ΑΕΙ ΠΕΙΡΑΙΑ Τ. Τ.

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ

ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ Τ. Ε.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και υλοποίηση ενός Chat Web Application

Καπασάκης Κώστας

Εισηγητής: Πρεζεράκος Γεώργιος , Καθηγητής

ΑΘΗΝΑ ΦΕΒΡΟΥΑΡΙΟΣ 2018

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και υλοποίηση ενός Chat Web Application

Κωνσταντίνος Καπασάκης

Α.Μ. 40817

Εισηγητής:

Πρεζεράκος Γεώργιος , Καθηγητής

Εξεταστική Επιτροπή:

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο/Η κάτωθι υπογεγραμμένος Καπασάκης Κωνσταντίνος του Ιωσήφ με αριθμό μητρώου 40817 φοιτητής/τρία του Τμήματος Μηχανικών Η/Υ Συστημάτων Τ.Ε. του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή εργασία ολοκληρώθηκε έπειτα από μεγάλη προσπάθεια για κατανόηση δύσκολων αλλά εξίσου ενδιαφέροντών τεχνολογιών. Την προσπάθειά μου αυτή υποστήριξε ο επιβλέπων καθηγητής μου, τον οποίο θα ήθελα να ευχαριστήσω.

Ακόμα θα ήθελα να ευχαριστήσω τους συναδέλφους για τις πολύτιμες συμβουλές του και την οικογένειά μου.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία ασχολείται με τη σχεδίαση και την ανάπτυξη μίας Web Chat εφαρμογής χρησιμοποιώντας τις ποιο σύγχρονες τεχνολογίες του Web Development. Αντιμετωπίζει το πρόβλημα της real-time λειτουργικότητας και προτείνει συγκεκριμένες μεθοδολογίες που με τις οποίες μπορεί ο κάθε μηχανικός να καταφέρει παρόμοια αποτελέσματα έχοντας πρώτα καταλάβει συγκεκριμένες έννοιες

ABSTRACT

The present thesis concerned with the design and implementation of a Web Chat Application which uses the latest and up to date technologies of Web Development. Moreover, it confronts the problem of real-time functionality and suggests specific methodologies which every software engineer can use and achieve similar results after he or she has a good understanding of specific concepts.

Περιεχόμενα

1. ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο αναλύεται το αντικείμενο της πτυχιακής εργασίας και ο τρόπος με τον οποίο οργανώθηκε το κείμενο.

1.1 Σκοπός της πτυχιακής εργασίας

Το αντικείμενο της παρούσας πτυχιακής εργασίας είναι η ανάπτυξη μιας ολοκληρωμένης διαδικτυακής εφαρμογής αντιμετωπίζοντας παράλληλα το πρόβλημα της real-time λειτουργικότητας. Συγκεκριμένα παρέχει ένα online chat χώρο για όλους τους εγγεγραμμένους χρήστες.

Το Free Chat (όνομα εφαρμογής) δίνει της δυνατότητα στους χρήστες να γίνουν μέλη του site και να πάρουν μέρος σε όποιο chat room επιθυμούν. Επίσης δίνεται η δυνατότητα οι χρήστες να φτιάξουν το δικό τους chat room το οποίο θα είναι μέρος μιας από τις κύριες κατηγορίες που προσφέρει η εφαρμογή. Οι χρήστες κατηγοριοποιούνται σε ρόλους όπως εγγεγραμμένοι, μη εγγεγραμμένοι, διαχειριστές και ανάλογα με το ρόλους τους έχουν πρόσβαση σε διαφορετικές λειτουργίες της εφαρμογής. Οι διαχειριστές έχουν τη δυνατότητα να επηρεάσουν το status των χρηστών και να κάνουν ένα συνολικό monitoring της εφαρμογής.

Η εφαρμογή υλοποιήθηκε χρησιμοποιώντας την πλατφόρμα Microsoft Visual Studio 2015.Πιο συγκεκριμένα χρησιμοποιήθηκε η γλώσσα προγραμματισμού C# για το Server κομμάτι της εφαρμογής όπως και η βιβλιοθήκη της Microsoft , SignalR για την real-time λειτουργικότητα .Επίσης η γλώσσα προγραμματισμού JavaScript για το client κομμάτι της εφαρμογής. Για σχεδίαση και την υλοποίησης της Βάσης Δεδομένων αλλά και για την πρόσβαση στα δεδομένα χρησιμοποιήθηκε το Entity Framework 6.

1.2 Οργάνωση της πτυχιακής εργασίας

Στα πλαίσια της πτυχιακής αναλύεται τόσο η μεθοδολογία σχεδιασμού της εφαρμογής όσο και η υλοποίησή της. Αποτελείται από 7 κεφάλαια.

Στο 1ο κεφάλαιο γίνεται μια εισαγωγή όπου αναφέρονται το αντικείμενο της πτυχιακής εργασίας και η οργάνωση της.

Στο 2^{ο} κεφάλαιο γίνεται αναφορά στο συνολικό λογισμικό πακέτο που χρησιμοποιήθηκε και σε διάφορα design patterns που οργάνωσαν την εφαρμογή

Στο 3^{ο} κεφάλαιο γίνεται μια λεπτομερής περιγραφή της εφαρμογής τόσο στο client κομμάτι όσο και στο server κομμάτι.

Στο 4^{ο} κεφάλαιο γίνεται μια λεπτομερής περιγραφή στο τρόπο υλοποίησης της βάσης με το Entity Framework αλλά και πώς υλοποιήθηκε το Repository Pattern

Στο 5^{ο} κεφάλαιο γίνεται μία περιγραφή του client side της εφαρμογής και του τρόποι που οργανώθηκε

Στο 6^{ο} κεφάλαιο αναλύεται το SignalR και ο τρόπο με τον οποίο χρησιμοποιήθηκε στην εφαρμογή

Στο 7^{ο} κεφάλαιο γίνεται ανασκόπηση της πτυχιακής εργασίας και παρουσιάζονται συμπεράσματα και τυχόν μελλοντικές επεκτάσεις που θα μπορούσαν να γίνουν με σκοπό τη βελτίωση της εφαρμογής

Αναφορές σε βιβλία, σε διαδικτυακές σελίδες και σε επιστημονικά άρθρα που κατέστησαν δυνατή τη πραγματοποίηση της εργασίας αυτής τόσον όσον αφορά στην σύνταξη του βιβλίου όσο και την υλοποίηση της εφαρμογής υπάρχουν στο τέλος κάθε κεφαλαίου

2. ΤΟ ΛΟΓΙΣΜΙΚΟ ΠΑΚΕΤΟ

2.1- NET framework

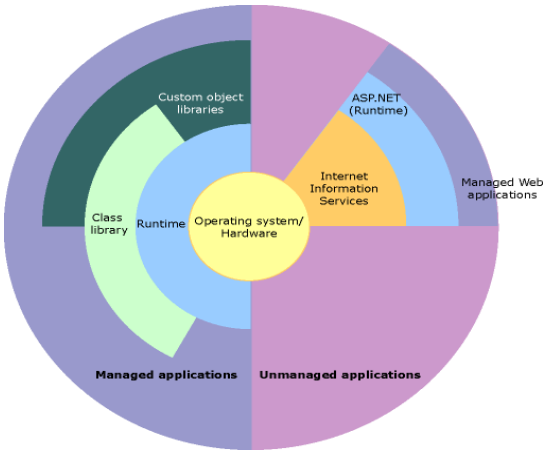
Το .Net framework είναι μία τεχνολογία που υποστηρίζει την ανάπτυξη και την υποστήριξη εφαρμογών. Είναι σχεδιασμένο για να ικανοποιεί τα παρακάτω:

- Να παρέχει ένα συνεπές αντικειμενοστραφή προγραμματιστικό περιβάλλον .
- Να παρέχει ένα περιβάλλον εκτέλεσης κώδικα που ελαχιστοποιεί κινδύνους versioning
- Να παρέχει ένα περιβάλλον εκτέλεσης κώδικα που προμοτάρει την ασφαλή εκτέλεση κώδικα συμπεριλαμβανομένου κώδικα από άγνωστου η ημι-εμπιστεύσιμης πηγής (third party libraries)
- Να προσφέρει συνεπή εμπειρία στον προγραμματιστή σε όλα τα είδη εφαρμογών

Το. NET αποτελείται από το common language run time (CLR) και το .NET class library.

- The CLR είναι ένα run-time περιβάλλον το οποίο εκτελεί τον κώδικα και παρέχει υπηρεσίες που κάνουν την διαδικασία παραγωγής κώδικα ευκολότερη.
- Το .NET class library είναι μία βιβλιοθήκη από κλάσεις , interfaces, και άλλου τύπους που παρέχουν λειτουργικότητα στο σύστημα

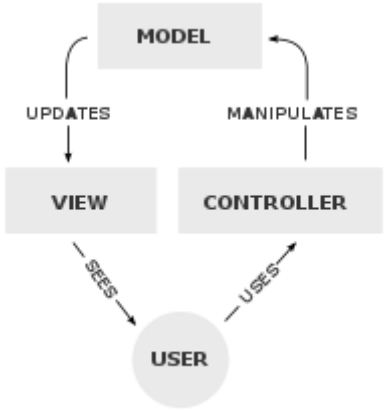
Ο κώδικας που εκτελείται στο CLR έρχεται υπό μορφή assemblies. Το .NET χαρακτηρίζεται από την λεγόμενη just in time μεταγλώττιση. Ο κώδικας στη γλώσσα προγραμματισμού που χρησιμοποιεί ο χρήστης μεταγλωττίζεται αρχικά σε κάποια ενδιάμεση γλώσσα (IL) η οποία αποθηκεύεται σε ένα εκτελέσιμο αρχείο η σε μια βιβλιοθήκη. Όταν ο χρήστης τρέξει το πρόγραμμα το CLR διαβάζει το IL(intermediate language) το μεταγλωττίζει σε κώδικα windows και στην συνέχεια τον εκτελεί.



Εικόνα 2-1: Το παραπάνω σχήμα μας δείχνει τη σχέση του CLR και του class library στις εφαρμογές μας αλλά και στο συνολικό σύστημα

2.2 MVC μοντέλο αρχιτεκτονικής λογισμικού

Το Model–view–controller (σε συντομογραφία αναφέρεται ως MVC) είναι ένα μοντέλο αρχιτεκτονικής λογισμικού το οποίο χρησιμοποιείται για την δημιουργία περιβαλλόντων αλληλεπίδρασης χρήστη. Στο μοντέλο αυτό η εφαρμογή διαιρείται σε τρία διασυνδεδεμένα μέρη ώστε να διαχωριστεί η παρουσίαση της πληροφορίας στον χρήση από την μορφή που έχει αποθηκευτεί στο σύστημα. Το κύριο μέρος του μοντέλου είναι το αντικείμενο *Model* το οποίο διαχειρίζεται την ανάκτηση/αποθήκευση των δεδομένων στο σύστημα. Το αντικείμενο *View* χρησιμοποιείται μόνο για να παρουσιάζεται η πληροφορία στον χρήστη (π.χ. με γραφικό τρόπο). Το τρίτο μέρος είναι ο *Controller* ο οποίος δέχεται την είσοδο και στέλνει εντολές στο αντικείμενο *Model* και στο *View*



Εικόνα 2-2 :Στη συγκεκριμένη εφαρμογή χρησιμοποιήθηκε το ASP.NET MVC 5 το οποίο είναι framework της Microsoft που υλοποιεί αυτήν την αρχιτεκτονική

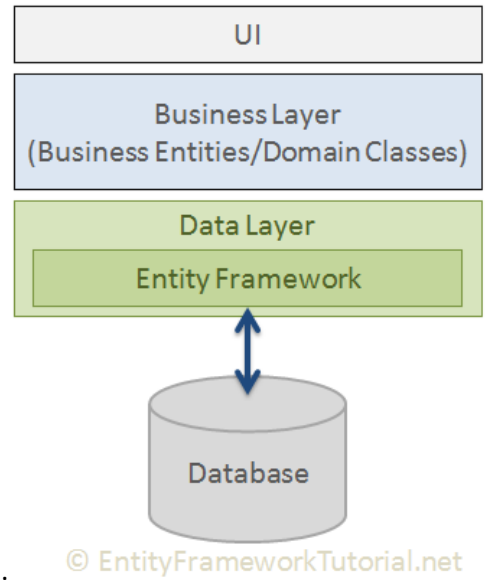
2.3 Entity Framework

Μέχρι και την έκδοση .NET 3.5 οι μηχανικοί που δουλεύανε με .NET αναγκαζόντουσαν να γράφουνε ADO-NET ή Enterprise Data access κώδικα ώστε να αποθηκεύσουνε ή να ανακτήσουν δεδομένα από τη βάση δεδομένων. Η τότε συνήθης διαδικασία ήταν:

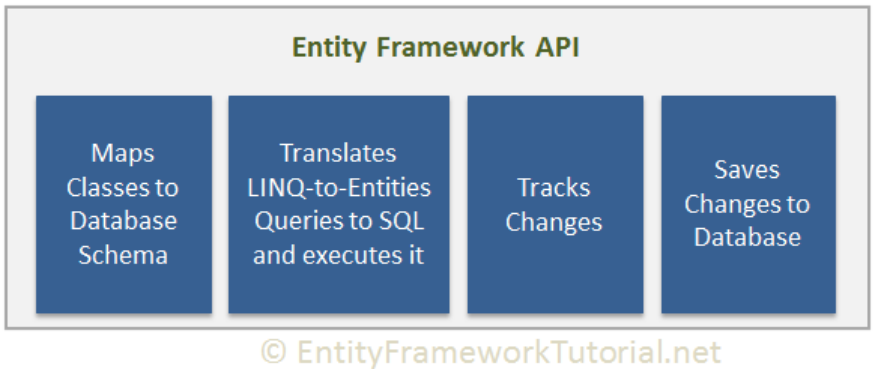
- Άνοιγμα σύνδεσης με την βάση
- Δημιουργία ενός Dataset για να πάρουνε η να αποθηκεύσουνε δεδομένα
- Μετατροπή αυτού του dataset σε .NET αντικείμενα ή το αντίθετο και εφαρμογή Business κανόνων

Αυτή η διαδικασία ήταν χρονοβόρα και ανοικτή σε πολλά λάθη. Έτσι η Microsoft δημιούργησε το Entity Framework για να παρέχει ένα framework που αυτοματοποιεί αυτή την διαδικασία για τις εφαρμογές μας.

Το Entity framework επομένως είναι ένα Open Source ORM framework που υποστηρίζεται από την Microsoft. Χρησιμοποιώντας το Entity Framework οι developers μπορούνε να δουλεύουνε σε ένα πιο αφηρημένο επίπεδο όταν ασχολούνται με δεδομένα και μπορούνε να δημιουργήσουν και να συντηρήσουν data-oriented εφαρμογές γράφοντας λιγότερο κώδικα από ότι με τις παραδοσιακές μεθόδους



Εικόνα 2-3:Το παραπάνω σχήμα δείχνει που προσδιορίζεται το Entity Framework στις εφαρμογές μας.



Εικόνα 2-4:Το παραπάνω σχήμα δείχνει τη βασική ροή λειτουργίας του Entity Framework

Χρησιμοποιώντας το Entity Framework ένας προγραμματιστής πρέπει να διαλέξει μία από τις προγραμματιστικές προσεγγίσεις που υποστηρίζει.

- Code First
- Database-First
- Model-first

Πλέον η Microsoft έχει σταματήσει να υποστηρίζει στα καινούργια της πλάνα τις δύο τελευταίες προσεγγίσεις και όλη η αγορά στρέφεται προς το code first όπως και αυτή η πτυχιακή.

Χρησιμοποιώντας την Code-First προσέγγιση δεν χρειάζεται να έχουμε ήδη μια στημένη βάση δεδομένων με την οποία θα συνδέσουμε την εφαρμογή μας ,αντιθέτως ξεκινάμε το στήσιμο της εφαρμογής μας από τις βασικές κλάσεις που αποτελούν την εφαρμογή μας και έπειτα φτιάχνουμε τη βάση μας από αυτές τις κλάσεις. Οι προγραμματιστές οι οποίοι ακολουθούν τους κανόνες του DDD(Domain Driven Development) προτιμούν συνήθως να χρησιμοποιούν αυτή τη προσέγγιση

αλλά όπως προαναφέρθηκε αρχίζει πλέον η code-first προσέγγιση να είναι η κύρια εκπρόσωπος του Entity Framework.

2.4 Front End τεχνολογίες

JAVASCRIPT

Η JavaScript είναι μια αντικειμενοστραφείς γλώσσα προγραμματισμού που δεν χρειάζεται μεταγλώττισή. Ενώ είναι κυρίως γνωστή ως γλώσσα σεναρίου(scripting language) για ιστοσελίδες χρησιμοποιείται σε πολλά περιβάλλοντα χωρίς τη χρήση προγράμματος περιήγησης όπως στη node.js apache Couch db. Πρόκειται για μία πρωτότυπη πολλαπλών παραδειγμάτων γλώσσα προγραμματισμού που είναι δυναμική και υποστηρίζει αντικειμενοστραφή προγραμματισμό.

jquery

Η jquery είναι μια μικρή γρήγορη βιβλιοθήκη της JavaScript. Χρησιμοποιώντας τα εργαλεία που μας δίνει μπορούμε να επεξεργαστούμε με όλους τους δυνατούς τρόπους τα HTML αντικείμενα μιας ιστοσελίδας πολύ πιο απλά από ότι χρησιμοποιώντας καθαρή JavaScript. Στην ουσία στόχος της jquery είναι να κάνει την γραφή της JavaScript πιο εύκολη. Συγκεκριμένα η jquery αναλαμβάνει πολλές εργασίες που αν γίνουν με καθαρή JavaScript χρειάζονται πολλές γραμμές κώδικα και τίς ενθυλακώνει μέσα σε μεθόδους που έπειτα ο προγραμματιστής μπορεί να καλέσει σε μία γραμμή.

2.5 Design μοτίβα στο MVC

Repository Pattern

Σε πολλές εφαρμογές η βασική business λογική έχει πρόσβαση σε δεδομένα από την βάση δεδομένων από Web υπηρεσίες κτλ. Η άμεση πρόσβαση σε δεδομένα από το business layer μπορεί να έχει τα ακόλουθα αποτελέσματα

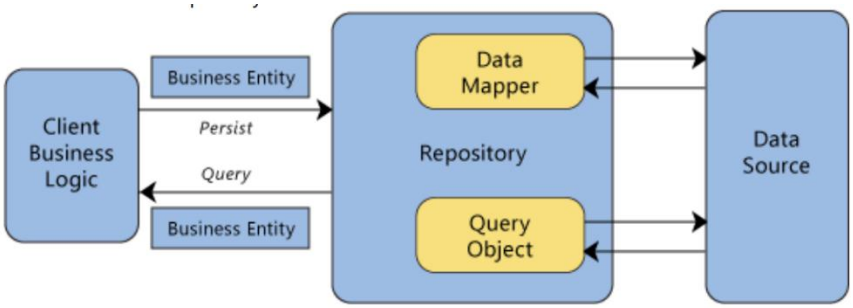
- Διπλότυπο κώδικα
- Μεγαλύτερο ρίσκο για προγραμματιστικά λάθη
- Δυσκολία στο τεστάρισμα της εφαρμογής λόγω αδυναμίας απομόνωσης της business λογικής από άλλες εξαρτήσεις

Χρησιμοποιούμε το repository pattern για να καταφέρουμε τα παρακάτω

- Τη μεγιστοποίηση της ποσότητας κώδικα που μπορεί να ελεγχθεί με αυτοματισμό και απομονωμένο από το λογισμικό επίπεδο των δεδομένων
- Για να μπορούμε να αποκτήσουμε τη πρόσβαση σε δεδομένα από πολλά σημεία και για να μπορέσουμε να εφαρμόσουμε κεντρική διαχείριση
- Για να επιτύχουμε καλύτερη αναγνωσιμότητα κώδικα και συντήρηση αυτού

Τρόπος Χρήσης

Χρησιμοποιούμε repository κλάσεις που διαχωρίζουν τη λογική από τα δεδομένα. Με αυτό το τρόπο το επίπεδο που συγκροτεί την business λογική δεν γνωρίζει τι είδους δεδομένα συνθέτουν το επίπεδο δεδομένων.



Εικόνα 2-5:5 Repository Pattern

Το repository υπάρχει μεταξύ το data source layer δηλαδή του επίπεδου των δεδομένων (βάση δεδομένων) και των business επιπέδων της εφαρμογής. Ζητάει δεδομένα από το data layer και τα μετατρέπει σε μια business οντότητα(αντικείμενο).Με άλλα λόγια τα repositories είναι γέφυρες μεταξύ πολλών υποσυστημάτων που μπορεί να βρίσκονται όλα κάτω από το ίδιο σύστημα.

Inversion of Control ή Dependency Injection Μοτίβα

Αυτά τα μοτίβα ασχολούνται με την απομάκρυνση εξαρτήσεων από τον πηγαίο κώδικα. Η βασική ιδέα πίσω από την υλοποίησή τους είναι ότι αντί να δένονται οι κλάσεις που συνθέτουν τις εφαρμογές μεταξύ τους δηλαδή να δημιουργούν τις εξαρτήσεις αυτές αλλάζει τη ροή και περνάνε αυτές οι εξαρτήσεις κατά την δημιουργία του αντικειμένου.

Πλεονεκτήματα:

- Δημιούργει πολύ καλές συνθήκες για έλεγχο πηγαίου κώδικά. Απομονώνει δηλαδή κάθε κομμάτι κώδικα και έτσι μπορεί να γίνει πολύ πιο εύκολα ελεγχόμενο
- Αυτά τα μοτίβα γίνονται υλοποιήσιμα είτε χρησιμοποιώντας κάποια ελεύθερη βιβλιοθήκη είτε δημιουργώντας μια. .

2.5 Third Party Βιβλιοθήκες

Auto Mapper

Ο AutoMapper είναι μια μικρή βιβλιοθήκη η οποία χρησιμοποιήθηκε στη server side κομμάτι της εφαρμογής. Σκοπός του είναι να λύσει το πρόβλημα της συχνά προβληματικής διαδικασίας της χαρτογράφησης (mapping) ενός τύπου αντικειμένου C# γλώσσας σε ένα άλλο. Αυτή η διαδικασία είναι πολύ συχνά εκτεθειμένη σε πολλά λάθη. Ο AutoMapper προσφέρει μια ασφαλή αυτοματοποιημένη διαδικασία που μας προφυλάσσει από τέτοιους κινδύνους.

AutoFac

Το AutoFac είναι μία βιβλιοθήκη η οποία σχεδιάστηκε για να υλοποιεί το dependency injection αρχιτεκτονικό μοτίβο. Είναι από τα πολυχρησιμοποιημένα containers στην αγορά λόγω της ευκολίας χρήσης ,της πληθώρας από documentation αλλά και λειτουργειών

jquery Data tables plugin

Η συγκεκριμένη βιβλιοθήκη χρησιμοποιήθηκε σε όλη την εφαρμογή για την παρουσίαση των δεδομένων σε πίνακες που παρέχουν φίλτρα , ταξινόμηση και αναζήτηση .

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] http://www.microsoft.com/net

[2] http://www.entityframeworktutorial.net/what-is-entityframework.aspx

[3] https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx

[4] http://javascript.com

[5] https://www.w3schools.com/jquery/jquery_get_started.asp

[6] https://msdn.microsoft.com/en-us/library/ff649690.aspx

3.ΕΠΙΛΟΓΗ ΤΕΧΝΟΛΟΓΙΩΝ

3.1 ΕΠΙΛΟΓΗ MVC PATTERN

3.1.1 Εναλλακτικές

1)SINGLE PAGE APPLICATIONS(SPA)

Τα Spas είναι Web εφαρμογές που φορτώνουν μόνο μια HTML σελίδα στον browser και ανανεώνουν το interface σε κάθε αλληλεπίδραση του χρήστη με την εφαρμογή χρησιμοποιώντας javascript . Στην ουσία για κάθε request ξαναγράφεται το περιεχόμενο της σελίδας αντί να φορτώνεται καινούργια από τον server. Μέσω αυτής της προσέγγισης βελτιώνεται σε μεγάλο βαθμό η εμπειρία του χρήστη χρησιμοποιώντας την εφαρμογή δίνοντας της αίσθηση ότι πρόκειται για desktop εφαρμογή. Στα Spas είτε όλος ο απαραίτητος κώδικας(HTML,CSS,JAVASCRIPT) φορτώνεται στην αρχή είτε φορτώνεται δυναμικά ανάλογα με τις αλληλεπιδράσεις του χρήστη με την εφαρμογή. Στην πλειοψηφία των περιπτώσεων τα Spas χρησιμοποιούνται σε mobile applications και γενικότερα εφαρμογές που απαιτούν πλουσιότερο και πολυπλοκότερο User Interface.

2)WEB FORMS

Τα ASP.NET Web Forms είναι ένα μέρος του ASP.NET framework .Είναι στην ουσία σελίδες που συνδυάζουν HTML κώδικα μαζί με client side κώδικα για παράδειγμα JavaScript αλλά και server side κώδικα. Όταν ο χρήστης ζητήσει μια σελίδα αυτή μεταγλωττίζεται και εκτελείται στον server από το framework και επιστρέφει μια HTML σελίδα. Στις Web Forms εφαρμογές ο προγραμματιστής χειρίζεται τις αλληλεπιδράσεις του χρήστη μέσω των events που ενεργοποιούνται.(Event Driven Development).Οι Web forms εφαρμογές επίσης χαρακτηρίζονται για την σχετικά γρήγορη ανάπτυξη τους καθώς οι βιβλιοθήκες προσφέρουν πολλά server controls που κρύβουν την πολυπλοκότητα.

3.1.2 Πλεονεκτήματα MVC

1-MVC vs SPA

- Στην MVC αρχιτεκτονική δεν χρειάζεται να φορτωθεί κάποιο βαρύ UI framework στον browser.
- Στα Spas ο client side κώδικας δεν είναι μεταγλωττισμένος συνεπώς είναι πολύ δύσκολο να εντοπιστούν και να διορθωθούν τα σφάλματα.
- Στην MVC αρχιτεκτονική η ασφάλεια στον χειρισμό των αιτημάτων είναι σαφώς καλύτερη μιας και τα request τα διαχειρίζεται από τον server.

2-MVC vs Web Forms

- Η MVC αρχιτεκτονική επιβάλει τον διαχωρισμό των διαφορετικών τμημάτων της εφαρμογής μας(Separation of Concerns) σε αντίθεση με τα Web Forms με αποτέλεσμα να προωθεί μία πιο καθαρή αρχιτεκτονική.
- Σαν ένα ακόμα αποτέλεσμα αυτού του διαχωρισμού είναι πιο εύκολη η διαδικασία ελέγχου και testing της εφαρμογής σε αντίθεση με τα Web forms που περιλαμβάνουν στο ίδιο αρχείου server side αλλά και client side κώδικα
- Επίσης υπάρχει πλήρης έλεγχος πάνω στην HTML. Στην MVC αρχιτεκτονική λόγω ότι δεν υπάρχουν server controls χρησιμοποιούνται μόνο HTML input elements συνεπώς μπορούμε να είμαστε βέβαιοι για το τελικό HTML που θα παραχθεί καθώς και τα ids που θα έχουν τα elements.

3.1.3 Ανάγκες Εφαρμογής

Κατά την διάρκεια του αρχικού σχεδιασμού της εφαρμογής έγινε μία καταμέτρηση των αναγκών που χρειάζεται σε επίπεδο υλοποίησης. Σε αυτό το στάδιο επίσης έγινε και η επιλογή της αρχιτεκτονικής που θα βασιζόταν η εφαρμογή. Η επιλογή αυτή σε κάθε έργο είναι απόρροια μιας κατανόησης των αναγκών που θα προκύψουν σκεπτόμενοι όλες τις εναλλακτικές αρχιτεκτονικές που αρμόζουν σε αυτό το έργο. Σύμφωνα και με τις παραπάνω πληροφορίες οι πιο βασικές εναλλακτικές αρχιτεκτονικές που ταιριάζουνε σε αυτή την εφαρμογή είναι:

Web Forms

Χρησιμοποιώντας Web Forms η ανάπτυξη της εφαρμογής θα ολοκληρώνονταν σε μικρότερο χρονικό διάστημα. Η διευκόλυνση του API που προσφέρεται μας δίνει την δυνατότητα να περιορίσουμε την ανάπτυξη σε πολύ λιγότερα αρχεία από ότι στην MVC αρχιτεκτονική. Η διαδικασία δημιουργίας καινούργιων σελίδων είναι σχετικά γρήγορη εφόσον συνδυάζουμε server side κώδικα και client side κώδικα στο ίδιο αρχείο. Σε αντιπαράθεση με αυτό το επιχείρημα έρχεται η ανάγκη της συγκεκριμένης εφαρμογής όπου απαιτεί περισσότερα επίπεδα. Πιο συγκεκριμένα μια εφαρμογή όπου μπορεί να έχει αρκετή πολυπλοκότητα στο user interface κάποιας σελίδας δεν εφαρμόζει κατάλληλα με αρχιτεκτονικές και apis όπως τα Web forms γιατί ακόμα και αν επιτευχθεί η ζητούμενη λειτουργικότητα ο κώδικας που παράγεται δεν είναι συντηρήσιμος ούτε δίνει την δυνατότητα του testing.Επιπροσθέτως η διευκόλυνση που δίνεται με τον διαχωρισμό των κομματιών της εφαρμογής από το MVC κάνει ξεκάθαρη την σωστή θέση του κάθε πηγαίου κώδικα. Χρησιμοποιώντας επίσης repository pattern μπορούμε να το συνδέσουμε καλύτερα με μια MVC αρχιτεκτονική οπού τα end points της εφαρμογής δηλαδή οι controllers θα είναι τα σημεία όπου θα μιλάνε με τα προηγούμενα επίπεδα(services,repositories) σε αντίθεση με τα web forms όπου η σύνδεση αυτή δεν είναι ούτε ξεκάθαρη ούτε εύκολα υλοποιήσιμη.

SPA

Σε αντίθεση με τα Web Forms η ενναλακτική των single page application είναι αρκετά καλύτερη. Όλη η υποδομή των κατώτερων επιπέδων της εφαρμογής μπορεί εύκολα να συνδυαστεί με κάποιο Javascript framework που συχνά χρησιμοποιείται σήμερα για την δημιουργία μιας SPA εφαρμογής. Επίσης χρησιμοποιώντας end points όπως Web Api controllers μπορούμε πολύ εύκολα να προσεγγίσουμε τα data που μεταφέρονται από και προς της βάση δεδομένων. Χρησιμοποιώντας την ενναλακτική των SPA σε εφαρμογές με πολλές διαφορετικές σελίδες και λειτουργίες όμως δεν

συνιστάται. Η εφαρμογή αυτή απαιτεί αρκετές διαφορετικές σελίδες όπου σε ένα υποσύνολο αυτών εφαρμόζονται διαφορετικά δικαιώματα οπότε πρέπει να υπάρξει κλήση στο server για τη διαχείριση αυτών. Επίσης λόγω του μεγάλου όγκου πόρων που χρειάζεται να κατεβάσει ο browser στον υπολογιστή του χρήστη αναπτύσσοντας την εφαρμογή ως single page application έχουμε ως αποτέλεσμα χαμηλή απόδοση στην αρχική πλοήγηση στο web site.

User Interface

Στην επιλογή των τεχνολογιών για το front end της εφαρμογής καθοριστικό ρόλο έπαιξε η πολυπλοκότητα που υπολογίστηκε ότι θα έχει κάθε view,σελίδα της εφαρμογής. Ο συνδυασμός της Javascript και της βιβλιοθήκης της jQuery είναι αρκετός για εφαρμογές που απαιτούν απλό χειρισμό του DOM(Document Object Model),animations,αυτόματη ανανέωση περιεχομένων όπως και αυτή η εφαρμογή. Η χρησιμοποίηση κάποιου javascript framework δικαιολογείται πολλές φορές σε περιπτώσεις που κάθε σελίδα χρειάζεται κάποια ανώτερη business λογική, διαχειρίζεται συνδεδεμένα περιεχόμενα σελίδων οπού σε αυτές τις περιπτώσεις αυτοματοποιεί και απλουστεύει την ανάπτυξη.

Real time Framework

Στην επιλογή της βιβλιοθήκης που θα καταστήσει δυνατή την real time λειτουργικότητα ,οι ενναλακτικές ήταν:

- SignalR
- Fleck
- SuperWebSocket

Εκτός από το SignalR οι υπόλοιπές τεχνολογίες είναι third-party βιβλιοθήκες υλοποιημένες σε C# και στηριζόμενες στο Web Socket πρωτόκολλο. Το SignalR αποτελεί κομμάτι του .NET framework και είναι στηριζόμενο από τη Microsoft. Πέραν αυτού του επιχειρήματος το SignalR παρέχει το ευκολότερο στη χρήση API και από τις 3 επιλογές έχοντας πάντα τη δυνατότητα για low level διαχείριση της real time λειτουργικότητας μέσω του Persistent Connection API.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] https://www.asp.net/web-forms

[2] https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

[3] https://deepstreamhub.com/blog/realtime-framework-overview/

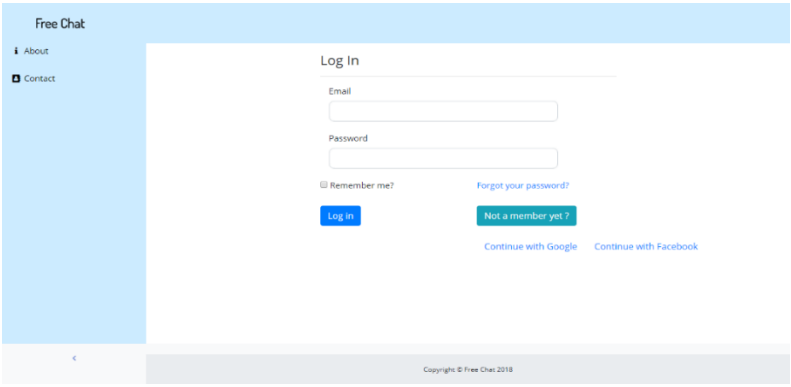
4.ΓΕΝΙΚΗ ΠΕΡΙΓΡΑΦΗ ΕΦΑΡΜΟΓΗΣ

4.1 Περιγραφή του User interface κομμάτι της εφαρμογής

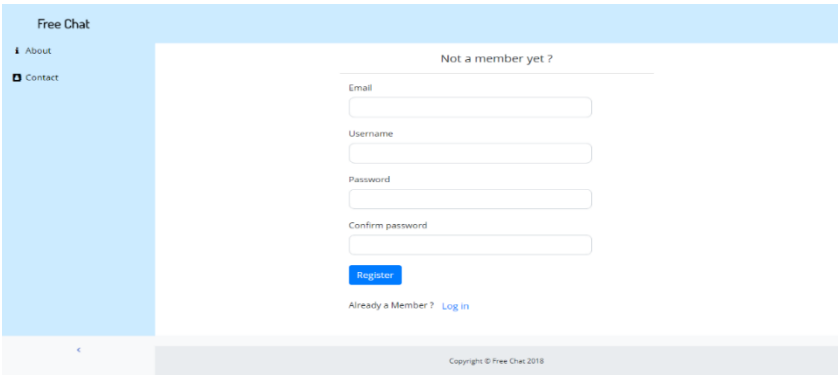
4.1.1 Login και Register σελίδα

Στις παρακάτω εικόνες απεικονίζονται η σελίδα σύνδεσης(Σχήμα 3.1) και η σελίδα εγγραφής(Σχήμα 4.2) με την εφαρμογή. .

Πέρα από τις βασικές λειτουργίες σύνδεσης και εγγραφής δίνεται η δυνατότητα στο χρήστη να συνδεθεί στην εφαρμογή είτε χρησιμοποιώντας κάποιον Google λογαριασμό είτε το Facebook λογαριασμό του.



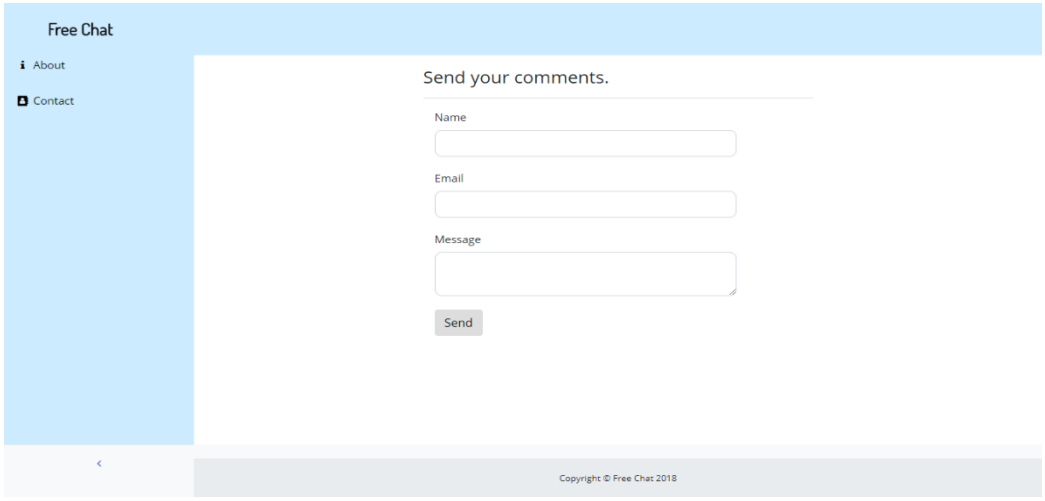
Εικόνα 4-6



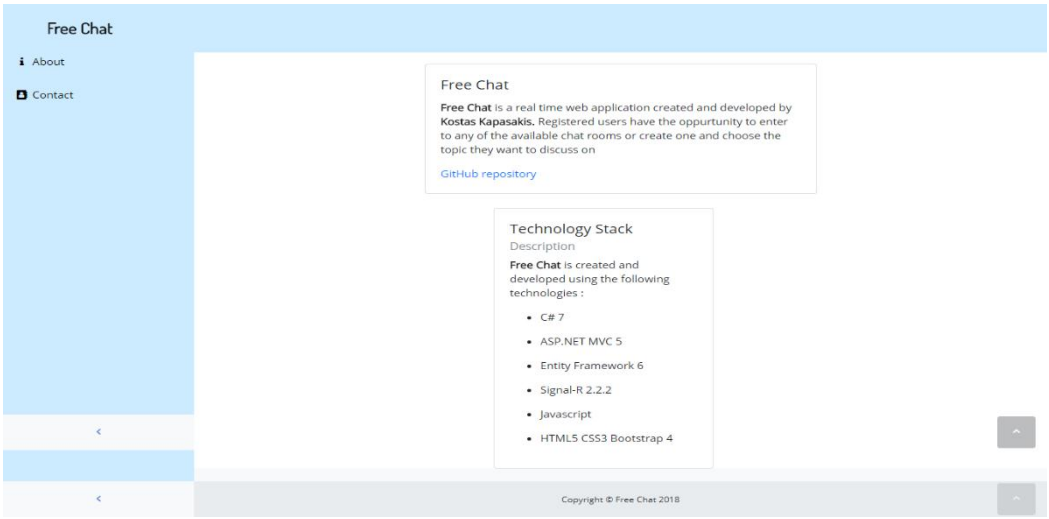
Εικόνα 4-7

4.1.2 About και Contact σελίδα

Στις παρακάτω εικόνες απεικονίζονται η σελίδα επικοινωνίας των χρηστών με την πλατφόρμα (Σχήμα 3.3) στην οποία οι χρήστες έχουν την δυνατότητα να στείλουνε email στον διαχειριστή της εφαρμογής είτε είναι εγγεγραμμένοι χρήστες είτε όχι. Επίσης απεικονίζεται η σελίδα About της εφαρμογής όπου δίδονται κάποιες πληροφορίες όπως τι είδους εφαρμογή είναι και ποιες τεχνολογίες χρησιμοποιήθηκαν.



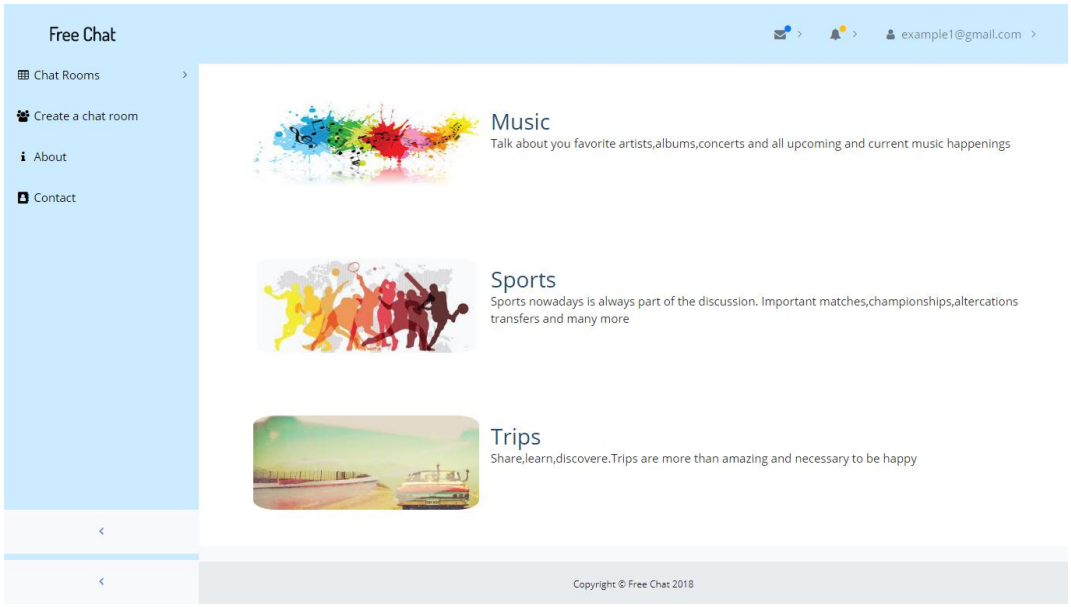
Εικόνα 4-8



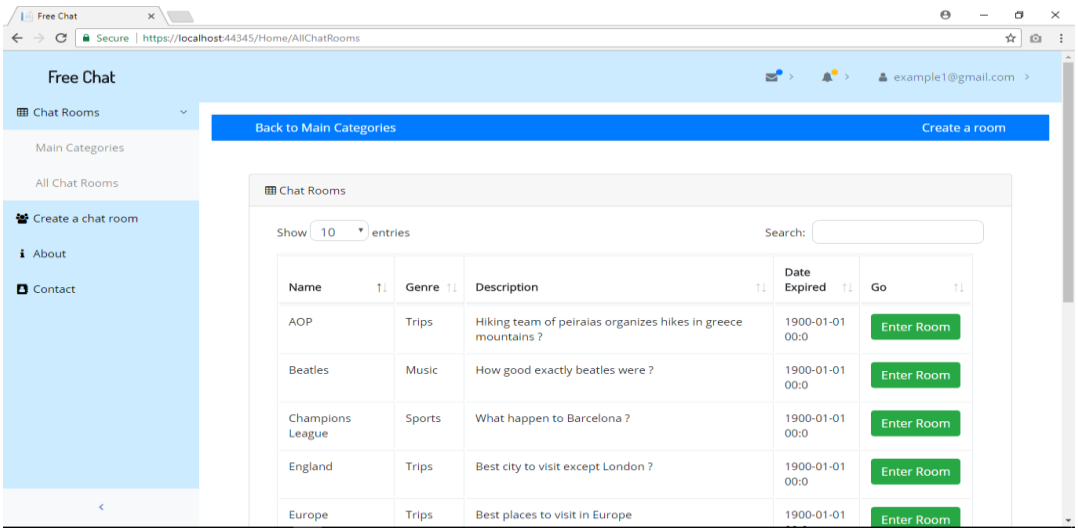
Εικόνα 4-9

4.1.3 Βασικές σελίδες και λειτουργίες

Στις παρακάτω εικόνες απεικονίζονται η πρώτη σελίδα που βλέπει ο χρήστης αφού εγγραφτεί στην εφαρμογή ή συνδεθεί(Σχήμα 3.5). Στην σελίδα αυτή βλέπει τις βασικές κατηγορίες θεμάτων στις οποίες χωρίζονται τα διαθέσιμα chat rooms αλλά και όποια φτιαχτούν μελλοντικά. Εφόσον ο χρήστης επιλέξει κάποια κατηγορία μπορεί να δει τα δωμάτια κάθε κατηγορίας και να εισέλθει από εκεί στο chat room. Επίσης στο σχήμα 3.6 ο χρήστης αφού έχει επιλέξει το Chat Rooms εικονίδιο στο αριστερό μενού εμφανίζονται δύο επιλογές όπου η $1^{η}$ είναι το σχήμα που μόλις προανέφερα και η $2^{η}$ επιλογή είναι η εμφάνιση όλων την διαθέσιμων-ενεργών chat rooms.



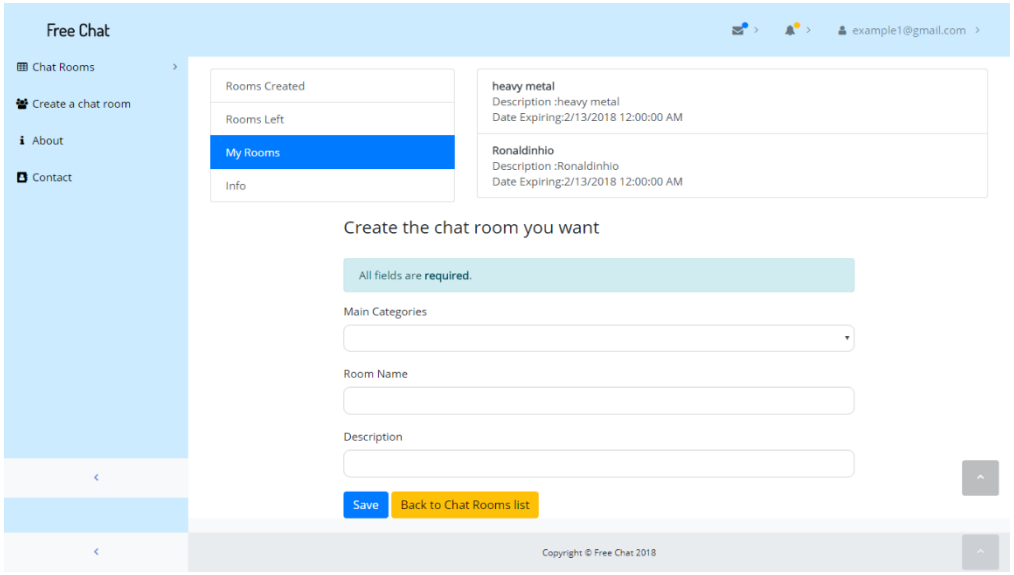
Εικόνα 4-10



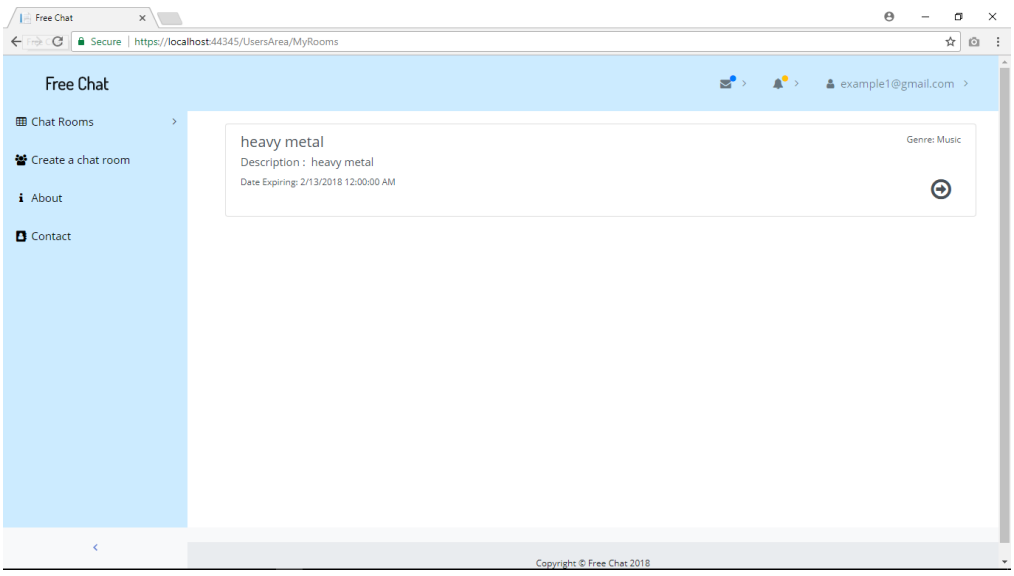
Εικόνα 4-11

Στο παρακάτω σχήμα (Σχήμα 3.7) φαίνεται μία από τις πιο σημαντικές παρεχόμενες λειτουργίες κάθε εγγεγραμμένου χρήστη, η δημιουργία chat room μέσα από μια διαδικασία συμπλήρωσης φόρμας. Ειδικότερα ο χρήστης επιλέγει στο 1^{ο} πεδίο την βασική κατηγορία θέματος που θα ανήκει το chat room έπειτα διαλέγει ένα όνομα και μία περιγραφή στο τέλος. Επιλέγοντας το save κουμπί ανακατευθύνεται στη σελίδα (Σχήμα 3.8) με τα δικά του δωμάτια συζητήσεων και από εκεί μπορεί να εισέλθει στο επιθυμητό δωμάτιο. Επιπλέον κάθε χρήστης έχει τη δυνατότητα δημιουργίας 10 δωματίων. Στη σελίδα δημιουργίας των δωματίων (Σχήμα 3.7) υπάρχει όπως βλέπουμε και παρακάτω ένα χώρος πληροφοριών οπού εκεί ο χρήστης μπορεί δει όλα τα δημιουργημένα

δωμάτιά του ,πόσα είναι, πόσα του μένουν και ένα σετ βημάτων για να βοηθηθεί κατά την δημιουργία.



Εικόνα 4-12

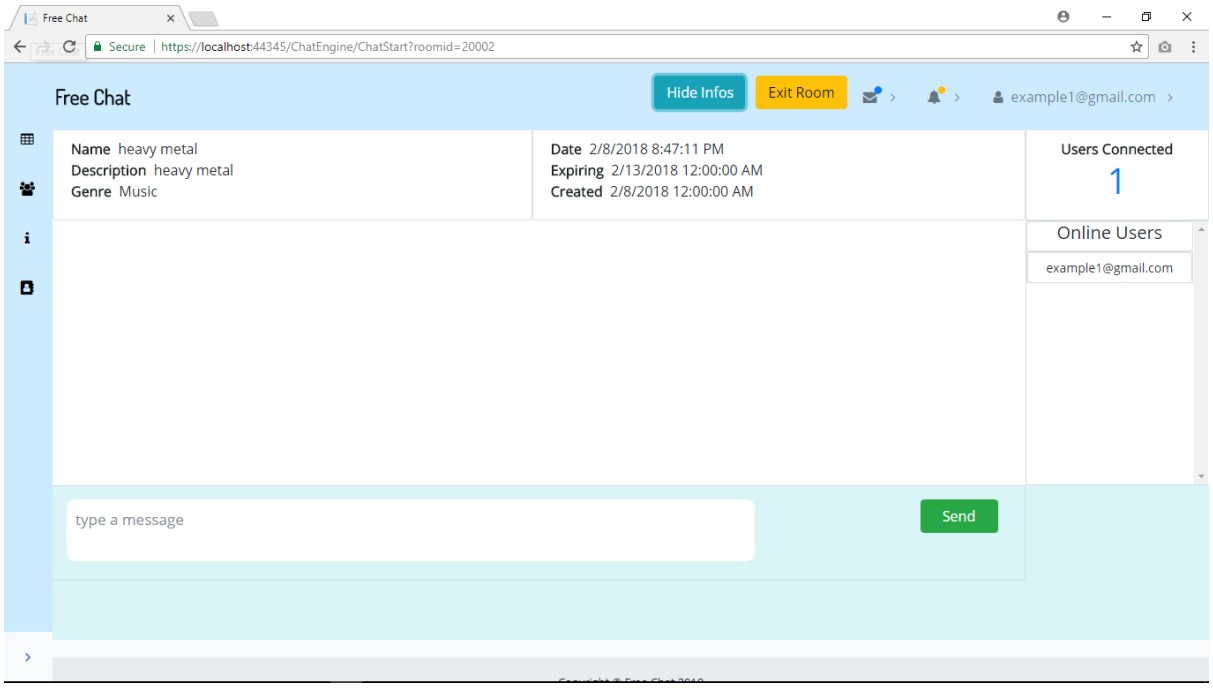


Εικόνα 4-13

4.1.4 Σελίδα Chat Room

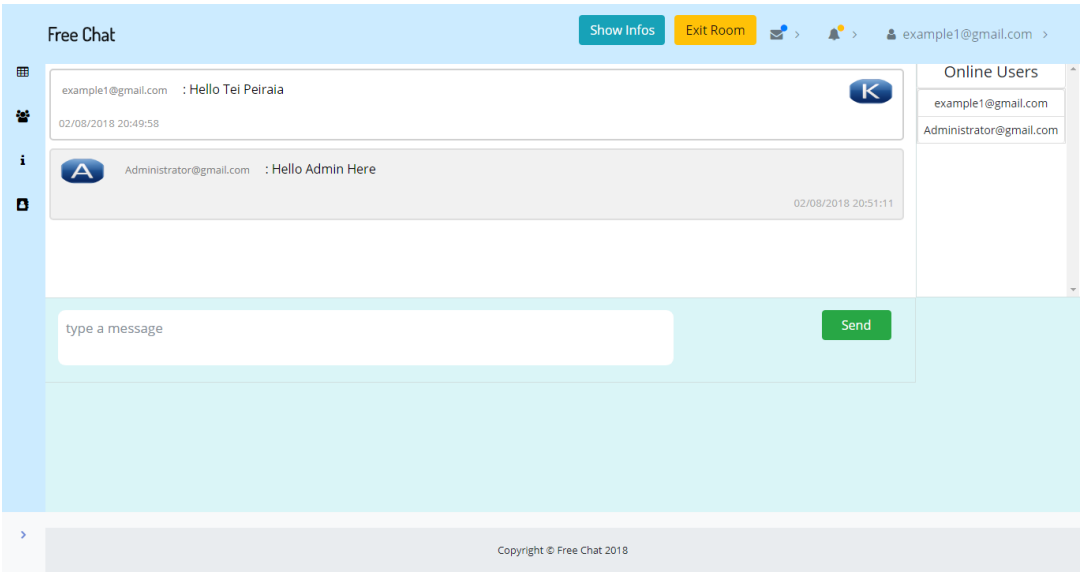
Παρακάτω στο σχήμα 3.9 βλέπουμε το πιο ουσιαστικό κομμάτι της εφαρμογής το Chat Room η αλλιώς την σελίδα συζητήσεων. Ο χρήστης έχοντας επιλέξει ένα δωμάτιο ανακατευθύνεται σε αυτήν την σελίδα και παρατηρεί

- Το βασικό αριστερό μενού έχει συρρικνωθεί ώστε να δίνεται μεγαλύτερος χώρος στην οθόνη
- Στην κορυφή της σελίδας έχουν προστεθεί κουμπιά για την απομάκρυνση από το δωμάτιο και επιστροφή στη σελίδα όλων των διαθέσιμων δωμάτιων
- Πάνελ με πληροφορίες για το συγκεκριμένο δωμάτιο όπως όνομα, περιγραφή, κατηγορία, ημερομηνία δημιουργίας, ημερομηνία λήξης του δωματίου και αριθμός χρηστών που είναι συνδεδεμένοι με αυτό το δωμάτιο.



Εικόνα 4-14

Στο σχήμα 4.10 φαίνεται η όψη της σελίδας ενώ έχει κρυφτεί το πάνελ πληροφοριών και έχοντας σταλθεί μηνύματα από τους συνδεδεμένους χρήστες.



Εικόνα 4-15

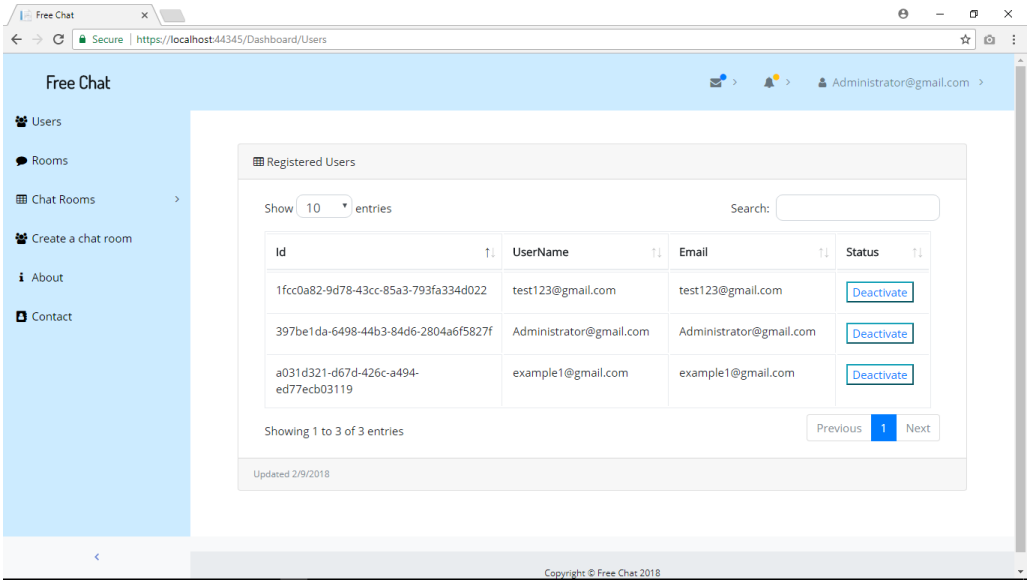
4.1.5 Ρόλος Διαχειριστή

Η εφαρμογή προσφέρει επίσης δύο τύπους χρηστών:

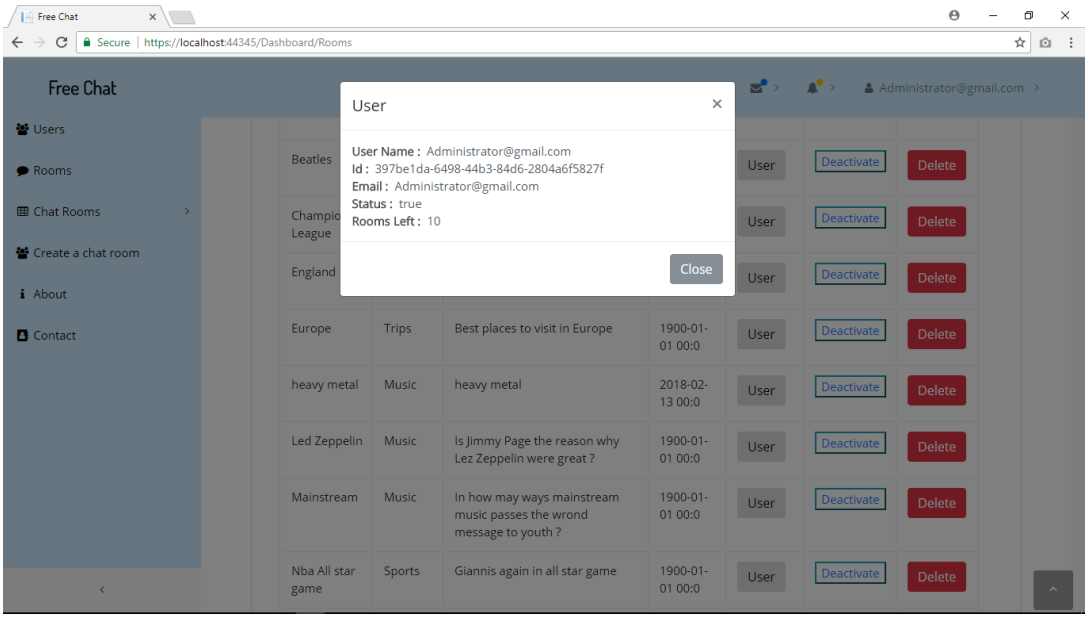
- Διαχειριστής
- Εγγεγραμμένοι χρήστες

Ο χρήστης που είναι διαχειριστής έχει τη δυνατότητα να ελέγχει την εφαρμογή βλέποντας

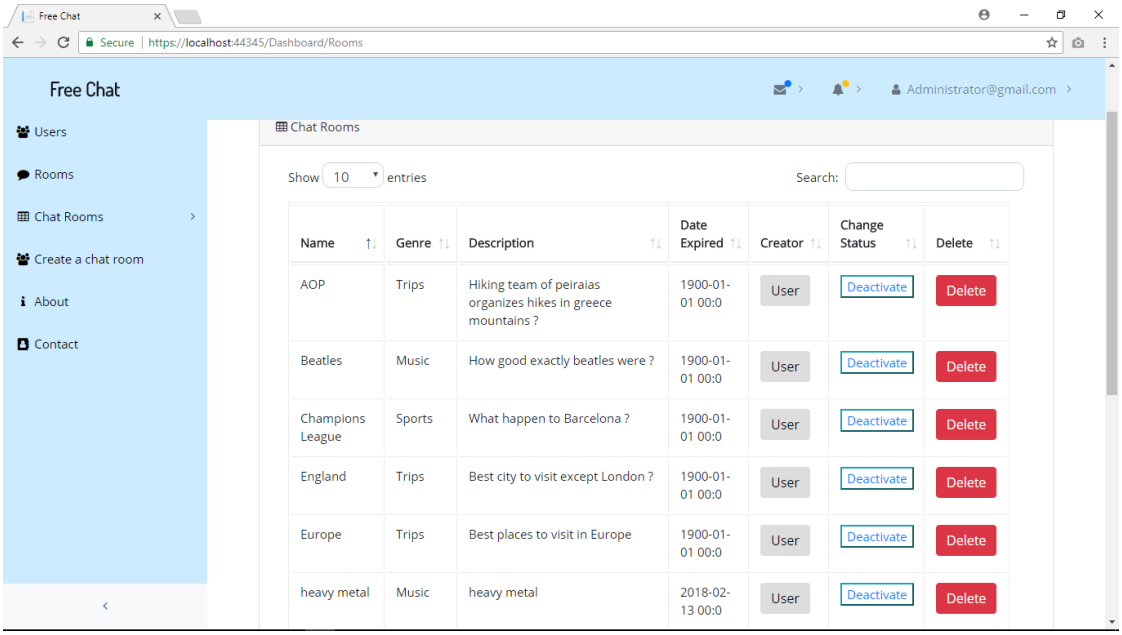
- Τους εγγεγραμμένους χρήστες(Σχήμα 4.11) και ενεργοποιώντας η απενεργοποιώντας τους λογαριασμούς τους. Επίσης επιλέγοντας το user κουμπί (Σχήμα 4.12) μπορεί να δει επιπλέον πληροφορίες για τον χρήστη όπως πόσα δωμάτια έχει φτιάξει ,πόσα του μένουν κτλ.
- Τα δημιουργημένα δωμάτια(Σχήμα 4.13) , ενεργοποιώντας η απενεργοποιώντας τα ή και διαγράφοντας τα.



Εικόνα 4-16



Εικόνα 4-17


Free Chat
Secure | https://localhost:44345/Dashboard/Rooms
Free Chat
Administrator@gmail.com
Users
Rooms
Chat Rooms
Create a chat room
About
Contact
Chat Rooms
Show 10 entries
Search:
Name
Genre
Description
Date Expired
Creator
Change Status
Delete
AOP
Trips
Hiking team of peiraias organizes hikes in greece mountains ?
1900-01-01 00:0
User
Deactivate
Delete
Beatles
Music
How good exactly beatles were ?
1900-01-01 00:0
User
Deactivate
Delete
Champions League
Sports
What happen to Barcelona ?
1900-01-01 00:0
User
Deactivate
Delete
England
Trips
Best city to visit except London ?
1900-01-01 00:0
User
Deactivate
Delete
Europe
Trips
Best places to visit in Europe
1900-01-01 00:0
User
Deactivate
Delete
heavy metal
Music
heavy metal
2018-02-13 00:0
User
Deactivate
Delete

Εικόνα 4-18

5.ΟΡΓΑΝΩΣΗ ΒΑΣΗΣ ΚΑΙ BACK END

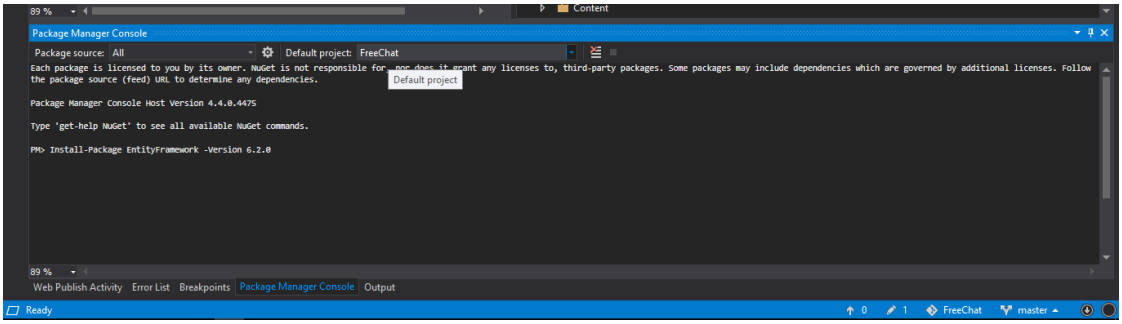
5.1 Entity Framework και Code First

Στην συγκεκριμένη πτυχιακή χρησιμοποιήθηκε το Entity Framework για την οργάνωση υλοποίηση και πρόσβαση σε δεδομένα της βάσης δεδομένων .Επίσης ακολουθήθηκε η μεθοδολογία της Code First προσέγγισης όπου δεν χρειάστηκε να δημιουργηθεί μια βάση δεδομένων πάνω στην οποία θα δεθεί η εφαρμογή ,αντιθέτως η βάση δημιουργήθηκε με τη βοήθεια της παραπάνω προσέγγισης από τον κώδικα.

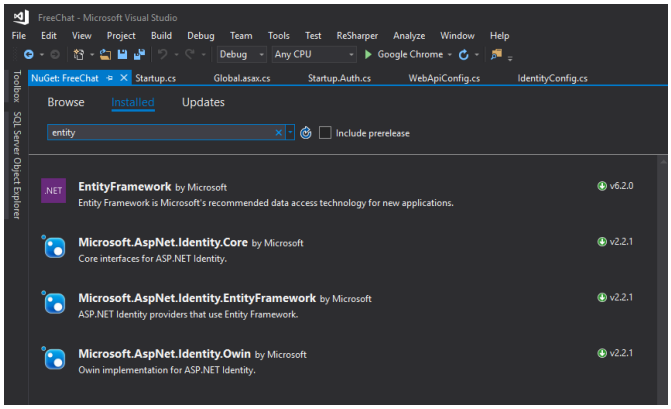
5.1.1 Code First Μεθοδολογία

1)Εισαγωγή Βιβλιοθηκών Entity Framework

Προσθέτουμε στην εφαρμογή μας τις βιβλιοθήκες της έκδοσης του Entity Framework που θέλουμε είτε μέσω του Package Manager Console (Σχήμα 5.1) είτε μέσω του NuGet manager (Σχήμα 5.2)



Εικόνα 5-19:View->Other Windows->Package Manager Console



Εικόνα 5-20:Nuget Manager

2)Ενεργοποίηση των Code First Migrations

Όπως έχει ήδη προαναφερθεί χρησιμοποιώντας την code first τεχνική κάθε φορά που αλλάζουμε τις βασικές κλάσεις μας που αποτελούν την εφαρμογή μας αλλάζει και το Schema η αλλιώς η δομή της βάσης μας. Για να ενεργοποιήσουμε τα code first migrations πληκτρολογούμε στο Package Manager Console παράθυρο του Visual Studio την εντολή Enable-Migrations

Επίσης με την εκτέλεση αυτής της εντολής δημιουργείτε ο φάκελος migrations που θα περιέχει όλες τις διαφορετικές καταστάσεις από τις οποίες θα περάσει η βάση μας.

3)Δημιουργία του 1ο Migration.

Η εφαρμογή χρησιμοποιεί την προκαθορισμένη βιβλιοθήκη ττης Microsoft για την πιστοποίηση χρηστών και άλλων διάφορων λειτουργιών σχετικά με τους χρήστες την ASP.NET identity.Με αυτήν την βιβλιοθήκη υπάρχουν έτοιμοι controllers,views αλλά και models που υλοποιούν λειτουργίες σχετικά με τους χρήστες. Στην δημιουργία του 1^{ο} Migration αν και δεν έχουν οριστεί άλλες κλάσεις δημιουργούνται πίνακες στη βάση δεδομένων όπως :

- AspNetRoles
- AspNetUserClaims
- AspNetUserLogins
- AspNetUserRoles
- AspNetUsers

Η εντολή που πληκτρολογούμε στο package manager console για την πρόσθεση του Migration είναι: add-migration ExampleMigrationName

Το αποτέλεσμα της εντολής είναι η δημιουργία μιας csharp κλάσης που ορίζει, δημιουργεί τους παραπάνω πίνακες

Παρακάτω δίνεται ένα μέρος αυτής της κλάσης

Παρατηρούμε ότι κάθε migration αποτελείται από μια Up μέθοδο και μια Down. Στην ουσία αυτές οι δύο μέθοδοι περιέχουν ακριβώς τις αντίθετες εντολές. Στην Up μέθοδο παρατηρούμε ότι δημιουργούνται οι sql πίνακες που δηλώνονται αυτόματα όπως προαναφέραμε μέσω του Identity ενώ στην down διαγράφονται.

4)Ενημέρωση της Βάσης

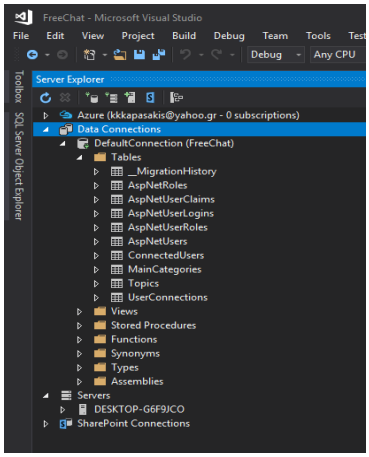
Το επόμενο βήμα μετά από το κάθε migration είναι η ενημέρωση της βάσης και η πραγματική εκτέλεση των εντολών που δημιουργήθηκαν από το migration.Η εντολή έχει ως εξής: Update-database

Με την εκτέλεση αυτής της εντολής δημιουργείτε ένα αρχείο .mdf κάτω από τον φάκελο app data που στην ουσία αποτελεί την βάση .(Σχήμα 5.3). Όπως παρατηρούμε δημιουργείτε και ένας φάκελος **_MigrationsHistory** ο οποίος περιέχει το migration που έτρεξε τελευταίο δηλαδή την παρούσα κατάσταση της βάσης κάθε στιγμή.

Για να δούμε το συγκεκριμένο παράθυρο επιλέγουμε View->Server Explorer

```
namespace FreeChat.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class InitialMigrationCreationOfMainModels : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "dbo.AspNetRoles",
                c => new
                    {
                        Id = c.String(nullable: false, maxLength: 128),
                        Name = c.String(nullable: false, maxLength: 256),
                    })
                .PrimaryKey(t => t.Id)
                .Index(t => t.Name, unique: true, name: "RoleNameIndex");

            CreateTable(
                "dbo.AspNetUserRoles",
                c => new
                    {
                        UserId = c.String(nullable: false, maxLength: 128),
                        RoleId = c.String(nullable: false, maxLength: 128),
                    })
```

Μέρος του Migration 1



Εικόνα 5-21

5.1.2 Domain Models Εφαρμογής.

MainCategories.cs

```
public byte Id { get; set; }

[Required]
[StringLength(255)]
public string Name { get; set; }

public bool Active { get; set; }

public string CategoryImage { get; set; }

public string CategoryDescription { get; set; }
```

Η συγκεκριμένη κλάση αντιπροσωπεύει τον πίνακα που θα κρατάει τις βασικές κατηγορίες θεμάτων που προσφέρει η εφαρμογή. Βάση αυτών έπειτα τα δωμάτια συζητήσεων θα κατηγοριοποιούνται.

```
        public long Id { get; set; }

        [Required]
        [StringLength(50)]
        [DisplayName("Room Name")]
        public string Name { get; set; }
        public string Genre { get; set; }
        [StringLength(250)]
        [Required]
        public string Description { get; set; }
        public bool Active { get; set; }
        public DateTime DateCreated { get; set; }
        public DateTime DateExpired { get; set; }
        public long MaxClientsOnline { get; set; }

        public ApplicationUser UserCreator { get; set; }
        public string UserCreatorId { get; set; }
```

Topics.cs 1

Η συγκεκριμένη κλάση αντιπροσωπεύει το πίνακα των δωματίων .Όπως φαίνεται κάθε δωμάτιο έχει ένα όνομα, ένα είδος ,μία περιγραφή ,ένα flag για το αν είναι ενεργό η όχι, ημερομηνία δημιουργίας, ημερομηνία λήξης, πόσους χρήστες μπορεί να φιλοξενήσει.

Επίσης παρατηρούμε μεταβλητές που υπακούουν σε συμβάσεις που δίνει το entity framework για να υποστηρίξει συσχετίσεις μεταξύ πινάκων.

Το property UserCreator και UserCreatorId υλοποιούν το foreign key constraint όπου εφόσον δημιουργηθεί το migration θα δημιουργηθεί στήλη στο συγκεκριμένο πίνακα ξένου κλειδιού με τον πίνακα AspNetUsers του Identity.Επιπλέον το ίδιο θα γίνει και για το property MainCategory και MainCategoryId όπου θα δημιουργηθεί περιορισμός ξένου κλειδιού με τον πίνακα MainCategories.

Data annotations

Στην δήλωση των ιδιοτήτων των παραπάνω κλάσεων παρατηρούμε συγκεκριμένες δεσμευμένες λέξεις όπως [REQUIRED],[StringLength(250).

Η συγκεκριμένη τεχνική ονομάζετε data annotations και δίνει έλεγχο εγκυρότητας στα δεδομένα που πάμε να περάσουμε στα συγκεκριμένα αντικείμενα αλλά και στα δεδομένα που πάνε να αποθηκευτούν στους αντίστοιχους πίνακες στη βάση. Συγκεκριμένα το Required στην ουσία θα μεταφραστεί ως NOT NULL στην βάση και το StringLength(250) ως ένα varchar με μέγιστο αριθμό χαρακτήρων ίσο με 250.

Δημιουργία πινάκων από τα Domain Models

Το Entity Framework παρέχει ένα set από κλάσεις που αντιπροσωπεύουν τη σύνδεση μας με τη βάση και τους πίνακες της.

DbContext: Αυτή η κλάση παρέχει όλες τις μεθόδους που μας δίνουν την δυνατότητα να συνδεόμαστε με την βάση.

DbSet: Αυτή η κλάση στην ουσία αντιπροσωπεύει τους πίνακες μας.

Παρακάτω φαίνεται η κλάση ApplicationDbContext η οποία υλοποιεί τα παραπάνω.

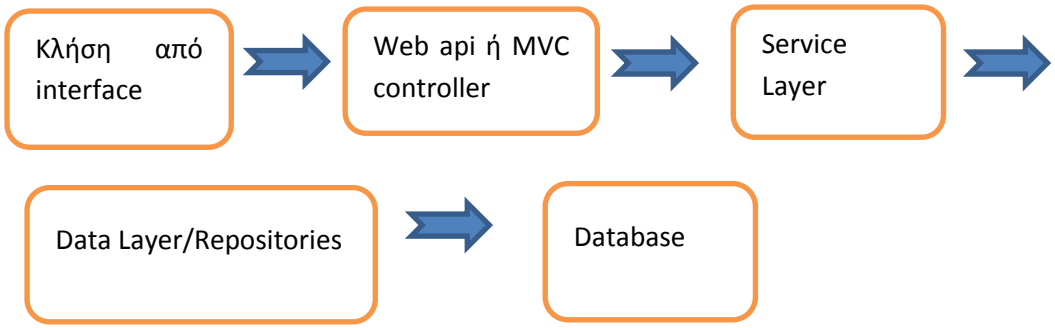
ApplicationDbContext.cs 1

```
public class ApplicationDbContext: IdentityDbContext<ApplicationUser>
    {
        public DbSet<Topics> Topics { get; set; }
        public DbSet<MainCategories> MainCategories { get; set; }

        public ApplicationDbContext()
            : base("DefaultConnection", throwIfV1Schema: false)
        {
        }

        public static ApplicationDbContext Create()
        {
            return new ApplicationDbContext();
        }
    }
```

Στην δήλωση των DbSet δίνουμε σαν παράμετρο τα domain models που φτιάξαμε παραπάνω. Έπειτα μπορούμε να προσθέσουμε το νέο migration ακολουθώντας τα παραπάνω βήματα τα οποία θα δημιουργήσουν τους καινούργιους πίνακες.

5.2 Repository pattern

Ροή κλήσεων από το interface μέχρι το Data Layer.



5.2.1 Data Access Layer

Repositories

Τα Repositories αντιπροσωπεύουν το Data Access Layer.Στην ουσία είναι κλάσεις με ονοματολογία που στο τέλος κάθε κλάσεις υπάρχει η λέξη repository.Τα repositories είναι υπεύθυνα για CRUD λειτουργίες. Δηλαδή μέλη αυτών των κλάσεων είναι μέθοδοι που δεν περιέχουν κάποια λογική πέραν από το διάβασμα ,ενημέρωση, διαγραφή δεδομένων. Μέσα σε αυτές τις κλάσεις δημιουργείται ένα DbContext που είναι το αντικείμενο που μας επιτρέπει να μιλάμε με την βάση και μέσω αυτού μπορούμε να υλοποιήσουμε αυτές τις λειτουργίες. Τα κάθε repository πολύ συχνά αντιστοιχεί σε κάθε οντότητα δηλαδή σε κάθε domain αντικείμενο. Επίσης για κάθε repository

υπάρχει ένα αντίστοιχο interface που το repository υλοποιεί με στόχο την εξάλειψη ρητών εξαρτήσεων και την διευκόλυνση testing στον κώδικά μας.

Repositories Εφαρμογής

```
public interface ITopicsRepo
    {
        Topics GetTopicById(long id);
        IEnumerable<Topics> GetActiveTopics();
        IEnumerable<Topics> GetActiveTopicsByGenreId(long id);
        IEnumerable<MainCategories> GetMainCategories();
        bool AddTopic(Topics chatRoom);
        int DeleteTopicById(long id);
        IEnumerable<Topics> GetUserTopics(string id);
        int RoomsRemainingForUser(string id);


        bool ChangeTopicStatus(long id, bool status);
        IEnumerable<Topics> GetTopicsFull();
}
```

ItopicsRepository

```
public class TopicsRepoRepository : ITopicsRepo
{
  private readonly ApplicationDbContext _context;
  private readonly IUsersService _usersService;

  public   TopicsRepoRepository(ApplicationDbContext   context,   IUsersService
usersService)
  {
    _context = context;
    _usersService = usersService;
  }

  public Topics GetTopicById(long Id)
  {
   return
   _context.Topics.Include(c  =>  c.MainCategory).FirstOrDefault(x  =>  x.Id  ==
Id);
   }

  public IEnumerable<Topics> GetActiveTopics()
  {
    return
      _context.Topics.Include(c => c.MainCategory).Where(x => x.Active);
  }

  public bool AddTopic(Topics topic){
    var user =
          _context.Users.SingleOrDefault(x => x.Id == topic.UserCreatorId);
    if (user == null || user.RoomsLeft == 0)
        return false;

    if (!_usersService.IsAdmin(user.Id))
        user.RoomsLeft--;

        _context.Users.Attach(user);
        _context.Entry(user).Property(x => x.RoomsLeft).IsModified = true;
        _context.Topics.Add(topic);
        _context.SaveChanges();

        return true;}

    }
```

Το ItopicsRepo interface δηλώνει 10 μεθόδους που πρέπει να ορίζονται από οποιαδήποτε κλάση το υλοποιεί. Η κλάση TopicsRepoRepository υλοποιώντας το παραπάνω interface δίνει και την υλοποίηση αυτών των μεθόδων.

Επεξήγηση παραπάνω μεθόδων:

1) GetTopicById: Φέρνει το chat room με το συγκεκριμένο id
2) GetActiveTopics Φέρνει όλα τα chat rooms που είναι ενεργά
3) AddTopic: Προσθέτει ένα topic/chat room στον πίνακα Topic

```
public interface IUsers
    {
        IEnumerable<ApplicationUser> GetRegisteredUsers();

        ApplicationUser GetUser(string id);

        long CountRegisteredUsers();

        bool UpdateUserStatus(bool status, string userId);

        bool IsAdmin(string userId);


    }
```

IUsers.cs

```
public class UsersRepository : IUsers
    {
        private readonly ApplicationDbContext _context;

        public UsersRepository(ApplicationDbContext context)
        {
            _context = context;
        }

        public ApplicationUser GetUser (string id)
        {
            return _context.Users.FirstOrDefault(i => i.Id == id);
        }

        public long CountRegisteredUsers()
        {
            return _context.ConnectedUsers.Count();
        }

        public IEnumerable<ApplicationUser> GetRegisteredUsers()
        {
            return _context.Users;
        }

        public bool UpdateUserStatus(bool status, string userId)
        {
            var user = _context.Users.FirstOrDefault(x => x.Id == userId);

            if (user == null) { return false; }

            user.Active = status;
            _context.Entry(user).State = EntityState.Modified;
            _context.SaveChanges();

            return true;
        }

        public bool IsAdmin(string userId)
        {
            var user = _context.Users.FirstOrDefault(x => x.Id == userId);

            if (user == null)
                return false;

            return user.Role == UsersRole.Admin;
        }
```

UsersRepository.cs

Επεξήγηση μεθόδων.

1) GetUser: Επιστρέφει τον Χρήστη με το συγκεκριμένο id
2) CountRegisteredUsers: Επιστρέφει το πλήθος των εγγεγραμμένων χρηστών
3) GetRegisteredUsers: Επιστρέφει όλους τους χρήστες ανεξαρτήτως αν είναι ενεργοί η όχι.
4) UpdateUserStatus: Ενημερώνει την κατάσταση του χρήστη
5) IsAdmin: Επιστρέφει αν ο χρήστης είναι ο Διαχειριστής

Παρατηρήσεις Repositories

Σε κάθε repository παρατηρούμε ότι δημιουργούμε ένα αντικείμενο τύπου ApplicationDbContext το οποίο μας δίνει την δυνατότητα να προσπελάσουμε τους πίνακες της βάσης και με την βοήθεια των βιβλιοθηκών της csharp όπως το LINQ εκτελούμε τα query που θέλουμε.

5.2.2 Service Layer

Services Εφαρμογής

Οι κλάσεις αυτές αντιπροσωπεύουν το business logic layer της εφαρμογής αν και πολλές φορές επειδή δεν υπάρχει η ανάγκη κάποιας συγκεκριμένης λογικής δρουν και σαν ένα απλό περιτύλιγμα του data access layer.Είναι κλάσεις που ακολουθούν τη ονοματολογική σύμβαση να τελειώνει το όνομα τους σε Services.Συνήθως η αντιστοιχία με τα repositories είναι ένα προς ένα. Σε κάθε service περνιέται ένα αντικείμενο του αντίστοιχου repository. Το συγκεκριμένο layer είναι πολύ σημαντικό και για το λόγο ότι μπορεί να απομονώσει όλο το back end και να χρησιμοποιηθεί και με διαφορετικές τεχνολογίες στο end point layer δηλαδή τους controllers.Για παράδειγμα με ακριβώς την ίδια υλοποίηση που θα δείξω παρακάτω θα μπορούσε ένα console application να μιλήσει με το service και να πάρει τα ίδια δεδομένα χωρίς να χρειαστεί να γίνει η παραμικρή αλλαγή.Τα services επίσης υλοποιούν ένα interface ακριβώς για τους ίδιους λόγους όπως και τα repositories.

```
public interface IUsersService
    {
        UserDto GetUser(string id);
        long CountRegisteredUsers();
        IEnumerable<ApplicationUser> GetRegisteredUsers();
        bool UpdateUserStatus(bool status, string userId);
        bool IsAdmin(string userId);
    }
```

IUsersService.cs

```
public class UsersService : IUsersService
    {
        private readonly IUsers _userRepo;

        public UsersService(IUsers userRepo)
        {
            _userRepo = userRepo;
        }

        public UserDto GetUser (string id)
        {
            return Mapper.Map<ApplicationUser, UserDto>(_userRepo.GetUser(id));
        }

        public long CountRegisteredUsers()
            => _userRepo.CountRegisteredUsers();

        public IEnumerable<ApplicationUser> GetRegisteredUsers()
        {
            return _userRepo.GetRegisteredUsers();

        }

        public bool UpdateUserStatus(bool status, string userId)
        {
            return _userRepo.UpdateUserStatus(status, userId);
        }

        public bool IsAdmin(string userId)
        {
            return _userRepo.IsAdmin(userId);
        }
    }
```

UsersService.cs

Παρατηρήσεις Services

Στη συγκεκριμένη περίπτωση παρατηρούμε ότι τα services έχουν ακριβώς τις ίδιες μεθόδους με τα αντίστοιχα repositories και ως επί το πλείστων απλά μεταφέρουν τη κλήση στο επόμενο επίπεδο. Επίσης παρατηρούμε ότι τα services επιστρέφουν Dto αντικείμενα και όχι τα domain αντικείμενα. Αυτό γίνεται κυρίως για 2 λόγους. Ο πρώτος είναι γιατί συνήθως δεν θέλουμε να φορτώσουμε ένα ολόκληρο domain αντικείμενο αν γνωρίζουμε ότι θα χρησιμοποιήσουμε μόνο 2 μέλη του. Ο δεύτερος είναι για να διασφαλίσουμε ότι δεν θα σπάσει ο κώδικας μας σε μια πιθανή αλλαγή στο domain αντικείμενο. Η μετατροπή από το domain αντικείμενο στο Dto αντικείμενό και αντίθετα γίνεται σε αυτό το layer με την βοήθεια στη συγκεκριμένη περίπτωση του AutoMapper .

Για παράδειγμα βλέποντας την παρακάτω μέθοδο

Παρατηρούμε ότι το service layer ενώ παίρνει αντικείμενο τύπου ApplicationUser σαν αποτέλεσμα από την κλήση της _userRepo.GetUser(id) μέσω του AutoMapper μετατρέπουμε αυτό το

```
public UserDto GetUser (string id)
 {

   return Mapper. Map<ApplicationUser,UserDto>(_userRepo.GetUser(id));
  }
```

αντικείμενο σε UserDto αντικείμενο και το δίνουμε στο πιο πάνω layer.

5.2.3 Controllers

Στην παρούσα πτυχιακή χρησιμοποιούνται και Mvc controllers και WebApi controllers. Αυτό το layer είναι το ενδιάμεσο μεταξύ back end και front end.Οργανώνει τις κλήσεις που γίνονται από το front end και είτε αν μιλάμε για MVC controller σερβίρει κάποιο view είτε αν μιλάμε για WebApi controller σερβίρει μόνο δεδομένα. Σε κάθε controller περνιέται μια εξάρτηση σε κάποιο service.

1)MVC Controllers

- ManageController
- AccountController

OI παραπάνω controllers είναι by default μέσα σε ένα ASP.NET MVC 5 project που διαθέτει το identity.Περιλαμβάνουν μεθόδους σχετικά με τη επεξεργασία των λογαριασμών των χρηστών.

- HomeController
- DashBoardController
- ChatEngineController
- UsersAreaController

Οι παραπάνω controllers διαθέτουν μεθόδους που ονομάζονται actions και μεταφέρουν τη κλήση δίνοντας ένα view στο interface.

Για παράδειγμα στον HomeController παρακάτω παρατηρούμε ότι εφόσον θα αναλάβει την κλήση κάποια action method έπειτα το σύνηθες είναι η μέθοδος να επιστρέψει κάποιο view. Στο MVC υπάρχει η σύμβαση ότι τα views έχουν ίδια ονόματα με τα αντίστοιχα action methods.

```
public class HomeController : Controller
  {
      private readonly ITopicsService _topicsService;

      public HomeController(ITopicsService topicsService)
      {
          _topicsService = topicsService;
      }

      public ActionResult MainCategories()
      {
          var mainCategories = _topicsService.GetMainCategories();
          return View("MainCategories", new MainCategoriesViewModel
          {
              MainCategories = mainCategories
          });
      }

      public ActionResult AllChatRooms()
      {
          return View();
      }
  }
```

```
public class RoomListController : ApiController
  {
      private readonly ITopicsService _service;

      public RoomListController(ITopicsService service)
      {
          _service = service;
      }

      [HttpGet]
      public IHttpActionResult GetAllRooms()
      {
          var activeRooms = _service.GetActiveTopics();

          if (activeRooms == null)
              return NotFound();

          return Ok(activeRooms);
      }

      [HttpGet]
      public IHttpActionResult GetRoomsForSpecificGenre(long id)
      {
          var genreRooms = _service.GetActiveTopicsByGenreId(id);

          if (genreRooms == null)
              return BadRequest();

          return Ok(genreRooms);
      }

      [HttpGet]
      public IHttpActionResult GetUserTopics(string id)
          => Ok(_service.GetUserTopics(id));


      [HttpGet]
      public IHttpActionResult GetTopicsFull()
      {
          var topics = _service.GetTopicsFull();

          return Ok(topics);
      }
```

2)Api Controllers

- ChatEngineApiController
- RoomCreatorController
- RoomListController
- UsersController

Οι Api controllers δέχονται κλήσεις στην εφαρμογή σχεδόν πάντα από ajax calls από το interface. Έχουν σαν στόχο να μιλήσουν με το back end και να κάνουν κάποια ενέργεια επιστρέφοντας κάποια data πίσω.

Για παράδειγμα στον RoomListController παρατηρούμε ότι πάνω από κάθε action method επιλέγουμε το http verb με το οποίο να καλεστεί η συγκεκριμένη μέθοδος. Επίσης παρατηρούμε ότι σαν επιστρεφόμενες τιμές δεν έχουμε Views αλλά data συνοδευόμενα με ένα status code.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] https://msdn.microsoft.com/en-us/library/ff649690.aspx

[2] https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

[3] http://deviq.com/repository-pattern/

[4] https://www.codeproject.com/Articles/842869/Repository-Pattern-using-Dependency-Injection-Auto

6.CLIENT SIDE ΚΟΜΜΑΤΙ ΕΦΑΡΜΟΓΗΣ

6.1 BOOTSTRAP 4

Το bootstrap είναι από τα πιο διάσημα html,CSS και JavaScript frameworks.Χρησιμοποιήθηκε στα razor views για την δημιουργία του HTML σκελετού κάθε σελίδας. Χρησιμοποιώντας τις κλάσεις CSS του bootstrap πετυχαίνουμε responsive UI σε όλες τις συσκευές με πολύ λιγότερη ενασχόληση στο CSS κομμάτι του interface.

6.1.1 Τρόπος οργάνωσης των Razor Views

MainCategories.chtml

```
@model FreeChat.Models.ViewModels.MainCategoriesViewModel

@{
    ViewBag.Title = "Free Chat";
    Layout = "~/Views/Shared/_AdminTemplate.cshtml";
}
@Styles.Render("~/bundles/IndexViewStyle")

<div class="container" id="container-categories-img-list">
    <ul class="list img-list" id="imglistMainCategories">
     @{
      foreach (var categ in Model.MainCategories)
      {
       <li data-categ-id='@categ.Id'>
         <a href='#' class='inner'>
          <div class='li-img'>
            <img src='@categ.CategoryImage' alt='@categ.Name'       height='120'/>
          </div>
                  <div class='li-text'>
                      <h3 class='li-head'>@categ.Name</h3>
                      <div class='li-sub'>
                          <p>@categ.CategoryDescription</p>
                      </div>
                  </div>
              </a>
              <button type="button" class="btn btn-primary goToRoomsBtn">
              See Rooms</button>
           </li>
           }
       }
    </ul>
</div>

<div class="card text-center" id="redirectPanel">
<div class="card-block" id="cardBlock">
@Html.ActionLink("Back to Main Categories","MainCategories","Home", null,new {
@class = "btn btn-primary"})

@Html.ActionLink("Create a room","Create","ChatRoom",null, new { @class = "btn
btn-primary" })
    </div>
</div>

@section scripts{
    @Scripts.Render("~/bundles/IndexView")
    <script type="text/javascript">
        $(function() {
            IndexController.Init();
        });
    </script>

}
```

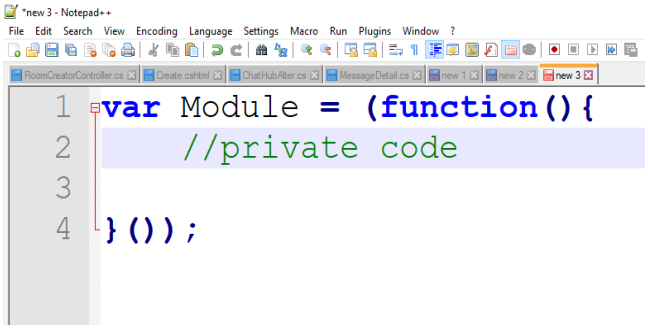
Παρατηρούμε ότι στην αρχή του chtml αρχείου επιλέγουμε το ViewModel που θέλουμε να τροφοδοτήσουμε το view αφού βέβαια το έχουμε περάσει από το controller στο view.Έπειτα επιλέγουμε το τίτλο της σελίδας και το layout αρχείο που θέλουμε να περικυκλώνει τη σελίδα. Στην συνέχεια οργανώνουμε τον HTML σκελετό της σελίδας χρησιμοποιώντας bootstrap κλάσεις όπως card m-5.Στο τέλος του αρχείου βλέπουμε ένα section που καλούμε τον JavaScript Controller της επιλογής μας θέμα το οποίο θα αναλύσω στην επόμενη ενότητα.

6.2 Οργάνωση JavaScript κώδικα

6.2.1 Module Pattern

Το Module Pattern είναι ένα design pattern σχεδιασμένο για να οργανώνει τον JavaScript κώδικα. Έχει σαν αποτέλεσμα ένα πολύ πιο καθαρό και αναγνώσιμο κώδικα. Επιπλέον πλεονεκτήματα είναι η δυνατότητα να δημιουργούμε namespaces και να επαναχρησιμοποιούμε τον κώδικά μας.

Το module pattern βασίζεται στον όρο Immediately-Invoked-Function Expression (Σχήμα 6.1) Η συγκεκριμένη function δηλώνεται και καλείται απευθείας. Μέσα σε αυτή την μέθοδο δημιουργείται εμβέλεια (scope) και δυνατότητα να έχουμε private μεταβλητές.



Εικόνα 0-22

6.2.2 Controllers Services στην JavaScript

Στην συγκεκριμένη εφαρμογή για λόγους οργάνωσης αλλά και πιο ευανάγνωστου κώδικα υλοποιήθηκε μία μορφή repository pattern και στο front end.Σχεδόν κάθε view διαθέτει ένα JavaScript Controller οργανωμένος σε ένα module υπεύθυνος για την οργάνωση του interface για αυτό το view.Επίσης όπου κρινόταν σκόπιμο δημιουργήθηκαν και JavaScript Services αρχεία που σαν στόχο έχουν να ομαδοποιούν τις ajax call κλήσεις προς τους WebApi controllers.

Η εφαρμογή έχει τους ακόλουθους JavaScript Controllers:

1) allchatRoomsPartialController.js
2) ChatEngineController.js
3) chatRoom.js

4) ChatRoomAdminPartialController.js

5) IndexController.js

```
(function (self, $, _document, _console, _indexService, undefined) {
    "use strict";

    var $doc;
    var $html;
    var $config;
    var $table;

    self.Init = function (config) {
        digestConfig(config);
        initImpl(config);
        listeners();
    };

    function digestConfig(config) {
        $config = config;
    };

    function initImpl(config) {
        $table = $("#allChatRooms").DataTable({
            ajax: {
                url: "/api/RoomList/GetAllRooms",
                dataSrc: ""
            },
            columns: [
                {
                    data: "Name"
                },
                {
                    data: "Genre"
                },
                {
                    data: "Description"
                },
                {
                    data: "DateExpired",
                },
                {
                    data: "Id",
                    render: function (data, type, room) {
                    return "<button class='btn btn-success roominitBtn'
                    id='" + data + "'>Enter Room</button>";
                    }
                }
            ]
        });
    }

    function listeners() {
        $(_document).on("click", ".roominitBtn", function () {
            var roomId = $(this).attr("id");

            $.ajax({
                method: "get",
                url: "/api/ChatEngineApi/Chatengine?roomId="+roomId,

                success: function(data) {
                    if (data) {
                    window.location = "/ChatEngine/ChatStart?roomid=" + roomId;
                    } else {
                        alert("Room Anavailable");
                    }
            }
            });
        });
    }
}
```

7. SignalR

7.1 Τι είναι το SignalR?

Το ASP.NET SignalR είναι μια βιβλιοθήκη της Microsoft η οποία απλουστεύει τη διαδικασία της ενσωμάτωσης λειτουργικότητας σε πραγματικό χρόνο μέσα στις εφαρμογές μας. Η λειτουργικότητα πραγματικού χρόνου στην ουσία είναι η δυνατότητα του διακομιστή(Server) να δίνει δεδομένα στους συνδεδεμένους πελάτες αμέσως όταν τα δεδομένα είναι διαθέσιμα χωρίς να περιμένει να ζητήσουν οι πελάτες καινούργια δεδομένα. Μέσω του SignalR δίνεται η δυνατότητα να υπάρξουν εφαρμογές που χρειάζονται μεγάλης συχνότητας ενημερώσεων από το διακομιστή όπως για παράδειγμα το Online Gaming.

Το SignalR χρησιμοποιεί το καινούργιο πρωτόκολλο Web Sockets όπου είναι διαθέσιμο και αυτομάτως χρησιμοποιεί παλιότερες τεχνολογίες όπου δεν είναι ,όπως long polling ,Server Send Events. Επίσης ο κάθε developer μπορεί να δημιουργήσει την εφαρμογή του χρησιμοποιώντας το SignalR αποκομίζοντας τα πλεονεκτήματα των Web Sockets χωρίς να χρειάζεται να νοιαστεί να φτιάξει ξεχωριστό κώδικα για παλιότερους σε τεχνολογία πελάτες μιας και το SignalR το διαχειρίζεται αυτόματα.

Αν και ο κάθε developer μπορεί να δημιουργήσει την εφαρμογή του με web sockets μόνος του το SignalR παρέχει όλη αυτήν την λειτουργικότητα που γλυτώνει από πολλές έννοιες και κώδικα. Για παράδειγμα αναλαμβάνει επίσης την αυτόματη ενημέρωση του Web Socket πρωτόκολλου παρέχοντας πάντα ένα ενημερωμένο API.

Η SignalR βιβλιοθήκη παρέχει δύο μοντέλα για την αμφίδρομη επικοινωνία μεταξύ των πελατών και των διακομιστών:

1) Persistent Connections
2) Hubs

1)Μια σύνδεση αντιπροσωπεύει ένα σημείο επικοινωνίας για αποστολή ενός η πολλών μυνημάτων.Το πακέτο των Persistent Connections δίνει την δυνατότητα στον developer να έχει άμεση πρόσβαση στο χαμηλό επίπεδο της επικοινωνίας του πρωτοκόλλου.

2)Τα Hubs είναι χτισμένα πάνω στο API των Persistent Connections και παρέχουν πιο φιλική βιβλιοθήκη προς τον προγραμματιστή. Η παρούσα εφαρμογή χρησιμοποίησε την HUB βιβλιοθήκη μιας και δεν υπήρχε ανάγκη για έλεγχο του χαμηλού επιπέδου επικοινωνίας.

Τρόπος Λειτουργίας Hubs

Όταν από τη μεριά του server καλεστεί μια μέθοδο στον client στέλνεται ένα πακέτο μέσω του ενεργού πρωτοκόλλου μεταφοράς που περιέχει το όνομα και τις παραμέτρους αυτής της μεθόδου κωδικοποιημένα σε JSON μορφή. Από την μεριά του client ,αυτός συσχετίζει το όνομα της μεθόδου με τις μεθόδους που υπάρχουν στο client side κώδικα. Αν βρεθεί αντιστοίχιση τότε εκτελείται η μέθοδος χρησιμοποιώντας τα κωδικοποιημένα δεδομένα σε JSON μορφή.

7.2 Τρόπος Χρησιμοποίησης Hubs

1)Αρχικά προσθέτουμε τις βιβλιοθήκες του SignalR στην εφαρμογή μας είτε μέσω του package manager console είτε μέσω του NuGet manager .(Βλέπε 5.1.1 σελ. 28).

2)Δημιουργούμε ένα φάκελο Hubs στην εφαρμογή μας και προσθέτουμε μια κλάση με το όνομα της επιλογής μας που κληρονομεί από την Hub κλάση.

3)Για να καθορίσουμε την διεύθυνση όπου οι πελάτες θα χρησιμοποιούν για να συνδέονται με το Hub καλούμε τη μέθοδο MapSignalR όταν ξεκινά η εφαρμογή μας. Το κατάλληλο αρχείο για να γίνει αυτό είναι η StartUp κλάση.

4)Υλοποιηθούμε τις μεθόδους της εφαρμογής μας για το server side κομμάτι και αντίστοιχα κάνουμε για το Client side κομμάτι.

7.3 Chat Engine Εφαρμογής

```
    public class Chat : Hub
    {

        // (1) Send the Callers Client Username to the view
        public void SendUsername()
        {
            var name = Context.User.Identity.Name;
            Clients.Caller.SendName(name);

        }
        //(2) Send Messages to room and cashing <=50 messages for this room
        public void SendMessageToRoom(string room, string message)
        {
            var date = DateTime.Now;
            var messageSend =
             new    List<string>    {    Context.User.Identity.Name,    message,
             date.ToString(CultureInfo.InvariantCulture) };


            Clients.Group(room).newMessage(messageSend);


             if((MessageCaching.TryGetValue(room,out                          var
             existingMessageCashing)) && (existingMessageCashing.Count <= 50))
            {
                existingMessageCashing.Add(new MessageDetail
                {
                    Message = message,
                    UserName = Context.User.Identity.Name,
                    TimeSend = date.ToString(CultureInfo.InvariantCulture)
                });
            }
            messageSend.Clear();
        }

        //(3)Send Message to Specific user inside the room
   public  void  SendMessageToUser(string  nameToChat,  string  message,  string
group2Name)
        {
            var date = DateTime.Now;
            var user = RoomsUsers
                .FirstOrDefault(x => x.Value.Contains(nameToChat))
                .Key;
                  var        messageSend        =        new        List<string>
      {Context.User.Identity.Name,                                        message,
      date.ToString(CultureInfo.InvariantCulture) };

            Clients.User(user).newMessage(messageSend);
        }
```

ChatHub.cs--- SendUsername, SendMessageToRoom, SendMessageToUser Methods

```
(function(self,$,_document,_console,_chatEngineService,undefined) {
    "use strict";

    self.Init = function (config) {
        listeners();

        initImpl(config);

    };
    function initImpl(config) {
        $(".content-wrapper").addClass("chatEngineMode");
              $("#sidenavToggler").trigger("click");

              roomName = config.RoomName;
            _$chat = $.connection.chat;
             $.connection.hub.logging = true;
            _$chat.client.newMessage = onNewMessage;
            _$chat.client.newMessagePrivate = onNewMessagePrivate;
            _$chat.client.onlineUsers = connectedUsers;
            _$chat.client.loadHistory = showHistory;
            _$chat.client.sendName = saveUsernameGotFromHub;
            _$chat.client.private = isInPrivateChat;

            $.connection.hub.start().done(function () {

                    _$chat.server.joinRoom(_roomName);
                    _$chat.server.sendUsername();
                    _$chat.server.sendRoomConnectedUsers(_roomName);
                    _$chat.server.sendSavedRoomMessages(_roomName);
                    $("#send").click(onSend);

            })
                .fail(function () {
                    alert("Error connecting to group : " + _roomName);
                    window.location.href = '/Home/Index';
                });
    function onNewMessage(message) {

        var d = new Date();
        d.toLocaleTimeString();

        userFullName = message[0];
        realMessage = message[1];
        timeSend = message[2];
        var user = message[0].substring(0, 4);

        if (userFullName.substring(0, 4) !== connectedUser.substring(0,4)) {
            $(".inner-list").append(html);
        } else {
            $(".inner-list").append(html);
        }
        var selectorForImg = "#" + user + messageCount + "img";

        messageCount = messageCount + 1;

        $(selectorForImg).attr("src", imagePath);
```

ChatEngineController.js

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] https://docs.microsoft.com/en-us/aspnet/signalr/overview/guide-to-the-api/hubs-api-guide-server

8. Μελλοντικές Επεκτάσεις Εφαρμογής

Real time notifications

Στην εφαρμογή θα μπορούσε να προστεθεί μηχανισμός real time ειδοποιήσεων σε όλους τους συνδεδεμένους χρήστες σε ποικίλες περιπτώσεις όπως:

- Δημιουργία καινούργιου chat room
- Λήξη chat room
- Συμμετοχή χρήστη σε chat room
- Έξοδος χρήστη από chat room

Προτάσεις Δωματίων σε χρήστες αναλόγως τις προτιμήσεις τους

Στην εφαρμογή επίσης θα μπορούσε να προστεθεί μόνιμη αποθήκευση των δωματίων που παίρνει μέρος ο χρήστης και μέσω αυτών των προτιμήσεων να τον/την ενημερώνουμε για παρόμοια δωμάτια που δημιουργούνται ή λήγουνε.

Bonus διαθέσιμων δωματίων σε συχνούς χρήστες

Επίσης σε συχνούς χρήστες της εφαρμογής με βάση κάποιο έλεγχο να αυξάνονται τα διαθέσιμα δωμάτια που έχει στη διάθεσή του να φτιάξει ο χρήστης ευχαριστώντας τον έτσι για την προτίμηση του στην εφαρμογή.

Δημιουργία private group εντός κάποιου δωματίου

Μια επιπλέον προσθήκη θα μπορούσε να ήταν η δημιουργία ενός private group από συχνούς χρήστες που κερδίζουνε αυτό το δικαίωμα με βάση των αριθμό των δωματίων που έχουνε φτιάξει κατά την διάρκεια συνομιλίας εντός κάποιου chat room.

Εισαγωγή news letter στους χρήστες

Θα μπορούσε να προστεθεί συχνό ενημερωτικό email προς τους χρήστες και να ενημερώνει για τη λήξη των δωματίων που έχουν φτιάξει αλλά επίσης και να τους δίνεται η ευκαιρία να βαθμολογήσουνε τις βασικές κατηγορίες του Free chat και να προτείνουνε και άλλες.

Σελίδα στατιστικών εφαρμογής

Στο user interface της εφαρμογής και συγκεκριμένα στο landing page θα μπορούσε να προστεθεί τμήμα παρουσίασης στατιστικών της εφαρμογής που θα περιλαμβάνει πληροφορίες και δεδομένα όπως:

- Σύνολο συνδεδεμένων χρηστών σε κάθε δωμάτιο
- Σύνολο εγγεγραμμένων χρηστών
- Αριθμός δωματίων που δημιουργούνται
- Τάση επισκεψημότητας στο site