφίλ χρήστη

4.5.1.3 Εγγραφή ως επιχείρηση

Η διαδικασία εγγραφής ως επιχείρηση είναι η ίδια ακριβώς με την εγγραφή χρήστη. Το μόνο που αλλάζει είναι πως στην σελίδα ολοκλήρωσης προφίλ, ζητούνται στην φόρμα εισαγωγής στοιχεία σχετικά με την επιχείρηση και όχι στοιχεία χρήστη όπως την προηγούμενη περίπτωση (Εικόνα 4.15).



Εικόνα 4.15: Σελίδα ολοκλήρωσης προφίλ επιχείρησης



Εικόνα 4.16: Σελίδα επιβεβαίωσης στοιχείων που εισάχθηκαν στην προηγούμενη φόρμα

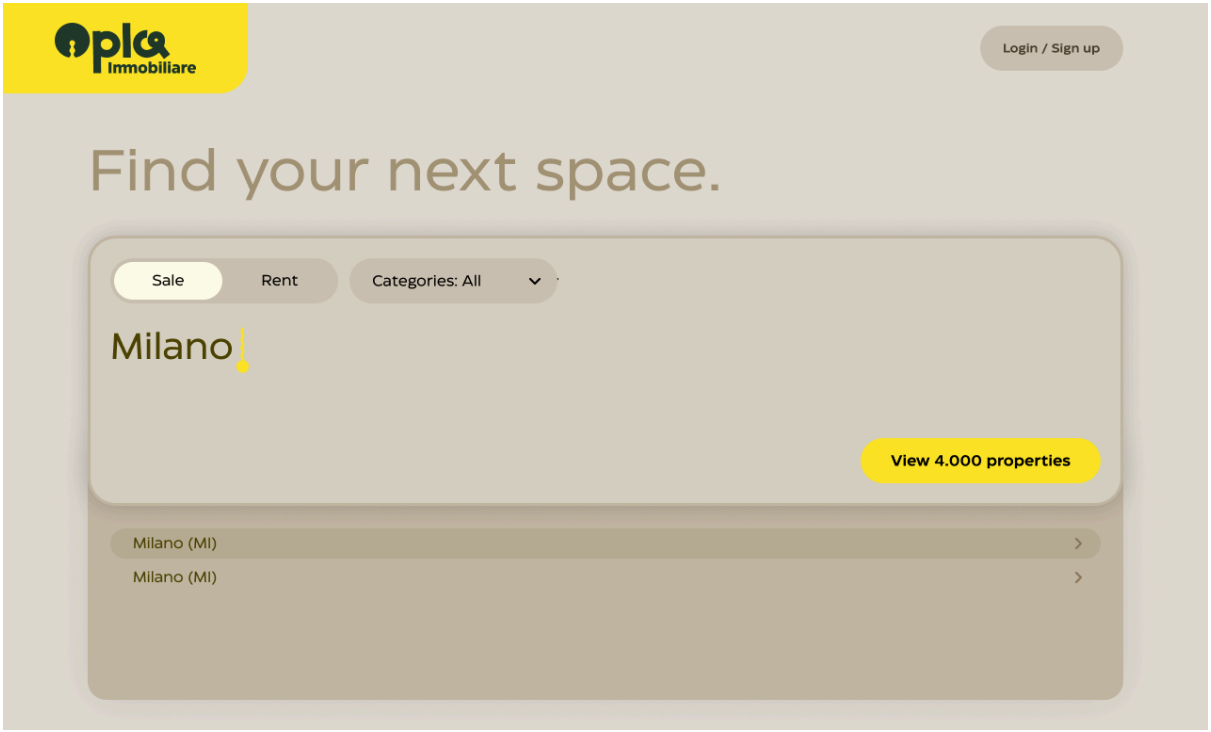
Πατώντας "Ολοκλήρωση & Επιλογή Πλάνου" μεταφέρεται στην σελίδα επιλογής πλάνου συνδρομής η οποία αναλύεται παρακάτω.

4.5.2 Αναζήτηση ακινήτου

Για την αναζήτηση ακινήτου ο χρήστης στην αρχική σελίδα βλέπει ένα μεγάλο πλαίσιο (Εικόνα 4.16) που εξυπηρετεί την αναζήτηση περιοχής που ενδιαφέρει τον χρήστη, καθώς μπορεί πάνω στο πλαίσιο να φιλτράρει την κατηγορία της αναζήτησης του και επιλέγει αν ενδιαφέρεται για ενοικίαση ή αγορά ακινήτου.

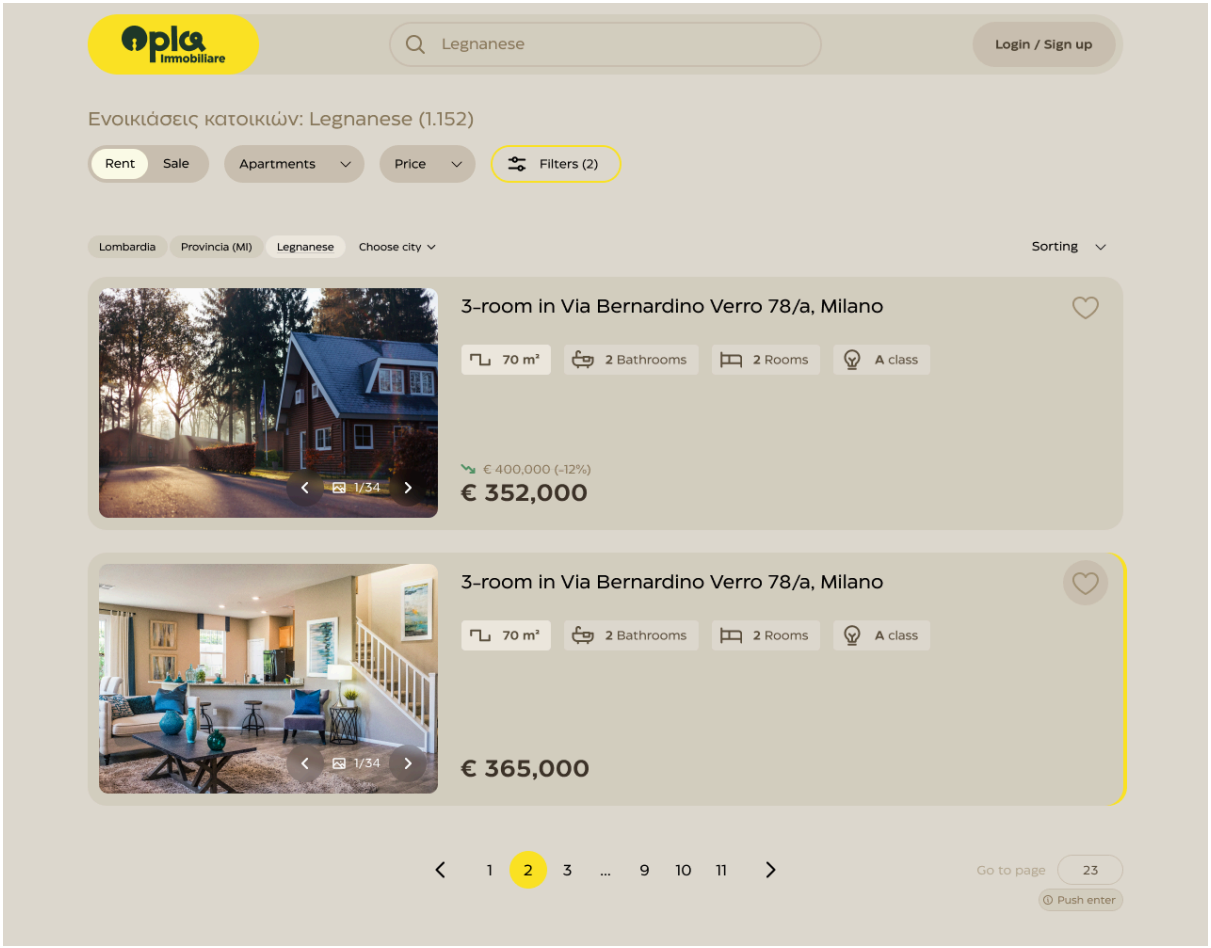
Εικόνα 4.17: Αρχική σελίδα, φόρμα αναζήτησης

Καθώς πληκτρολογεί την περιοχή που τον ενδιαφέρει ένα αναπτυσσόμενο μενού εμφανίζεται κάτω από το πλαίσιο ώστε να επιλέξει υποπεριοχές (π.χ. αναζητά την πόλη και του εμφανίζει τους δήμους της εκάστοτε πόλης, Εικόνα 4.18).

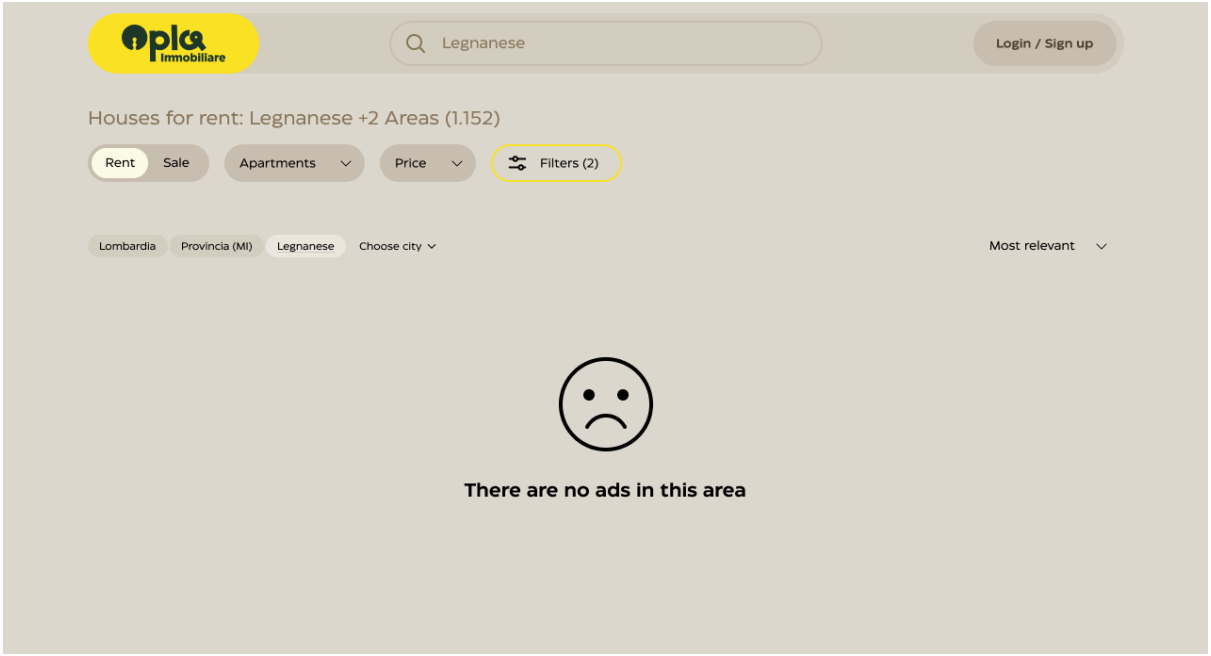


Εικόνα 4.18: Αναπτυσόμενο μενού αναζήτησης περιοχής

Στην συνέχεια ο χρήστης μεταφέρεται στην λίστα με τα ακίνητα που έχει αναζητήσει (Εικόνα 4.19), αν δεν υπάρχουν αποτελέσματα του εμφανίζεται η λίστα κενή με το μήνυμα πως δεν βρέθηκαν ακίνητα για την συγκεκριμένη αναζήτηση (Εικόνα 4.20).



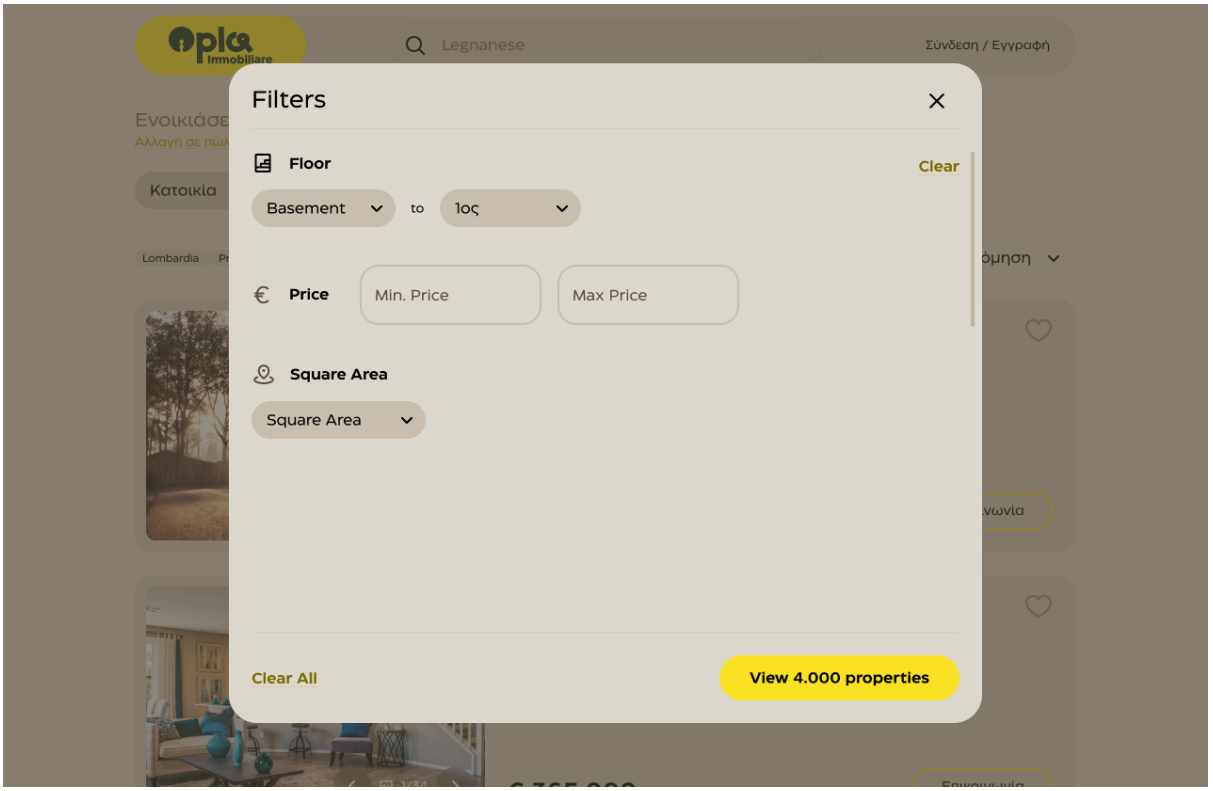
Εικόνα 4.19: Λίστα ακινήτων μετά από αναζήτηση



Εικόνα 4.20: Κενή λίστα ακινήτων μετά από αναζήτηση

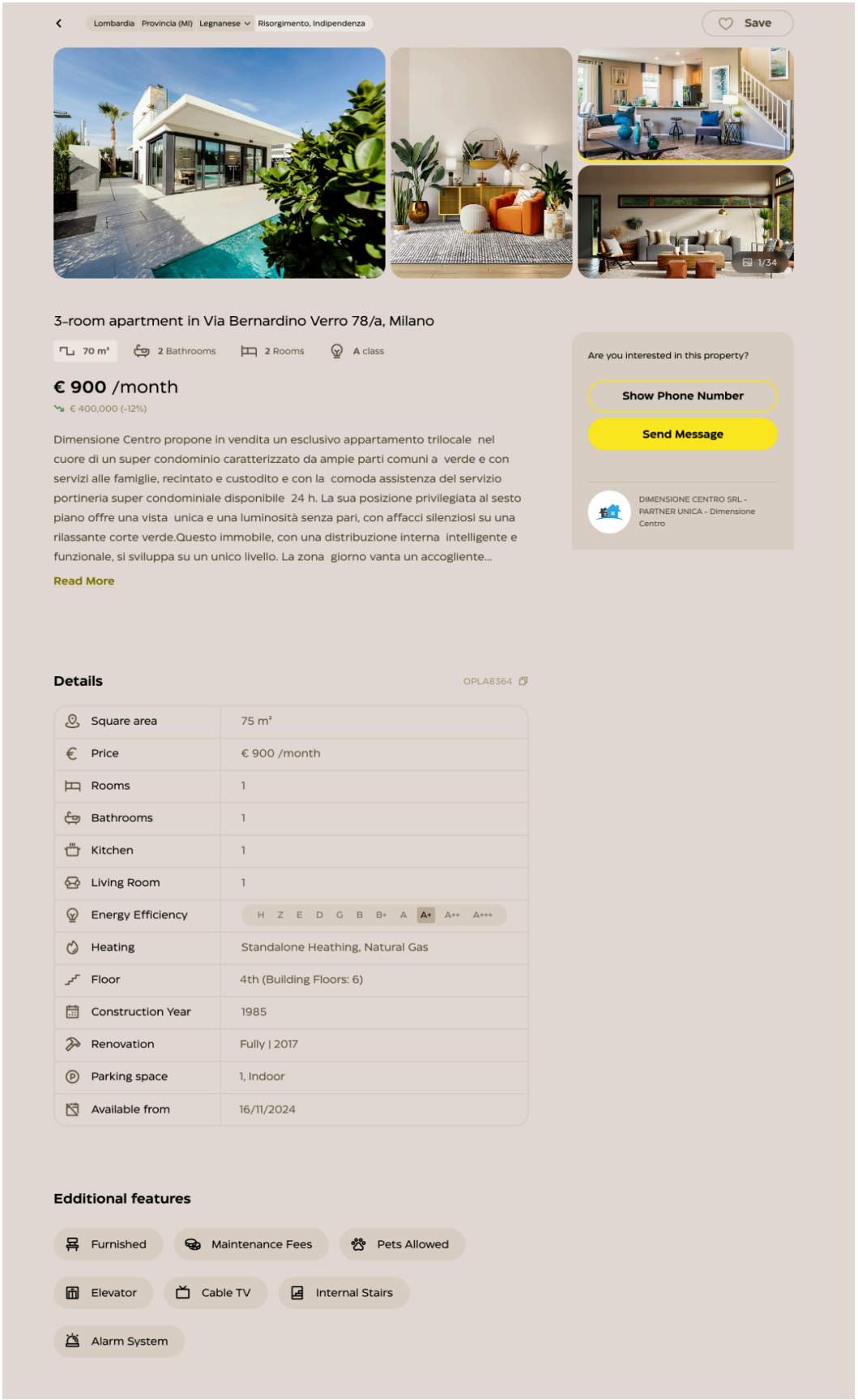
4.5.2.1 Φίλτρα

Από την λίστα ο χρήστης μας μπορεί να επιλέξει επιπλέον φίλτρα αναζήτησης πατώντας το κουμπί “Φίλτρα” που βρίσκεται στην αρχή της λίστας. Πατώντας το κουμπί εμφανίζεται ένα αναδυόμενο παράθυρο με επιπλέον φίλτρα για την αναζήτηση του (Εικόνα 4.21).



Εικόνα 4.21: Αναδυόμενο παράθυρο φίλτρων

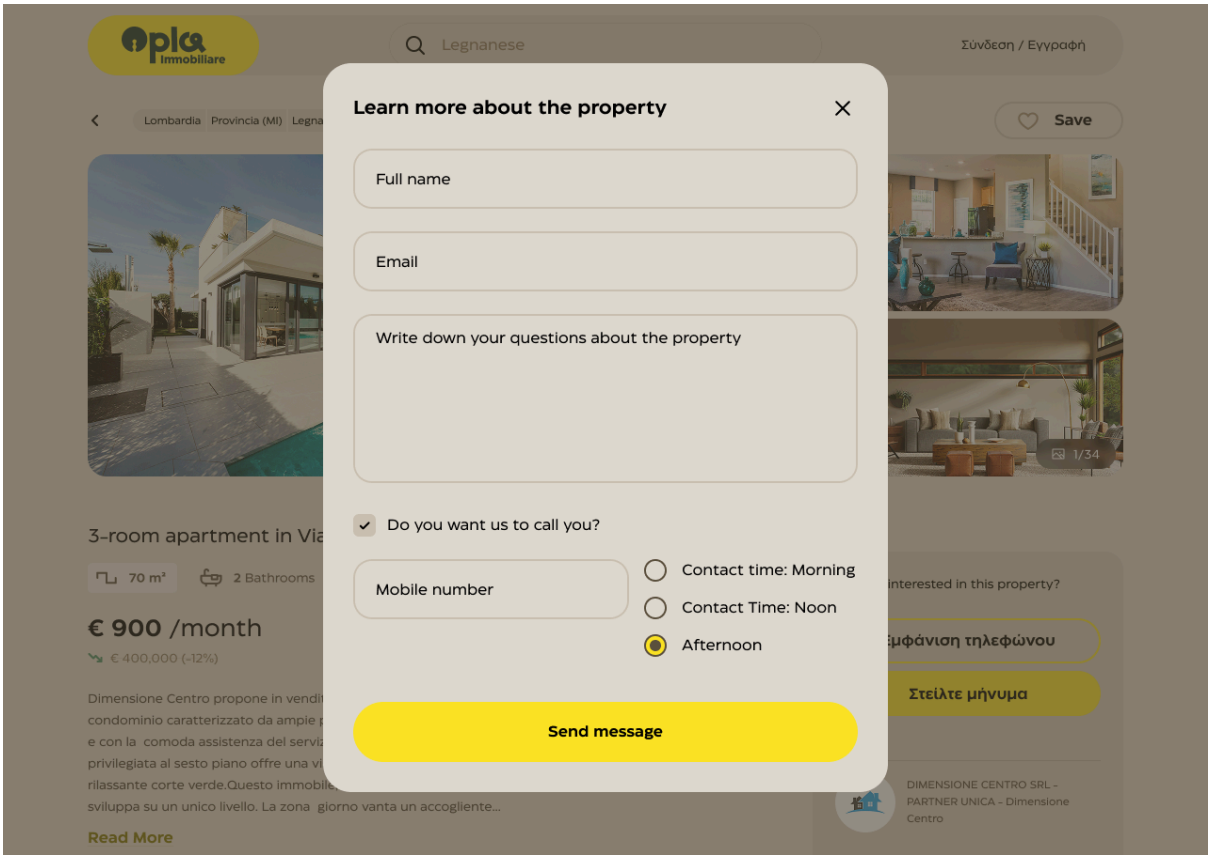
Πατώντας ο χρήστης σε κάποιο από τα ακίνητα της λίστας μεταφέρεται στην σελίδα του ακινήτου για να δει περισσότερες λεπτομέρειες για το ακίνητο (Εικόνα 4.22).



Εικόνα 4.22: Σελίδα ακινήτου

4.5.2.2 Δήλωση ενδιαφέροντος

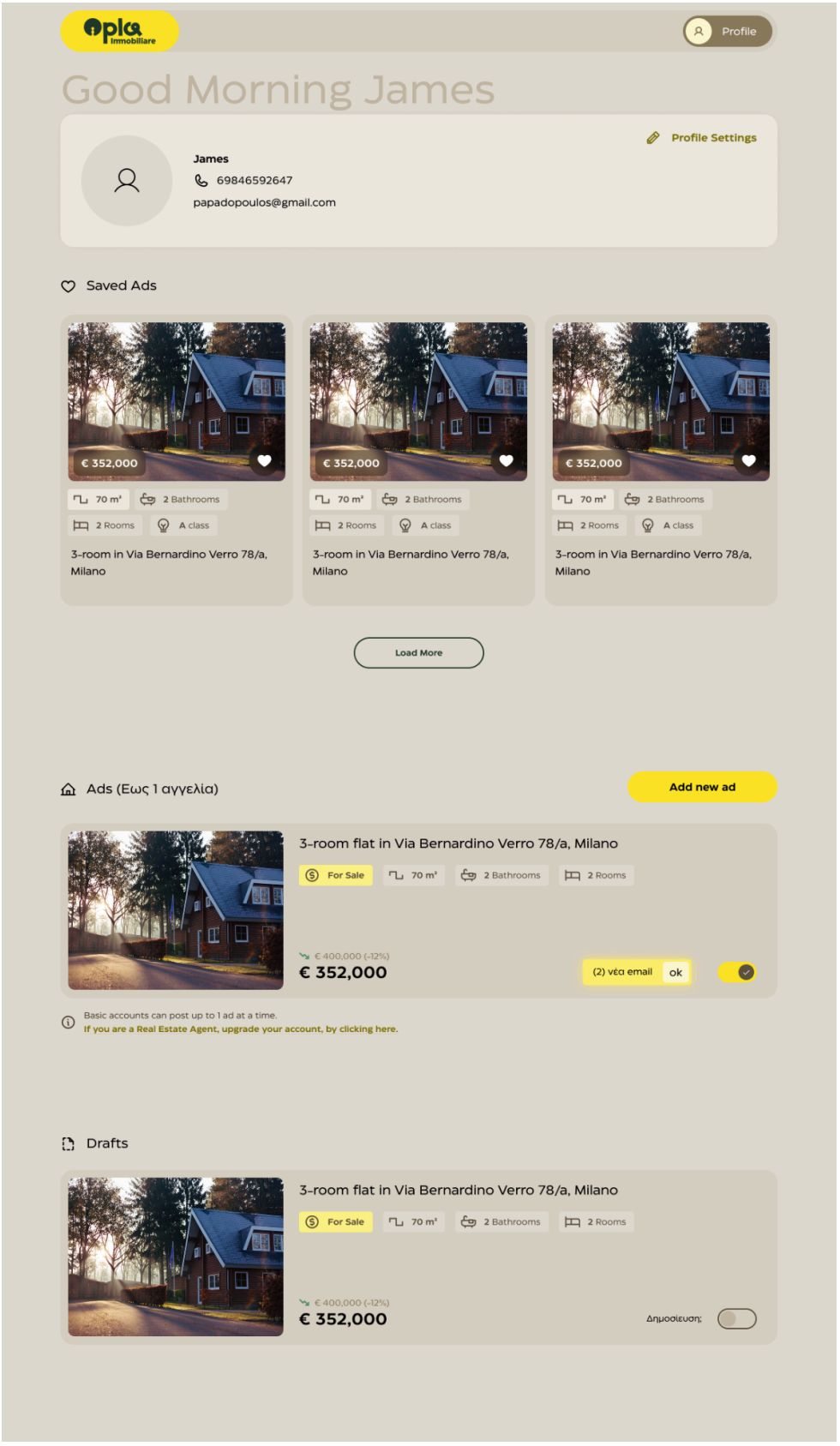
Ο χρήστης καθώς έχει επισκεφτεί την σελίδα ακινήτου δεξιά βρίσκεται ένα τμήμα το οποίο έχει δύο κουμπιά. Το κουμπί "Προβολή Αριθμού Τηλεφώνου" εμφανίζει τον αριθμό τηλεφώνου του δημιουργού της αγγελίας. Το κουμπί "Στείλε Μήνυμα" εμφανίζει ένα αναδυόμενο παράθυρο το οποίο περιέχει μια φόρμα επικοινωνίας με τον δημιουργό της αγγελίας (Εικόνα 4.23).



Εικόνα 4.23: Αναδυόμενο παράθυρο φόρμας ενδιαφέροντος

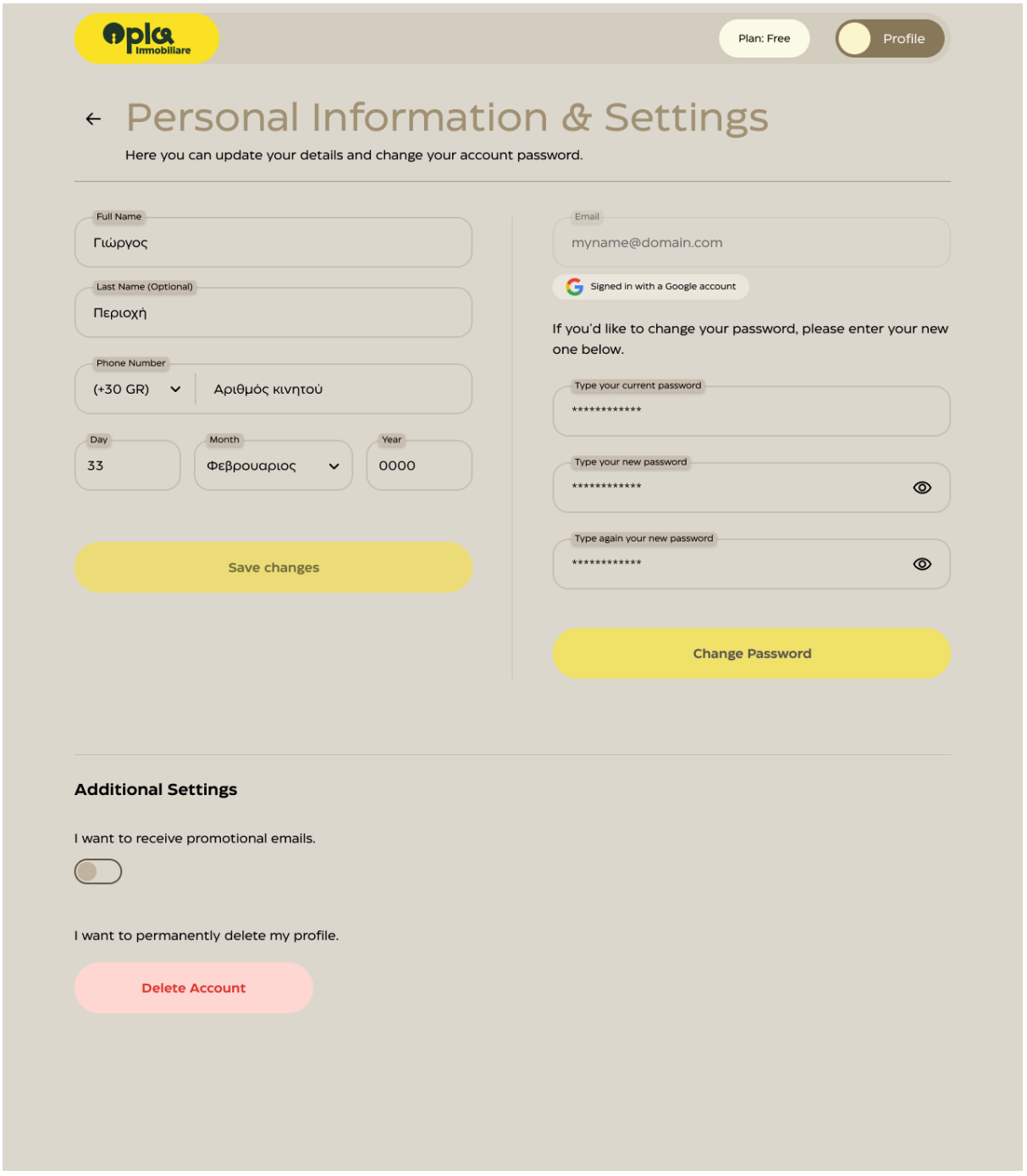
4.5.3 Προφίλ χρήστη

Στην σελίδα προφίλ χρήστη ο χρήστης βλέπει τα ακίνητα που έχει στα αποθηκευμένα, τα ακίνητα που έχει δημοσιεύσει ο ίδιος καθώς και τα ακίνητα που έχει ως προσχέδια για δημοσίευση όπως φαίνονται στην Εικόνα 4.24.



Εικόνα 4.24: Σελίδα προφίλ χρήστη

Ο Χρήστης πατώντας το κουμπί “Ρυθμίσεις προφίλ” μεταφέρεται στην σελίδα ρυθμίσεις όπου μπορεί να διαχειριστεί τα στοιχεία του προφίλ του (όνομα, email, κωδικό πρόσβασης) Εικόνα 4.25.

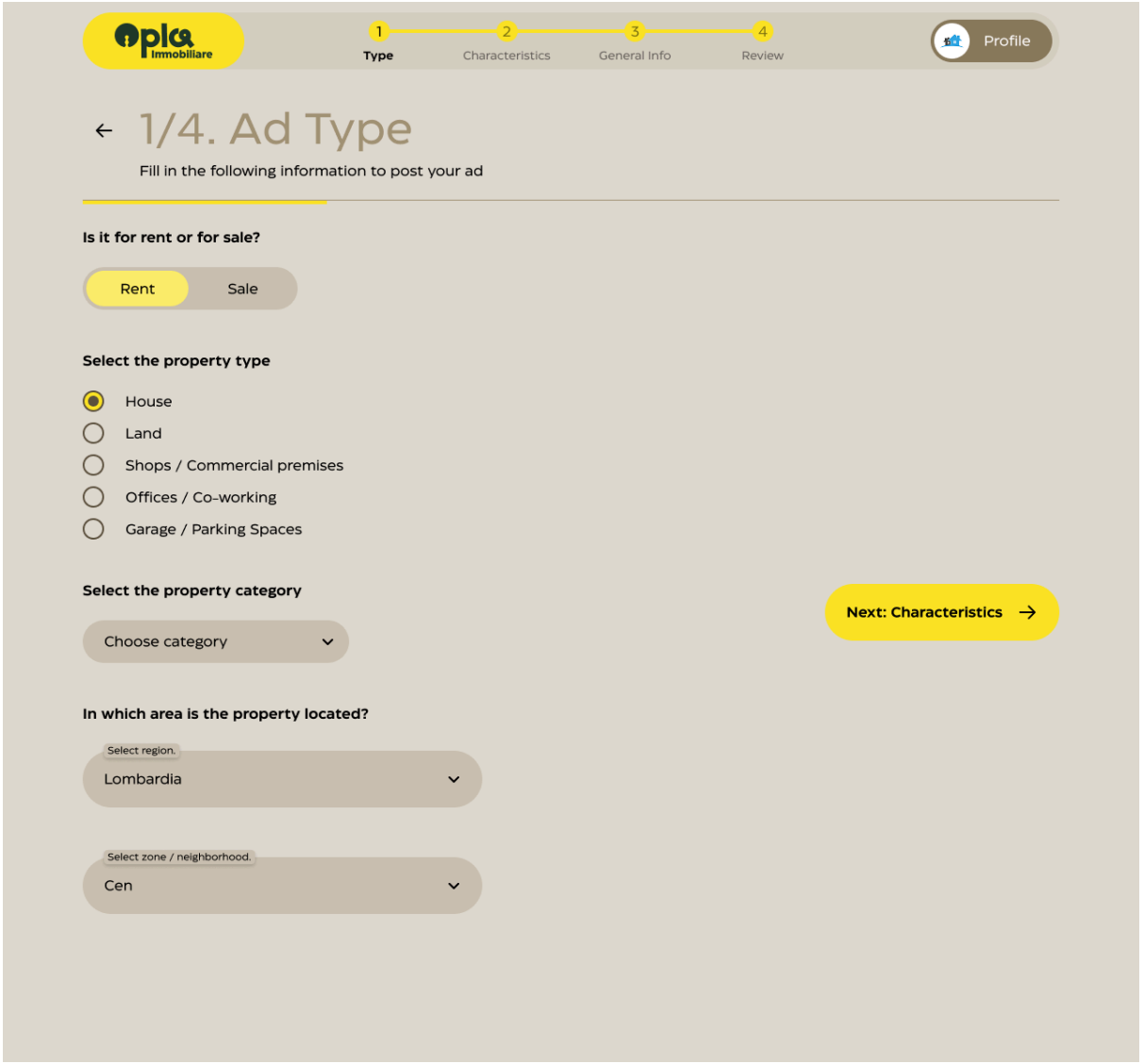


Εικόνα 4.25: Ρυθμίσεις προφίλ

4.5.4 Δημοσίευση Ακινήτου

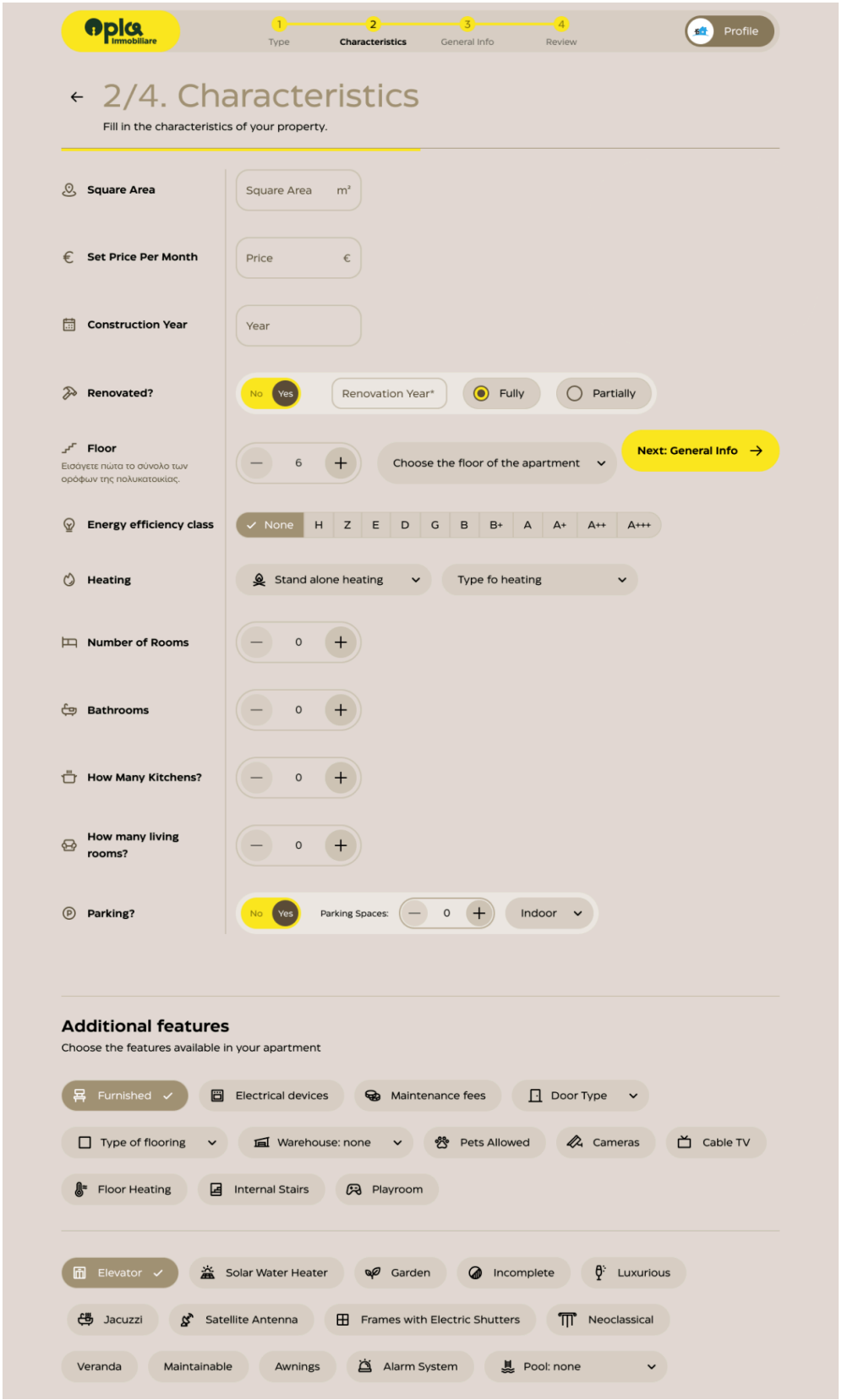
Η δημοσίευση ακινήτου χωρίζεται σε τέσσερα βήματα.

1. Επιλογή αν το ακίνητο μας διατίθεται για αγορά ή ενοικίαση, έπειτα γίνεται επιλογή των κατηγοριών που το ακίνητο μας ανήκει και επιλογή της διεύθυνσης του ακινήτου (Εικόνα 4.26).



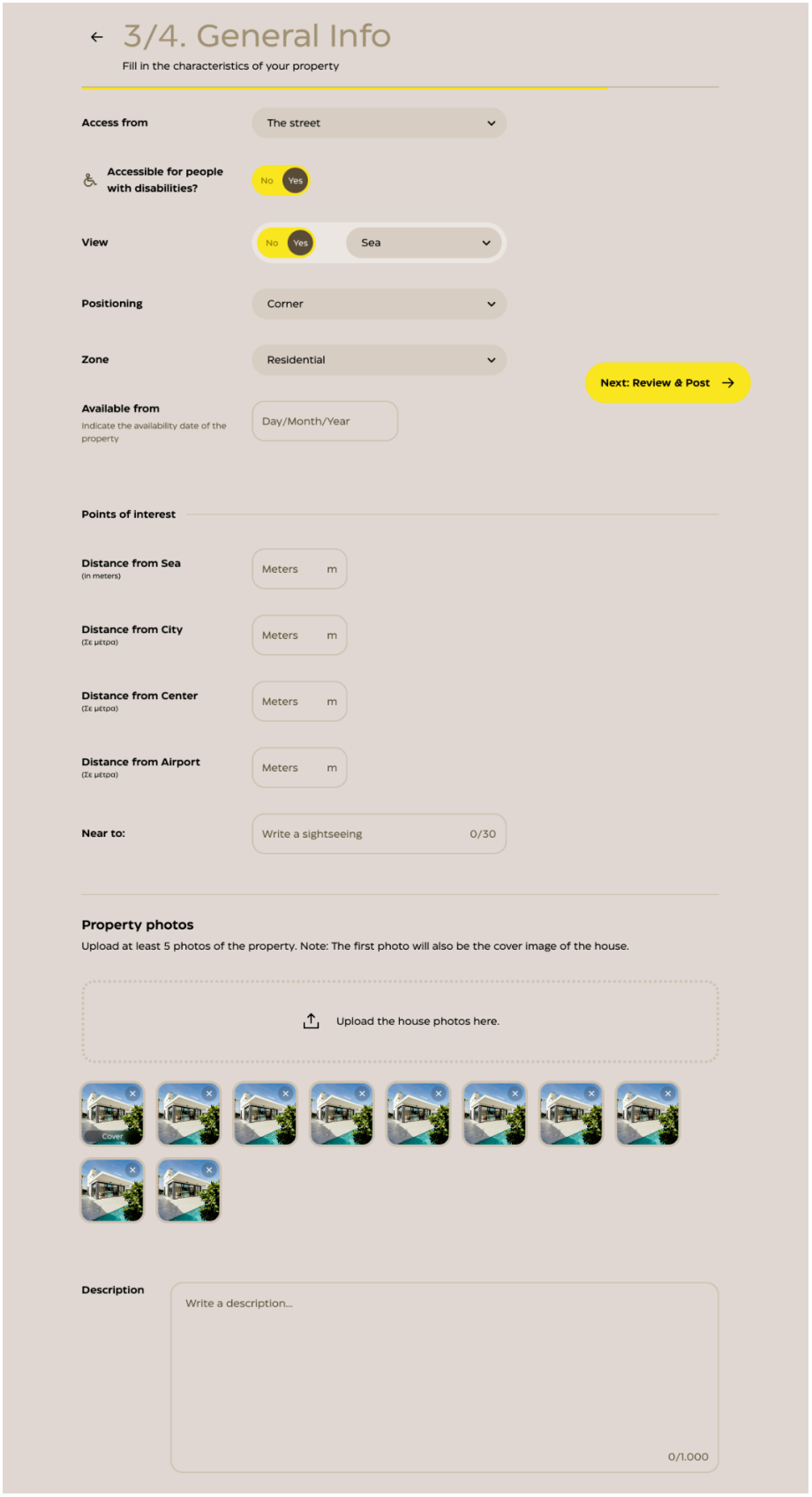
Εικόνα 4.26: Βήμα 1ο δημοσίευσης ακινήτου

2. Στην συνέχεια ο χρήστης προσθέτει τα χαρακτηριστικά του ακινήτου (Εικόνα 4.27).



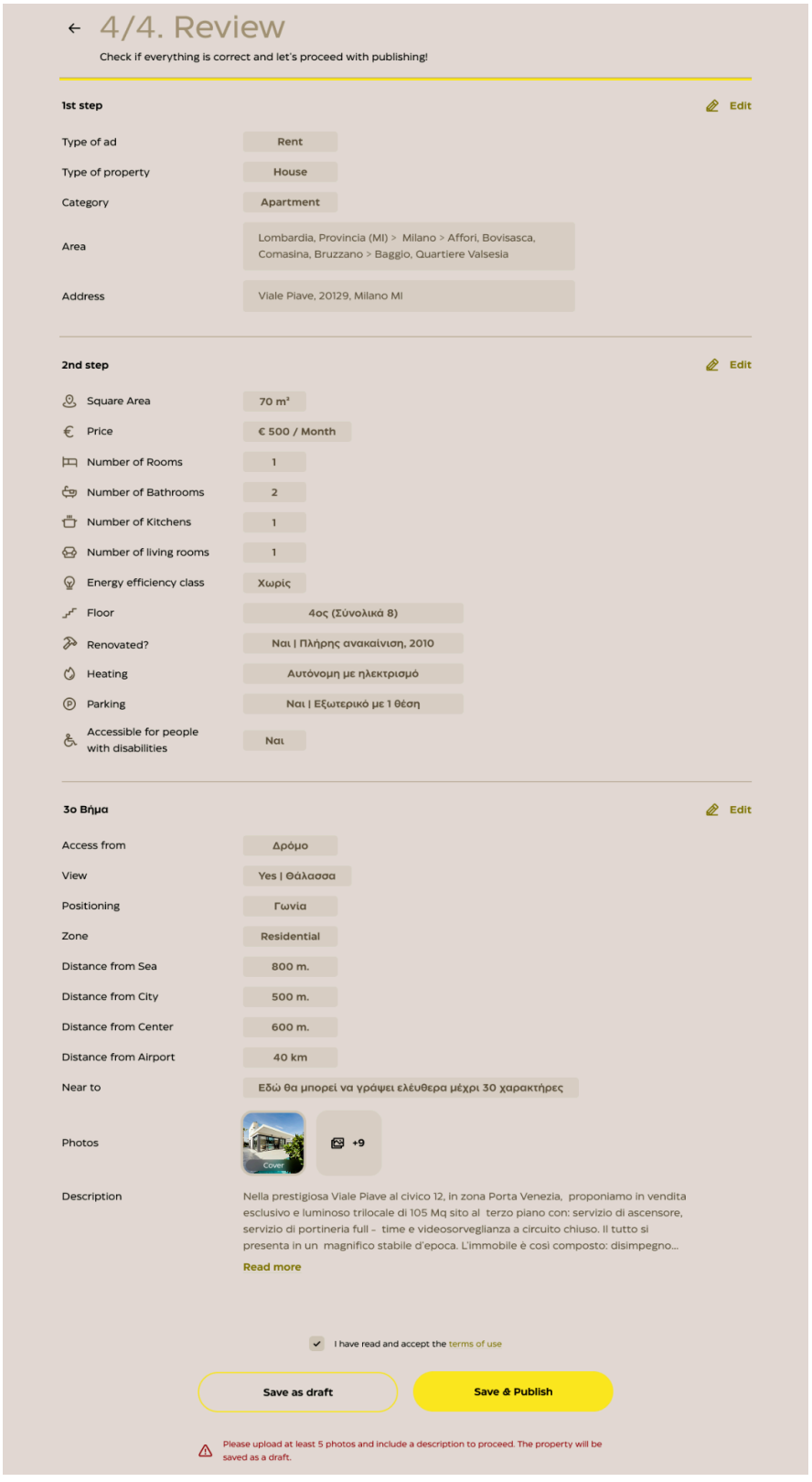
Εικόνα 4.27: Βήμα 2ο δημοσίευσης ακινήτου

3. Στην συνέχεια ο χρήστης επιλέγει τα γενικά χαρακτηριστικά για το ακίνητο που θέλει να δημοσιεύσει, επιλέγοντας και το φωτογραφικό υλικό της δημοσίευσής του (Εικόνα 4.28).



Εικόνα 4.28: Βήμα 3ο δημοσίευσης ακινήτου

4. Και το τελευταίο βήμα είναι η σελίδα επιβεβαίωσης των στοιχείων που έχει προσθέσει ο χρήστης (Εικόνα 4.29).

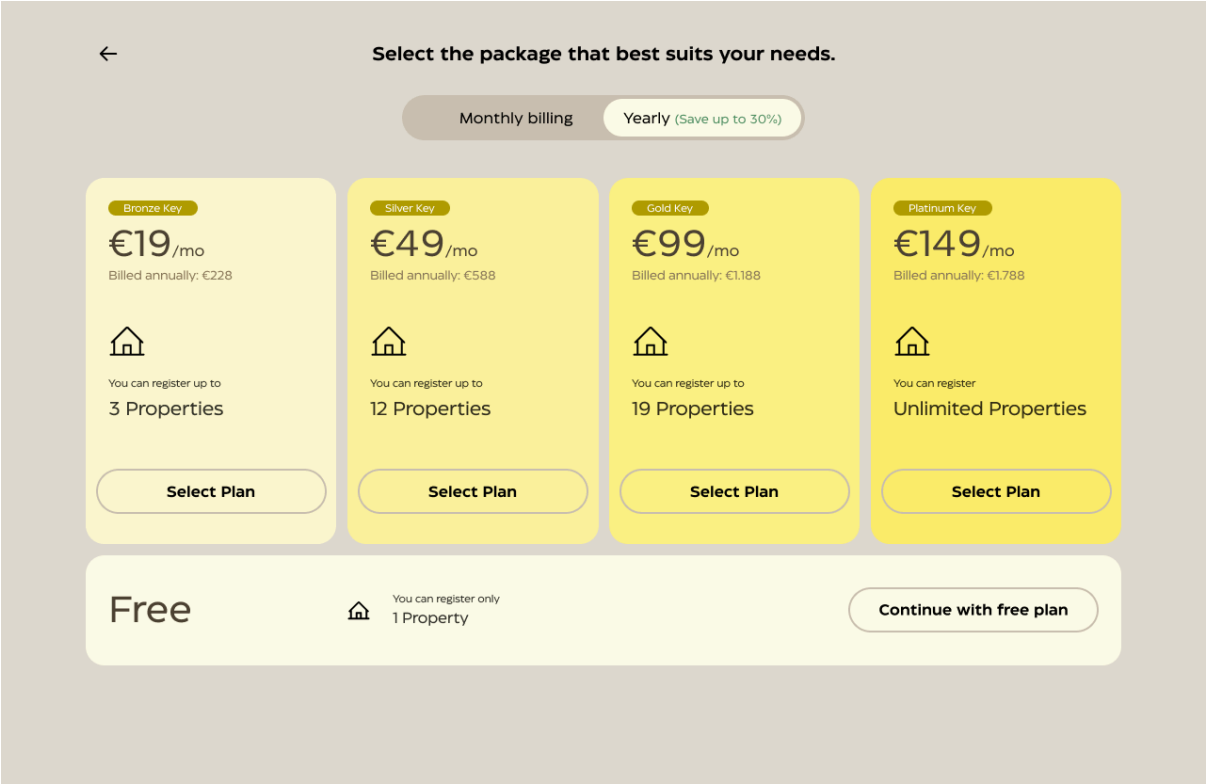


Εικόνα 4.29: Βήμα 4ο δημοσίευσης ακινήτου

Μετά την δημοσίευση ο χρήστης μεταφέρεται στην σελίδα προβολής ακινήτου (Εικόνα 4.22).

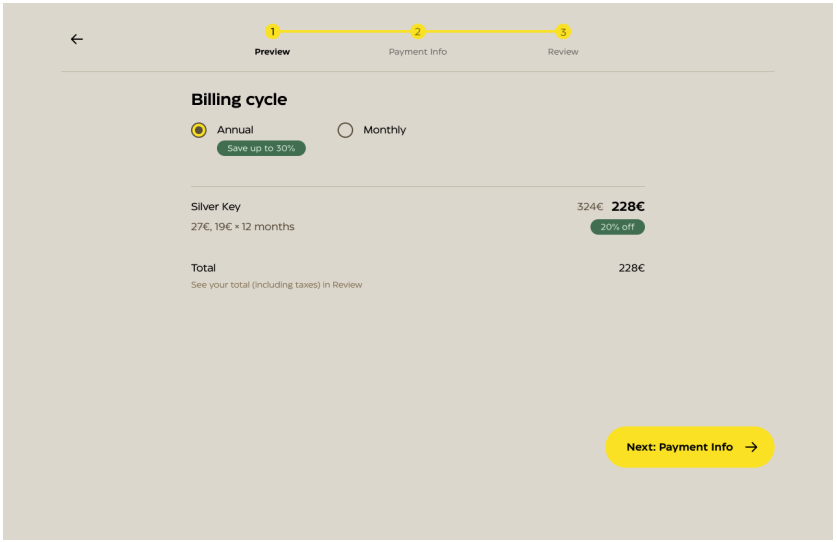
4.5.5 Συνδρομές

Κατά την ολοκλήρωση εγγραφή ως επαγγελματίας ο χρήστης μεταφέρεται στην σελίδα του καταστήματος όπου πρέπει να επιλέξει ένα πλάνο συνδρομής για να συνεχίσει Εικόνα 4.30.



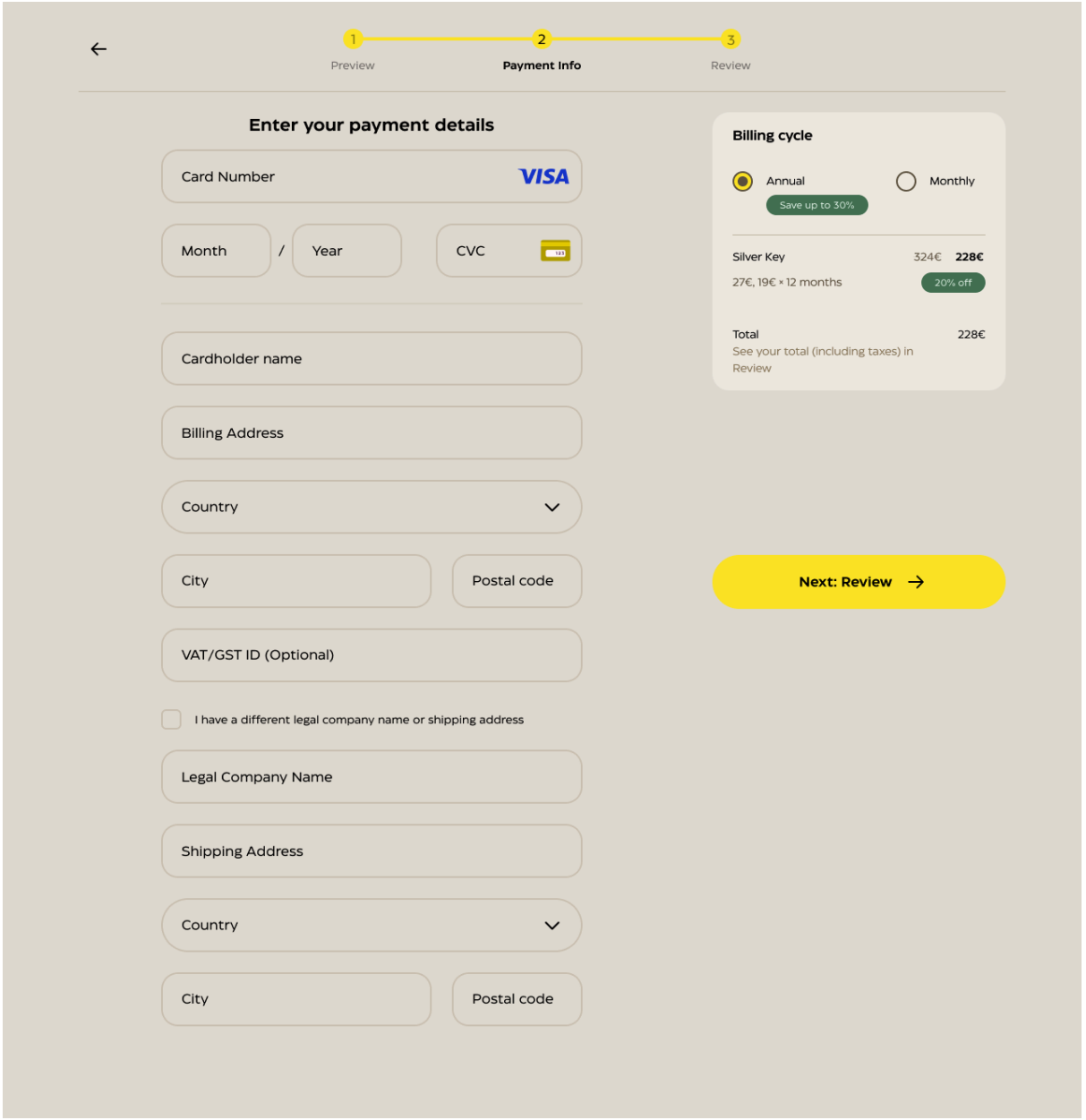
Εικόνα 4.30: Σελίδα πλάνου συνδρομής

Επιλέγοντας το πλάνο μεταφέρεται στην σελίδα καλαθιού όπου βλέπει και μπορεί να επεξεργαστεί την επιλογή του Εικόνα 4.31.



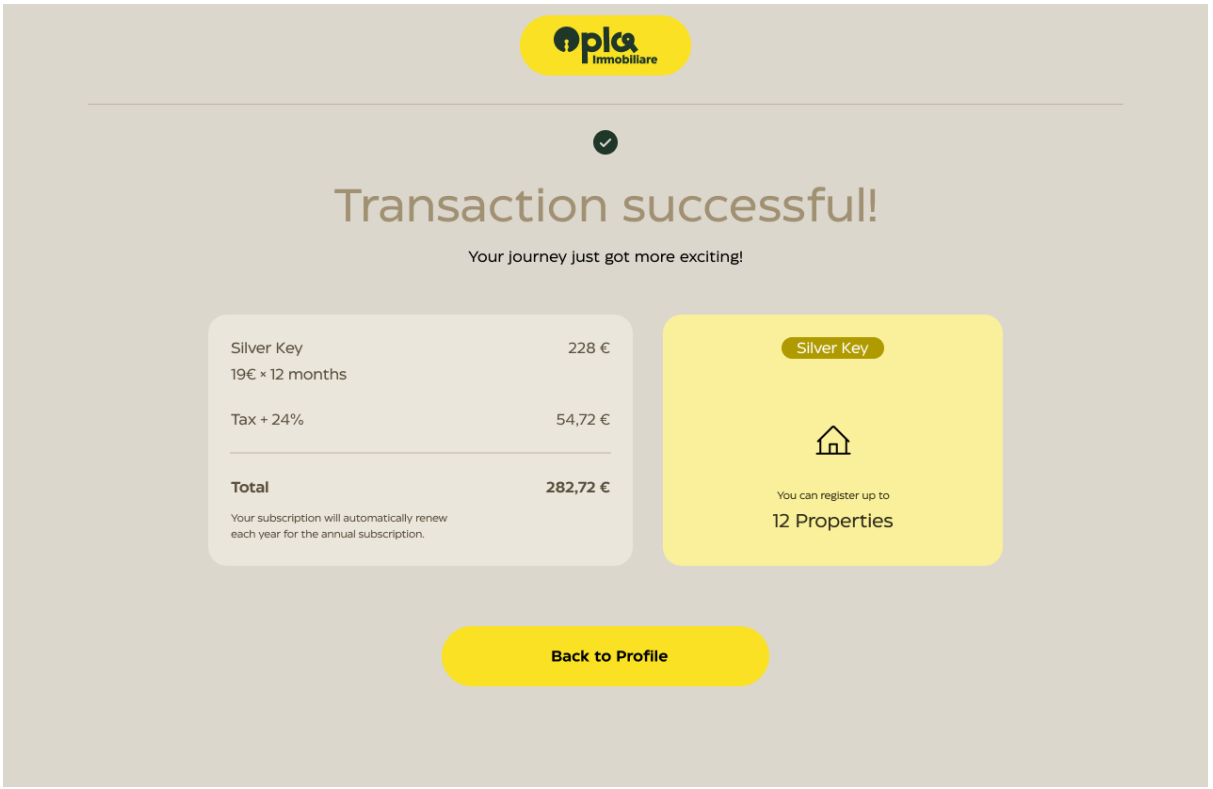
Εικόνα 4.31: Σελίδα καλαθιού

Στην συνέχεια ο χρήστης μεταφέρεται στην σελίδα ολοκλήρωσης πληρωμής όπου και προσθέτει τα στοιχεια της κάρτας του για να προχωρήσει στην συνδρομή Εικόνα 4.32.



Εικόνα 4.32: Σελίδα Ολοκλήρωσης Πληρωμής

Σε περίπτωση ανεπιτυχής πληρωμής ένα μήνυμα εμφανίζεται στον χρήστη. Εάν η πληρωμή είναι επιτυχής ο χρήστης μεταφέρεται στην σελίδα επιτυχημένης πληρωμής Εικόνα 4.33.



Εικόνα 4.33: Σελίδα Ολοκλήρωσης Πληρωμής

Κεφάλαιο 5ο: Προτάσεις βελτίωσης και επεκτασιμότητα της εφαρμογής.

Η πτυχιακή εργασία που παρουσιάστηκε είχε ως στόχο τη σχεδίαση και υλοποίηση μιας διαδικτυακής εφαρμογής δημοσίευσης και διαχείρισης ακινήτων. Η εφαρμογή αυτή περιέχει δύο ρόλους χρηστών για απλούς χρήστες και μεσιτικά γραφεία αλλά είναι προσβάσιμη και για επισκέπτες. Τα μεσιτικά γραφεία είναι υποχρεωτικό να έχουν συνδρομή στην πλατφόρμα για να μπορέσουν να ξεκλειδώσουν ακίνητα που μπορούν να δημοσιεύσουν.

5.1 Προτάσεις βελτίωσης εφαρμογής

Η εφαρμογή αποτελείται από διάφορες τεχνολογίες και έχει αρκετές συναρτήσεις. Όπως κάθε εφαρμογή, μπορεί να βρεθεί καλύτερος τρόπος για την υλοποίηση διαφόρων συναρτήσεων της εφαρμογής μας, τεχνολογιών και αρχιτεκτονικών.

5.1.1 Χρήση Cloud Storage

Στην τρέχουσα έκδοση το φωτογραφικό υλικό και τα βίντεο των ακινήτων αποθηκεύονται σε έναν φάκελο τοπικά στον εξυπηρετητή της εφαρμογής. Αυτό έχει ως αποτέλεσμα την επιβάρυνση του εξυπηρετητή με αποτέλεσμα να γεμίζει ο αποθηκευτικός χώρος του. Το πρόβλημα που δημιουργείται εδώ είναι πως αν γεμίσει ο χώρος του εξυπηρετητή η ιστοσελίδα μας δεν θα μπορεί να είναι προσβάσιμη αν δεν επεκτείνουμε τον εξυπηρετητή μας.

Επιπλέον το κόστος του εξυπηρετητή είναι σταθερό και αυτό έχει ως αποτέλεσμα να αγοράζουμε χώρο τον οποίο δεν χρησιμοποιούμε κάθε φορά.

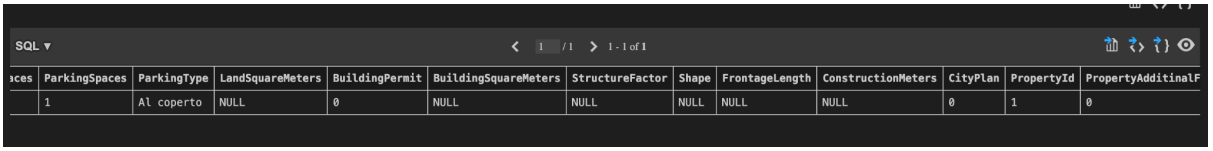
Σε κάποιο Cloud Storage όπως το S3 Bucket της Amazon ή το Azure Blob της Microsoft αποθηκεύει τα αρχεία μας ως JSON αντικείμενα σε κάποιο Storage και είναι προσβάσιμα μέσω CDN οποιαδήποτε στιγμή και να τα χρειαστούμε.

Για τον ίδιο παραπάνω λόγο θα επιλέγαμε να έχουμε ένα Cloud εξυπηρετητή για ολόκληρη την εφαρμογή μας ώστε να χρεώνεται η επιχείρηση ανάλογα με την χρήση των πόρων που χρειαζόμαστε.

5.1.2 Χρήση μη σχεσιακής βάσης δεδομένων (NoSQL)

Η τωρινή έκδοση χρησιμοποιεί δύο σχεσιακές βάσης δεδομένων στην εφαρμογή μας. Η βάση που διαχειρίζεται τους χρήστες θα μπορούσαμε να την κρατήσουμε ώς έχει. Από την στιγμή όμως που έχουμε αποφασίσει να έχουμε δύο διαφορετικές βάσεις δεδομένων στην δεύτερη μας βάση η οποία διαχειρίζεται τα ακίνητα θα μπορούσαμε να έχουμε μια μη σχεσιακή βάση (NoSql) όπως είναι η MongoDB ή το Elastic Search.

Ο λόγος που θα μπορούσαμε να χρησιμοποιήσουμε την αντίστοιχη βάση είναι κυρίως γιατί αυτήν την στιγμή οι πίνακες της βάσης έχουν κάποια πεδία. Αυτά τα πεδία, δεν αποτελουν πεδία κάθε ακινήτου και αυτό έχει ως αποτέλεσμα σε κάθε ακίνητο μας να έχουμε πάρα πολλά πεδία στην βάση μας ως κενά (NULL). Όπως φαίνεται στο στιγμιότυπο της βάσης στην Εικόνα 5.1.



SQL ▾ 1 / 1 1 - 1 of 1

aces	ParkingSpaces	ParkingType	LandSquareMeters	BuildingPermit	BuildingSquareMeters	StructureFactor	Shape	FrontageLength	ConstructionMeters	CityPlan	PropertyId	PropertyAdditinalF
	1	Al coperto	NULL	0	NULL	NULL	NULL	NULL	NULL	0	1	0

Εικόνα 5.1: Πεδία της βάσης τα οποία είναι κενά

Αν αλλάξουμε σε μια μη σχεσιακή βάση δεδομένων τα δεδομένα μας αποθηκεύονται ως JSON αντικείμενα, οπότε θα αποθηκεύεται σε κάθε ακίνητο μόνο τα πεδία που αντιπροσωπεύουν το συγκεκριμένο ακίνητο οπότε έτσι δεν θα είχαμε κενά πεδία στην βάση μας να διαχειριστούμε.

Στο παρακάτω απόσπασμα κώδικα φαίνεται το ακίνητο που φαίνεται με τα κενά πεδία στην Εικόνα 5.1 ως ένα JSON αντικείμενο σε μια μη σχεσιακή βάση δεδομένων.


```
{
   "propertyFor": 0,
   "squareMetters": 150,
   "price": 600,
   "description": "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.",
   "disabilitiesAccess": true,
   "view": "Sea",
   "positioning": "Corner",
   "zone": "Residential",
   "nearTo": "Stadio Olympico di Roma",
   "distanceFromSea": 10,
   "disatnceFromCity": 20,
   "distanceFromCenter": 30,
   "distanceFromAirport": 40,
   "propertyImages": [
      "kkdkdd.jpg",
      "kkdkdkkd.jpg"
   ],
   "propertyCharacteristics": {
      "constractionYear": 1960,
      "renovatedYear": 2007,
      "renovationType": "Fully",
      "buildingFloors": 6,
      "apartmentFloor": 1,
      "energyClass": "A+",
      "heating": "Stand Alone",
      "typeOfHeating": "Gas",
      "numberOfRooms": 1,
      "bathrooms": 1,
      "kitchens": 1,
      "livingRooms": 1,
      "propertyAdditionalFeatures": {
         "fournished": true,
         "electricDevices": true,
         "maintenanceFees": 0,
         "petsAllowed": true,
         "cameras": true,
         "cableTv": true,
         "floorHeating": true,
         "internalStairs": true,
         "playroom": true,
         "elevator": true,
         "solarWaterHeater": true,
         "garden": true,
```

```
        "incomplete": true,
        "luxurious": true,
        "jacuzzi": true,
        "sateliteAntenna": true,
        "framesWithElectricShutters": true,
        "neoclassical": true,
        "veranda": true,
        "maintainable": true,
        "awnings": true,
        "alarmSystem": true,
        "pool": "string"
      }
    },
    "propertyTypeId": 1,
    "propertyCategoryId": 1,
    "dateCreated": "2024-10-03",
    "lastUpdate": "2024-10-03"
}
```


5.1.3 Χρήση Stripe ολοκληρωτικά για την διαχείριση των συνδρομών

Η εφαρμογή μας χρησιμοποιεί έναν συνδυασμό του Stripe και ειδικής κατασκευής μέσα στην εφαρμογή με πίνακες διαχείρισης συνδρομών στην βάση. Αυτό έχει ως αποτέλεσμα να αυξάνεται η πολυπλοκότητα της εφαρμογής και να είναι πιο δύσκολη η επεκτασιμότητα της. Κάποια πράγματα όπως τα πλάνα συνδρομών πρέπει να δημιουργούνται δύο φορές, μία στο Stripe και μια στην εφαρμογή μας απευθείας.

Το Stripe παρέχει την δυνατότητα να χρησιμοποιηθεί ολοκληρωτικά σαν διαχειριστής συνδρομών καθώς το API του προσφέρει όλες τις δυνατότητες που χρειαζόμαστε. Με αυτόν τον τρόπο θα απλοποιηθεί πολύ η εφαρμογή μας στο σύστημα συνδρομών και θα είναι πιο εύκολα επεκτάσιμο.

5.1.4 Χρήση εφαρμογής για την καταγραφή

Αυτή την στιγμή η καταγραφή χρήσης και σφαλμάτων γίνεται στην εφαρμογή μας τοπικά στον εξυπηρετητή σε έναν φάκελο Logs, ο οποίος περιέχει δύο αρχεία:

1. error.log: το οποίο περιέχει τα σφάλματα της εφαρμογής μας
2. info.log: το οποίο περιέχει την καταγραφή χρήσης της εφαρμογής μας

Για την καλύτερη διαχείριση, γρηγορότερη ανάγνωση και εύρεση των λαθών αλλά και την καλύτερη ασφάλεια, θα μπορούσαμε να συνδέσουμε την βιβλιοθήκη καταγραφής μας με το API κάποις εξωτερικής πλατφόρμας όπως είναι το Elasticsearch ώστε αυτόματα τα δεδομένα καταγραφής να στέλνονται στην πλατφόρμα και να μην αποθηκεύονται τοπικά. Έτσι τα δεδομένα καταγραφείς θα είναι προσβάσιμα μέσω κάποιας ιστοσελίδας. Αυτό είναι ασφαλέστερο καθώς δεν θα συνδεόμαστε κάθε φορα στον εξυπηρετητή για να βλέπουμε τα αρχεία μας αλλά είναι και πιο εύχρηστο καθώς μπορούμε στην ιστοσελίδα να φιλτράρουμε και να κάνουμε αναζήτηση.

5.2 Επεκτασιμότητα εφαρμογής

Η εφαρμογή μας μπορεί να επεκταθεί πολύ σε διάφορους τομείς όπως εμπειρία χρήστη, στατιστική ανάλυση επιχειρήσεων και διαφήμιση.

5.2.1 Ηλεκτρονικά μηνύματα ειδοποίησης χρηστών

Βασικότερη και κυριότερη επέκταση είναι η δυνατότητα του χρήστη να μπορεί να δέχεται ηλεκτρονικά μηνύματα με βάση κάποια κριτήρια που ορίζει ο ίδιος, ώστε να δέχεται ηλεκτρονικά μηνύματα (email) κάθε φορά που δημοσιεύεται ένα ακίνητο για παράδειγμα σε κάποια συγκεκριμένη πόλη ή περιοχή. Αυτό θα μπορούσε να επιτευχθεί με την χρήση κάποιας εξωτερικής πλατφόρμας όπως είναι το MailChimp ή το Brevo για να μπορούμε να διαμορφώνουμε δυναμικά τα email και να στέλνουμε μαζικά emails στους χρήστες.

5.2.2 Κλείσουμε ραντεβού υπόδειξης

Ο χρήστης κατά την αναζήτηση μπορεί κατά την προβολή ενός ακινήτου αυτή την στιγμή να επικοινωνήσει με τον δημιουργό της αγγελίας τηλεφωνικά ή να στείλει ένα αίτημα επικοινωνίας στον δημιουργό της αγγελίας. Θα μπορούσε να υλοποιηθεί μια φόρμα όπου ο χρήστης θα βλέπει στην πλατφόρμα τα διαθέσιμα ραντεβού και θα κλείνει απευθείας από την πλατφόρμα ένα ραντεβού για την υπόδειξη του ακινήτου.

Βέβαια η παραπάνω επέκταση της εφαρμογής θα πρέπει να μπορεί να ενεργοποιείται και να απενεργοποιείται από τον εκάστοτε δημιουργώ καθώς επιφέρει έναν επιπλέον φόρτο στην διαχείριση των ακινήτων του στην πλατφόρμα.

5.2.3 Χάρτης προβολής ακινήτων

Για την βέλτιστη εμπειρία χρήστη θα πρέπει να ενσωματωθεί ένας χάρτης ο οποίος θα δείχνει τα ακίνητα πάνω στον χάρτη. Έτσι ώστε ο χρήστης εκτός από την λίστα με τα εκλινητα να βλέπει και σε ποιο σημείο βρίσκονται στον χάρτη στην αντίστοιχη περιοχή.

5.2.4 Σελίδα στατιστικών

Αυτό που θα βοηθούσε πολύ τα μεσιτικά γραφεία θα ήταν να δημιουργηθεί μια σελίδα στατιστικής ανάλυσης η οποία θα περιέχει όλα τα απαραίτητα στατιστικά στοιχεία που θα βοηθούσαν τα μεσιτικά γραφεία να καταλάβουν καλύτερα τους χρήστες που κάνουν αναζήτηση στην ιστοσελίδα. Θα μπορούσα για παράδειγμα να περιέχει τον μέσο όρο αναζήτησης τιμών πώλησης ανά περιοχή, τύπος ακινήτων αναζήτησης ή προδιογραφές σε μια στατιστική ανάλυση για τους διαχειριστές των μεσιτικών γραφείων.

5.2.5 Παρακολούθηση και ειδοποίηση

Μια πολύ σημαντική επέκταση της εφαρμογής μας είναι η διασύνδεση των αρχείων καταγραφής χρήσης της εφαρμογής και των σφαλμάτων σε μία εφαρμογή προβολής αλλά και η ειδοποίηση για σφάλματα που μπορούν να προκαλέσουν σημαντικά προβλήματα στην εφαρμογή μας και κατά συνέπεια και στην επιχείρηση μας. Για παράδειγμα θα μπορούσαμε να χρησιμοποιήσουμε Grafana Dashboards για την παρακολούθηση της χρήσης της εφαρμογής (αιτήματα το λεπτό, συνδεδεμένοι χρήστης), Grafana Alerts για την ειδοποίηση σε περίπτωση που έχουμε πολλά αιτήματα το λεπτό από συγκεκριμένη διεύθυνση δικτύου (IP). Επιπλέον θα μπορούσαμε να χρησιμοποιήσουμε Elasticsearch για την προβολή και αναζήτηση των δεδομένων καταγραφής της εφαρμογής μας.

5.2.6 API διασύνδεσης

Ακόμα ένα σημαντικό σημείο επεκτασιμότητας της ιστοσελίδας μας θα μπορούσε να είναι η αναπροσαρμογή του API μας ώστε να μπορούν τα μεσιτικά γραφεία να επικοινωνήσουν απευθείας από τα ενδοεταιρικά τους συστήματα με το δικό μας. Έτσι όταν δημοσιεύουν κάποιο ακίνητο στο δικό τους τοπικό πρόγραμμα να μπορεί αυτό να δημοσιεύεται και απευθείας στην Ιστοσελίδα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] https://www.spitogatos.gr/page/aboutUs

[2] https://www.onlinemarketplaces.com/articles/greek-portal-spitogatos-launches-data-insights-business/

[3] https://play.google.com/store/apps/details?id=gr.spitogatos.android

[4] https://apps.apple.com/us/app/spitogatos-property-search/id1057971912

[5] https://www.wappalyzer.com/lookup/spitogatos.gr/

[6] 4 Reasons to Use Angular for Your Next Web App – Telerik Blog, 2024 https://www.telerik.com/blogs/4-reasons-use-angular-your-next-web-app

[7] Angular vs. React: Choosing the Right Framework for Your Next Big Idea - Medium Blog, 2024 https://medium.com/@gavandiroshan/angular-vs-react-choosing-the-right-framework-for-your-next-big-idea-40a5318abd9b

[8] Why we use Tailwind CSS as our primary framework? - Medium Blog, 2021 https://medium.com/@mrkaluzny/why-we-use-tailwind-css-as-our-primary-framework-ed8994dec949

[9] Why use Redux? Reasons with clear examples - Medium Blog, 2018 https://medium.com/@neo/why-use-redux-reasons-with-clear-examples-d21bffd5835

[10] 9 Advantages of Using .NET Core for Developing Your Software - Yuki Solutions Blog, https://yuktisolutions.com/blog/benefits-of-asp-net-core-for-web-application-development

[11] Why use .NET for Your Project? - Medium Blog, 2022 https://medium.com/@dreamix.eu/why-use-net-for-your-project-e8a073a02110

[12] PostgreSQL Benefits - Prisma Data Guide https://www.prisma.io/dataguide/postgresql/benefits-of-postgresql

[13] What is PostgreSQL? Key Features, Benefits, and Real-World Uses - Percona Blog, 2025 https://www.percona.com/blog/what-is-postgresql-used-for/

[14] Should I Use Entity Framework? - IAMTICOREY.COM Blog, 2024 https://blog.iamtimcorey.com/should-i-use-entity-framework/

[15] Advantages and disadvantages of Entityframework. - Medium Blog, 2023 https://medium.com/@anandugnath/advantages-and-disadvantages-of-entityframework-4c7b0e016788

[16] Introduction to Redis https://redis.io/about/

[17] Why You Should Use Redis Cache? - Medium Blog, 2023 https://medium.com/@vndpal/why-you-should-use-redis-cache-2e48bdd2c0be

[18] Stripe: Top Reasons Why It's The Best Payment Processor - WP Simple Pay Blog, 2025
https://wpsimplepay.com/reasons-to-use-stripe-to-accept-payments-on-wordpress/

[19] Why Use Docker: Real-life Use Cases, Examples, and Takeaways - Oursky Blog
https://www.oursky.com/blogs/why-use-docker-real-life-use-cases-examples-and-takeaways

[20] Visual Studio Code: Read this before you get started - daily.dev Blog, 2024
https://daily.dev/blog/visual-studio-code-read-this-before-you-get-started

[21] About GitHub and Git - Github Docs
http://docs.github.com/en/get-started/start-your-journey/about-github-and-git

[22] Microsoft Identity Platform: Introduction - Safetica, 2022
https://www.safetica.com/resources/blogs/microsoft-identity-platform-introduction#:~:text=It%20allows%20all%20identities%20and,into%20any%20application%20or%20infrastructure.

[23] MVC Framework Introduction - GeeksForGeeks Blog, 2025
https://www.geeksforgeeks.org/software-engineering/mvc-framework-introduction/

[24] Introduction to JSON Web Tokens - JWT.IO
https://www.jwt.io/introduction

[25] What is a Generic Repository? - Medium Blog, 2024
https://medium.com/@valentin.osidach/simple-unit-of-work-with-generic-repository-c-and-entity-framework-core-57845c5e277f

[26] Simplifying Business Rules with the Specification Pattern - DEV.TO Blog, 2025
https://dev.to/horse_patterns/simplifying-business-rules-with-the-specification-pattern-4o4o

[27] How to split an Angular app into micro-frontend apps - DEV.TO Blog, 2023
https://dev.to/michaeljota/how-to-split-an-angular-app-into-micro-frontend-apps-1fi9

ΠΑΡΑΡΤΗΜΑ Α

Κώδικας .NET έργου στο Back End

Program.cs

```
using API.Extensions;
using Core.Models.Auth;
using Infrastructure.Data;
using Infrastructure.Data.Identity;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Serilog;

var builder = WebApplication.CreateBuilder(args);

// Καταγραφή χρήσης και σφαλμάτων
Log.Logger = new LoggerConfiguration()
    .ReadFrom
    .Configuration(builder.Configuration)
    .CreateLogger();

// Χρήση βιβλιοθήκης καταγραφής
builder.Host.UseSerilog();

// Μέθοδος δημιουργία σύνδεσης στην βάση
builder.Services.AddDBConnection(builder.Configuration);

// Πρόσθετα διαχείρισης ηλεκτρονικών μηνυμάτων
builder.Services.AddEmailConfiguration(builder.Configuration);

// Πρόσθετα χρήσης Microsoft Identity
builder.Services.AddIdentityServiceExtensions(builder.Configuration);

// Χρήση Swagger για προβολή API Endpoints
builder.Services.AddSwaggerConfiguration();

// Χρήση Controllers αντί για MinimalAPI
builder.Services.AddControllers()
    // .AddJsonOptions(options => {
    //     options.JsonSerializerOptions.DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull;
    // })
    .AddNewtonsoftJson(x => {
        x.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore;
    });

// Εγγραφή DI Υπηρεσιών
builder.Services.AddServices();
builder.Services.AddStripeIntegration(builder.Configuration);

var app = builder.Build();

app.UseCors(
    options => options.WithOrigins("http://localhost:4200").AllowAnyMethod().AllowAnyHeader()
);

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment()) app.UseSwaggerDocumentation();

// Χρήση στατικών αρχείων από το API / εικόνες
app.UseStaticFilesConfigurations();

app.UseHttpsRedirection();
```

```
// JWT Auth
app.UseAuthentication();
app.UseAuthorization();

// Χρήση Controllers αντί για MinimalAPI
app.MapControllers();
app.MapFallbackToController("Index", "Fallback");

// Κώδικας που κατά την εκτέλεση γεμίζει την βάση με βασικά δεδομένα
using var scope = app.Services.CreateScope();
var services = scope.ServiceProvider;
var context = services.GetRequiredService<AppIdentityDbContext>();
var contextData = services.GetRequiredService<DatabaseContext>();
var userManager = services.GetRequiredService<UserManager<UserModel>>();
var roleManager = services.GetRequiredService<RoleManager<RoleModel>>();
var logger = services.GetRequiredService<ILogger<Program>>();
try
{
   await context.Database.MigrateAsync();
   await contextData.Database.MigrateAsync();
   await AppIdentityDbContextSeed.SeedUserAsync(userManager, roleManager);
   await DatabaseContextSeed.SeedDataAsync(contextData);
}
catch (Exception ex)
{
   logger.LogError(ex, "An error occured during migration");
}

app.Run(); // εκτέλεση API
```

DBConnectionServicesExtension.cs

Περιέχει την μέθοδο που εκτελεί την σύνδεση με την βάση δεδομένων μας

```
using Infrastructure.Data;
using Infrastructure.Data.Identity;
using Microsoft.EntityFrameworkCore;

// Μέθοδος που εκτελεί την σύνδεση με την βάση
namespace API.Extensions
{
   public static class DBConnectionServicesExtension
   {
      public static IServiceCollection AddDBConnection(this IServiceCollection services, IConfiguration config)
      {
         // Postgres
         services.AddDbContext<DatabaseContext>(x => x.UseNpgsql(config.GetConnectionString("DefaultConnection")));
         services.AddDbContext<AppIdentityDbContext>(x =>
x.UseNpgsql(config.GetConnectionString("IdentityConnection")));

         return services;
      }
   }
}
```

ServicesConfigurationsExtensions.cs

Στον παρακάτω κώδικα έχουμε δύο μεθόδους. Η μέθοδος AddServices() η οποία αρχικοποιεί και ορίζει τις διεπαφές του Dependency Injection απο ποιες υπηρεσίες χρησιμοποιούνται.

Η δεύτερη μεθοδος περιέχει την μέθοδο AddStripeIntegration() όπου περιέχει την εγγραφή και την Σύνδεση του Stripe API και του Redis.

```
using API.Helpers;
using API.Services.Interfaces;
using API.Services.Services;
using Core.Interfaces;
using Infrastructure.Repositories;
using Infrastructure.Repositories.BasketRepository.Interface;
using Infrastructure.Repositories.BasketRepository.Services;
using Infrastructure.Repositories.ProoopertiesRepositories;
using StackExchange.Redis;
using StripeIntegration;

namespace API.Extensions
{
    public static class ServicesConfigurationsExtensions
    {
        public static IServiceCollection AddServices(this IServiceCollection services)
        {
            services.AddScoped<ITokenService, TokenService>();
            services.AddScoped<IEmailSender, EmailSender>();
            services.AddScoped(typeof(IGenericRepository<>), (typeof(GenericRepository<>)));
            services.AddScoped<IBasketRepository, BasketRepository>();
            services.AddScoped(typeof(IPropertyRepository<>), (typeof(PropertyRepository<>)));
            services.AddAutoMapper(typeof(AutoMapperProfile));
            return services;
        }

        public static IServiceCollection AddStripeIntegration(this IServiceCollection services,
            IConfiguration configuration)
        {
            var stripeConfigurationSection = configuration.GetRequiredSection("Stripe");
            var stripeApiKey = stripeConfigurationSection.GetValue<string>("ApiKey");

            if (string.IsNullOrWhiteSpace(stripeApiKey))
            {
                StripeIntegrationException.ThrowMissingApiKeyException();
            }

            services.AddSingleton(new StripeCredentials(stripeApiKey!));
            services.AddSingleton<IStripeService, StripeService>();

            services.AddSingleton<IConnectionMultiplexer>(config => {
                var connString = configuration.GetConnectionString("Redis");
                if (connString is null) throw new Exception("Cannot get redis connection string");
                var conf = ConfigurationOptions.Parse(connString);
                return ConnectionMultiplexer.Connect(conf);
            });

            return services;
        }
    }
}
```

EmailSender.cs

Στην εφαρμογή μας χρειάζεται πολλές φορές να στείλουμε μηνύματα ηλεκτρονικού ταχυδρομειου στον χρήστη. Σε αυτό μας βοηθάει η κλάση EmailSender που υλοποιεί την Διεπαφή IEmailSender.

```
using API.Services.Interfaces;
using Core.Models.Emails;
using MailKit.Net.Smtp;
using MimeKit;

namespace API.Services.Services
{
    public class EmailSender : IEmailSender
```

```
    {
        private readonly EmailConfigurationModel _emailConfiguration;
        private readonly ILogger<EmailSender> _logger;
        public EmailSender(EmailConfigurationModel emailConfiguration, ILogger<EmailSender> logger)
        {
            _logger = logger;
            _emailConfiguration = emailConfiguration;
        }

        public async Task SendEmailAsync(MessageModel message)
        {
            var emailMessage = CreateEmailMessage(message);
            await SendAsync(emailMessage);
        }

        // Μέθοδος δημιουργίας μηνύματος ταχυδρομείου
        private MimeMessage CreateEmailMessage(MessageModel message)
        {
            var emailMessage = new MimeMessage();
            emailMessage.From.Add(new MailboxAddress(_emailConfiguration.SenderName!, _emailConfiguration.From!));
            emailMessage.To.AddRange(message.To);
            emailMessage.Subject = message.Subject;
            emailMessage.Body = new TextPart(MimeKit.Text.TextFormat.Html) { Text = message.Content };

            return emailMessage;
        }

        // Μέθοδος αποστολής μηνύματος
        private async Task SendAsync(MimeMessage message)
        {
            using (var client = new SmtpClient())
            {
                try
                {
                    await client.ConnectAsync(_emailConfiguration.SmtpServer, _emailConfiguration.Port,
_emailConfiguration.EnableSsl);
                    // client.AuthenticationMechanisms.Remove("XOAUTH2");
                    await client.AuthenticateAsync(_emailConfiguration.Username, _emailConfiguration.Password);

                    await client.SendAsync(message);
                }
                catch (Exception ex)
                {
                    // log
                    _logger.LogError($"Error sending message: {ex}");
                    throw;
                }
                finally
                {
                    await client.DisconnectAsync(true);
                    client.Dispose();
                }
            }
        }
    }
}
```

PropertiesController.cs

Η κλάση PropertiesController περιέχει τα τελικά σημεία του API μας για την διαχείριση των ακινήτων.

```
using AutoMapper;
using Core.DTOs.Properties;
using Core.Interfaces;
using Core.Models.Auth;
using Core.Models.Properties;
using Infrastructure.Data;
using Infrastructure.Repositories.ProopertiesRepositories;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace API.Controllers.Properties
{
   public class PropertyController : BaseApiController
   {
      private readonly ILogger<PropertyController> _logger;
      private readonly DatabaseContext _context;
      private readonly IMapper _mapper;
      private readonly IConfiguration _config;
      private readonly UserManager<UserModel> _userManager;
      private readonly IPropertyRepository<PropertyModel> _propertyRepo;
      public PropertyController(ILogger<PropertyController> logger, DatabaseContext context, IMapper mapper,
IConfiguration config, UserManager<UserModel> userManager, IPropertyRepository<PropertyModel> propertyRepo)
      {
         _propertyRepo = propertyRepo;
         _userManager = userManager;
         _config = config;
         _mapper = mapper;
         _context = context;
         _logger = logger;
      }

      // Δημιουργία νέας δημοσίευσης ακινήτων
      [Authorize] // Συνδεδεμένος χρήστης
      [HttpPost]
      public async Task<ActionResult<int>> NewPropertyAsync([FromBody] PropertyToAddDto model)
      {
         try
         {
            var identity = HttpContext.User.Identity;
            var property = await AddPropertyAsync(model, identity.Name);
            await AddPropertyImagesAsync(model.PropertyImages, property.Id);
            var characteristicsId = await AddPropertyCharacteristicsAsync(model.PropertyCharacteristics, property.Id);
            await AddPropertyAdditionalFeatureModelAsync(model.PropertyCharacteristics.AdditionalFeatures,
characteristicsId);
            return Ok(property.Id);
         }
         catch (Exception ex)
         {
            _logger.LogError(ex.Message);
            return BadRequest(ex.Message);
         }
      }

      // Εύρεση ακινήτου από χρήστη με βάση τον κωδικό ακινήτου
      [AllowAnonymous] // Επιτρέπεται από όλους τους χρήστες
      [HttpGet("{id}")]
      public async Task<ActionResult<PropertyDto>> GetPropertyByIdAsync(int id)
      {
```

```
        var property = await _context.Properties.Where(x => x.Id == id).Include(x => x.PropertyImages).FirstAsync();

        if (property is null) return NotFound();

        PropertyDto prop = _mapper.Map<PropertyDto>(property);

        var characteristics = await _context.PropertyCharacteristics.Where(x => x.PropertyId == property.Id).FirstAsync();

        PropertyCharacteristicsDto charact = _mapper.Map<PropertyCharacteristicsDto>(characteristics);

        var features = await _context.PropertyAdditionalFeatures.Where(x => x.PropertyCharacteristicId ==
characteristics.Id).FirstAsync();

        AdditionalFeaturesDto addFeatures = _mapper.Map<AdditionalFeaturesDto>(features);

        List<string> images = new List<string>();
        foreach (var image in property.PropertyImages)
        {
           images.Add(_config["ApiUrl"]+image.PropertyImagePath);
        }

        var user = await _userManager.FindByEmailAsync(property.UserEmail);

        charact.AdditionalFeatures = addFeatures;
        prop.PropertyCharacteristics = charact;
        prop.PropertyImages = images;
        prop.UserPhoneNumber = user!.PhoneNumber!;

        return Ok(prop);
    }

    // Αναζήτηση ακινήτων στην ιστοσελίδα
    [AllowAnonymous]
    [HttpGet]
    public async Task<ActionResult<IReadOnlyList<PropertyToReturnDto>>> GetPropertiesListAsync([FromQuery]
GetPropertiesDto parameters)
    {

        var spec = new PropertySpecifications(parameters.sort, parameters.categoryId, parameters.cityId, parameters.type);
        var properties = await _propertyRepo.ListAsync(spec);

        // Areas
        if (parameters.areaId?.Length > 0) properties = properties.Where(x =>
parameters.areaId.Contains(x.ComuneId)).ToList();

        // MinPrice
        if (parameters.minPrice.HasValue) properties = properties.Where(p => p.Price >= parameters.minPrice).ToList();

        // MaxPrice
        if (parameters.maxPrice.HasValue) properties = properties.Where(p => p.Price <= parameters.maxPrice).ToList();

        // SquareMeters
        if (parameters.squareMeters.HasValue) properties = properties.Where(p => p.SquareMetters >=
(parameters.squareMeters - 10) && p.SquareMetters <= (parameters.squareMeters + 10)).ToList();

        // MinRooms
        if (parameters.minRooms.HasValue) properties = properties.Where(p =>
p.PropertyCharacteristics?.NumberOfRooms >= parameters.minRooms).ToList();

        // MaxRooms
        if (parameters.maxRooms.HasValue) properties = properties.Where(p =>
p.PropertyCharacteristics?.NumberOfRooms <= parameters.maxRooms).ToList();

        // EnergyClass
        if (!string.IsNullOrEmpty(parameters.energyClass))
        {
           switch(parameters.energyClass)
```

```
            {
                case "low":
                    properties = properties
                        .Where(p => p.PropertyCharacteristics?.EnergyClass == "None" &&
                            p.PropertyCharacteristics.EnergyClass == "H" &&
                            p.PropertyCharacteristics.EnergyClass == "Z" &&
                            p.PropertyCharacteristics.EnergyClass == "E"
                        )
                        .ToList();
                    break;
                case "medium":
                    properties = properties
                        .Where(p => p.PropertyCharacteristics?.EnergyClass == "D" &&
                            p.PropertyCharacteristics.EnergyClass == "G" &&
                            p.PropertyCharacteristics.EnergyClass == "B" &&
                            p.PropertyCharacteristics.EnergyClass == "B+"
                        )
                        .ToList();
                    break;
                case "high":
                    properties = properties
                        .Where(p => p.PropertyCharacteristics?.EnergyClass == "A" &&
                            p.PropertyCharacteristics.EnergyClass == "A+" &&
                            p.PropertyCharacteristics.EnergyClass == "A++" &&
                            p.PropertyCharacteristics.EnergyClass == "A+++"
                        )
                        .ToList();
                    break;
            }
        }

        // MinConstYear
        if (parameters.minConstYear.HasValue) properties = properties.Where(p =>
p.PropertyCharacteristics?.ConstractionYear >= parameters.minConstYear).ToList();

        // MaxConstYear
        if (parameters.maxConstYear.HasValue) properties = properties.Where(p =>
p.PropertyCharacteristics?.ConstractionYear <= parameters.maxConstYear).ToList();

        var propertiesToReturn = _mapper.Map<IReadOnlyList<PropertyToReturnDto>>(properties);
        return Ok(propertiesToReturn);
    }

    // Προδιαγραφές ακινήτων με βάση τον κωδικό ακινήτου
    [AllowAnonymous]
    [HttpGet("{id}/Features")]
    public async Task<ActionResult<PropertyAdditionalFeatureModel>> GetPropertyFeaturesByProperyIdAsync(int id)
    {
        var characteristics = _context.PropertyCharacteristics
            .FirstAsync(f => f.PropertyId == id);
        var features = _context.PropertyAdditionalFeatures
            .FirstAsync(f => f.PropertyCharacteristicId == characteristics.Id);
        return Ok(features);
    }


    // Ιδιωτική μέθοδος που δημιουργεί το μοντέλο για την προσθήκη ακινήτου
    private async Task<PropertyModel> AddPropertyAsync(PropertyToAddDto model, string userEmail)
    {
        PropertyModel property = new PropertyModel()
        {
            PropertyFor = model.PropertyFor,
            SquareMetters = model.SquareMetters,
            Price = model.Price,
            Description = model.Description,
            DisabilitiesAccess = model.DisabilitiesAccess,
            View = model.View,
```

```
            Positioning = model.Positioning,
            Zone = model.Zone,
            DistanceFromSea = model.DistanceFromSea,
            DistanceFromCity = model.DistanceFromCity,
            DistanceFromCenter = model.DistanceFromCenter,
            DistanceFromAirport = model.DistanceFromAirport,
            UserEmail = userEmail,
            PropertyTypeId = model.PropertyTypeId,
            Address = model.Address,
            Alias = model.Alias,
            AccessFrom = model.AccessFrom
        };

        if (!string.IsNullOrEmpty(model.PropertyCategoryId))
        {
            var cat = await _context.PropertyCategories.Where(x => x.PropertyCategoryName == model.PropertyCategoryId
&& x.PropertyTypeId == model.PropertyTypeId).FirstAsync();
            property.PropertyCategoryId = cat.Id;
        }

        var citta = await _context.Citta.Where(c => c.Citta == model.Citta).FirstAsync();
        property.CittaId = citta.Id;
        var comune = await _context.Comuni.Where(c => c.CittaId == citta.Id && c.Comune ==
model.Comune).FirstAsync();
        property.ComuneId = comune.Id;
        var response = await _context.Properties.AddAsync(property);
        await _context.SaveChangesAsync();
        return response.Entity;
    }

    // Add PropertyImages
    private async Task AddPropertyImagesAsync(ICollection<string> images, int propertyId)
    {
        foreach (var image in images)
        {
            await _context.PropertyImages.AddAsync(new PropertyImageModel() {
                PropertyId = propertyId,
                PropertyImagePath = image
            });
        }
        await _context.SaveChangesAsync();
    }

    //  Ιδιωτική μέθοδος που δημιουργεί το μοντέλο για την προσθήκη χαρακτηριστικών ακινήτου
    private async Task<int> AddPropertyCharacteristicsAsync(PropertyCharacteristicsDto model, int propertyId)
    {
        PropertyCharacteristicModel property = new PropertyCharacteristicModel()
        {
            ConstractionYear = model.ConstructionYear,
            RenovatedYear = model.RenovatedYear,
            RenovationType = model.RenovationType,
            ApartmentFloor = model.ApartmentFloor,
            EnergyClass = model.EnergyClass,
            Heating = model.Heating,
            TypeOfHeating = model.TypeOfHeating,
            NumberOfRooms = model.NumberOfRooms,
            Bathrooms = model.Bathrooms,
            Kitchens = model.Kitchens,
            // LivingRooms = model.LivingRooms,
            Balconies = model.Balconies,
            Tarraces = model.Tarraces,
            ParkingSpaces = model.ParkingSpaces,
            ParkingType = model.ParkingType,
            LandSquareMeters = model.LandSquareMeters,
            BuildingPermit = model.BuildingPermit,
            BuildingSquareMeters = model.BuildingSquareMeters,
            Shape = model.Shape,
```

```
            FrontageLength = model.FrontageLength,
            CityPlan = model.CityPlan,
            PropertyId = propertyId,
            WCs = model.WCs,
            ConstructionMeters = model.ConstructionMeters
        };
        var characteristics = await _context.PropertyCharacteristics.AddAsync(property);
        await _context.SaveChangesAsync();
        return characteristics.Entity.Id;
    }

    //  Ιδιωτική μέθοδος που δημιουργεί το μοντέλο για την προσθήκη προδιαγραφών ακινήτου
    private async Task AddPropertyAdditionalFeatureModelAsync(AdditionalFeaturesDto model, int characteristicsId)
    {
        PropertyAdditionalFeatureModel property = new PropertyAdditionalFeatureModel()
        {
            Fournished = model.Fournished,
            ElectricDevices = model.ElectricDevices,
            MaintenanceFees = model.MaintenanceFees,
            DoorType = model.DoorType,
            FloorType = model.FloorType,
            Warehouse = model.Warehouse,
            PetsAllowed = model.PetsAllowed,
            Cameras = model.Cameras,
            CableTv = model.CableTv,
            FloorHeating = model.FloorHeating,
            InternalStairs = model.InternalStairs,
            Playroom = model.Playroom,
            Elevator = model.Elevator,
            SolarWaterHeater = model.SolarWaterHeater,
            Garden = model.Garden,
            // Incomplete = model.Incomplete,
            HousingStatus = model.HousingStatus,
            Luxurious = model.Luxurious,
            Jacuzzi = model.Jacuzzi,
            SateliteAntenna = model.SateliteAntenna,
            MotorisedShutters = model.MotorisedShutters,
            Neoclassical = model.Neoclassical,
            Veranda = model.Veranda,
            // Maintainable = model.Maintainable,
            Awnings = model.Awnings,
            AlarmSystem = model.AlarmSystem,
            Pool = model.Pool,
            PropertyCharacteristicId = characteristicsId
        };

        var features = await _context.PropertyAdditionalFeatures.AddAsync(property);
        await _context.SaveChangesAsync();
    }
  }
}
```

Για την εκτέλεση του API μας θα πρέπει να τρέξουμε τις παρακάτω εντολές σε σύστημα UNIX.

1. cd API: μπαίνουμε στον φάκελο API
2. dotnet run: εκτελούμε το Program.cs
3. Επισκεπτόμαστε την σελίδα http://localhost:5270/swagger/index.html όπου μπορούμε να δούμε τα τελικά σημεία (endpoints) του API μας.

ΠΑΡΑΡΤΗΜΑ Β

Κώδικας Angular Έργου στο Front End.

app.module.ts

Το αρχείο app.module.ts περιέχει ότι χρειαζόμαστε για την δημιουργία και εγγραφή του front end μας.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { CoreModule } from './core/core.module';
import { StoreModule } from '@ngrx/store';
import { metaReducers, reducers } from './reducers';
import { StoreDevtoolsModule } from '@ngrx/store-devtools';
import { EffectsModule } from '@ngrx/effects';
import { HttpClientModule } from '@angular/common/http';
import { RouterModule, Routes } from '@angular/router';
import { environment } from 'src/environments/environment';
import { EntityDataModule } from '@ngrx/data';
import { RouterState, StoreRouterConnectingModule } from '@ngrx/router-store';
import { AuthModule } from './modules/auth/auth.module';
import { NotFoundComponent } from './modules/pages/not-found/not-found.component';
import { AuthGuard } from './modules/auth/services/auth.guard';

const routes: Routes = [
  {
    path: '',
    loadChildren: () => import('./modules/pages/pages.module').then(m => m.PagesModule)
  },
  {
    path: 'auth',
    loadChildren: () => import('./modules/auth/auth.module').then(m => m.AuthModule)
  },
  {
    path: 'profile',
    loadChildren: () => import('./modules/profile/profile.module').then(m => m.ProfileModule),
    canActivate: [AuthGuard]
  },
  {
    path: 'properties',
    loadChildren: () => import('./modules/properties/properties.module').then(m => m.PropertiesModule)
  },
  {
    path: 'shop',
    loadChildren: () => import('./modules/shop/shop.module').then(m => m.ShopModule)
  },
  // 404
  {
    path: '**',
    pathMatch: 'full',
    component: NotFoundComponent
  }
];

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    CoreModule,
    HttpClientModule,
    RouterModule.forRoot(routes, {}),
    AuthModule.forRoot(),
```

```
    StoreModule.forRoot(reducers, {
      metaReducers,
      runtimeChecks: {
        strictActionImmutability: true,
        strictStateImmutability: true,
        strictActionSerializability: true,
        strictStateSerializability: true,
      }
    }),
    StoreDevtoolsModule.instrument({
      maxAge: 25,
      logOnly: !environment.production
    }),
    EffectsModule.forRoot([]),
    EntityDataModule.forRoot({}),
    StoreRouterConnectingModule.forRoot({
      stateKey: 'router',
      routerState: RouterState.Minimal
    })
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.html

Το αρχείο app.component.html περιέχει τον βασικό html κώδικα το οποίο φορτώνει την εφαρμογή μας. HTML ετικέτα:

- app-header φορτώνει το Component που περιέχει το Header της ιστοσελίδας μας
- app-footer φορτώνει το footer της ιστοσελίδας μας
- router-outlet φορτώνει τα component που φορτώνονται από την δρομολόγηση (routing) των σελίδων μας.

```
<main class="w-full flex justify-center items-center flex-col">
    <header class="w-full max-w-[1100px] lg:px-6">
        <app-header class="w-full"></app-header>
    </header>
    <section class="w-full max-w-[1100px] px-4 lg:px-6">
        <router-outlet *ngIf="!loading" (activate)="onActivate($event)"></router-outlet>
        <div *ngIf="this.loading" role="status" class="w-full h-[500px] inline-flex justify-center items-center">
                    <img src=”./logo.png” alt=”logo”/>
        </div>
    </section>
    <footer class="w-full max-w-[1100px] px-4 lg:px-6">
        <app-footer class="w-full"></app-footer>
    </footer>
</main>
```

app.component.ts

Η κλαση AppComponent περιέχει των TypeScript κώδικα που κατά την σύνταξη (compile) του κώδικα γίνεται Javascript δημιουργεί. Εκτελείτε και φορτώνει το app.component.html. Κατά την εκτέλεση του κάνει έλεγχο εάν ο χρήστης είναι συνδεδεμένος η όχι.

```
import { Component, inject, OnInit } from '@angular/core';
import { initFlowbite } from 'flowbite';
import { AppState } from './reducers';
import { select, Store } from '@ngrx/store';
import { NavigationCancel, NavigationEnd, NavigationError, NavigationStart, Router } from '@angular/router';
import { login } from './modules/auth/services/auth.actions';
import { Observable } from 'rxjs';
import { isLoggedIn, isLoggedOut } from './modules/auth/services/auth.selectors';
import { AuthService } from './modules/auth/services/auth.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {
  private authService = inject(AuthService);
  title = 'Opla Immobilliare';

  loading = true;

  isLoggedIn$: Observable<boolean> | undefined;

  isLoggedOut$: Observable<boolean> | undefined;

  constructor(private store: Store<AppState>, public router: Router) {}

  ngOnInit(): void {
    initFlowbite();

    const userProfile = localStorage.getItem("user");

    if (userProfile) {
      this.store.dispatch(login({ user: JSON.parse(userProfile) }));
      let user = JSON.parse(userProfile);
      if (user.role == "agency") this.authService.setAgency(true);
      if (user.role == "user") this.authService.setUser(true);
    }

    // Loading
    this.router.events.subscribe(event => {
      switch (true) {
        case event instanceof NavigationStart: {
          this.loading = true;
          break;
        }

        case event instanceof NavigationEnd:
        case event instanceof NavigationCancel:
        case event instanceof NavigationError: {
          this.loading = false;
          break;
        }

        default: {
          break;
        }
      }
    });
```

```
    this.isLoggedIn$ = this.store
      .pipe(
        select(isLoggedIn)
      );

    this.isLoggedOut$ = this.store
      .pipe(
        select(isLoggedOut)
      );
  }

  onActivate(event: any) {
    window.scroll(0,0);
  }
}
```

home.component.ts

Το αρχείο home.component.ts εκτελεί και φορτώνει την αρχική σελίδα της ιστοσελίδας μας. η κλάση περιέχει και τις απαραίτητες μεθόδους για την αναζήτηση ενός ακινήτου από κάποιον χρήστη.

```
import { Component, inject, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';
import { SearchCriteria } from './models/searchCriteria.model';
import { SearchCriteriaService } from './services/search-criteria-service.service';
import { Observable } from 'rxjs';
import { CittaEComune } from './models/cittaEComune.model';
import { SearchService } from './services/search.service';
import { PropertyCategories, PropertyTypesWithCategories } from './models/propertyTypesWithCategories.model';
import { PropertyCategoriesEntityService } from './services/property-categories-entity.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})
export class HomeComponent implements OnInit {
  private searchService = inject(SearchService);
  private propertyCategories = inject(PropertyCategoriesEntityService);
  private router = inject(Router);

  searchForm: FormGroup;
  searchCriteriaModel?: SearchCriteria;
  areas: string[] = [];
  isfrmChecked: any;
  categories: string[] = [];
  isCatChecked: any;

  cittaEComune$: CittaEComune[] = [];
  typesWithCategories$: Observable<PropertyTypesWithCategories[]> = new
Observable<PropertyTypesWithCategories[]>();

  areasModal: boolean = false;
  categoriesModal: boolean = false;
  categoriesDropdown: boolean = false;
  cityId: number = 0;

  constructor(private searchCriteriaService: SearchCriteriaService) {
    this.searchForm = this.generateSearchForm();
  }

  // Δημιουργία φόρμας αναζήτησης
  generateSearchForm(): FormGroup {
    return new FormGroup({
```

```
    searchType: new FormControl<string>("rent"),
    category: new FormControl<string>("Tutte"),
    categoryId: new FormControl<number | undefined>(undefined),
    municipality: new FormControl<string | undefined>(undefined, [Validators.required]),
    municipalityId: new FormControl()
  })
}

// Δημιουργία κριτηρίων αναζήτησης ως παράμετροι στο αίτημα του API
searchCriteria(): void {
  let params = new URLSearchParams();
  this.areas.forEach(res => {
    params.append("area", res);
  });
  this.searchCriteriaModel = {
    searchType: this.searchForm.value.searchType,
    areas: params.toString(),
    category: this.searchForm.value.categoryId,
    municipality: this.searchForm.value.municipality,
  }
  params.append("type", this.searchForm.value.searchType);
  if (this.searchForm.value.categoryId) params.append("category", this.searchForm.value.categoryId);
  if (this.cityId != 0) params.append("cityId", this.cityId.toString());
  this.searchCriteriaService.updateSearchCriterria(this.searchCriteriaModel!);
  // Ανακατεύθυνση στην λίστα ακινήτων
  this.router.navigateByUrl(`/properties/${this.cittaEComune$[0].citta}?${params.toString()}`);
}

ngOnInit(): void {
  this.propertyCategories.entities$.pipe(res => this.typesWithCategories$ = res);
}

// HTTP αίτημα για αναζήτηση πόλης και δήμου
getCittaEComune(): void {
  if (this.searchForm.value.municipality != undefined && this.searchForm.value.municipality.length >= 4) {
    this.searchService.getComune(this.searchForm.value.municipality).subscribe( res =>{
      this.cittaEComune$ = res;
    });
  }
}


addAreas(event: any, isChecked: boolean) {
  if (isChecked) {
    this.areas.push(event.target.value);
  }
  else {
    let index = this.areas.indexOf(event.target.value);
    this.areas.splice(index, 1);
  }
}

// Ορισμός κατηγορίας αναζήτησης
setCategory(val: PropertyCategories): void {
  this.searchForm.patchValue({
    category: val.categoryName,
    categoryId: val.id,
  });
  this.categoriesModal = false;
  this.categoriesDropdown = false;
}

// Ορισμός πόλης αναζήτησης
setCity(cityId: number): void {
  this.cityId = cityId;
}
}
```

property-categories-data.service.ts

Το αρχείο property-categories-data.service.ts στέλνει ένα HTTP αίτημα στο backend ώστε να ζητήσει την λίστα με τα ακίνητα που έχει ζητήσει ο πελάτης.

```
import { DefaultDataService, HttpUrlGenerator } from "@ngrx/data";
import { PropertyTypesWithCategories } from "../models/propertyTypesWithCategories.model";
import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { HttpOptions } from "@ngrx/data/src/dataservices/interfaces";
import { Observable } from "rxjs";
import { environment } from "src/environments/environment";

@Injectable()
export class PropertyCategoriesDataService extends DefaultDataService<PropertyTypesWithCategories> {

   constructor(http: HttpClient, httpUrlGenerator: HttpUrlGenerator) {
      super('PropertyTypesWithCategories', http, httpUrlGenerator);
   }

   // HTTP αίτημα στο API
   override getAll(options?: HttpOptions): Observable<PropertyTypesWithCategories[]> {
      return this.http.get<PropertyTypesWithCategories[]>(`${environment.apiUrl}/PropertyTypes/categories`)
   }
}
```

login.component.ts

Το Αρχείο login.compoent.ts φορτώνει το component με την σελίδα σύνδεσης του χρήστη.

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';
import { AuthService } from '../services/auth.service';
import { Router } from '@angular/router';
import { AppState } from 'src/app/reducers';
import { Store } from '@ngrx/store';
import { login } from '../services/auth.actions';
import { noop, tap } from 'rxjs';
import { environment } from 'src/environments/environment';
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss'],
})

export class LoginComponent implements OnInit {

  loginForm: FormGroup;
  type: string = 'login';

  constructor(
    private auth: AuthService,
    private router: Router,
    private store: Store<AppState>
  ) {
    this.loginForm = this.generateLoginForm();
  }

  ngOnInit(): void {
  }

  // Σύνδεση χρήστη
  doLogin(): void {
    let obj = new Object({
      email: this.loginForm.value.email,
```

```
    password: this.loginForm.value.password
   });
   this.auth.login(obj)
    .pipe(
     tap(user => {
      this.store.dispatch(login({ user }));
      if (user.role === 'agency') this.auth.setAgency(true);
      if (user.role === 'user') this.auth.setUser(true);
      if (user.completedProfile) {
       // this.router.navigateByUrl('/');
       window.location.href = `${environment.clientUri}/profile`
      }
      window.location.href = `${environment.clientUri}/profile/complete`
     })
    )
    .subscribe(
     noop,
     () => alert("Login failed")
    )
  }

  // Δημιουργία φόρμας σύνδεσης
  generateLoginForm(): FormGroup{
   return new FormGroup({
    email: new FormControl(undefined, [Validators.email, Validators.required]),
    password: new FormControl(undefined, [Validators.required])
   });
  };

  labelControl(name: string): string {
   if (this.loginForm.controls[name].valid && this.loginForm.controls[name].touched) return 'bg-[#CCC2B2] px-1 py-0.5';
   if (this.loginForm.controls[name].invalid && (this.loginForm.controls[name].touched ||
this.loginForm.controls[name].dirty)) return 'bg-[#FF5449] peer-focus:bg-[#FF5449] py-0.5 px-1';
   return 'bg-none';
  }
}
```

basket.service.ts

Το Αρχείο basket.service.ts περιέχει τις απαραίτητες συναρτήσεις για την επικοινωνία και την διαχείριση του καλαθιού. Το καλάθι χρησιμοποιείται ως σελίδα χρήσης πριν την ολοκλήρωση παραγγελίας όπου ο χρήστης μπορεί να διαχειριστεί την συνδρομή του.

```
import { Injectable, signal } from "@angular/core";
import { BehaviorSubject } from "rxjs";
import { Basket } from "../models/basket.model";
import { HttpClient } from "@angular/common/http";
import { environment } from "src/environments/environment";

@Injectable({
   providedIn: 'root'
})
export class BasketService {
   private basketSource = new BehaviorSubject<Basket | null>(null);
   basketSource$ = this.basketSource.asObservable();
   basket = signal<Basket | null>(null);

   constructor(private http: HttpClient) {}

   // HTTP αίτημα στο API για να μας φέρει το καλάθι από το Redis
   getBasket(id: string) {
      return this.http.get<Basket>(`${environment.apiUrl}/Basket?id=${id}`).subscribe({
         next: basket => this.basketSource.next(basket)
      })
   }
```

```
  // HTTP αίτημα στο API για την δημιουργία νέου καλαθιού στο Redis
  setBasket(basket: Basket) {
    const createdBasket = this.createBasket();
    basket.id = createdBasket.id;
    return this.http.post<Basket>(`${environment.apiUrl}/Basket`, basket).subscribe({
      next: basket => this.basketSource.next(basket)
    })
  }

  // Ιδιωτική μέθοδος η οποία δημιουργεί ένα μοντέλο καλαθιού
  private createBasket(): Basket {
    const basket = new Basket();
    localStorage.setItem('basket_id', basket.id);
    return basket;
  }
}
```

Για την εκτέλεση του FrontEnd μας θα πρέπει να τρέξουμε τις παρακάτω εντολές σε σύστημα UNIX.

1. npm i ώστε να κατεβάσουμε τα απαραίτητα npm modules
2. ng server για να τρέξει ο angular εξυπηρετητής
3. Επισκεπτόμαστε την σελίδα http://localhost:4200 για να δούμε την ιστοσελίδα μας

ΠΑΡΑΡΤΗΜΑ Γ

Κώδικας δημιουργίας Docker Container

```
services:
  redis:
    image: redis:latest
    ports:
      - 6379:6379
    command: ["redis-server", "--appendonly", "yes"]
    volumes:
      - redis-data:/data
   db:
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: *****
      POSTGRES_USER: appuser
    ports:
      - 5432:5432
    volumes:
      - postgres-data:/data
volumes:
  redis-data:
  postgres-data:
```