ΑΤΕΙ ΗΡΑΚΛΕΙΟΥ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Πλατφόρμα διαχείρισης προσωπικού σε ASP.NET με αρχιτεκτονική MVC

Του Λιάπη Χρήστου

Επιβλέπων καθηγητής: Νικόλαος Παπαδάκης

Ηράκλειο | Οκτώβρης 2018

Περίληψη

Οι web developers και designers, ανέκαθεν ήταν σημαντικοί παράγοντες για την ανάπτυξη εταιριών που χρησιμοποιούσαν το διαδίκτυο για να διαφημιστούν. Πλέον, με τα καινούργια εργαλεία που παρουσιάζονται τα τελευταία χρόνια, αυτές οι εταιρίες μπορούν να προσφέρουν και παραπάνω υπηρεσίες στους χρήστες τους, από απλό interactivity, μέχρι και ένα ολόκληρο eshop.

Ο σκοπός της διπλωματικής, είναι η δημιουργία μιας εύχρηστης εφαρμογής, στην οποία μια εταιρία θα μπορεί εύκολα να διαχειρίζεται το προσωπικό και τα projects τα οποία έχει αναλάβει. Η υλοποίηση αυτής της εφαρμογής, θα γίνει με τη χρήση του ASP.NET Core της Microsoft, καθώς και άλλων Frameworks για την επέκταση της λειτουργίας της.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την οικογένειά μου, η οποία με στήριξε και ήταν δίπλα μου σε όλα μου τα βήματα.

Ακόμη, δεν θα μπορούσα να μην ευχαριστήσω τους φίλους μου, οι οποίοι με υποστήριξαν, με βοήθησαν και μου χάρισαν ωραίες στιγμές στην μέχρι τώρα πορεία μας.

Περιεχόμενα

Περίληψη 2
Ευχαριστίες 3
Περιεχόμενα 4
Κατάλογος Εικόνων 6
Κεφάλαιο 1 8
Εισαγωγή 8
1.1 Αντικείμενο διπλωματικής εργασίας 8
1.2 Σύνοψη διπλωματικής εργασίας 9
Κεφάλαιο 2 10
Τεχνολογίες Ανάπτυξης Εφαρμογής 10
2.1 ASP.NET 10
2.2 SQL Server 12
2.3 Entity Framework 13
2.4 ASP.NET Identity 14
Κεφάλαιο 3 16
Υλοποίηση εφαρμογής 16
3.1 Δομή της εφαρμογής 16
3.2 Δομή της βάσης δεδομένων 17
3.3 MVC (Model View Controller) Framework 21
3.3.1 Controllers 23
3.3.2 Views 26
3.3.3 Models 28
3.4 Interface της εφαρμογής 31
3.4.1 Λίστα με τους εργαζομένους 33
3.4.2 Λίστα με τα Tasks 35
3.4.3 Λίστα με τα Projects 36
3.4.4 Ρυθμίσεις και πληροφορίες χρήστη 40
Παράρτημα 45
Controllers 45
Views 54
Models 66

Υλοποίηση του IdentityConfig 73
Υλοποίηση του BundleConfig 75
Κεφάλαιο 4 43
4.1 Συμπεράσματα 43
4.2 Προοπτικές Βελτίωσης 44
Βιβλιογραφία 76

Κατάλογος Εικόνων

Εικόνα 1: Αρχιτεκτονική ASP.NET 11
Εικόνα 2: Αρχιτεκτονική ASP.NET Core 12
Εικόνα 3: Entity Framework 14
Εικόνα 4: Αρχιτεκτονική ASP.NET Identity 15
Εικόνα 5: Προσθήκη Authorization στους χρήστες 17
Εικόνα 6: Λίστα με τα tables της εφαρμογής. 18
Εικόνα 7: Πίνακας AspNetUsers 19
Εικόνα 8: Πίνακας Projects 19
Εικόνα 9: Πίνακας ProjectTasks 20
Εικόνα 10: Πίνακας AspNetRoles 20
Εικόνα 11: Πίνακας ApplicationUserProjects 20
Εικόνα 12: Πίνακας AspNetUserRoles 21
Εικόνα 13: Αναπαράσταση του MVC Pattern. 22
Εικόνα 14: Default Routing μέσα σε μια εφαρμογή ASP.NET MVC 23
Εικόνα 15: Παράδειγμα από Controller 24
Εικόνα 16: Παράδειγμα από action του Controller 24
Εικόνα 17: Σχέση μεταξύ Controller action και View 25
Εικόνα 18: Παραδείγματα του Razor Syntax σε διάφορες περιπτώσεις. 27
Εικόνα 19: Σχέση μεταξύ του action με όνομα Details και του αντίστοιχου View. 28
Εικόνα 20: Ορισμός της κλάσης ProjectTask ως Model. 29
Εικόνα 21: Δήλωση της κλάσης ProjectTasks ως μέλος της βάσης δεδομένων. 29
Εικόνα 22: Το τελικό table του ProjectTasks στη βάση δεδομένων. 30
Εικόνα 23: Παράδειγμα ενός migration. 31
Εικόνα 24: Είσοδος χρήστη στην πλατφόρμα. 32
Εικόνα 25: Εγγραφή νέου χρήστη στην πλατφόρμα. 32
Εικόνα 26: Επιλογές του χρήστη στην πλατφόρμα. 33
Εικόνα 27: Διαφορές στη λίστα των εργαζομένων ανάμεσα σε administrators και απλών users. 34
Εικόνα 28: Διαφορές στην επιλογή Details των εργαζομένων (α). 34
Εικόνα 29: Διαφορές στην επιλογή Details των εργαζομένων (β). 35
Εικόνα 30: Σελίδα επιβεβαίωσης διαγραφής ενός χρήστη. 35
Εικόνα 31: Λίστα με τα tasks που υπάρχουν. 36
Εικόνα 32: Λίστα με τα projects όπως την βλέπουν οι administrators. 36
Εικόνα 33: Λίστα με τα projects όπως την βλέπουν οι χρήστες. 37
Εικόνα 34: Επιλογή Edit για τον Project Manager. 38
Εικόνα 35: Επιπλέον δυνατότητες του Project Manager. 38
Εικόνα 36: Επιλογή εργαζομένων για να μπουν στο project. 38

Εικόνα 37: Κατάσταση ενός task όταν δεν το έχει αναλάβει κάποιος χρήστης. 39
Εικόνα 38: Κατάσταση ενός task όταν το έχει αναλάβει κάποιος χρήστης, αλλά δεν το έχει τελειώσει ακόμα. 39
Εικόνα 39: Κατάσταση ενός task όταν το έχει αναλάβει κάποιος χρήστης και το έχει τελειώσει. 39
Εικόνα 40: Πληροφορίες του λογαριασμού του χρήστη. 40
Εικόνα 41: Φόρμα αλλαγής κωδικού χρήστη. 40
Εικόνα 42: Μήνυμα επιτυχούς αλλαγής κωδικού. 41
Εικόνα 43: Μήνυμα ανεπιτυχούς αλλαγής κωδικού. 41
Εικόνα 44: Ορισμός πολιτικών για τους κωδικούς των χρηστών, όπως έχει οριστεί από το Identity Framework. 42

Κεφάλαιο 1

Εισαγωγή

Στο παρόν κεφάλαιο γίνεται μια μικρή παρουσίαση του αντικειμένου της εργασίας, της δομής αυτού και των σκοπών του. Ακόμη, παρουσιάζεται η σύνοψη των επόμενων κεφαλαίων ώστε να μπορέσει ο αναγνώστης να κατατοπιστεί επαρκώς σχετικά με το τι πρόκειται να διαβάσει στις επόμενες σελίδες.

1.1 Αντικείμενο διπλωματικής εργασίας

Αντικείμενο της εργασίας αυτής αποτελεί η δημιουργία μίας διαδικτυακής πλατφόρμας διαχείρισης προσωπικού. Στην εφαρμογή υπάρχουν διαφορετικοί ρόλοι χρηστών με τα αντίστοιχα δικαιώματα, όπως για παράδειγμα διαχειριστής και υπάλληλος. Η πλατφόρμα θα περιέχει όλα τα στοιχεία των εργαζόμενων, ενώ θα υπάρχει και διαχείριση εργασιών μέσω της οποίας οι χρήστες θα ενημερώνουν και θα ενημερώνονται σχετικά με τις εργασίες που έχουν αναλάβει. Στόχος της εφαρμογής είναι να είναι εύχρηστη και αξιόπιστη. Με σκοπό τα παραπάνω, η κύρια τεχνολογία που χρησιμοποιήθηκε για την ανάπτυξη της ήταν το framework ASP.NET της Microsoft, σε συνδυασμό με το Entity framework για το μοντέλο οντοτήτων, ενώ τα δεδομένα θα αποθηκεύονται σε μία βάση δεδομένων τύπου SQL server.

1.2 Σύνοψη διπλωματικής εργασίας

Η παρούσα εργασία αποτελείται από πέντε κεφάλαια. Στο πρώτο κεφάλαιο περιγράφεται το αντικείμενο της εργασίας. Στο δεύτερο κεφάλαιο πραγματοποιείται μία σύντομη παρουσίαση των τεχνολογιών που χρησιμοποιήθηκαν. Στο τρίτο κεφάλαιο ακολουθεί η περιγραφή της ανάπτυξης της εφαρμογής, παρουσιάζοντας τον κώδικα και την εφαρμογή. Στο τέταρτο κεφάλαιο παρατίθενται παραδείγματα από κώδικα της εφαρμογής, για την καλύτερη κατανόηση της υλοποίησης της. Στο πέμπτο και τελευταίο κεφάλαιο, καταγράφονται τα συμπεράσματα και οι προοπτικές βελτίωσης της υλοποιηθείσας εφαρμογής.

Κεφάλαιο 2

Τεχνολογίες Ανάπτυξης Εφαρμογής

Στο κεφάλαιο αυτό γίνεται μια παρουσίαση των τεχνολογιών που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής της παρούσας εργασίας. Πιο συγκεκριμένα, παρουσιάζονται το framework ASP.NET, ο SQL Server, το Entity Framework και το ASP.NET Identity. Στα πλαίσια της εφαρμογής χρησιμοποιήθηκαν οι εκδόσεις ASP.NET Core 2.1, SQL Server 2017, Entity Framework 6 και ASP.NET Identity 2.0.

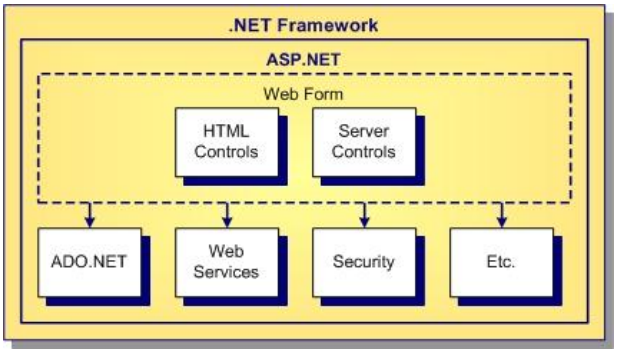
2.1 ASP.NET

Το ASP.NET είναι ένα framework της Microsoft που εξυπηρετεί τη δημιουργία δυναμικών ιστοσελίδων στο διαδίκτυο (Microsoft, ASP.NET, 2018). Αποτελεί το διάδοχο της τεχνολογίας Active Server Pages (ASP) και από το 2014 διατίθεται σε μορφή ανοιχτού κώδικα. Η τεχνολογία ASP συναντάται σε αρκετά μοντέλα ανάπτυξης: ASP, ASP.NET Web Forms, ASP.NET MVC, ASP.NET Web Pages, ASP.NET API και ASP.NET Core. Το ASP.NET είναι τεχνολογία που εκτελείται στην πλευρά του server, ενώ το ASP.NET Core κυκλοφόρησε το 2016 ενώνοντας όλα τα προηγούμενα μοντέλα σε ένα κοινό framework. Το ASP.NET 4.6 είναι η πιο σταθερή έκδοση του framework που συνεχίζει να χρησιμοποιείται ακόμα, ενώ με το ASP.NET Core οι εφαρμογές μπορούν να τρέξουν σε οποιαδήποτε πλατφόρμα[i].

Με την έκδοση ASP.NET 2.0 η Microsoft εισήγαγε το μοντέλο code-behind που επιτρέπει στατικό κείμενο να παραμένει στη σελίδα .aspx, ενώ ο δυναμικός κώδικας παραμένει σε αρχείο .aspx.cs ή .aspx.vb ή .aspx.fs ανάλογα με τη γλώσσα προγραμματισμού που χρησιμοποιείται. Με αυτόν τον τρόπο διαχωρίζεται η εμφάνιση της σελίδας με το περιεχόμενο της.

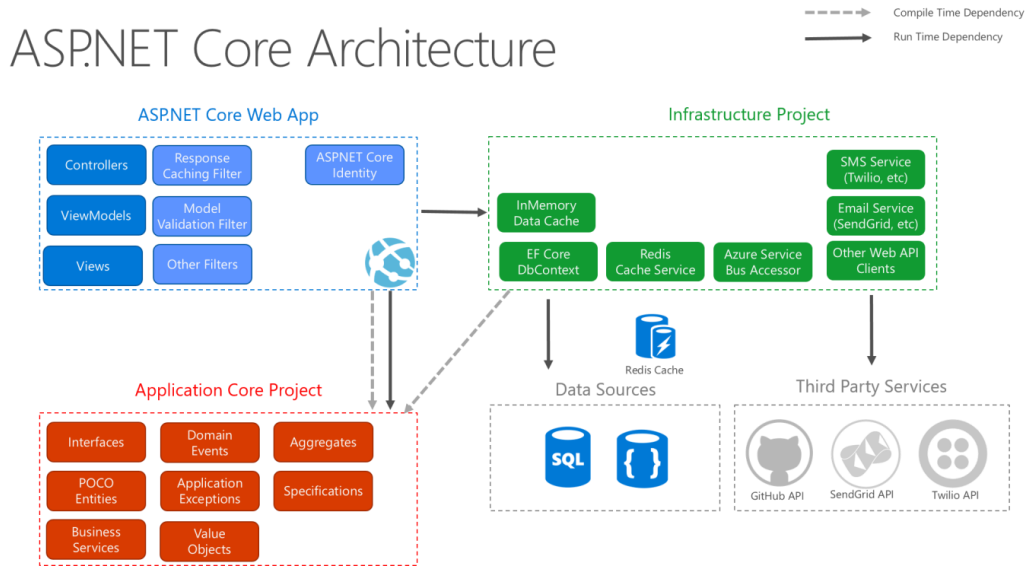
Χαρακτηριστικά του ASP.NET είναι οι ντιρεκτίβες, ειδικές εντολές για το πως πρέπει το ASP.NET να επεξεργαστεί τη σελίδα. Το .NET χρησιμοποιεί την τεχνική

“visited composites” για την απόδοση της σελίδας. Κατά τη διάρκεια της μεταγλώττισης το πρότυπο αρχείο .aspx μετατρέπεται σε κώδικα αρχικοποίησης που χτίζει ένα δέντρο ελέγχου (που ονομάζεται composite), το οποίο αναπαριστά το αρχικό αρχείο. Ο κώδικας αρχικοποίησης συνδυάζεται με τον κώδικα που έχει γράψει ο προγραμματιστής και καταλήγει σε μία συγκεκριμένη κλάση για τη σελίδα.



Εικόνα 1: Αρχιτεκτονική ASP.NET

Στην **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.** (Πηγή: http://docs.embarcadero.com/products/rad_studio/radstudio2007/RS2007_helpupdates/HUpdate4/EN/html/devnet/aspnetov_xml.html) παρουσιάζεται η αρχιτεκτονική των πρώτων εκδόσεων ASP.NET. Τα Web Forms ορίζουν τη διεπαφή του χρήστη με την εφαρμογή και περιέχουν HTML στοιχεία και λειτουργίες ελέγχου του ASP.NET server. Τα Web Forms έχουν πρόσβαση στα δεδομένα μέσω του ADO.NET, ενώ τα Web Services παρέχουν στοιχεία της εφαρμογής σε πολλά κατανεμημένα συστήματα χρησιμοποιώντας μηνύματα XML. Στην Εικόνα 2 (Πηγή: https://docs.microsoft.com/en-us/dotnet/standard/modern-web-apps-azure-architecture/common-web-application-architectures) παρουσιάζεται η αρχιτεκτονική μίας εφαρμογής ASP.NET Core αν ακολουθηθούν οι προτεινόμενες οδηγίες του framework. Η πρόοδος της αρχιτεκτονικής είναι σαφής καλύπτοντας όλο και μεγαλύτερο εύρος εφαρμογών.



Εικόνα 2: Αρχιτεκτονική ASP.NET Core

2.2 SQL Server

Ο SQL Server είναι ένα σύστημα διαχείρισης σχεσιακής βάσης δεδομένων που αναπτύχθηκε από τη Microsoft, το οποίο μπορεί να τρέξει είτε τοπικά είτε σε έναν άλλον υπολογιστή μέσω ενός δικτύου (Microsoft, Microsoft, 2018). Η Microsoft έχει παρουσιάσει πολλές διαφορετικές εκδόσεις του εργαλείου στοχεύοντας σε διαφορετικά κοινά και ποικιλία στο φόρτο δεδομένων με την πρώτη έκδοση του εργαλείου να κυκλοφορεί το μακρινό 1989.

Οι κύριες εκδόσεις του SQL Server είναι οι εξής:

- Enterprise: Διαθέτει τον πυρήνα της βάσης δεδομένων, επιπλέον υπηρεσίες, αλλά κι ένα εύρος εργαλείων για τη δημιουργία και διαχείριση ενός SQL Server cluster.
- Standard: Έχει τα ίδια χαρακτηριστικά με την έκδοση Enterprise με τη διαφορά ότι υποστηρίζει λιγότερα ενεργά στιγμιότυπα της βάσης, ενώ δεν παρέχει και κάποιες λειτουργίες, όπως η προσθήκη μνήμης ενώ ο server εξακολουθεί να τρέχει.
- Web: Η έκδοση για φιλοξενία ιστοσελίδων.

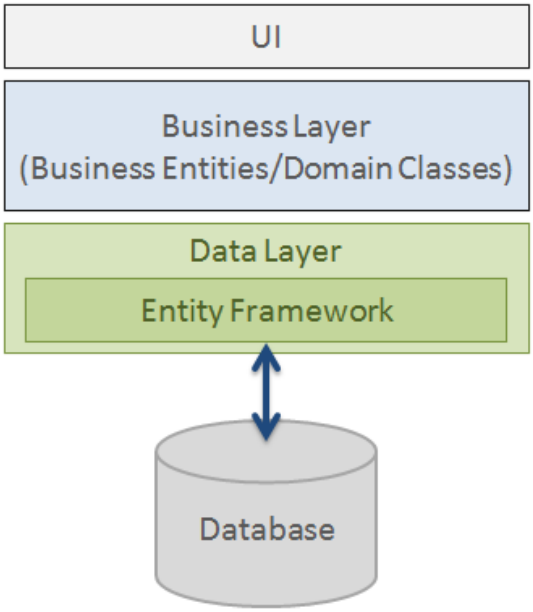
- Business Intelligence: Περιλαμβάνει την έκδοση Standard και επιπλέον εργαλεία σχετικά με επιχειρησιακές δραστηριότητες.
- Express: Η δωρεάν έκδοση του SQL Server που διαθέτει τον πυρήνα της βάσης δεδομένων με περιορισμό στους πόρους που μπορεί να χρησιμοποιήσει.

Όλες οι λειτουργίες που μπορεί να γίνουν σε έναν SQL Server διαχέονται μέσω του Tabular Data Stream (TDS), ένα πρωτόκολλο επιπέδου εφαρμογής που έχει ορίσει η Microsoft και χρησιμοποιείται για τη μεταφορά δεδομένων μεταξύ ενός server βάσης δεδομένων και ενός πελάτη. Τα TDS πακέτα μπορούν να ενθυλακωθούν σε άλλα πρωτόκολλα, όπως το TCP/IP και η κοινόχρηστη μνήμη.

2.3 Entity Framework

Πριν από την έκδοση .NET 3.5 οι προγραμματιστές αναγκάζονταν να γράψουν επιπλέον κώδικα για τη διαχείριση της βάσης δεδομένων, ανοίγοντας μία σύνδεση με τη βάση, δημιουργώντας DataSet για τη ανάγνωση ή εγγραφή δεδομένων. Η διαδικασία αυτή ήταν επίπονη για τους προγραμματιστές κι επιρρεπής σε σφάλματα. Για το λόγο αυτό η Microsoft δημιούργησε το Entity Framework που αυτοματοποιεί όλες τις σχετικές δραστηριότητες μεταξύ βάσης δεδομένων και της εφαρμογής.

Ο επίσημος ορισμός που δίνει η Microsoft είναι πως το Entity Framework είναι ένα object-relational mapping (ORM) framework που επιτρέπει στους προγραμματιστές .NET να εργαστούν με σχεσιακά δεδομένα χρησιμοποιώντας domain-specific αντικείμενα (Microsoft, 2018). Το Entity Framework επιτρέπει τη δημιουργία ενός μοντέλου γράφοντας κώδικα ή χρησιμοποιώντας κουτιά και γραμμές στον EF Designer. Και οι δύο προσεγγίσεις μπορούν να χρησιμοποιηθούν σε μία υπάρχουσα βάση δεδομένων ή για τη δημιουργίας μίας νέας. Στην Εικόνα 3 (Πηγή: http://www.entityframeworktutorial.net/what-is-entityframework.aspx) παρουσιάζεται η θέση του Entity Framework σε μία εφαρμογή.

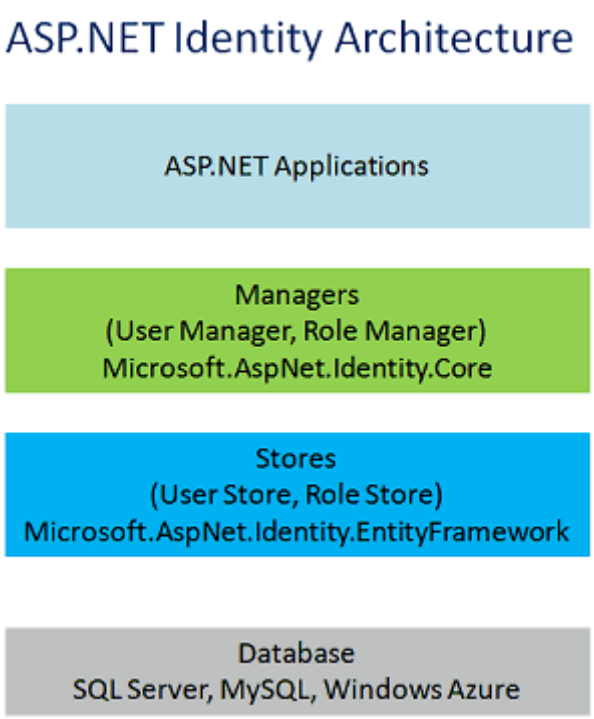


Εικόνα 3: Entity Framework

2.4 ASP.NET Identity

Το Identity Framework του ASP.NET, είναι ένας εύκολος τρόπος που έχει προωθήσει η Microsoft για να δημιουργούνται εύκολα συστήματα διαχείρισης χρηστών, σε εφαρμογές και πλατφόρμες που το χρειάζονται. Παλιά το σύστημα αυτό ονομαζόταν ASP.NET Membership και στη συνέχεια, μετά από αλλαγές, μετονομάστηκε σε ASP.NET Identity. Στην Εικόνα 4 (Πηγή: https://www.tektutorialshub.com/wp-content/uploads/2015/10/ASP.NET-Identity-Architecture.png) παρουσιάζεται η αρχιτεκτονική του Identity Framework.

Με αυτό το Framework, η Microsoft παρουσίασε έναν εύκολο τρόπο να μπορούν οι προγραμματιστές να δημιουργήσουν ένα σύστημα διαχείρισης χρηστών. Πιο συγκεκριμένα, κατευθείαν με τη δημιουργία ενός project με membership system μέσω του Identity, το web application έχει τη δυνατότητα registration νέων χρηστών καθώς και login και log off. Επιπλέον, δίνει τη δυνατότητα στον προγραμματιστή να δημιουργήσει εύκολα ρόλους για τους χρήστες, όπως για παράδειγμα τον ρόλο του administrator ο οποίος θα έχει παραπάνω δικαιώματα από τους υπόλοιπους χρήστες.



Εικόνα 4: Αρχιτεκτονική ASP.NET Identity

Η αρχιτεκτονική του Identity χωρίζεται σε τέσσερα επίπεδα. Αρχικά έχουμε την εφαρμογή μας που καλεί το Framework για να πάρει πληροφορίες, ή να διαχειριστεί κάποιο χρήστη. Στη συνέχεια, καλεί έναν από τους Managers για να διαχειριστεί την κλήση αυτή. Οι Managers είναι υπεύθυνοι για τη διαχείριση των λεγόμενων Stores. Τα Stores διαχειρίζονται τα λεγόμενα models, όπως αποκαλούνται από το Model View Controller convention, τα οποία ουσιαστικά είναι κάποια tables στη βάση δεδομένων που έχει δημιουργηθεί αυτόματα από το ASP.NET, τα οποία έχουν την πληροφορία για τους χρήστες και τους ρόλους τους. Μόλις, λοιπόν, πάρουν τα Stores την πληροφορία που χρειάζονται από τους Managers, πηγαίνουν στο τέταρτο και τελευταίο επίπεδο, τη βάση δεδομένων της εφαρμογής, για να τραβήξουν τα στοιχεία που ζητήθηκαν.

Κεφάλαιο 3

Υλοποίηση εφαρμογής

Στο κεφάλαιο αυτό, γίνεται μια περιγραφή του τρόπου υλοποίησης της πλατφόρμας Διαχείρισης Προσωπικού. Πιο συγκεκριμένα, παρουσιάζεται η κυρίως δομή της εφαρμογής αυτής, καθώς και η βασική λειτουργία της. Όπως περιεγράφηκε παραπάνω, τα εργαλεία που χρησιμοποιήθηκαν, ήταν το ASP.NET. το Entity Framework, ο SQL Server και το ASP.NET Core Identity για την εισαγωγή χρηστών στο σύστημα. Επίσης, χρησιμοποιήθηκε το μοντέλο MVC (Model View Controller), το οποίο θα αναλυθεί περισσότερο παρακάτω.

3.1 Δομή της εφαρμογής

Η εφαρμογή διαχείρισης προσωπικού που υλοποιήθηκε, φτιάχτηκε με στόχο τη δημιουργία μιας ενιαίας πλατφόρμας, μέσω της οποίας τα μέλη μιας υποτιθέμενης επιχείρησης, θα μπορούσαν να δημιουργούν projects, να θέτουν tasks σε άλλα μέλη της επιχείρησης και να ενημερώνονται γενικά για τις εκκρεμότητες που έχουν. Εκτός από τη λίστα των projects και των tasks όμως, η εφαρμογή θα παρέχει και μια λίστα όλων των μελών της επιχείρησης την οποία θα μπορούν να βλέπουν όλα τα μέλη.

Αρχικά, εφόσον αυτή η εφαρμογή θα αφορούσε μόνο την επιχείρηση, οι υπηρεσίες που μπορεί να προσφέρει αυτή, δεν θα πρέπει να είναι προσβάσιμες σε οποιονδήποτε, παρά μόνο στα μέλη της επιχείρησης. Χρησιμοποιώντας το Identity Framework, μπορούμε εύκολα να βάλουν ένα είδος φίλτρου, όπως ονομάζεται από το ASP.NET, το οποίο θα εμποδίζει τη χρήση της πλατφόρμας, εκτός αν κάνεις login σε αυτή. Στην Εικόνα 5 φαίνεται πως ορίζεται ακριβώς το Authorization των χρηστών μέσω ενός configuration αρχείου που έχει δημιουργηθεί αυτόματα.

Μόλις συνδεθούν οι χρήστες στην πλατφόρμα, έχουν πρόσβαση στη λίστα με τα υπόλοιπα μέλη, μια λίστα με τα projects που υπάρχουν και μια λίστα με όλα τα tasks που υπάρχουν καθώς και links που σε στέλνουν από το task στο project στο οποίο αντιστοιχεί. Σε κάθε μια από αυτές τις λίστες μπορούν να δουν πληροφορίες για κάθε

αντίστοιχη εγγραφή. Επίσης, μπορούν να αλλάξουν τα στοιχεία της κάθε εγγραφής, εφόσον όμως έχουν τα κατάλληλα δικαιώματα. Για τα projects και τα tasks, πρέπει να είσαι ο Administrator της πλατφόρμας, ή να είσαι ο χρήστης που δημιούργησε το task ή το project. Επιπλέον, όταν βλέπουν τις πληροφορίες ενός project, βλέπουν ποιοι χρήστες συμμετέχουν σε αυτό, καθώς και ποια tasks υπάρχουν σε αυτό. Αυτές οι λίστες μπορούν να αλλάξουν από τον Administrator και τον Project Manager. Project Manager ορίζεται αρχικά αυτός που δημιουργεί το αντίστοιχο project και στη συνέχεια μπορεί να αλλάξει και να μεταφέρει τα δικαιώματα του σε άλλο μέλος του project αυτού.

```
using System.Web;
using System.Web.Mvc;

namespace MvcEmployeeManager
{
    public class FilterConfig
    {
        public static void RegisterGlobalFilters(GlobalFilterCollection filters)
        {
            filters.Add(new HandleErrorAttribute());
            filters.Add(new AuthorizeAttribute()); //add authorization to users
        }
    }
}
```

Εικόνα 5: Προσθήκη Authorization στους χρήστες

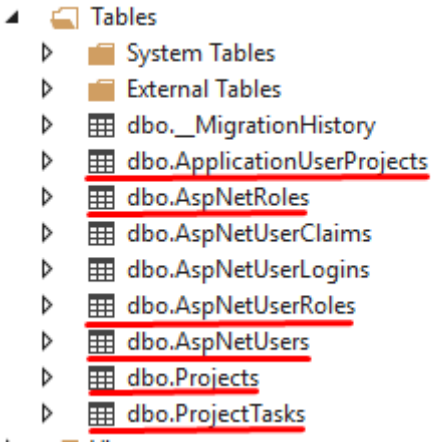
3.2 Δομή της βάσης δεδομένων

Η εφαρμογή αυτή, στηρίζεται πάνω σε μια βάση δεδομένων πάνω στην οποία θα αποθηκεύονται οι χρήστες, τα projects και τα tasks. Χρησιμοποιώντας αυτή τη βάση λοιπόν θα μπορεί να λειτουργήσει σωστά η πλατφόρμα.

Αρχικά έχουμε τα τρία βασικά tables που έχουμε αναφέρει. Έχουμε τους AspNetUsers, τα Projects, και τα ProjectTasks. Το table Application Users είναι ένα table που το παρέχει από μόνο του το ASP.NET, εφόσον δημιουργούμε πλατφόρμα που χρησιμοποιεί ένα membership system. Στην Εικόνα 6 φαίνονται όλα τα tables που έχουν δημιουργηθεί για την εφαρμογή. Υπογραμμισμένα με κόκκινο είναι αυτό που χρησιμοποιούνται. Τα υπόλοιπα είναι tables τα οποία έχουν παραχθεί από το

ASP.NET αυτόματα για άλλες λειτουργίες, σε περίπτωση που ο προγραμματιστής επιλέξει να τις αξιοποιήσει.

Η σύνδεση των tables αυτών είναι η εξής. Ένας AspNetUsers μπορεί να συμμετέχει σε πολλά Projects και να έχει και πολλά Tasks. Ένα project μπορεί να έχει πολλούς AspNetUsers και πολλά Tasks. Τέλος, ένα Task μπορεί να ανήκει μόνο σε ένα Project και να είναι κατειλημμένο (claimed) μόνο από έναν AspNetUsers. Εσωτερικά επίσης το Identity Framework έχει ορίσει και τον πίνακα AspNetRoles για να κρατάει τους ρόλους των χρηστών.



Εικόνα 6: Λίστα με τα tables της εφαρμογής.

Φαίνεται από την περιγραφή ότι στα tables αυτά δημιουργούνται οι παρακάτω σχέσεις. Το table AspNetUsers έχει σχέση Many to Many με αυτό των Projects. Για αυτό το λόγο το Entity Framework έχει δημιουργήσει ένα επιπλέον table ApplicationUserProjects, το οποίο ενώνει τα προηγούμενα δυο. Έχει επίσης και σχέση Many to One με το table των ProjectTasks. Ο πίνακας Projects έχει επίσης Many to One σχέση με αυτόν των ProjectTasks. Τέλος ο πίνακας AspNetUsers έχει σχέση Many to Many με αυτόν των AspNetRoles, καθώς κάθε χρήστης μπορεί να έχει παραπάνω από ένα ρόλο. Πάλι το Entity Framework, έχει δημιουργήσει έναν ενδιάμεσο πίνακα AspNetUserRoles, για να ενώσει τους προηγούμενους δυο.

Παρακάτω φαίνεται η υλοποίηση των παραπάνω πινάκων καθώς και μια μικρή εξήγηση για τον καθένα.

Update | Script File: dbo.AspNetUsers.sql

Name	Data Type	Allow Nulls	Default
Id	nvarchar(128)	☐	
Email	nvarchar(256)	☑	
EmailConfirmed	bit	☐	
PasswordHash	nvarchar(MAX)	☑	
SecurityStamp	nvarchar(MAX)	☑	
PhoneNumber	nvarchar(MAX)	☑	
PhoneNumberConfirmed	bit	☐	
TwoFactorEnabled	bit	☐	
LockoutEndDateUtc	datetime	☑	
LockoutEnabled	bit	☐	
AccessFailedCount	int	☐	
UserName	nvarchar(256)	☐	
FirstName	nvarchar(100)	☐	('')
LastName	nvarchar(100)	☐	('')
HomeAddress	nvarchar(100)	☐	('')
HireDate	datetime	☐	('1900-01-01T00:00:00.000')

Εικόνα 7: Πίνακας AspNetUsers

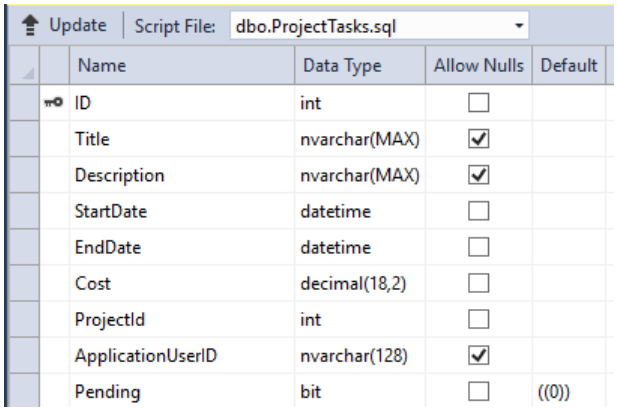
Ο πίνακας AspNetUsers (Εικόνα 7), που χρησιμοποιήθηκε για να εγγράφονται οι employees ως users της εφαρμογής, κρατήθηκε σε μεγάλο κομμάτι όπως ήταν δημιουργημένος από το ίδιο το Identity Framework. Τα μόνα columns που έχουν δημιουργηθεί μετά είναι τα FirstName, LastName, HomeAddress και HireDate. Επίσης αυτό που δεν φαίνεται στην περιγραφή αυτή είναι και τα foreign keys που δείχνουν στους projects και τα project tasks.

Update | Script File: dbo.Projects.sql

Name	Data Type	Allow Nulls
ID	int	☐
Title	nvarchar(MAX)	☑
Description	nvarchar(MAX)	☑
StartDate	datetime	☐
EndDate	datetime	☐
Cost	decimal(18,2)	☐
ManagerId	nvarchar(MAX)	☑

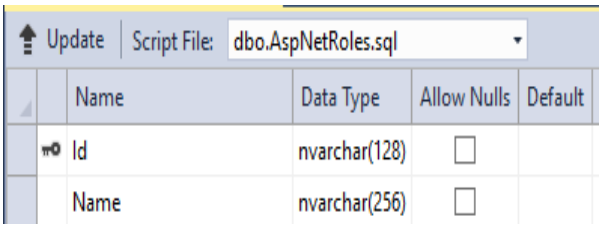
Εικόνα 8: Πίνακας Projects

Ο πίνακας Projects (Εικόνα 8) περιέχει τα στοιχεία των projects, όπως το όνομα και η περιγραφή του, πότε αρχίζει και πότε πρέπει να τελειώσει, καθώς και το Id του Project Manager.



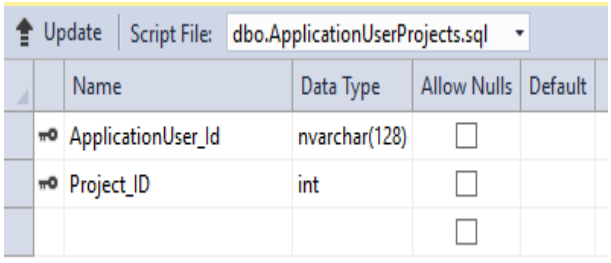
Εικόνα 9: Πίνακας ProjectTasks

Ο πίνακας ProjectTasks (Εικόνα 9) περιέχει τα στοιχεία τους, όπως το όνομα και η περιγραφή του, πότε αρχίζει και πότε πρέπει να τελειώσει, σε ποιο Project αναφέρεται σύμφωνα με το ProjectId που έχει, και ποιος user το έχει αναλάβει σύμφωνα με το ApplicationUserId. Έχει το status του task με το όνομα Pending, για να δείξει αν είναι τελειωμένο ή όχι.



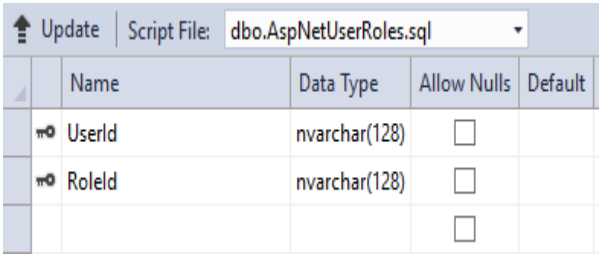
Εικόνα 10: Πίνακας AspNetRoles

Ο πίνακας AspNetRoles (Εικόνα 10), περιέχει τους διαθέσιμους ρόλους που είναι να πάρει ο κάθε χρήστης, Κάθε ρόλος περιγράφεται από ένα Id και ένα όνομα.



Εικόνα 11: Πίνακας ApplicationUserProjects

Ο πίνακας ApplicationUserProjects (Εικόνα 11), είναι ο ενδιάμεσος πίνακας που έχουν οι Users, με τα Projects. Αυτός δημιουργείται αυτόματα από το Entity Framework.

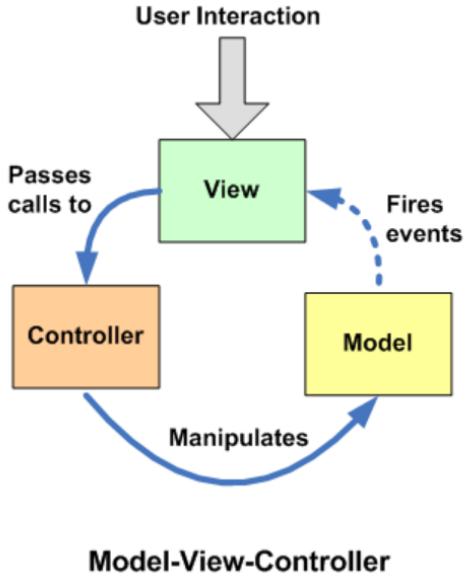


Εικόνα 12: Πίνακας AspNetUserRoles

Τέλος, ο πίνακας AspNetUserRoles (Εικόνα 12), είναι άλλος ένας πίνακας που δημιουργείται πάλι από το Entity Framework για να ενώσει τους πίνακες των Users και των Roles.

3.3 MVC (Model View Controller) Framework

Βασιζόμενο στο MVC Pattern, το οποίο είναι ένα μοντέλο αρχιτεκτονικής λογισμικού, το ASP.NET δημιουργεί τη λειτουργικότητα της πλατφόρμας. Το pattern αυτό χωρίζει τη βασική λειτουργικότητα σε τρία κομμάτια, αυτό του Model, του View και του Controller. Αυτά τα τρία κομμάτια συνεργάζονται μεταξύ τους για να πάρουν το input από τον χρήστη και να παράγουν κάποιο response σε αυτόν. Η Εικόνα 13 δείχνει μια γραφική αναπαράσταση του πως συνεργάζονται τα τρία αυτά στοιχεία[ii].



Εικόνα 13: Αναπαράσταση του MVC Pattern.

Ουσιαστικά ο τρόπος λειτουργίας ενός τέτοιου συστήματος είναι ως εξής:

- Ο χρήστης βλέπει το interface μιας πλατφόρμας μέσω του View. Όλο το interaction, τα κουμπιά, οι φόρμες κ.α. βρίσκονται μέσα σε αυτό.
- Όταν ο χρήστης πατήσει κάποιο από τα links ή τα κουμπιά, πληροφορία για το τι πατήθηκε, που ουσιαστικά σημαίνει το τι θέλει να κάνει ο χρήστης, στέλνεται στους αντίστοιχους Controllers που είναι συνδεδεμένοι με το αντίστοιχο interface.
- Οι Controllers επεξεργάζονται την πληροφορία που έστειλε ο χρήστης, και στη συνέχεια επεξεργάζονται τα Models που υπάρχουν σε αυτή την εφαρμογή. Τα Models αυτά, είναι η βάση δεδομένων και ότι άλλες κλάσεις χρειάζονται για τη λειτουργία της πλατφόρμας.
- Τέλος, μόλις ενημερωθούν και τα Models, ενημερώνεται το νέο View που βλέπει ο χρήστης, πάλι μέσω των αντίστοιχων Controllers που στέλνουν την απαραίτητη πληροφορία.

Η πλατφόρμα διαχείρισης προσωπικού φτιάχτηκε με το ASP.NET MVC μοντέλο, το οποίο λειτουργεί και αυτό με αντίστοιχο τρόπο.

3.3.1 Controllers

Οι Controllers σε ένα Web Application φτιαγμένο μέσω του ASP.NET, είναι οι ουσιαστικά οι διαχειριστές όλου του interface. Είναι ο συνδετικός κρίκος μεταξύ των Models και των Views που θα αναλυθούν παρακάτω.

Ο τρόπος λειτουργίας τους είναι λίγο διαφορετικός από τον συνηθισμένο. Όταν μπαίνει κάποιος σε μια σελίδα, ή γενικά αλληλοεπιδρά με αυτή, ένα URL στέλνεται ως πληροφορία για ποια σελίδα πρέπει να φορτωθεί μετά στον browser. Μέσα σε αυτό το URL υπάρχουν και παραπάνω πληροφορίες για το τι αρχεία πρέπει να φορτωθούν, με τι παραμέτρους κ.α.

Σε μια εφαρμογή υλοποιημένη μέσω του ASP.NET MVC όμως το URL δεν συνδέεται με κάποια σελίδα ή κάποιο αρχείο. Το URL αντιστοιχεί σε ένα action ενός Controller, και μέσα σε αυτό βρίσκεται η πληροφορία για ποιο Controller θέλουμε να καλέσουμε και με τι παραμέτρους θέλουμε να τον καλέσουμε. Αυτή η σύνδεση μεταξύ του URL και των Controllers ονομάζεται ASP.NET Routing, ή απλά Routing. Στην Εικόνα 14 φαίνεται και στο ίδιο το application πως ορίζεται το Routing της εφαρμογής, από το ίδιο το Framework[iv].

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.Web.Mvc;
6   using System.Web.Routing;
7
8   namespace MvcEmployeeManager
9   {
10      public class RouteConfig
11      {
12          public static void RegisterRoutes(RouteCollection routes)
13          {
14              routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
15
16              routes.MapRoute(
17                  name: "Default",
18                  url: "{controller}/{action}/{id}",
19                  defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
20              );
21          }
22      }
23  }
24
```

Εικόνα 14: Default Routing μέσα σε μια εφαρμογή ASP.NET MVC.

Όπως φαίνεται και στην εικόνα, με την εντολή MapRoute, γίνεται ένα default Routing μεταξύ του URL και των Controllers που πρέπει να καλούνται. Αρχικά ορίζει το όνομα αυτού του Route με το όρισμα name, ενώ στη συνέχεια ορίζει το πως πρέπει

να συνταχθεί ένα URL σωστά, για να το μεταφράσει όπως πρέπει και να καλέσει τον αντίστοιχο Controller[iii].

Πιο συγκεκριμένα, το όρισμα url ορίζει πως μετά το domain του application μας, πρέπει να ακολουθεί το όνομα της κλάσης του Controller που θέλουμε να καλέσουμε, μετά το όνομα του action που θέλουμε να κάνουμε και τέλος το όρισμα που θέλουμε να περάσουμε για αυτό το action, το οποίο ανάλογα με το action μπορεί να είναι προαιρετικό. Η διαφοροποίηση ανάμεσα σε κλάση του Controller και το action γίνεται επειδή άλλον Controller θα έχει η λίστα με τα Project Tasks για παράδειγμα, και άλλον η λίστα με τα Projects και οι δύο όμως μπορεί να έχουν το ίδιο όνομα action, όπως φαίνεται στην Εικόνα 15 και Εικόνα 16.

```
namespace MvcEmployeeManager.Controllers
{
    public class EmployeesController : Controller
    {
        private ApplicationDbContext db = new ApplicationDbContext();

        // GET: Employees
        public ActionResult Index()
        {
            //exclude admins from list
            var users = db.Users.ToList();
            var roleManager = new RoleManager<IdentityRole>(new RoleStore<IdentityRole>(n
            var role = roleManager.FindByName("IsAdmin").Users.First();
            var admins = db.Users.Where(u => u.Roles.Select(r => r.RoleId).Contains(role.I

            return View(users.Except(admins).ToList());
        }
```

Εικόνα 15: Παράδειγμα από Controller

```
namespace MvcEmployeeManager.Controllers
{
    public class ProjectsController : Controller
    {
        private ApplicationDbContext db = new ApplicationDbContext();

        // GET: Projects
        public ActionResult Index()
        {
            return View(db.Projects.ToList());
        }
```

Εικόνα 16: Παράδειγμα από action του Controller

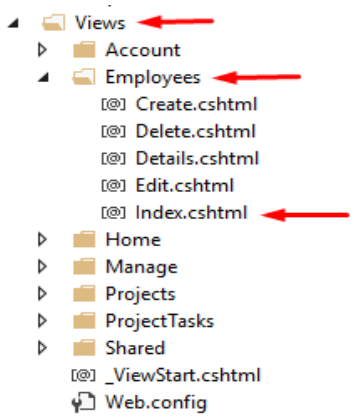
Είναι φανερό ότι έχουμε δυο διαφορετικές κλάσεις από Controllers, με το ίδιο όνομα action, τα οποία όμως κάνουν διαφορετικές λειτουργίες.

Μόλις λοιπόν, γίνει η κλήση σε κάποιον Controller και ενεργοποιηθεί κάποιο action του, αυτός με τη σειρά του καλεί τη βάση δεδομένων που υπάρχει στο Application, που όπως ορίσαμε πριν είναι το Model, μαζεύει όση πληροφορία είναι απαραίτητη σύμφωνα με ότι ζήτησε ο χρήστης, και επιστρέφει ένα ActionResult. Αυτό το ActionResult μπορεί να είναι μια κλήση σε κάποιο View της εφαρμογής, ή κάποια κλήση σε άλλο action του ίδιου ή ακόμα και άλλου Controller.

```
public class EmployeesController : Controller
{
    private ApplicationDbContext db = new ApplicationDbContext();

    // GET: Employees
    public ActionResult Index()
    {
        //exclude admins from list
        var users = db.Users.ToList();
        var roleManager = new RoleManager<IdentityRole>(new RoleStore<Identi
        var role = roleManager.FindByName("IsAdmin").Users.First();
        var admins = db.Users.Where(u => u.Roles.Select(r => r.RoleId).Conta

        return View(users.Except(admins).ToList());
    }
```



Εικόνα 17: Σχέση μεταξύ Controller action και View

Για να γίνει μια κλήση σε κάποιο View, το ASP.NET MVC ακολουθεί ένα συγκεκριμένο convention. Υπάρχουν δυο περιπτώσεις για μια τέτοια κλήση, ή να ορίσει ρητά ο προγραμματιστής το action, τον Controller και τα ορίσματα που θέλει να περάσει σε αυτόν, έτσι ώστε αυτός να καλέσει το View που χρειαζόμαστε, ή μέσω του naming convention που ορίζει ότι το όνομα του View πρέπει να είναι το ίδιο με το όνομα του action του Controller, όπως επίσης και ότι το View πρέπει να είναι μέσα σε φάκελο ο οποίος θα έχει το ίδιο όνομα με την κλάση του Controller αυτού. Για παράδειγμα, αν θέλουμε να βρεθούμε στη σελίδα με το όνομα Index η οποία αφορά τον EmployeeController, τότε θα πρέπει να καλέσουμε το action με όνομα Index που

βρίσκεται στην κλάση με όνομα EmployeeController. Στη συνέχεια θα πρέπει να υπάρχει ένας φάκελος με όνομα Employee, ο οποίος θα περιέχει ένα View μέσα του με το όνομα Index, όπως ακριβώς φαίνεται στην Εικόνα 17. Αν ισχύουν τα προηγούμενα, το ASP.NET είναι αρκετά έξυπνο για να κάνει την σύνδεση μεταξύ τους και να δρομολογήσει σωστά το request του χρήστη.

3.3.2 Views

Τα Views σε μια ASP.NET MVC εφαρμογή, είναι ουσιαστικά οι ιστοσελίδες που παρουσιάζονται στον χρήστη για να αλληλοεπιδράσει με αυτές. Ένα View περιέχει ένα συντακτικό που η Microsoft ονομάζει Razor. Το συντακτικό αυτό είναι υπεύθυνο για την παρουσίαση της σελίδας στον χρήστη.

Το Razor περιέχει ένα συνδυασμό από HTML, JQuery και C#, έτσι ώστε να μπορεί εύκολα ο προγραμματιστής να διαμορφώσει όπως θέλει αυτός τη σελίδα, και ταυτόχρονα να πάρει την πληροφορία που θέλει από τον Controller πολύ απλά χρησιμοποιώντας C#. Τα αρχεία Views παράγονται ως αρχεία .cshtml[vii].

Η default γλώσσα στο Razor syntax είναι η HTML οπότε μπορούμε εύκολα να χρησιμοποιήσουμε ότι ξέρουμε από αυτή τη γλώσσα. Αν όμως θέλει ο προγραμματιστής να χρησιμοποιήσει C# το μόνο που χρειάζεται να κάνει είναι να βάλει το σύμβολο @ μπροστά από μια έκφραση και να γράψει την λογική που θέλει. Το Razor syntax μάλιστα, υποστηρίζει πλήρως και if statements καθώς και loops, αλλά και απλό κώδικα σε C#. Στην Εικόνα 18 φαίνονται διάφορα παραδείγματα χρήσης του. Όπως φαίνεται στα παραδείγματα, με αυτή τη σύνταξη, μπορεί ο προγραμματιστής εύκολα να συνδυάσει λογική C# με HTML tags για να έχει interactivity η σελίδα του[v].

```
@if (value % 2 == 0)
{
    <p>The value was even.</p>
}
else if (value >= 1337)
{
    <p>The value is large.</p>
}
else
{
    <p>The value is odd and small.</p>
}
```

```
@{
    var people = new Person[]
    {
        new Person("Weston", 33),
        new Person("Johnathon", 41),
        ...
    };
}

 @for (var i = 0; i < people.Length; i++)
 {
     var person = people[i];
     <p>Name: @person.Name</p>
     <p>Age: @person.Age</p>
 }
```

Εικόνα 18: Παραδείγματα του Razor Syntax σε διάφορες περιπτώσεις.

Για να πάρει την πληροφορία από τα models όμως, αν αυτό χρειαστεί, πρέπει ο αντίστοιχος Controller να του στείλει κάπως τα απαραίτητα αντικείμενα. Η εντολή View() που καλείται από το action μπορεί να πάρει και όρισμα το αντικείμενο που θέλουμε να περάσουμε στο View αρκεί να ξέρουμε τι τύπο αντικείμενο είναι, καθώς το View πρέπει να ξέρει τι θα πάρει. Όταν δημιουργούμε ένα αρχείο View, στην αρχή του πρέπει να ορίσουμε τι τύπος θα είναι το model που θα πάρουμε. Στην Εικόνα 19, φαίνεται ακριβώς πως γίνεται αυτή η σύνδεση μεταξύ του action και του View.

```
// GET: Projects/Details/5
public ActionResult Details(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Project project = db.Projects.Find(id);
    if (project == null)
    {
        return HttpNotFound();
    }

    var projectEmployees = project.Employees;
    var projectTasks = project.Tasks;
    var projectIndexData = new ProjectIndexData
    {
        Project = project,
        Tasks = projectTasks,
        Employees = projectEmployees
    };

    return View(projectIndexData);
}
```

(local variable) ProjectIndexData projectIndexData

```
1
2
3   @model MvcEmployeeManager.Models.ProjectIndexData
4
5   @using Microsoft.AspNet.Identity;
6
7   @{
8       ViewBag.Title = "Details";
9   }
10
11  <div>
12      <h3>Project Details</h3>
13      <hr />
14      <dl class="dl-horizontal">...</dl>
56  </div>
57  <p>
58      @if (Model.Project.ManagerId == User.Identity.GetUserId())
59      {
60          @Html.ActionLink("Edit", "Edit", new { id = Model.Project.ID }) <span>|</span>
61      }
62
63      @Html.ActionLink("Back to List", "Index")
64  </p>
```

Εικόνα 19: Σχέση μεταξύ του action με όνομα Details και του αντίστοιχου View.

Βλέπουμε στην παραπάνω εικόνα πως αρχικά το action δημιουργεί ένα αντικείμενο τύπου ProjectIndexData. Στη συνέχεια καλούμε το View() περνώντας το αντικείμενο αυτό στο View μας, δίνοντας του έτσι όλη την πληροφορία που χρειάζεται για να παράγει την τελική σελίδα. Το View από τη μεριά του πρέπει να ξέρει τι είδους αντικείμενο θα πάρει από τον Controller. Αυτό το κάνει με την εντολή

```
@model MvcEmployeeManager.Models.ProjectIndexData
```

η οποία ορίζει ποιο ακριβώς θα είναι το model που θα χρησιμοποιήσει. Ταυτόχρονα δημιουργεί μια μεταβλητή με όνομα Model η οποία αποθηκεύει το αντικείμενο που έστειλε ο Controller.

3.3.3 Models

Τα Models σε μια εφαρμογή τύπου MVC, αποτελούν όλες τις κλάσεις τις οποίες χρησιμοποιεί η εφαρμογή μας για να περάσει οποιαδήποτε πληροφορία, από ένα action σε ένα view ή ακόμα και σε άλλο action. Αυτά τα αντικείμενα μπορεί να είναι πληροφορία από τη βάση δεδομένων της εφαρμογής, ή ακόμα και απλά αντικείμενα που περιέχουν άλλα αντικείμενα από απλές κλάσεις.

Για τα Models σε ένα ASP.NET web application, χρησιμοποιείται πλέον το Entity Framework, το οποίο δημιουργεί τη βάση δεδομένων της εφαρμογής. Το Framework αυτό έχει δυο διαφορετικά workflows. Αυτά είναι το Database First και

το Code First. Στο Database First ο προγραμματιστής είναι υπεύθυνος στο να δημιουργήσει και να συνδέσει τα tables της βάσης δεδομένων της εφαρμογής. Στο Code First αντιθέτως, ο προγραμματιστής χρειάζεται να ορίσει μόνο τις κλάσεις με C# όπως αυτός θέλει να είναι στη βάση δεδομένων, και μετά να ορίσει explicitly ότι αυτές πρέπει να μεταφραστούν σε tables. Την μετάφραση αυτή την κάνει μόνο του το Entity Framework[vi].

Για την υλοποίηση της πλατφόρμας χρησιμοποιήθηκε το Code First workflow, καθώς το Entity Framework μπορεί αρκετά εύκολα να δημιουργήσει τα tables και τις μεταξύ τους συνδέσεις. Στις παρακάτω εικόνες φαίνεται ένα παράδειγμα μιας κλάσης που ορίστηκε ότι θα είναι στη βάση δεδομένων μας.

```
namespace MvcEmployeeManager.Models
{
    public class ProjectTask
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }

        [DisplayFormat(DataFormatString = "{0:dd-MM-yyyy}", ApplyFormatInEditMode = true)]
        public DateTime StartDate { get; set; }

        [DisplayFormat(DataFormatString = "{0:dd-MM-yyyy}", ApplyFormatInEditMode = true)]
        public DateTime EndDate { get; set; }

        public decimal Cost { get; set; }
        public bool Pending { get; set; }

        public string ApplicationUserID { get; set; } //employee id
        public int ProjectID { get; set; }

        public virtual ApplicationUser Employee { get; set; }
        public virtual Project Project { get; set; }
    }
}
```

Εικόνα 20: Ορισμός της κλάσης ProjectTask ως Model.

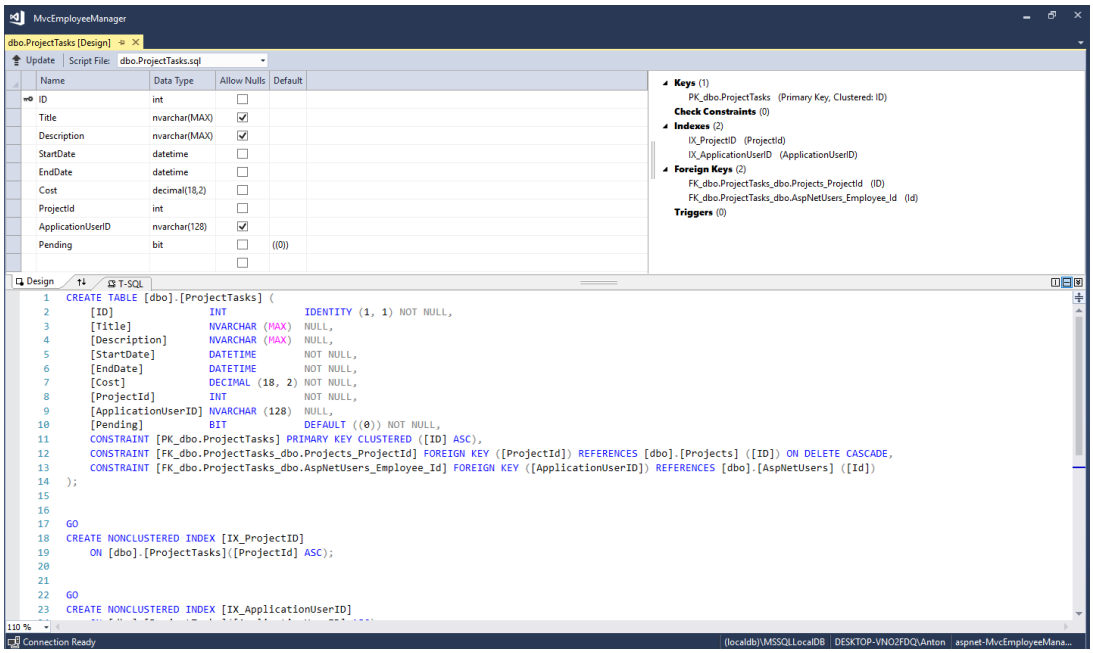
```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext()
        : base("DefaultConnection", throwIfV1Schema: false)
    {
    }

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

    public System.Data.Entity.DbSet<MvcEmployeeManager.Models.Project> Projects { get; set; }

    public System.Data.Entity.DbSet<MvcEmployeeManager.Models.ProjectTask> ProjectTasks { get; set; }
}
```

Εικόνα 21: Δήλωση της κλάσης ProjectTasks ως μέλος της βάσης δεδομένων.



Εικόνα 22: Το τελικό table του ProjectTasks στη βάση δεδομένων.

Μέσω της Code First προσέγγισης γίνεται αρκετά εύκολη η δουλειά των προγραμματιστών, που δεν θέλουν να κάνουν από την αρχή όλα τα tables της βάσης δεδομένων μαζί με όλα τα constraints τους. Πλέον μπορούν απλά να γράφουν μια κλάση και μέσω του Entity Framework να μεταφραστούν σε tables της βάσης[ix].

Το Entity Framework όμως, απαιτεί άλλο ένα βήμα για να δημιουργήσει το table και αυτό είναι το λεγόμενο migration. Συγκεκριμένα χρησιμοποιήθηκαν τα Code First Migrations, εφόσον χρησιμοποιήσαμε την Code First προσέγγιση. Τα migrations χρησιμοποιούνται όταν θέλει ο προγραμματιστής να αλλάξει κάπως την βάση δεδομένων, είτε αλλάζοντας κάποια από τα στοιχεία σε κάποιο υπάρχων table, είτε προσθέτοντας ένα καινούργιο, ή ακόμα και να κάνει populate κάποιον πίνακα με κάποιες τιμές[viii].

Για να προσθέσει ένα νέο Migration ο προγραμματιστής πρέπει να πάει στο Package Manager Console του Visual Studio και να γράψει την εντολή:

add-migration "όνομα του migration χωρίς τα quotes"

Στη συνέχεια δημιουργείται ένα καινούργιο C# script που αυτό δημιουργεί ή αλλάζει τα tables της βάσης. Η επόμενη εντολή που πρέπει να γράψει μετά, για να εφαρμοστούν οι αλλαγές, είναι η:

update-database [xiii]

Αυτή η εντολή, εφαρμόζει τις αλλαγές από το τελευταίο pending migration που έχει δημιουργήσει ο προγραμματιστής[xi].

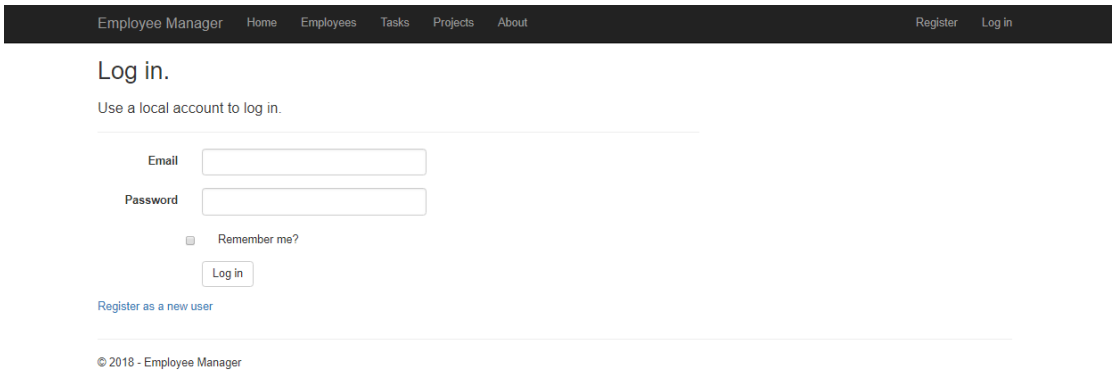
Τα migrations λειτουργούν επίσης ως και ένα απλό version control της βάσης, καθώς ο προγραμματιστής μπορεί να επιλέξει να κάνει rollback τη βάση σε προηγούμενο migration. Με αυτή τη λογική, το C# script του migration έχει δυο συναρτήσεις την Up() και την Down(). Η Up() είναι υπεύθυνη για την ανανέωση της βάσης, ενώ η Down() είναι υπεύθυνη για το rollback. Ένα παράδειγμα ενός migration φαίνεται στηνΕικόνα 23, όπου προστίθεται ένα νέο column στον πίνακα Projects με το όνομα ManagerId[x].

```
1  namespace MvcEmployeeManager.Migrations
2  {
3      using System;
4      using System.Data.Entity.Migrations;
5
6      public partial class ProjectFix : DbMigration
7      {
8          public override void Up()
9          {
10             AlterColumn("dbo.Projects", "ManagerId", c => c.String());
11         }
12
13         public override void Down()
14         {
15             AlterColumn("dbo.Projects", "ManagerId", c => c.Int(nullable: false));
16         }
17     }
18 }
```

Εικόνα 23: Παράδειγμα ενός migration.

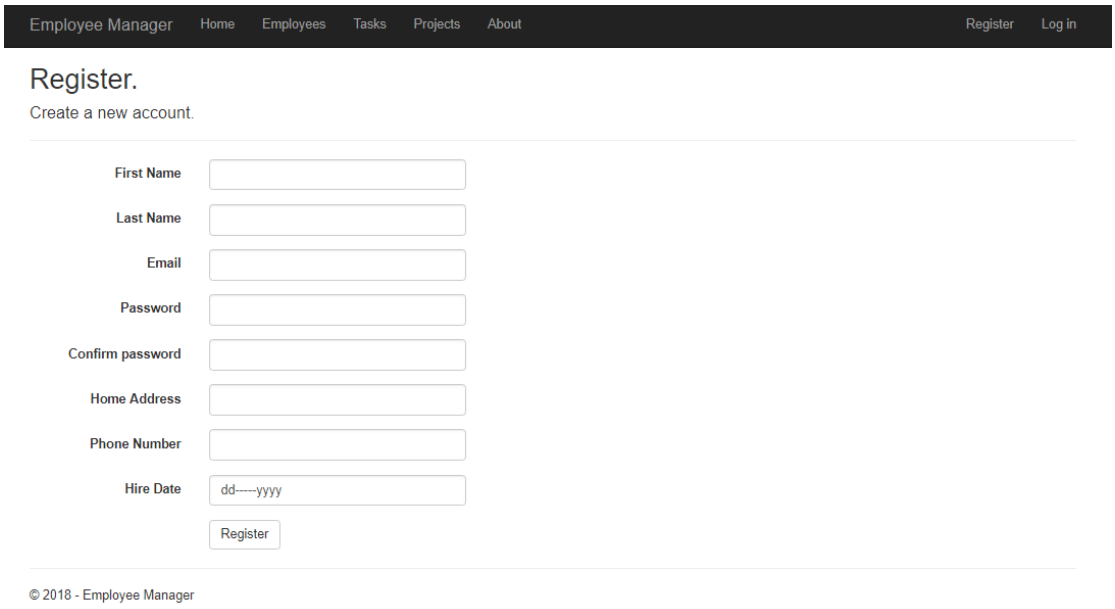
3.4 Interface της εφαρμογής

Το βασικό interface της πλατφόρμας διαχείρισης προσωπικού, δημιουργήθηκε με τρόπο που είναι αρκετά απλό για τον χρήστη να πλοηγηθεί. Με την είσοδο του χρήστη στη σελίδα, του ζητείται να συνδεθεί με το λογαριασμό του, όπως φαίνεται στην Εικόνα 24[xiv].



Εικόνα 24: Είσοδος χρήστη στην πλατφόρμα.

Αν ο χρήστης δεν έχει λογαριασμό στην πλατφόρμα, μπορεί εύκολα να δημιουργήσει ένα καινούργιο πατώντας στο κουμπί Register πάνω δεξιά, ή το κουμπί Register as a new user. Η φόρμα για τη δημιουργία ενός νέου χρήστη φαίνεται παρακάτω.



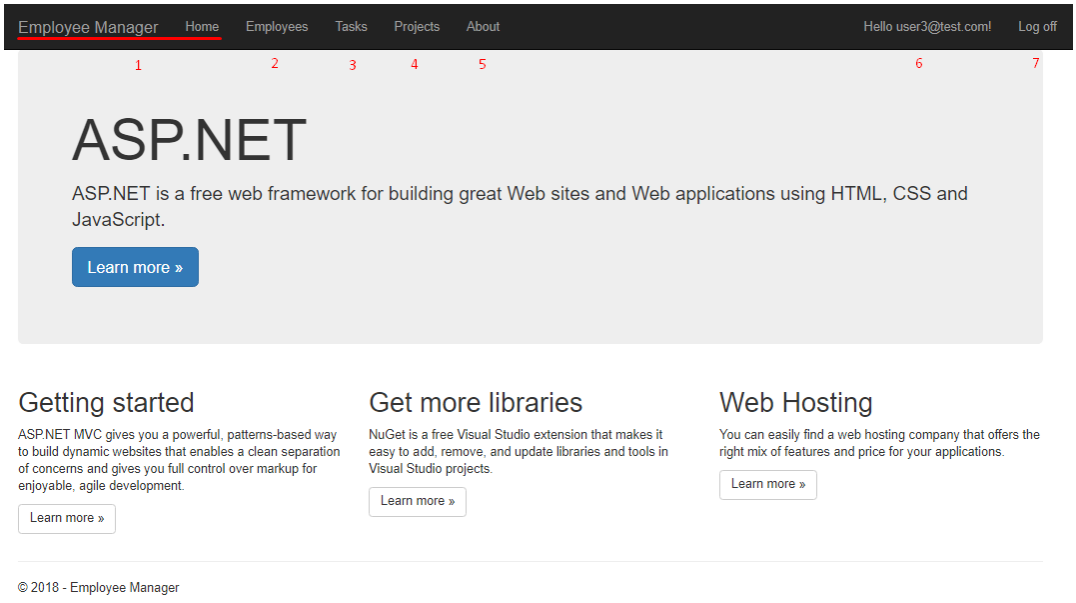
Εικόνα 25: Εγγραφή νέου χρήστη στην πλατφόρμα.

Με το που εισέρχεται ο χρήστης στο σύστημα, έχει 7 επιλογές:

- Να πάει στην κεντρική σελίδα της πλατφόρμας
- Να δει τη λίστα με όλους τους υπόλοιπους χρήστες της πλατφόρμας
- Να δει όλα τα tasks που υπάρχουν γραμμένα στη πλατφόρμα
- Να δει όλα τα projects που υπάρχουν γραμμένα στη πλατφόρμα

- Να δει περισσότερες πληροφορίες που αφορούν τη σελίδα (About)
- Να δει περισσότερες πληροφορίες για τον λογαριασμό του
- Να αποσυνδεθεί

Στην Εικόνα 26 φαίνονται και οι επιλογές πάνω στη πλατφόρμα.

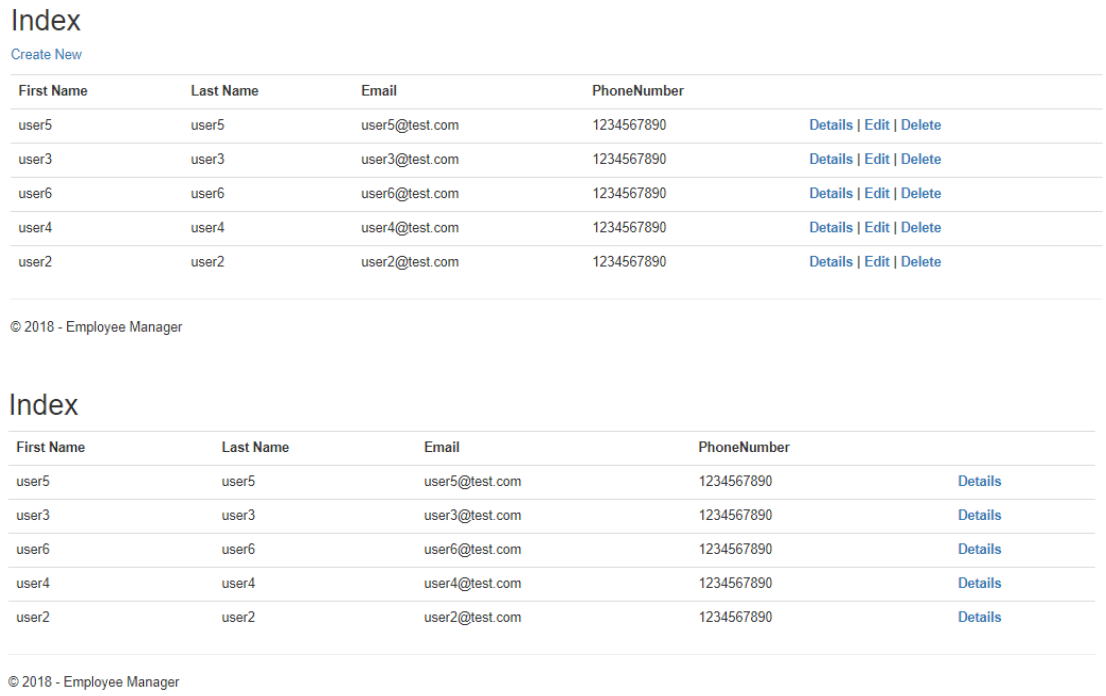


Εικόνα 26: Επιλογές του χρήστη στην πλατφόρμα.

3.4.1 Λίστα με τους εργαζομένους

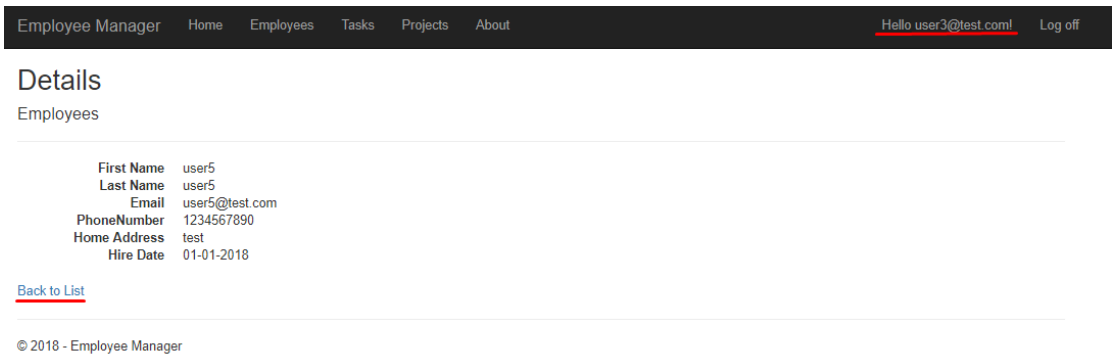
Η επιλογή Employees, ανακατευθύνει τον χρήστη σε μια λίστα που φαίνονται όλοι οι χρήστες του συστήματος. Αυτή η λίστα έχει στήλες με τα ονοματεπώνυμα των χρηστών, το email τους, το τηλέφωνό τους και η τελευταία στήλη έχει διαθέσιμες κάποιες λειτουργίες. Για τους απλούς χρήστες, η μόνη λειτουργία που έχουν διαθέσιμη είναι η Details που δίνει περισσότερες πληροφορίες για τον χρήστη που θέλουν να δουν. Για τους administrators, εμφανίζονται άλλες δύο επιλογές, το Edit και το Delete, που αλλάζουν τα στοιχεία και διαγράφουν έναν χρήστη αντίστοιχα. Επίσης, στους administrators εμφανίζεται η επιλογή Create New για να έχουν τη δυνατότητα να δημιουργήσουν νέο χρήστη γρήγορα. Μια μικρή λεπτομέρεια είναι ότι οι administrators δεν εμφανίζονται στη παραπάνω λίστα. Αυτό γίνεται γιατί ο ρόλος ενός administrator είναι μόνο για να διαχειρίζεται τη σελίδα και όχι να είναι ενεργό μέλος της. Αυτό σημαίνει πως ο χρήστης με τον ρόλο admin πρέπει να δηλωθεί στη

βάση manually και όχι μέσω της πλατφόρμας. Στην Εικόνα 27 φαίνονται οι διαφορές στη σελίδα ανάμεσα σε administrators και απλών χρηστών.

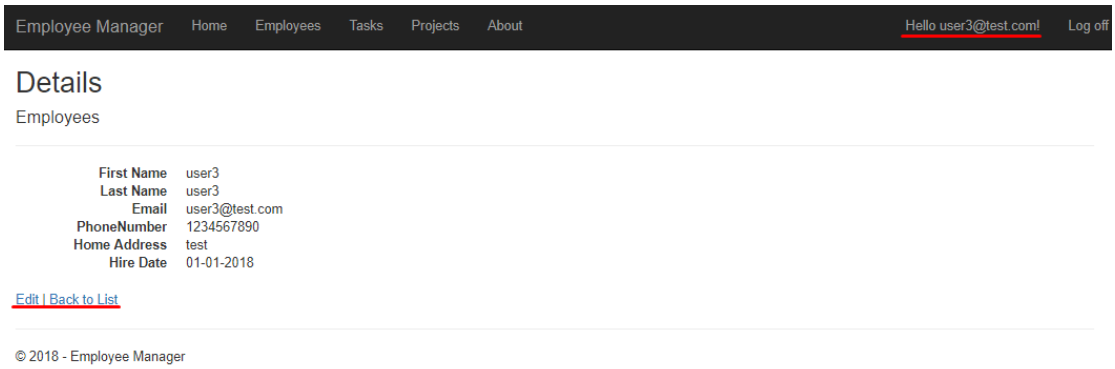


Εικόνα 27: Διαφορές στη λίστα των εργαζομένων ανάμεσα σε administrators και απλών users.

Αν ο χρήστης πατήσει την επιλογή Details, τότε μπορεί να δει περισσότερα στοιχεία για τον συγκεκριμένο χρήστη που επέλεξε. Αν αυτός ο χρήστης που επέλεξε είναι ο ίδιος, τότε του εμφανίζεται μια επιπλέον επιλογή Edit, με την οποία μπορεί να αλλάξει όποια στοιχεία θέλει. Οι διαφορές φαίνονται στην παρακάτω εικόνα.

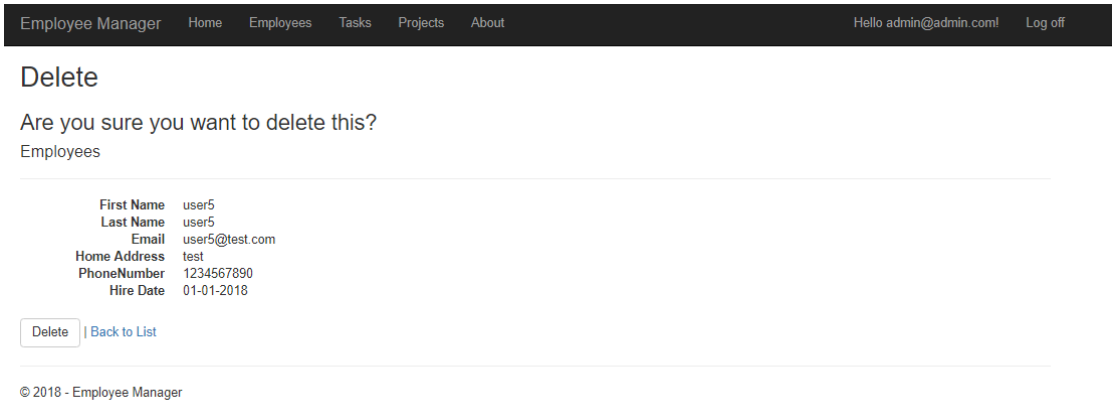


Εικόνα 28: Διαφορές στην επιλογή Details των εργαζομένων (α).



Εικόνα 29: Διαφορές στην επιλογή Details των εργαζομένων (β).

Η επιλογή Edit που εμφανίζεται στους admins, τους δίνει τη δυνατότητα να αλλάξουν τα στοιχεία όποιου χρήστη θέλουν. Η επιλογή Delete που εμφανίζεται πάλι μόνο στους admins, τους επιτρέπει να διαγράψουν κάποιο χρήστη από τη λίστα. Πριν τον διαγράψουν, παραπέμπονται σε μια σελίδα επιβεβαίωσης όπως η παρακάτω.

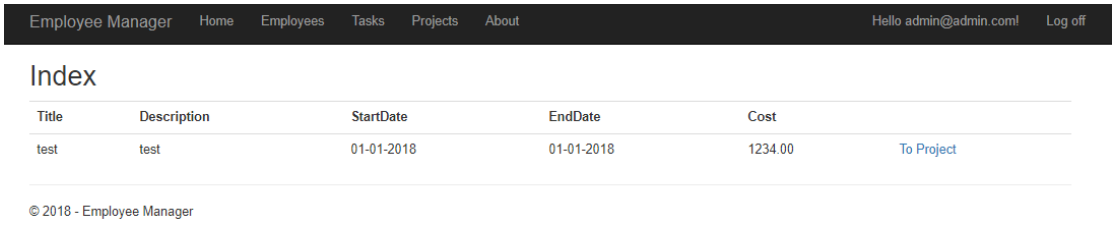


Εικόνα 30: Σελίδα επιβεβαίωσης διαγραφής ενός χρήστη.

Τέλος η επιλογή Create New για τους Employees, ανακατευθύνει τον administrator στη σελίδα Register.

3.4.2 Λίστα με τα Tasks

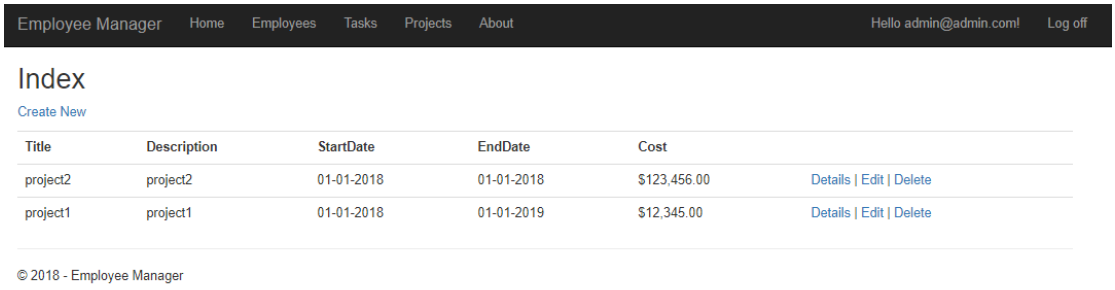
Μόλις ο χρήστης πατήσει την επιλογή Tasks, του παρουσιάζονται όλα τα tasks που έχουν δημιουργηθεί σε projects. Σε αυτή τη λίστα παρουσιάζονται οι στήλες με τον τίτλο, την περιγραφή, πότε αρχίζει και πότε τελειώνει η προθεσμία του και πόσο κοστίζει. Επίσης, υπάρχει η επιλογή To Project που σε ανακατευθύνει στο project για το οποίο ισχύει αυτό το task.



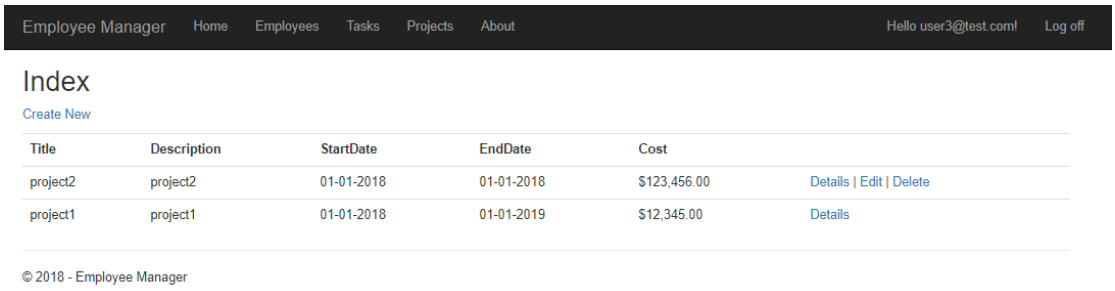
Εικόνα 31: Λίστα με τα tasks που υπάρχουν.

3.4.3 Λίστα με τα Projects

Πατώντας την επιλογή Projects, ο χρήστης ανακατευθύνεται σε μια λίστα με όλα τα projects που έχουν δημιουργηθεί στη πλατφόρμα. Σε αυτή τη λίστα, παρουσιάζεται ο τίτλος, η περιγραφή, πότε αρχίζει και πότε τελειώνει η προθεσμία του και πόσο κοστίζει συνολικά το project. Επίσης δίνονται πάλι οι επιλογές Edit, Details, Delete και Create New. Στην πλατφόρμα αυτή έχει δοθεί η επιλογή να φτιάξει όποιος χρήστης θέλει ένα νέο project στο οποίο αυτόματα γίνεται και Project Manager. Αυτό μπορεί να αλλάξει στη συνέχεια και να αναλάβει άλλος χρήστης. Οι επιλογές Edit και Delete δίπλα από κάθε project εμφανίζονται μόνο αν ο χρήστης είναι ο Project Manager αυτού του project, ή αν είναι ένας από τους administrators. Αυτή η διαφοροποίηση, φαίνεται στις παρακάτω εικόνες. Στην Εικόνα 33 φαίνεται πως ο χρήστης είναι Project Manager στο project2, αλλά όχι στο project1



Εικόνα 32: Λίστα με τα projects όπως την βλέπουν οι administrators.

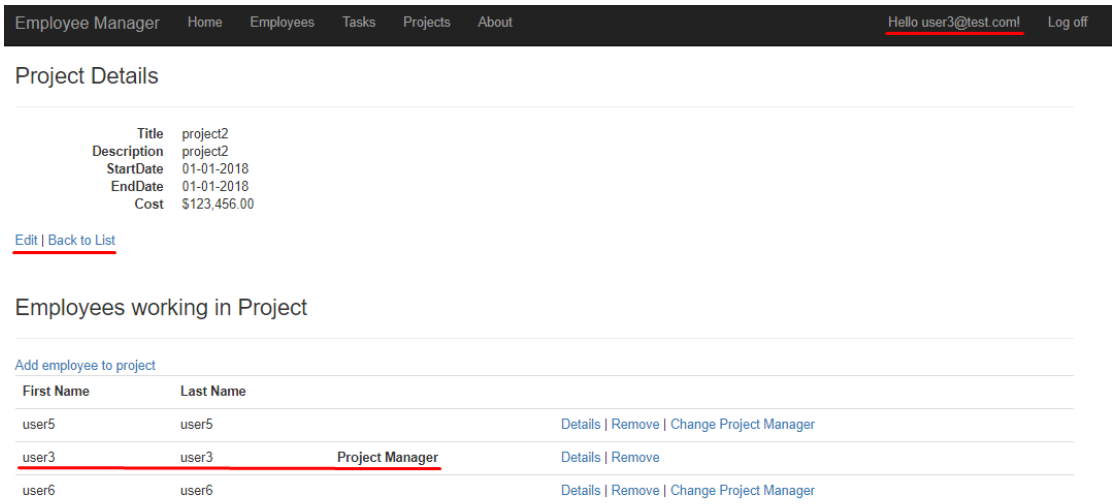


Εικόνα 33: Λίστα με τα projects όπως την βλέπουν οι χρήστες.

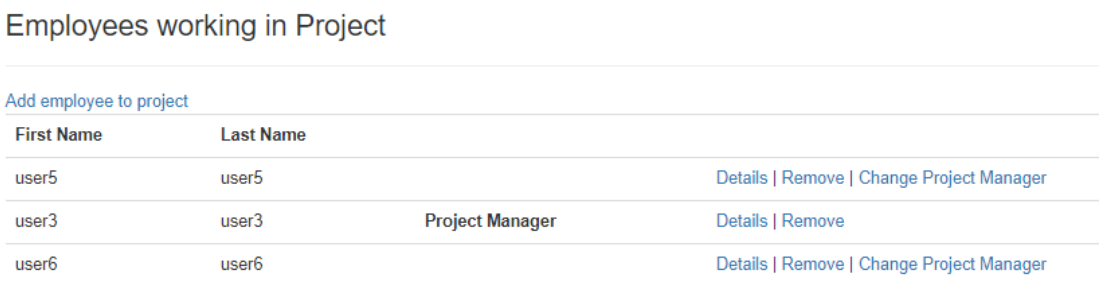
Με την επιλογή Details ο χρήστης μπορεί να δει περισσότερες λεπτομέρειες για το συγκεκριμένο project. Συγκεκριμένα, μπορεί να δει ποιοι άλλοι εργαζόμενοι δουλεύουν σε αυτό, ποιος είναι ο Project Manager και να δει ποια είναι τα tasks που υπάρχουν σε αυτό το project.

Αυτή η σελίδα, χωρίζεται σε τρία κομμάτια. Το πρώτο κομμάτι είναι οι πληροφορίες για το project. Κάτω από τις πληροφορίες, αν ο χρήστης που συνδέθηκε είναι και ο Project Manager του συγκεκριμένου project, τότε του δίνεται η δυνατότητα να κάνει edit τις πληροφορίες αν θέλει, όπως φαίνεται στην Εικόνα 34. Το ίδιο ισχύει και για τους administrators.

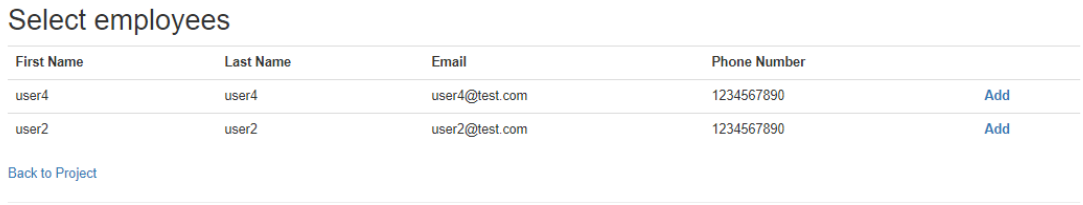
Το δεύτερο κομμάτι παρουσιάζει όσους εργαζόμενους δουλεύουν πάνω σε αυτό το project. Αν ο χρήστης δεν είναι ο Project Manager ή admin, τότε οι μόνες πληροφορίες που διαθέτει είναι το όνομα, το επώνυμο και τη δυνατότητα να μεταβεί στα Details του χρήστη που τον ενδιαφέρει. Αν είναι, τότε του παρουσιάζονται οι εξής επιπλέον δυνατότητες: Να προσθέσει νέο εργαζόμενο στο project, να βγάλει έναν εργαζόμενο από το project και τέλος να αλλάξει τον Project Manager. Προφανώς, μόλις αλλάξει ο Project Manager, ο προηγούμενος δεν θα βλέπει τις δυνατότητες αυτές πλέον. Αν πατήσει να προσθέσει καινούργιο εργαζόμενο στο project, του εμφανίζεται μια λίστα με όσους δεν βρίσκονται ήδη στο project, με την επιλογή να τους προσθέσει από δίπλα.



Εικόνα 34: Επιλογή Edit για τον Project Manager.



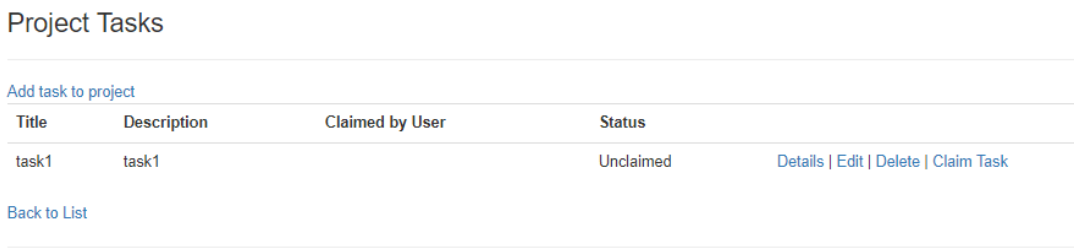
Εικόνα 35: Επιπλέον δυνατότητες του Project Manager.



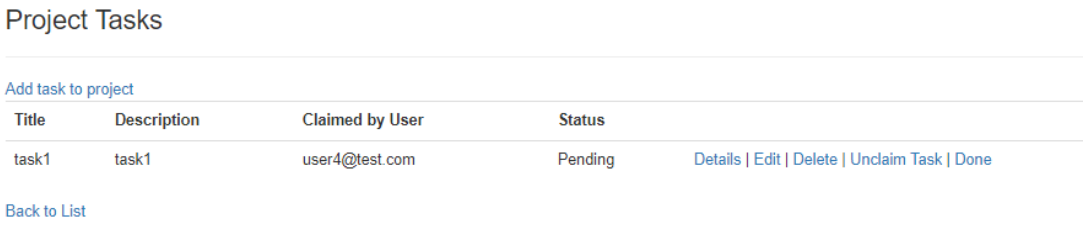
Εικόνα 36: Επιλογή εργαζομένων για να μπουν στο project.

Το δεύτερο κομμάτι παρουσιάζει όλα τα tasks που υπάρχουν για αυτό το project. Πάλι στη λίστα παρουσιάζονται κάποια βασικά στοιχεία για το task, καθώς και κάποιες λειτουργίες. Όπως και στα projects, έτσι και στα tasks οι απλοί χρήστες μπορούν να δουν details, ενώ οι admins και ο Project Manager μπορεί να κάνει Edit και Delete. Μόλις όμως δημιουργείται ένα task εμφανίζεται και η επιλογή Claim Task η οποία δίνει στον χρήστη τη δυνατότητα να αναλάβει αυτός το task.

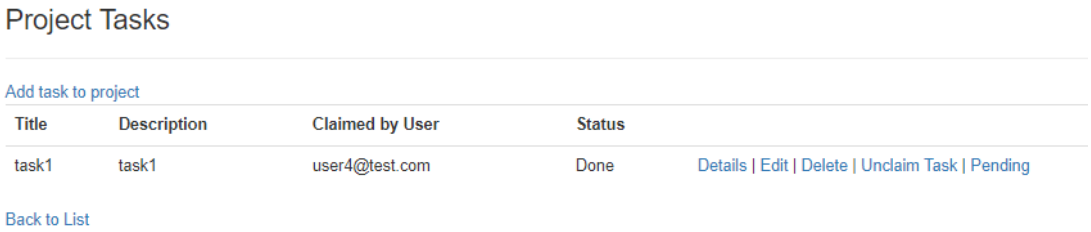
Υπάρχει επιπλέον και η στήλη Claimed by User, η οποία δείχνει αν κάποιος έχει αναλάβει αυτό το task και ποιος είναι. Μόλις ένας χρήστης αναλάβει κάποιο task, του εμφανίζεται επιπλέον η επιλογή να δηλώσει αν το έχει τελειώσει. Για να φαίνεται η κατάσταση του task, δηλαδή αν το έχει αναλάβει ένας χρήστης και αν έχει τελειώσει ή όχι, υπάρχει μια επιπλέον στήλη που δείχνει το status του task. Οι διαφορές ανάμεσα στις επιλογές φαίνονται στις παρακάτω εικόνες[xvi].



Εικόνα 37: Κατάσταση ενός task όταν δεν το έχει αναλάβει κάποιος χρήστης.



Εικόνα 38: Κατάσταση ενός task όταν το έχει αναλάβει κάποιος χρήστης, αλλά δεν το έχει τελειώσει ακόμα.

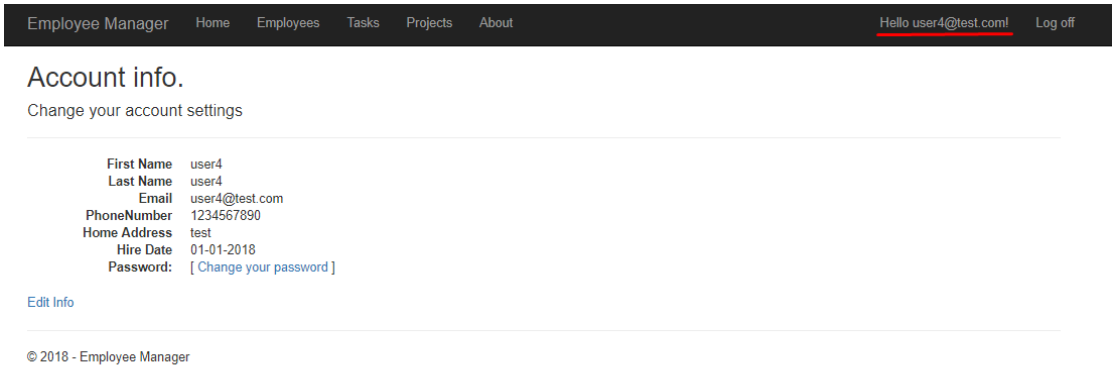


Εικόνα 39: Κατάσταση ενός task όταν το έχει αναλάβει κάποιος χρήστης και το έχει τελειώσει.

Με αυτά τα τρια κομμάτια στην επλογή Details του μενού Projects, κάθε χρήστης του συστήματος μπορεί να έχει πλήρη εικόνα για την κατάσταση ενός project.

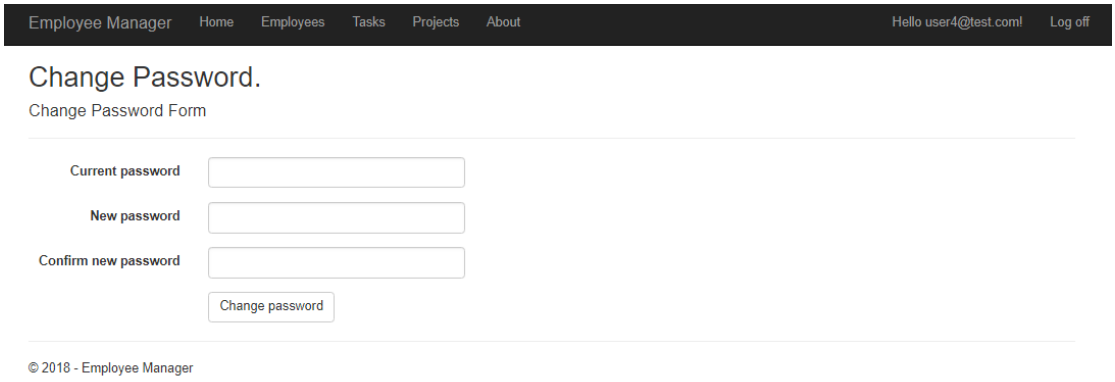
3.4.4 Ρυθμίσεις και πληροφορίες χρήστη

Αν ο χρήστης πατήσει πάνω δεξιά το όνομά του, όπως φαίνεται στην παρακάτω εικόνα, του εμφανίζονται οι πληροφορίες του λογαριασμού του στην πλατφόρμα. Επιπλέον, του δίνεται η δυνατότητα να αλλάξει τον κωδικό του λογαριασμού του.



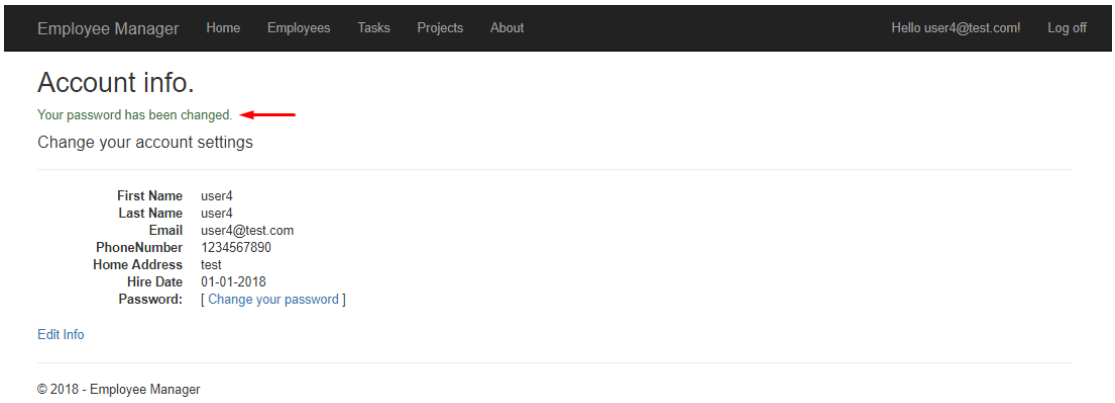
Εικόνα 40: Πληροφορίες του λογαριασμού του χρήστη.

Αν ο χρήστης επιλέξει να αλλάξει τον κωδικό του, ανακατευθύνεται σε μια νέα σελίδα, στην οποία καλείται να βάλει τον τωρινό του κωδικό, τον καινούργιο και να επιβεβαιώσει πάλι τον καινούργιο σε ένα ξεχωριστό text box. Η φόρμα αλλαγής κωδικού φαίνεται στην παρακάτω εικόνα.



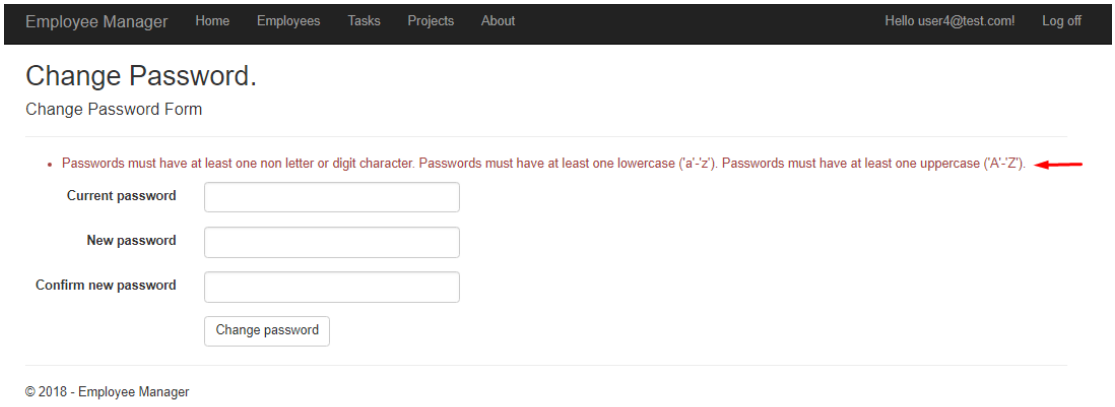
Εικόνα 41: Φόρμα αλλαγής κωδικού χρήστη.

Αν ο χρήστης αλλάξει τον κωδικό του τότε επιστρέφει πίσω στη προηγούμενη σελίδα με ένα μήνυμα επιτυχίας να εμφανίζεται, όπως στην παρακάτω εικόνα.



Εικόνα 42: Μήνυμα επιτυχούς αλλαγής κωδικού.

Αν ο χρήστης προσπαθήσει να αλλάξει τον κωδικό του αλλά για κάποιο λόγο, κάτι έχει γίνει λάθος, θα παραμείνει στη σελίδα και θα του εμφανιστεί ένα μήνυμα λάθους που θα του εξηγεί τι έχει κάνει λάθος. Στην παρακάτω εικόνα φαίνεται ένα παράδειγμα.



Εικόνα 43: Μήνυμα ανεπιτυχούς αλλαγής κωδικού.

Όπως φαίνεται στην Εικόνα 43, υπάρχουν κάποιοι κανόνες για το πως θα πρέπει να είναι ο κωδικός. Για λόγους ασφαλείας, οι κωδικοί πρέπει να έχουν τουλάχιστον ένα κεφαλαίο γράμμα, ένα μικρό, ένα σύμβολο και έναν αριθμό. Πρέπει επίσης να έχει μέγεθος τουλάχιστον 6 συμβόλων. Αυτή η ρύθμιση των πολιτικών ενός κωδικού, γίνεται πολύ εύκολα μέσω του ASP.NET Identity. Το Identity παρέχει μια βοηθητική κλάση που ονομάζεται PasswordValidator, μέσω του οποίου μπορούμε εύκολα να ρυθμίσουμε μια custom πολιτική για την πλατφόρμα, ή να χρησιμοποιήσουμε αυτή που δίνει by default το Framework. ΣτηνΕικόνα 44, φαίνεται πως ορίζονται οι πολιτικές αυτές και συγκεκριμένα, πως έχει οριστεί by default από το Identity[xv].

```
// Configure the application user manager used in this application. UserManager is defined in ASP.NET Identity and is used by the application.
public class ApplicationUserManager : UserManager<ApplicationUser>
{
    public ApplicationUserManager(IUserStore<ApplicationUser> store)
        : base(store)
    {
    }

    public static ApplicationUserManager Create(IdentityFactoryOptions<ApplicationUserManager> options, IOwinContext context)
    {
        var manager = new ApplicationUserManager(new UserStore<ApplicationUser>(context.Get<ApplicationDbContext>()));
        // Configure validation logic for usernames
        manager.UserValidator = new UserValidator<ApplicationUser>(manager)
        {
            AllowOnlyAlphanumericUserNames = false,
            RequireUniqueEmail = true
        };

        // Configure validation logic for passwords
        manager.PasswordValidator = new PasswordValidator
        {
            RequiredLength = 6,
            RequireNonLetterOrDigit = true,
            RequireDigit = true,
            RequireLowercase = true,
            RequireUppercase = true,
        };

        // Configure user lockout defaults
        manager.UserLockoutEnabledByDefault = true;
        manager.DefaultAccountLockoutTimeSpan = TimeSpan.FromMinutes(5);
        manager.MaxFailedAccessAttemptsBeforeLockout = 5;
```

Εικόνα 44: Ορισμός πολιτικών για τους κωδικούς των χρηστών, όπως έχει οριστεί από το Identity Framework.

Κεφάλαιο 4

4.1 Συμπεράσματα

Είμαστε σε μια εποχή που τα web applications γίνονται ολοένα και πιο περιζήτητα. Για αυτό είναι απαραίτητοι και οι web developers, οι οποίοι είναι ικανοί να παράγουν μια τέτοια εφαρμογή.

Καθώς όμως υπάρχουν πολλές διαδεδομένες γλώσσες προγραμματισμού, τις οποίες χρησιμοποιούν οι developers, υπήρξε η επιτακτική ανάγκη να παραχθούν εργαλεία τα οποία να μπορούσαν να επεκτείνουν τη λειτουργικότητα των γλωσσών αυτών. Χαρακτηριστικά παραδείγματα είναι για τη Javascript τα frameworks React, Angular και Node.js. Άλλο ένα χαρακτηριστικό παράδειγμα είναι και για την Python τα frameworks Django και TurboGears. Σκοπός αυτών των εργαλείων, είναι να βοηθούν τους προγραμματιστές να φτιάξουν εύκολα και πάνω απ' όλα γρήγορα web applications, σε μια γλώσσα που ήδη ξέρουν, τόσο στο front-end κομμάτι, που έχει να κάνει με το design μιας σελίδας, όσο και στο back-end κομμάτι, που είναι η βάση δεδομένων.

Η Microsoft με τη σειρά της, αποφάσισε να επεκτείνει μια δικιά της γλώσσα προγραμματισμού, τη C#, για να δώσει τη δυνατότητα σε όσους τη χρησιμοποιούν, να φτιάξουν ένα web application. Με το ASP.NET Core, μπορούν πλέον οι προγραμματιστές να φτιάξουν γρήγορα μια εφαρμογή χρησιμοποιώντας ένα αρκετά διαδεδομένο pattern, αυτό του Model View Controller (MVC). Το Core σε συνδυασμό με άλλα frameworks που παρουσιάστηκαν, συγκεκριμένα το Identity Framework για τη διαχείριση του membership system και το Entity Framework για τη διαχείριση του back-end, μπορεί να δώσει τεράστιες δυνατότητες στους developers, για να φτιάξουν από την πιο απλή εφαρμογή, μέχρι και την πιο περίπλοκη.

Γνωρίζοντας, λοιπόν, τα παραπάνω εργαλεία, ένας προγραμματιστής μπορεί να θεωρηθεί ως full-stack developer, χρησιμοποιώντας τη δύναμη και την ευκολία μιας πολύ δυνατής και ευέλικτης γλώσσας, της C#.

4.2 Προοπτικές Βελτίωσης

Με το ASP.NET Core οι προοπτικές βελτίωσης και επέκτασης της πλατφόρμας, είναι πραγματικά πολλές. Η δυνατότητα που παρέχει, να μπορεί ο προγραμματιστής να περνάει όποια βιβλιοθήκη θέλει στο front-end κομμάτι, σε συνδυασμό με την επιπλέον ευελιξία που δίνουν τα Entity και Identity Frameworks, δίνουν την άνεση για πειραματισμό.

Όσον αφορά το design της σελίδας, το ASP.NET έρχεται με προ εγκατεστημένο τη βιβλιοθήκη Bootstrap και τη JQuery, όπως προαναφέρθηκε. Αυτό σημαίνει πως χρησιμοποιώντας αυτά τα δυο frameworks, θα μπορούσε να βελτιωθεί σημαντικά η εμφάνιση της εφαρμογής. Αλλά ακόμα και αν αυτά δεν είναι αρκετά, γίνεται εύκολα η προσθήκη νέων βιβλιοθηκών στο project.

Σχετικά με τη λειτουργικότητα της εφαρμογής μπορούν να γίνουν αρκετές βελτιώσεις. Αρχικά, θα μπορούσε να μπει μια υπηρεσία για αποστολή email. Αυτό θα ήταν πολύ χρήσιμο για το Identity Framework, καθώς μπορεί για παράδειγμα με την εγγραφή ενός νέου χρήστη να στέλνεται ένα email για επιβεβαίωση. Επίσης, σε περίπτωση απώλειας του password του χρήστη, θα μπορούσε να στέλνεται πάλι αντίστοιχο email, για την αποκατάσταση του.

Επιπλέον, με τη χρήση email, θα μπορούσαν οι χρήστες να ενημερώνονται για τα νέα tasks που δημιουργούνται στο project που είναι μέλη του. Θα μπορούσαν, επίσης, να ενημερώνονται για την οποιαδήποτε αλλαγή γίνει σε κάποιο από τα tasks τους ή τα project τους και γενικά στην πλατφόρμα την ίδια.

Παράρτημα

Παρακάτω παρατίθενται παραρτήματα από διάφορες υλοποιήσεις των Controllers, των Views και των Models για την καλύτερη κατανόηση της λειτουργίας της πλατφόρμας.

Controllers

Υλοποίηση του EmployeesController.

```
namespace MvcEmployeeManager.Controllers
{
    public class EmployeesController : Controller
    {
        private ApplicationDbContext db = new ApplicationDbContext();

        // GET: Employees
        public ActionResult Index()
        {
            //exclude admins from list
            var users = db.Users.ToList();
            var roleManager = new RoleManager<IdentityRole>(new RoleStore<IdentityRole>(new ApplicationDbContext()));
            var role = roleManager.FindByName("IsAdmin").Users.First();
            var admins = db.Users.Where(u => u.Roles.Select(r => r.RoleId).Contains(role.RoleId)).ToList();

            return View(users.Except(admins).ToList());
        }

        // GET: Employees/Details/5
        public ActionResult Details(string id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            ApplicationUser employee = db.Users.Find(id);
            if (employee == null)
            {
                return HttpNotFound();
            }
            return View(employee);
        }

```

```
43          // GET: Employees/Edit/5
44          public ActionResult Edit(string id)
45          {
46              if (id == null)
47              {
48                  return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
49              }
50              ApplicationUser employee = db.Users.Find(id);
51              if (employee == null)
52              {
53                  return HttpNotFound();
54              }
55              return View(employee);
56          }
57
58          // POST: Employees/Edit/5
59          // To protect from overposting attacks, please enable the specific properties you want to bind to, for
60          // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
61          [HttpPost]
62          [ValidateAntiForgeryToken]
63          public ActionResult Edit([Bind(Include = "Id,FirstName,LastName,Email,HomeAddress,PhoneNumber,HireDate")] ApplicationUser employee)
64          {
65              if (ModelState.IsValid)
66              {
67                  db.Entry(employee).State = EntityState.Modified;
68
69                  var store = new UserStore<ApplicationUser>(new ApplicationDbContext());
70                  var manager = new UserManager<ApplicationUser>(store);
71                  var currentUser = manager.FindById(employee.Id);
72                  currentUser.FirstName = employee.FirstName;
73                  currentUser.LastName = employee.LastName;
74                  currentUser.Email = employee.Email;
75                  currentUser.HomeAddress = employee.HomeAddress;
76                  currentUser.PhoneNumber = employee.PhoneNumber;
77                  currentUser.HireDate = employee.HireDate;
78
79                  manager.Update(currentUser);
80                  var ctx = store.Context;
81                  ctx.SaveChanges();
82              }
83              return RedirectToAction("Index");
84          }
85
86          // GET: Employees/Delete/5
87          public ActionResult Delete(string id)
88          {
89              if (id == null)
90              {
91                  return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
92              }
93              ApplicationUser employee = db.Users.Find(id);
94              if (employee == null)
95              {
96                  return HttpNotFound();
97              }
98              return View(employee);
99          }
100
101         // POST: Employees/Delete/5
102         [HttpPost, ActionName("Delete")]
103         [ValidateAntiForgeryToken]
104         public ActionResult DeleteConfirmed(string id)
105         {
106             var store = new UserStore<ApplicationUser>(new ApplicationDbContext());
107             var manager = new UserManager<ApplicationUser>(store);
108             var temp = manager.FindById(id);
109
110             manager.Delete(temp);
111             return RedirectToAction("Index");
112         }
113
114         protected override void Dispose(bool disposing)
115         {
116             if (disposing)
117             {
118                 db.Dispose();
119             }
120             base.Dispose(disposing);
121         }
122     }
123 }
124
```

Υλοποίηση του ProjectsController.

```
13  namespace MvcEmployeeManager.Controllers
14  {
15      public class ProjectsController : Controller
16      {
17          private ApplicationDbContext db = new ApplicationDbContext();
18
19          // GET: Projects
20          public ActionResult Index()
21          {
22              return View(db.Projects.ToList());
23          }
24
25          // GET: Projects/Details/5
26          public ActionResult Details(int? id)
27          {
28              if (id == null)
29              {
30                  return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
31              }
32              Project project = db.Projects.Find(id);
33              if (project == null)
34              {
35                  return HttpNotFound();
36              }
37
38              var projectEmployees = project.Employees;
39              var projectTasks = project.Tasks;
40              var projectIndexData = new ProjectIndexData
41              {
42                  Project = project,
43                  Tasks = projectTasks,
44                  Employees = projectEmployees
45              };
46
47              return View(projectIndexData);
48          }

50          // GET: Projects/Create
51          public ActionResult Create()
52          {
53              return View();
54          }
55
56          // POST: Projects/Create
57          // To protect from overposting attacks, please enable the specific properties you want to bind to, for
58          // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
59          [HttpPost]
60          [ValidateAntiForgeryToken]
61          public ActionResult Create([Bind(Include = "ID,Title,Description,StartDate,EndDate,Cost,ManagerId")] Project project)
62          {
63              if (ModelState.IsValid)
64              {
65                  project.ManagerId = User.Identity.GetUserId();
66                  var emp = db.Users.Find(User.Identity.GetUserId());
67                  project.Employees.Add(emp);
68                  db.Projects.Add(project);
69                  db.SaveChanges();
70                  return RedirectToAction("Index");
71              }
72
73              return View(project);
74          }
```

```
76          // GET: Projects/Edit/5
77          public ActionResult Edit(int? id)
78          {
79              if (id == null)
80              {
81                  return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
82              }
83              Project project = db.Projects.Find(id);
84              if (project == null)
85              {
86                  return HttpNotFound();
87              }
88              return View(project);
89          }
90
91          // POST: Projects/Edit/5
92          // To protect from overposting attacks, please enable the specific properties you want to bind to, for
93          // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
94          [HttpPost]
95          [ValidateAntiForgeryToken]
96          public ActionResult Edit([Bind(Include = "ID,Title,Description,StartDate,EndDate,Cost,ManagerId")] Project project)
97          {
98              if (ModelState.IsValid)
99              {
100                 db.Entry(project).State = EntityState.Modified;
101                 db.SaveChanges();
102                 return RedirectToAction("Index");
103             }
104             return View(project);
105         }
106
```

```
107         // GET: Projects/Delete/5
108         public ActionResult Delete(int? id)
109         {
110             if (id == null)
111             {
112                 return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
113             }
114             Project project = db.Projects.Find(id);
115             if (project == null)
116             {
117                 return HttpNotFound();
118             }
119             return View(project);
120         }
121
122         // POST: Projects/Delete/5
123         [HttpPost, ActionName("Delete")]
124         [ValidateAntiForgeryToken]
125         public ActionResult DeleteConfirmed(int id)
126         {
127             Project project = db.Projects.Find(id);
128             db.Projects.Remove(project);
129             db.SaveChanges();
130             return RedirectToAction("Index");
131         }
132
```

```
133         // GET: Projects/AddEmployeeList
134         public ActionResult AddEmployeeList(int projectId)
135         {
136             //exclude admins from list
137             var users = db.Users.ToList();
138             var roleManager = new RoleManager<IdentityRole>(new RoleStore<IdentityRole>(new ApplicationDbContext()));
139             var role = roleManager.FindByName("IsAdmin").Users.First();
140             var admins = db.Users.Where(u => u.Roles.Select(r => r.RoleId).Contains(role.RoleId)).ToList();
141 
142             var viewModel = new ProjectIndexData
143             {
144                 Employees = users.Except(admins).ToList(),
145                 Project = db.Projects.Find(projectId),
146                 Tasks = null
147             };
148             return View(viewModel);
149         }
150 
151         // GET: Projects/AddEmployee/id+proj_id
152         public ActionResult AddEmployee(string id, int projectId)
153         {
154             var proj = db.Projects.Find(projectId);
155             var emp = db.Users.Find(id);
156             proj.Employees.Add(emp);
157             db.SaveChanges();
158             return RedirectToAction("Details", new { id = proj.ID });
159         }
160 
161         // GET: Projects/RemoveEmployee/id+proj_id
162         public ActionResult RemoveEmployee(string id, int projectId)
163         {
164             var proj = db.Projects.Find(projectId);
165             var emp = db.Users.Find(id);
166             proj.Employees.Remove(emp);
167             db.SaveChanges();
168             return RedirectToAction("Details", new { id = proj.ID });
169         }
170 
171         // GET: Projects/RemoveEmployee/id+proj_id
172         public ActionResult ChangeProjectManager(string id, int projectId)
173         {
174             var proj = db.Projects.Find(projectId);
175             proj.ManagerId = id;
176             db.SaveChanges();
177             return RedirectToAction("Details", new { id = proj.ID });
178         }
179 
180         // GET: Projects/EmployeeDetails/id+proj_id
181         public ActionResult EmployeeDetails(string id, int projectId)
182         {
183             List<ApplicationUser> temp = new List<ApplicationUser>();
184             temp.Add(db.Users.Find(id));
185             var viewModel = new ProjectIndexData
186             {
187                 Employees = temp,
188                 Project = db.Projects.Find(projectId),
189                 Tasks = null
190             };
191             return View(viewModel);
192         }
193 
194         protected override void Dispose(bool disposing)
195         {
196             if (disposing)
197             {
198                 db.Dispose();
199             }
200             base.Dispose(disposing);
201         }
202     }
203 }
```

Υλοποίηση του ProjectTasksController.

```
13  namespace MvcEmployeeManager.Controllers
14  {
15      public class ProjectTasksController : Controller
16      {
17          private ApplicationDbContext db = new ApplicationDbContext();
18
19          // GET: ProjectTasks
20          public ActionResult Index()
21          {
22              return View(db.ProjectTasks.ToList());
23          }
24
25          // GET: ProjectTasks/Details/5
26          public ActionResult Details(int? id)
27          {
28              if (id == null)
29              {
30                  return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
31              }
32              ProjectTask projectTask = db.ProjectTasks.Find(id);
33              if (projectTask == null)
34              {
35                  return HttpNotFound();
36              }
37              return View(projectTask);
38          }
39
40          // GET: ProjectTasks/Create
41          public ActionResult Create(int? id)
42          {
43              if (id == null)
44                  return View();
45
46              var viewModel = new NewTaskViewModel
47              {
48                  ProjectId = id
49              };
50              return View(viewModel);
51          }
```

```
53          // POST: ProjectTasks/Create
54          // To protect from overposting attacks, please enable the specific properties you want to bind to, for
55          // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
56          [HttpPost]
57          [ValidateAntiForgeryToken]
58          public ActionResult Create(NewTaskViewModel newTaskViewModel)
59          {
60
61              if (newTaskViewModel.ProjectTask == null)
62              {
63                  return RedirectToAction("Details", "Projects", new { id = newTaskViewModel.ProjectId });
64              }
65              newTaskViewModel.ProjectTask.ProjectID = (int) newTaskViewModel.ProjectId;
66              newTaskViewModel.ProjectTask.Pending = true;
67              db.ProjectTasks.Add(newTaskViewModel.ProjectTask);
68              db.Projects.Find(newTaskViewModel.ProjectId).Tasks.Add(newTaskViewModel.ProjectTask);
69              db.SaveChanges();
70
71              return RedirectToAction("Details", "Projects", new { id = newTaskViewModel.ProjectId });
72          }
73
```

```
74          // GET: ProjectTasks/Edit/5
75          public ActionResult Edit(int? id, int? projectId)
76          {
77              if (id == null || projectId == null)
78              {
79                  return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
80              }
81              ProjectTask projectTask = db.ProjectTasks.Find(id);
82              if (projectTask == null)
83              {
84                  return HttpNotFound();
85              }
86
87              NewTaskViewModel edited = new NewTaskViewModel
88              {
89                  ProjectId = projectId,
90                  ProjectTask = projectTask
91              };
92              return View(edited);
93          }
94
95          // POST: ProjectTasks/Edit/5
96          // To protect from overposting attacks, please enable the specific properties you want to bind to, for
97          // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
98          [HttpPost]
99          [ValidateAntiForgeryToken]
100         public ActionResult Edit(NewTaskViewModel newTaskViewModel)
101         {
102             var projectTask = newTaskViewModel.ProjectTask;
103             projectTask.ProjectID = (int) newTaskViewModel.ProjectId; // for consistency. DO NOT REMOVE
104             if (ModelState.IsValid)
105             {
106                 db.Entry(projectTask).State = EntityState.Modified;
107                 db.SaveChanges();
108                 return RedirectToAction("Details", "Projects", new { id = newTaskViewModel.ProjectId });
109             }
110
111             return View(projectTask);
112         }
113
---
114         // GET: ProjectTasks/Delete/5
115         public ActionResult Delete(int? id)
116         {
117             if (id == null)
118             {
119                 return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
120             }
121             ProjectTask projectTask = db.ProjectTasks.Find(id);
122             if (projectTask == null)
123             {
124                 return HttpNotFound();
125             }
126
127             return View(projectTask);
128         }
129
130         // POST: ProjectTasks/Delete/5
131         [HttpPost, ActionName("Delete")]
132         [ValidateAntiForgeryToken]
133         public ActionResult DeleteConfirmed(int id)
134         {
135             ProjectTask projectTask = db.ProjectTasks.Find(id);
136             db.Projects.Find(projectTask.ProjectID).Tasks.Remove(projectTask);
137             var projId = projectTask.ProjectID;
138             db.ProjectTasks.Remove(projectTask);
139             db.SaveChanges();
140             return RedirectToAction("Details", "Projects", new { id = projId });
141         }
142
```

```
143         public ActionResult Claim(int? id, string userId)
144         {
145             if (id == null || userId == null)
146             {
147                 return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
148             }
149             ProjectTask projectTask = db.ProjectTasks.Find(id);
150             var user = db.Users.Find(userId);
151             if (projectTask == null || user == null)
152             {
153                 return HttpNotFound();
154             }
155 
156             projectTask.Employee = user;
157             projectTask.ApplicationUserID = userId;
158             user.Tasks.Add(projectTask);
159             db.SaveChanges();
160 
161             return RedirectToAction("Details", "Projects", new { id = projectTask.ProjectID });
162         }
```

```
164         public ActionResult Unclaim(int? id, string userId)
165         {
166             if (id == null || userId == null)
167             {
168                 return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
169             }
170             ProjectTask projectTask = db.ProjectTasks.Find(id);
171             var user = db.Users.Find(userId);
172             if (projectTask == null || user == null)
173             {
174                 return HttpNotFound();
175             }
176 
177             projectTask.Employee = null;
178             projectTask.ApplicationUserID = null;
179             user.Tasks.Remove(projectTask);
180             db.SaveChanges();
181 
182             return RedirectToAction("Details", "Projects", new { id = projectTask.ProjectID });
183         }
```

```
185         public ActionResult Done(int? id)
186         {
187             if (id == null)
188             {
189                 return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
190             }
191             ProjectTask projectTask = db.ProjectTasks.Find(id);
192             if (projectTask == null)
193             {
194                 return HttpNotFound();
195             }
196 
197             projectTask.Pending = false;
198             db.SaveChanges();
199 
200             return RedirectToAction("Details", "Projects", new { id = projectTask.ProjectID });
201         }
202 
203         public ActionResult Pending(int? id)
204         {
205             if (id == null)
206             {
207                 return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
208             }
209             ProjectTask projectTask = db.ProjectTasks.Find(id);
210             if (projectTask == null)
211             {
212                 return HttpNotFound();
213             }
214 
215             projectTask.Pending = true;
216             db.SaveChanges();
217 
218             return RedirectToAction("Details", "Projects", new { id = projectTask.ProjectID });
219         }
220 
```

```
221
222         protected override void Dispose(bool disposing)
223         {
224             if (disposing)
225             {
226                 db.Dispose();
227             }
228             base.Dispose(disposing);
229         }
230     }
231 }
232
```

Views

Εδώ παρατίθενται ορισμένα από τα Views της πλατφόρμας καθώς πολλά έχουν την ίδια δομή, ενώ έχουν διαφορετικό Model και Controller συνδεδεμένο μεταξύ τους.

Υλοποίηση του View - Register των χρηστών (ApplicationUsers).

```
@model MvcEmployeeManager.Models.RegisterViewModel
@{
    ViewBag.Title = "Register";
}

<h2>@ViewBag.Title.</h2>

@using (Html.BeginForm("Register", "Account", FormMethod.Post, new { @class = "form-horizontal", role = "form" }))
{
    @Html.AntiForgeryToken()
    <h4>Create a new account.</h4>
    <hr />
    @Html.ValidationSummary("", new { @class = "text-danger" })
    <div class="form-group">
        @Html.LabelFor(m => m.FirstName, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.FirstName, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.LastName, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.LastName, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
        </div>
    </div>
```

```
38      <div class="form-group">
39          @Html.LabelFor(m => m.ConfirmPassword, new { @class = "col-md-2 control-label" })
40          <div class="col-md-10">
41              @Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-control" })
42          </div>
43      </div>
44      <div class="form-group">
45          @Html.LabelFor(m => m.HomeAddress, new { @class = "col-md-2 control-label" })
46          <div class="col-md-10">
47              @Html.TextBoxFor(m => m.HomeAddress, new { @class = "form-control" })
48          </div>
49      </div>
50      <div class="form-group">
51          @Html.LabelFor(m => m.PhoneNumber, new { @class = "col-md-2 control-label" })
52          <div class="col-md-10">
53              @Html.TextBoxFor(m => m.PhoneNumber, new { @class = "form-control" })
54          </div>
55      </div>
56      <div class="form-group">
57          @Html.LabelFor(m => m.HireDate, new { @class = "col-md-2 control-label" })
58          <div class="col-md-10">
59              @Html.EditorFor(m => m.HireDate, new { htmlAttributes = new { @class = "form-control" } })
60              @Html.ValidationMessageFor(m => m.HireDate, "", new { @class = "text-danger" })
61          </div>
62      </div>
63      <div class="form-group">
64          <div class="col-md-offset-2 col-md-10">
65              <input type="submit" class="btn btn-default" value="Register" />
66          </div>
67      </div>
68  }
69
70  @section Scripts {
71      @Scripts.Render("~/bundles/jqueryval")
72  }
73
```

Υλοποίηση του View - Login των χρηστών (ApplicationUsers).

```
@using MvcEmployeeManager.Models
@model LoginViewModel
@{
    ViewBag.Title = "Log in";
}

<h2>@ViewBag.Title.</h2>
<div class="row">
    <div class="col-md-8">
        <section id="loginForm">
            @using (Html.BeginForm("Login", "Account", new { ReturnUrl = ViewBag.ReturnUrl }, FormMethod.Post, new { @class = "form-horizontal", role = "form" }))
            {
                @Html.AntiForgeryToken()
                <h4>Use a local account to log in.</h4>
                <hr />
                @Html.ValidationSummary(true, "", new { @class = "text-danger" })
                <div class="form-group">
                    @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
                    <div class="col-md-10">
                        @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
                        @Html.ValidationMessageFor(m => m.Email, "", new { @class = "text-danger" })
                    </div>
                </div>
                <div class="form-group">
                    @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
                    <div class="col-md-10">
                        @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
                        @Html.ValidationMessageFor(m => m.Password, "", new { @class = "text-danger" })
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-md-offset-2 col-md-10">
                        <div class="checkbox">
                            @Html.CheckBoxFor(m => m.RememberMe)
                            @Html.LabelFor(m => m.RememberMe)
                        </div>
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-md-offset-2 col-md-10">
                        <input type="submit" value="Log in" class="btn btn-default" />
                    </div>
                </div>
                <p>
                    @Html.ActionLink("Register as a new user", "Register")
                </p>
                @* Enable this once you have account confirmation enabled for password reset functionality
                    <p>
                        @Html.ActionLink("Forgot your password?", "ForgotPassword")
                    </p>*@
            }
        </section>
    </div>

</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Υλοποίηση του View - Details ενός Project.

```
@model MvcEmployeeManager.Models.ProjectIndexData

@using Microsoft.AspNet.Identity;

@{
    ViewBag.Title = "Details";
}

<div>
    <h3>Project Details</h3>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Project.Title)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Project.Title)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Project.Description)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Project.Description)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Project.StartDate)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Project.StartDate)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Project.EndDate)
        </dt>


        <dd>
            @Html.DisplayFor(model => model.Project.EndDate)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Project.Cost)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Project.Cost)
        </dd>

    </dl>
</div>
<p>
    @if (Model.Project.ManagerId == User.Identity.GetUserId())
    {
        @Html.ActionLink("Edit", "Edit", new { id = Model.Project.ID }) <span>|</span>
    }

    @Html.ActionLink("Back to List", "Index")
</p>
<br />
```

```
<div>
    <h3>Employees working in Project</h3>
    <hr />
    @if (User.IsInRole("IsAdmin") ||
        User.Identity.GetUserId() == Model.Project.ManagerId)
    {
        @Html.ActionLink("Add employee to project", "AddEmployeeList", new { projectId = Model.Project.ID })
    }
    <table class="table">
        <tr>
            <th>
                First Name
            </th>
            <th>
                Last Name
            </th>
            <th></th>
            <th></th>
        </tr>

    @foreach (var item in Model.Employees)
    {
        <tr>
            <td>
                @item.FirstName
            </td>
            <td>
                @item.LastName
            </td>
            <th>
                @if (Model.Project.ManagerId == item.Id)
                {
                    <span>Project Manager</span>
                }
            </th>
            <td>
                @Html.ActionLink("Details", "EmployeeDetails", new { id = item.Id, projectId = Model.Project.ID }, null)
                @if (User.IsInRole("IsAdmin") ||
                    User.Identity.GetUserId() == Model.Project.ManagerId)
                {
                    <span>|</span>
                    @Html.ActionLink("Remove", "RemoveEmployee", new { id = item.Id, projectId = Model.Project.ID }, null)
                    if (Model.Project.ManagerId != item.Id)
                    {
                        <span>|</span>
                        @Html.ActionLink("Change Project Manager", "ChangeProjectManager", new { id = item.Id, projectId = Model.Project.ID }, null)
                    }
                }

            </td>
        </tr>
    }

    </table>
</div>
 <br />
<div>
    <h3>Project Tasks</h3>
    <hr />
    @Html.ActionLink("Add task to project", "Create", "ProjectTasks", new { id = Model.Project.ID }, null)
    <table class="table">
        <tr>
            <th>
                Title
            </th>
            <th>
                Description
            </th>
            <th>
                Claimed by User
            </th>
            <th>
                Status
            </th>
            <th></th>
        </tr>

        @foreach (var item in Model.Tasks)
        {
        <tr>
            <td>
                @item.Title
            </td>
            <td>
                @item.Description
            </td>
            <td>
```

```
151                 @if (item.Employee != null)
152                 {
153                     @item.Employee.UserName
154                 }
155             </td>
156             <td>
157                 @if (item.Employee != null)
158                 {
159                     if (item.Pending)
160                     {
161                         <span>Pending</span>
162                     }
163                     else
164                     {
165                         <span>Done</span>
166                     }
167                 }
168                 else
169                 {
170                     <span>Unclaimed</span>
171                 }
172 
173             </td>
174 
175             <td>
176                 @Html.ActionLink("Details", "Details", "ProjectTasks", new { id = item.ID }, null)
177 
178                 @if ((item.Employee != null &&
179                       User.Identity.GetUserId() == item.Employee.Id) ||
180                       Model.Project.ManagerId == User.Identity.GetUserId() ||
181                       User.IsInRole("IsAdmin"))
182                 {
183                     <span>|</span>
184                     @Html.ActionLink("Edit", "Edit", "ProjectTasks", new { id = item.ID, projectId = Model.Project.ID }, null) <span>|</span>
185                     @Html.ActionLink("Delete", "Delete", "ProjectTasks", new { id = item.ID, projectId = Model.Project.ID }, null)
186                 }
187 
188                 @if (item.Employee == null)
189                 {
190                     <span>|</span>
191                     @Html.ActionLink("Claim Task", "Claim", "ProjectTasks", new { id = item.ID, userId = User.Identity.GetUserId() }, null)
192                 }
193                 else if ((item.Employee != null &&
194                           User.Identity.GetUserId() == item.Employee.Id) ||
195                           Model.Project.ManagerId ==
196                           User.IsInRole("IsAdmin"))
197                 {
198                     <span>|</span>
199                     @Html.ActionLink("Unclaim Task", "Unclaim", "ProjectTasks", new { id = item.ID, userId = User.Identity.GetUserId() }, null) <span>|</span>
200 
201                     if (item.Pending)
202                     {
203                         @Html.ActionLink("Done", "Done", "ProjectTasks", new { id = item.ID }, null)
204                     }
205                     else
206                     {
207                         @Html.ActionLink("Pending", "Pending", "ProjectTasks", new { id = item.ID, projectId = Model.Project.ID }, null)
208                     }
209                 }
210             </td>
211         </tr>
212     }
213 
214 </table>
215 </div>
216 
217 <p>
218     @Html.ActionLink("Back to List", "Index")
219 </p>
220 
221 
222 
```

bool string.operator ==(string left, string right)

Υλοποίηση του View - Delete ενός ProjectTask.

```
@model MvcEmployeeManager.Models.ProjectTask

@{
    ViewBag.Title = "Delete";
}

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>


<div>
    <h4>ProjectTask</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Title)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Title)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Description)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Description)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.StartDate)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.StartDate)
        </dd>

```

```
40          <dt>
41              @Html.DisplayNameFor(model => model.EndDate)
42          </dt>
43
44          <dd>
45              @Html.DisplayFor(model => model.EndDate)
46          </dd>
47
48          <dt>
49              @Html.DisplayNameFor(model => model.Cost)
50          </dt>
51
52          <dd>
53              @Html.DisplayFor(model => model.Cost)
54          </dd>
55
56      </dl>
57
58      @using (Html.BeginForm()) {
59          @Html.AntiForgeryToken()
60
61      <div class="form-actions no-color">
62          <input type="submit" value="Delete" class="btn btn-default" /> |
63          @Html.ActionLink("Back to Project", "Details", "Projects", new { id = Model.ProjectID }, null)
64      </div>
65      }
66  </div>
67
```

Υλοποίηση του View - Index των Employees. Ουσιαστικά η λίστα με τους χρήστες.

```
1   @model IEnumerable<MvcEmployeeManager.Models.ApplicationUser>
2
3   @{
4       ViewBag.Title = "Index";
5   }
6
7   <h2>Index</h2>
8
9   <p>
10      @if  (User.IsInRole("IsAdmin"))
11      {
12          @Html.ActionLink("Create New", "Register", "Account")
13      }
14  </p>
15
16  <table class="table">
17      <tr>
18          <th>
19              @Html.DisplayNameFor(model => model.FirstName)
20          </th>
21          <th>
22              @Html.DisplayNameFor(model => model.LastName)
23          </th>
24          <th>
25              @Html.DisplayNameFor(model => model.Email)
26          </th>
27          <th>
```

```
27          <th>
28              @Html.DisplayNameFor(model => model.PhoneNumber)
29          </th>
30
31          <th></th>
32      </tr>
33      @foreach (var item in Model)
34      {
35          <tr>
36              <td>
37                  @Html.DisplayFor(modelItem => item.FirstName)
38              </td>
39              <td>
40                  @Html.DisplayFor(modelItem => item.LastName)
41              </td>
42              <td>
43                  @Html.DisplayFor(modelItem => item.Email)
44              </td>
45              <td>
46                  @Html.DisplayFor(modelItem => item.PhoneNumber)
47              </td>
48              <th>
49                  @Html.ActionLink("Details", "Details", new { id = item.Id })
50                  @if (User.IsInRole("IsAdmin"))
51                  {
52                      <span>|</span>
53                      @Html.ActionLink("Edit", "Edit", new { id = item.Id }) <span>|</span>
54                      @Html.ActionLink("Delete", "Delete", new { id = item.Id })
55                  }
56
57
58              </th>
59          </tr>
60      }
61  </table>
62
```

Υλοποίηση του View - Create των Projects.

```
@model MvcEmployeeManager.Models.Project

@{
    ViewBag.Title = "Create";
}

<h2>Create</h2>


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

<div class="form-horizontal">
    <h4>Project</h4>
    <hr />
    @Html.ValidationSummary(true, "", new { @class = "text-danger" })
    <div class="form-group">
        @Html.LabelFor(model => model.Title, htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Title, new { htmlAttributes = new { @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.Title, "", new { @class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Description, htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Description, new { htmlAttributes = new { @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.Description, "", new { @class = "text-danger" })
        </div>
    </div>
```

```
    <div class="form-group">
        @Html.LabelFor(model => model.StartDate, htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.StartDate, new { htmlAttributes = new { @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.StartDate, "", new { @class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.EndDate, htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.EndDate, new { htmlAttributes = new { @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.EndDate, "", new { @class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Cost, htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Cost, new { htmlAttributes = new { @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.Cost, "", new { @class = "text-danger" })
        </div>
    </div>
```

```
57
58      <div class="form-group">
59          <div class="col-md-offset-2 col-md-10">
60              <input type="submit" value="Create" class="btn btn-default" />
61          </div>
62      </div>
63
64      <div class="form-group">
65          <div class="col-md-10">
66              <h4><strong>You will be assigned as the Project Manager. This can be changed later.</strong></h4>
67          </div>
68      </div>
69
70  </div>
71  }
72
73  <div>
74      @Html.ActionLink("Back to List", "Index")
75  </div>
76
77  @section Scripts {
78      @Scripts.Render("~/bundles/jqueryval")
79  }
80
```

Models

Εδώ παρατίθενται Models της πλατφόρμας. Κάποια από αυτά αποτελούν μέλη της βάσης δεδομένων, ενώ κάποια άλλα είναι απλά για να χρησιμοποιούνται από τα Views, ως placeholders.

Παρακάτω είναι η υλοποίηση αρκετών placeholders κλάσεων για να περαστούν στα αντίστοιχα Views που τις χρειάζονται. Αυτό το αρχείο έχει δημιουργηθεί αυτόματα από το Identity Framework για όσα Views αφορούν τους χρήστες, όπως για παράδειγμα Register, Password Change κ.α. Αρκετές από αυτές τις κλάσεις δεν χρησιμοποιήθηκαν εφόσον δεν ήταν απαραίτητες για τη λειτουργία της εφαρμογής.

```
4   namespace MvcEmployeeManager.Models
5   {
6       public class ExternalLoginConfirmationViewModel
7       {
8           [Required]
9           [Display(Name = "Email")]
10          public string Email { get; set; }
11      }
12
13      public class ExternalLoginListViewModel
14      {
15          public string ReturnUrl { get; set; }
16      }
17
18      public class SendCodeViewModel
19      {
20          public string SelectedProvider { get; set; }
21          public ICollection<System.Web.Mvc.SelectListItem> Providers { get; set; }
22          public string ReturnUrl { get; set; }
23          public bool RememberMe { get; set; }
24      }
25
26      public class VerifyCodeViewModel
27      {
28          [Required]
29          public string Provider { get; set; }
30
31          [Required]
32          [Display(Name = "Code")]
33          public string Code { get; set; }
34          public string ReturnUrl { get; set; }
35
36          [Display(Name = "Remember this browser?")]
37          public bool RememberBrowser { get; set; }
38
39          public bool RememberMe { get; set; }
40      }
41
42      public class ForgotViewModel
43      {
44          [Required]
45          [Display(Name = "Email")]
46          public string Email { get; set; }
47      }
48
49      public class LoginViewModel
50      {
51          [Required]
52          [Display(Name = "Email")]
53          [EmailAddress]
54          public string Email { get; set; }
55
56          [Required]
57          [DataType(DataType.Password)]
58          [Display(Name = "Password")]
59          public string Password { get; set; }
60
61          [Display(Name = "Remember me?")]
62          public bool RememberMe { get; set; }
63      }
64
```

```
65      public class RegisterViewModel
66      {
67          [Required]
68          [StringLength(100)]
69          [Display(Name = "First Name")]
70          public string FirstName { get; set; }
71
72          [Required]
73          [StringLength(100)]
74          [Display(Name = "Last Name")]
75          public string LastName { get; set; }
76
77          [Required]
78          [EmailAddress]
79          [Display(Name = "Email")]
80          public string Email { get; set; }
81
82          [Required]
83          [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
84          [DataType(DataType.Password)]
85          [Display(Name = "Password")]
86          public string Password { get; set; }
87
88          [DataType(DataType.Password)]
89          [Display(Name = "Confirm password")]
90          [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
91          public string ConfirmPassword { get; set; }
92
93          [Required]
94          [StringLength(20)]
95          [Display(Name = "Phone Number")]
96          public string PhoneNumber { get; set; }
97
98          [Required]
99          [StringLength(100)]
100         [Display(Name = "Home Address")]
101         public string HomeAddress { get; set; }
102
103         [Required]
104         [Display(Name = "Hire Date")]
105         [DataType(DataType.Date)]
106         [DisplayFormat(DataFormatString = "{0:MM/dd/yyyy}", ApplyFormatInEditMode = true)]
107         public System.DateTime HireDate { get; set; }
108     }
109
110     public class ResetPasswordViewModel
111     {
112         [Required]
113         [EmailAddress]
114         [Display(Name = "Email")]
115         public string Email { get; set; }
116
117         [Required]
118         [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
119         [DataType(DataType.Password)]
120         [Display(Name = "Password")]
121         public string Password { get; set; }
122
123         [DataType(DataType.Password)]
124         [Display(Name = "Confirm password")]
125         [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
126         public string ConfirmPassword { get; set; }
127
128         public string Code { get; set; }
129     }
130
131     public class ForgotPasswordViewModel
132     {
133         [Required]
134         [EmailAddress]
135         [Display(Name = "Email")]
136         public string Email { get; set; }
137     }
138 }
139
```

Υλοποίηση κλάσεων που αφορούν τους χρήστες. Η πρώτη κλάση είναι η εκδοχή του ApplicationUser όπως ορίζεται από το ASP.NET Identity σε C#. Αυτό μετά περνάει στη βάση δεδομένων. Η δεύτερη κλάση ορίζει επίσης ποιες άλλες κλάσεις Models θα είναι στη βάση δεδομένων.

```
9   namespace MvcEmployeeManager.Models
10  {
11      // You can add profile data for the user by adding more properties to your ApplicationUser class,
12      // please visit https://go.microsoft.com/fwlink/?LinkID=317594 to learn more.
13      public class ApplicationUser : IdentityUser
14      {
15
16          public ApplicationUser()
17          {
18              Projects = new List<Project>();
19              Tasks = new List<ProjectTask>();
20          }
21
22          [Required]
23          [StringLength(100)]
24          [Display(Name = "First Name")]
25          public string FirstName { get; set; }
26
27          [Required]
28          [StringLength(100)]
29          [Display(Name = "Last Name")]
30          public string LastName { get; set; }
31
32          [Required]
33          [StringLength(100)]
34          [Display(Name = "Home Address")]
35          public string HomeAddress { get; set; }
36
37          [Required]
38          [Display(Name = "Hire Date")]
39          [DisplayFormat(DataFormatString = "{0:dd-MM-yyyy}", ApplyFormatInEditMode = true)]
40          public System.DateTime HireDate { get; set; }
41
42          public virtual ICollection<Project> Projects { get; set; }
43          public virtual ICollection<ProjectTask> Tasks { get; set; }
44
```

```
45          public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)
46          {
47              // Note the authenticationType must match the one defined in CookieAuthenticationOptions.AuthenticationType
48              var userIdentity = await manager.CreateIdentityAsync(this, DefaultAuthenticationTypes.ApplicationCookie);
49              // Add custom user claims here
50              return userIdentity;
51          }
52      }
53
54      public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
55      {
56          public ApplicationDbContext()
57              : base("DefaultConnection", throwIfV1Schema: false)
58          {
59          }
60
61          public static ApplicationDbContext Create()
62          {
63              return new ApplicationDbContext();
64          }
65
66          // Set Models to pass to the database
67          public System.Data.Entity.DbSet<MvcEmployeeManager.Models.Project> Projects { get; set; }
68
69          public System.Data.Entity.DbSet<MvcEmployeeManager.Models.ProjectTask> ProjectTasks { get; set; }
70      }
71  }
```

Υλοποίηση και άλλων βοηθητικών κλάσεων που αφορούν άλλες λειτουργίες χρηστών.

```
namespace MvcEmployeeManager.Models
{
    public class IndexViewModel
    {
        public ApplicationUser User { get; set; }
        public bool HasPassword { get; set; }
        public IList<UserLoginInfo> Logins { get; set; }
        public string PhoneNumber { get; set; }
        public bool TwoFactor { get; set; }
        public bool BrowserRemembered { get; set; }
    }

    public class ManageLoginsViewModel
    {
        public IList<UserLoginInfo> CurrentLogins { get; set; }
        public IList<AuthenticationDescription> OtherLogins { get; set; }
    }

    public class FactorViewModel
    {
        public string Purpose { get; set; }
    }

    public class SetPasswordViewModel
    {
        [Required]
        [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "New password")]
        public string NewPassword { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm new password")]
        [Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not match.")]
        public string ConfirmPassword { get; set; }
    }

    public class ChangePasswordViewModel
    {
        [Required]
        [DataType(DataType.Password)]
        [Display(Name = "Current password")]
        public string OldPassword { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "New password")]
        public string NewPassword { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm new password")]
        [Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not match.")]
        public string ConfirmPassword { get; set; }
    }

    public class AddPhoneNumberViewModel
    {
        [Required]
        [Phone]
        [Display(Name = "Phone Number")]
        public string Number { get; set; }
    }
```

```
69
70      public class VerifyPhoneNumberViewModel
71      {
72          [Required]
73          [Display(Name = "Code")]
74          public string Code { get; set; }
75
76          [Required]
77          [Phone]
78          [Display(Name = "Phone Number")]
79          public string PhoneNumber { get; set; }
80      }
81
82      public class ConfigureTwoFactorViewModel
83      {
84          public string SelectedProvider { get; set; }
85          public ICollection<System.Web.Mvc.SelectListItem> Providers { get; set; }
86      }
87  }
```

Υλοποίηση βοηθητικής κλάσης για τη δημιουργία νέου Project Task.

```
6   namespace MvcEmployeeManager.Models
7   {
8       public class NewTaskViewModel
9       {
10          public ProjectTask ProjectTask { get; set; }
11          public int? ProjectId { get; set; }
12      }
13  }
```

Υλοποίηση κλάσης Project η οποία είναι μέλος της βάσης δεδομένων.

```
7   namespace MvcEmployeeManager.Models
8   {
9       public class Project
10      {
11
12          public Project()
13          {
14              Employees = new List<ApplicationUser>();
15              Tasks = new List<ProjectTask>();
16          }
17
18          public int ID { get; set; }
19          public string Title { get; set; }
20          public string Description { get; set; }
21
22          [DisplayFormat(DataFormatString = "{0:dd-MM-yyyy}", ApplyFormatInEditMode = true)]
23          public DateTime StartDate { get; set; }
24
25          [DisplayFormat(DataFormatString = "{0:dd-MM-yyyy}", ApplyFormatInEditMode = true)]
26          public DateTime EndDate { get; set; }
27          [DataType(DataType.Currency)]
28          public decimal Cost { get; set; }
29
30          public string ManagerId { get; set; }
31
32          public virtual ICollection<ApplicationUser> Employees { get; set; }
33          public virtual ICollection<ProjectTask> Tasks { get; set; }
34      }
35  }
```

Υλοποίηση βοηθητικής κλάσης για τη δημιουργία νέου Project.

```
6   namespace MvcEmployeeManager.Models
7   {
8       public class ProjectIndexData
9       {
10          public Project Project;
11          public IEnumerable<ApplicationUser> Employees { get; set; }
12          public IEnumerable<ProjectTask> Tasks { get; set; }
13      }
14  }
```

Υλοποίηση κλάσης ProjectTask η οποία είναι μέλος της βάσης δεδομένων.

```
7   namespace MvcEmployeeManager.Models
8   {
9       public class ProjectTask
10      {
11          public int ID { get; set; }
12          public string Title { get; set; }
13          public string Description { get; set; }
14
15          [DisplayFormat(DataFormatString = "{0:dd-MM-yyyy}", ApplyFormatInEditMode = true)]
16          public DateTime StartDate { get; set; }
17
18          [DisplayFormat(DataFormatString = "{0:dd-MM-yyyy}", ApplyFormatInEditMode = true)]
19          public DateTime EndDate { get; set; }
20
21          public decimal Cost { get; set; }
22          public bool Pending { get; set; }
23
24          public string ApplicationUserID { get; set; } //employee id
25          public int ProjectID { get; set; }
26
27          public virtual ApplicationUser Employee { get; set; }
28          public virtual Project Project { get; set; }
29      }
30  }
```

Υλοποίηση του IdentityConfig

Το IdentityConfig είναι ένα αρχείο το οποίο ορίζει τις βασικές ρυθμίσεις για όλες τις λειτουργίες που δίνει by default το Identity Framework. Εδώ φαίνεται εύκολα οι δυνατότητες που μπορεί να δώσει σε ένα web application το ASP.NET.

```
15  namespace MvcEmployeeManager
16  {
17      public class EmailService : IIdentityMessageService
18      {
19          public Task SendAsync(IdentityMessage message)
20          {
21              // Plug in your email service here to send an email.
22              return Task.FromResult(0);
23          }
24      }
25
26      public class SmsService : IIdentityMessageService
27      {
28          public Task SendAsync(IdentityMessage message)
29          {
30              // Plug in your SMS service here to send a text message.
31              return Task.FromResult(0);
32          }
33      }
34
35      // Configure the application user manager used in this application. UserManager is defined in ASP.NET Identity and is used by the application.
36      public class ApplicationUserManager : UserManager<ApplicationUser>
37      {
38          public ApplicationUserManager(IUserStore<ApplicationUser> store)
39              : base(store)
40          {
41          }
42
43          public static ApplicationUserManager Create(IdentityFactoryOptions<ApplicationUserManager> options, IOwinContext context)
44          {
45              var manager = new ApplicationUserManager(new UserStore<ApplicationUser>(context.Get<ApplicationDbContext>()));
46              // Configure validation logic for usernames
47              manager.UserValidator = new UserValidator<ApplicationUser>(manager)
48              {
49                  AllowOnlyAlphanumericUserNames = false,
50                  RequireUniqueEmail = true
51              };
```

```
            // Configure validation logic for passwords
            manager.PasswordValidator = new PasswordValidator
            {
                RequiredLength = 6,
                RequireNonLetterOrDigit = true,
                RequireDigit = true,
                RequireLowercase = true,
                RequireUppercase = true,
            };

            // Configure user lockout defaults
            manager.UserLockoutEnabledByDefault = true;
            manager.DefaultAccountLockoutTimeSpan = TimeSpan.FromMinutes(5);
            manager.MaxFailedAccessAttemptsBeforeLockout = 5;

            // Register two factor authentication providers. This application uses Phone and Emails as a step of receiving a code for verifying the user
            // You can write your own provider and plug it in here.
            manager.RegisterTwoFactorProvider("Phone Code", new PhoneNumberTokenProvider<ApplicationUser>
            {
                MessageFormat = "Your security code is {0}"
            });
            manager.RegisterTwoFactorProvider("Email Code", new EmailTokenProvider<ApplicationUser>
            {
                Subject = "Security Code",
                BodyFormat = "Your security code is {0}"
            });
            manager.EmailService = new EmailService();
            manager.SmsService = new SmsService();
            var dataProtectionProvider = options.DataProtectionProvider;
            if (dataProtectionProvider != null)
            {
                manager.UserTokenProvider =
                    new DataProtectorTokenProvider<ApplicationUser>(dataProtectionProvider.Create("ASP.NET Identity"));
            }
            return manager;
        }
    }

    // Configure the application sign-in manager which is used in this application.
    public class ApplicationSignInManager : SignInManager<ApplicationUser, string>
    {
        public ApplicationSignInManager(ApplicationUserManager userManager, IAuthenticationManager authenticationManager)
            : base(userManager, authenticationManager)
        {
        }

        public override Task<ClaimsIdentity> CreateUserIdentityAsync(ApplicationUser user)
        {
            return user.GenerateUserIdentityAsync((ApplicationUserManager)UserManager);
        }

        public static ApplicationSignInManager Create(IdentityFactoryOptions<ApplicationSignInManager> options, IOwinContext context)
        {
            return new ApplicationSignInManager(context.GetUserManager<ApplicationUserManager>(), context.Authentication);
        }
    }
}
```

Υλοποίηση του BundleConfig

Το BundleConfig είναι ένα αρχείο το οποίο ορίζει όλα τα εξωτερικά plugins που θα χρησιμοποιηθούν στην εφαρμογή. Για παράδειγμα, εφόσον τα views χρησιμοποιούν κλασσική HTML και CSS, αλλά και JQuery, πρέπει πρώτα να κατεβούν τα αρχεία στο φάκελο που έχει η εφαρμογή και στη συνέχεια να δηλωθούν στο BundleConfig. Στην παρακάτω εικόνα φαίνεται συγκεκριμένα η δήλωση του Bootstrap και του JQuery.

```
4   namespace MvcEmployeeManager
5   {
6       public class BundleConfig
7       {
8           // For more information on bundling, visit https://go.microsoft.com/fwlink/?LinkId=301862
9           public static void RegisterBundles(BundleCollection bundles)
10          {
11              bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
12                          "~/Scripts/jquery-{version}.js"));
13
14              bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
15                          "~/Scripts/jquery.validate*"));
16
17              // Use the development version of Modernizr to develop with and learn from. Then, when you're
18              // ready for production, use the build tool at https://modernizr.com to pick only the tests you need.
19              bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
20                          "~/Scripts/modernizr-*"));
21
22              bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
23                        "~/Scripts/bootstrap.js"));
24
25              bundles.Add(new StyleBundle("~/Content/css").Include(
26                        "~/Content/bootstrap.css",
27                        "~/Content/site.css"));
28          }
29      }
30  }
31
```

Βιβλιογραφία

[i]Create a web app with ASP.NET Core MVC on Windows with Visual Studio: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/?view=aspnetcore-2.1

[ii]MVC Pattern από Microsoft: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.1

[iii]Routing in ASP.NET Core: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/core/fundamentals/routing?view=aspnetcore-2.1

[iv]Add a controller to an ASP.NET Core MVC app: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-controller?view=aspnetcore-2.1

[v]Add a view to an ASP.NET Core MVC app: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-view?view=aspnetcore-2.1

[vi]Add a model to an ASP.NET Core MVC app: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model?view=aspnetcore-2.1

[vvi]Razor Pages in ASP.NET Core: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/razor-pages-start?view=aspnetcore-2.1

[viii]Introduction to Entity Framework: Ανάκτηση από https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx

[ix]Entity Framework Code First Conventions: Ανάκτηση από https://msdn.microsoft.com/en-us/library/jj679962(v=vs.113).aspx

[x]Entity Framework Code First Migrations: Ανάκτηση από https://msdn.microsoft.com/en-us/library/jj591621(v=vs.113).aspx

[xi]Creating a More Complex Data Model for an ASP.NET MVC Application: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-a-more-complex-data-model-for-an-asp-net-mvc-application

[xii]Reading Related Data with the Entity Framework in an ASP.NET MVC Application: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/reading-related-data-with-the-entity-framework-in-an-asp-net-mvc-application

[xiii]Updating Related Data with the Entity Framework in an ASP.NET MVC Application: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/updating-related-data-with-the-entity-framework-in-an-asp-net-mvc-application

[xiv]Introduction to Identity on ASP.NET Core: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.1&tabs=visual-studio%2Caspnetcore2x

[xv]Add custom user data to Identity in an ASP.NET Core project: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/core/security/authentication/add-user-data?view=aspnetcore-2.1&tabs=visual-studio

[xvi]Introduction to authorization in ASP.NET Core: Ανάκτηση από https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction?view=aspnetcore-2.1