ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας	(Ελληνικά) "Διαδικτυακή εφαρμογή περιήγησης ταινιών με χρήση ASP.NET MVC" (Αγγλικά) "Web-based movie browsing application using ASP.NET MVC"
Ονοματεπώνυμο Φοιτητή	Γεώργιος Μεγαλιός
Πατρώνυμο	Νικόλαος
Αριθμός Μητρώου	Π/ 19103
Επιβλέπων	Αλέπης Ευθύμιος, Καθηγητής

Ημερομηνία Παράδοσης: Ιούλιος 2024

Copyright ©

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στα πλαίσια της ολοκλήρωσης των σπουδών μου στο Προπτυχιακό Πρόγραμμα Σπουδών του Τμήματος Πληροφορικής του Πανεπιστημίου Πειραιώς με κατεύθυνση Τεχνολογία Λογισμικού και Ευφυή Συστήματα.

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου τον κ. Ευθύμιο Αλέπη για την εμπιστοσύνη που μου έδειξε αναθέτοντάς μου θέμα διπλωματικής εργασίας και την πολύτιμη υποστήριξη και καθοδήγησή του, από την επιλογή του θέματος μέχρι την ολοκλήρωση της.

Τέλος, θέλω να ευχαριστήσω την οικογένειά μου, για την ανεκτίμητη στήριξή τους, τόσο στην παρούσα εργασία, όσο και σε όλο το διάστημα των σπουδών μου.

Περίληψη

Σκοπός της διπλωματικής εργασίας ήταν η δημιουργία μιας εφαρμογής περιήγησης και εξερεύνησης ταινιών με χρήση των νέων τεχνολογιών ASP.NET MVC. Η εφαρμογή επιτρέπει στους χρήστες να δημιουργήσουν λογαριασμό και στη συνέχεια να αναζητήσουν τις αγαπημένες τους ταινίες από μια συλλογή. Παρέχεται η δυνατότητα αναζήτησης ταινιών με βάση το είδος της ταινίας, τον τίτλο, την μέση βαθμολογία καθώς και με συνδυασμό των παραπάνω. Για κάθε ταινία παρουσιάζονται κάποιες βασικές πληροφορίες όπως τίτλος, είδος, ημερομηνία κυκλοφορίας, σκηνοθέτης, ηθοποιοί και trailer. Οι χρήστες έχουν την δυνατότητα να βαθμολογήσουν μια ταινία ή και να αφήσουν σχόλιο κάτω από αυτή. Η παροχή συστήματος σχολίων και βαθμολόγησης είναι σημαντική καθώς επιτρέπει στους χρήστες να εκφράσουν τις απόψεις τους και να παρέχουν ανατροφοδότηση σχετικά με τις ταινίες, κάτι που μπορεί να βοηθήσει άλλους χρήστες να αποφασίσουν ποιες να παρακολουθήσουν. Επίσης με την ανταλλαγή σχολίων οι χρήστες μπορούν να αλληλεπιδρούν μεταξύ τους, ανταλλάσσοντας απόψεις και συζητώντας για τις ταινίες δημιουργώντας έτσι μια πιο ζωντανή και ενεργή κοινότητα. Εκτός από αναζήτηση ταινιών υπάρχει η δυνατότητα προβολής ενός σύντομου βιογραφικού για τους σκηνοθέτες και ηθοποιούς που έλαβαν μέρος.

Η παρούσα διπλωματική λοιπόν αποτελεί μια πρώτη προσπάθεια στην δημιουργία σελίδας περιήγησης ταινιών παρέχοντας ένα μέρος των λειτουργιών που προσφέρουν οι μεγάλες ιστοσελίδες.

Abstract

The aim of the thesis was to create a movie browsing and exploration application using the new ASP.NET MVC technologies. The application allows users to create an account and then search for their favorite movies from a collection. The ability to search for movies by movie genre, title, average rating and a combination of the above is provided. For each movie some basic information is presented such as title, genre, release date, director, actors and trailer. Users have the option to rate a movie or even leave a comment below it. Providing a comment and rating system is important as it allows users to express their opinions and provide feedback on movies, which can help other users decide which ones to watch. Also by sharing comments users can interact with each other, exchanging views and discussing about the movies, thus creating a more vibrant and active community. In addition to searching for films there is the option to view a short biography of the directors and actors who took part.

This thesis is therefore a first attempt at creating a movie browser page providing some of the functionality offered by major websites.

Περιεχόμενα

1.Εισαγωγή

1.1 Web εφαρμογές

Η ανάπτυξη ***web εφαρμογών*** έχει εξελιχθεί σημαντικά τα τελευταία χρόνια, αξιοποιώντας διάφορες τεχνολογίες και πλατφόρμες για τη δημιουργία δυναμικών και διαδραστικών ιστοσελίδων. Οι web εφαρμογές είναι προγράμματα που λειτουργούν σε ένα διακομιστή και προσφέρουν αλληλεπίδραση με τους χρήστες μέσω ενός προγράμματος περιήγησης. Αυτές οι εφαρμογές χρησιμοποιούν μια ποικιλία τεχνολογιών τόσο στο ***frontend*** όσο και στο ***backend*** για να παρέχουν μια ολοκληρωμένη εμπειρία χρήστη.

Στο ***frontend***, οι βασικές τεχνολογίες περιλαμβάνουν **HTML**, **CSS** και **JavaScript**. Η **HTML (HyperText Markup Language)** αποτελεί τη ραχοκοκαλιά κάθε ιστοσελίδας, προσδιορίζοντας τη δομή και το περιεχόμενό της. Η **CSS (Cascading Style Sheets)** χρησιμοποιείται για τον καθορισμό της εμφάνισης και της διάταξης των στοιχείων της ιστοσελίδας, ενώ η **JavaScript** προσθέτει διαδραστικότητα και δυναμική συμπεριφορά, επιτρέποντας στους προγραμματιστές να δημιουργούν πλούσιες εμπειρίες χρήστη με άμεση απόκριση. Στο ***backend***, οι web εφαρμογές βασίζονται σε διακομιστές και βάσεις δεδομένων για τη διαχείριση δεδομένων και την εκτέλεση επιχειρησιακής λογικής. Οι διακομιστές εφαρμόζουν διάφορες τεχνολογίες και πλατφόρμες όπως **Node.js**, **Django**, **Ruby on Rails** και **ASP.NET** για την ανάπτυξη server-side λογικής. Οι βάσεις δεδομένων, όπως **MySQL**, **PostgreSQL** και **MongoDB**, αποθηκεύουν και ανακτούν δεδομένα που απαιτούνται από την εφαρμογή. Η αλληλεπίδραση μεταξύ του frontend και του backend επιτυγχάνεται μέσω αιτήσεων HTTP, όπου το frontend ζητά δεδομένα ή λειτουργίες από το backend και το backend ανταποκρίνεται με τα απαραίτητα δεδομένα ή αποτελέσματα. Αυτή η διαδικασία επιτρέπει τη δημιουργία δυναμικών σελίδων που μπορούν να ενημερώνονται και να αλληλεπιδρούν με τον χρήστη χωρίς να απαιτείται πλήρης επαναφόρτωση της σελίδας.

Μία από τις πιο ισχυρές και δημοφιλείς τεχνολογίες για την ανάπτυξη web εφαρμογών είναι το **ASP.NET MVC (Model-View-Controller)** της **Microsoft**. Το ASP.NET MVC είναι ένα πλαίσιο ανάπτυξης web εφαρμογών που βασίζεται στην

αρχιτεκτονική MVC, η οποία διαχωρίζει την εφαρμογή σε τρία κύρια συστατικά: το **Μοντέλο (Model)**, την **Προβολή (View)** και τον **Ελεγκτή (Controller)**. Αυτή η αρχιτεκτονική βοηθά στην οργάνωση του κώδικα, καθιστώντας την εφαρμογή πιο ευέλικτη, εύκολη στη συντήρηση και επεκτάσιμη. Το ASP.NET MVC παρέχει ενσωματωμένη υποστήριξη για την ανάπτυξη ισχυρών και κλιμακούμενων web εφαρμογών, προσφέροντας ένα πλούσιο σετ εργαλείων και βιβλιοθηκών που διευκολύνουν την ανάπτυξη τόσο στο frontend όσο και στο backend. Επιπλέον, υποστηρίζει τη χρήση του **Entity Framework** για τη διαχείριση της βάσης δεδομένων, διευκολύνοντας την αλληλεπίδραση με τα δεδομένα και μειώνοντας την ανάγκη για γράψιμο περίπλοκου SQL κώδικα.

Συνοψίζοντας, οι τεχνολογίες ανάπτυξης web εφαρμογών έχουν προοδεύσει σημαντικά, παρέχοντας στους προγραμματιστές τα εργαλεία που χρειάζονται για να δημιουργούν δυναμικές και διαδραστικές εφαρμογές. Το ASP.NET MVC, με την ισχυρή του αρχιτεκτονική και τα πλούσια χαρακτηριστικά του, αποτελεί μια από τις κορυφαίες επιλογές για την ανάπτυξη σύγχρονων web εφαρμογών, συνδυάζοντας τη δύναμη του .NET οικοσυστήματος με την ευελιξία της αρχιτεκτονικής MVC.

1.2 Αντικείμενο διπλωματικής

Αντικείμενο της παρούσας διπλωματικής είναι η ανάπτυξη μιας διαδικτυακής εφαρμογής χρησιμοποιώντας το ASP.NET MVC framework της Microsoft. Η εφαρμογή επικεντρώνεται στη διαχείριση μιας βάσης δεδομένων που αφορά ταινίες, ηθοποιούς και χρήστες, προσφέροντας λειτουργίες όπως εγγραφή και σύνδεση χρηστών, διαχείριση ταινιών και ηθοποιών, καθώς και σύστημα σχολιασμού και βαθμολόγησης ταινιών από τους χρήστες. Η βάση δεδομένων υλοποιείται μέσω του **Microsoft SQL Management Studio**, ενώ η σύνδεση και η διαχείριση της βάσης δεδομένων πραγματοποιούνται μέσω του **Entity Framework**.

Ο σκοπός της εργασίας είναι η κατανόηση και η εφαρμογή των τεχνολογιών και των αρχών ανάπτυξης web εφαρμογών, με έμφαση στην αρχιτεκτονική **Model-View-Controller (MVC)**. Μέσω της ανάπτυξης αυτής της εφαρμογής, επιδιώκεται η εξοικείωση με τη χρήση του ASP.NET MVC framework, η διαχείριση βάσεων δεδομένων, η εφαρμογή πρακτικών ασφαλούς κώδικα και η ανάπτυξη λειτουργιών που εξυπηρετούν τις ανάγκες των χρηστών. Επιπλέον, η εφαρμογή παρέχει ένα

πρακτικό παράδειγμα του πώς μπορεί να δημιουργηθεί και να συντηρηθεί μια πλήρης web εφαρμογή, καλύπτοντας όλο τον κύκλο ζωής της ανάπτυξης, από τη σχεδίαση και την υλοποίηση έως τη δοκιμή και την συντήρηση. Με αυτόν τον τρόπο, η εργασία στοχεύει όχι μόνο στην παροχή μιας λειτουργικής εφαρμογής αλλά και στην ενίσχυση των γνώσεων και δεξιοτήτων στον τομέα της ανάπτυξης web εφαρμογών.

2. Ανάλυση απαιτήσεων

Η εφαρμογή στοχεύει να παρέχει μια ολοκληρωμένη πλατφόρμα για τη διαχείριση και την αξιολόγηση ταινιών, προσφέροντας λειτουργίες όπως η δημιουργία, η επεξεργασία και η διαγραφή χρηστών και ταινιών, καθώς και η υποβολή σχολίων και βαθμολογιών από τους χρήστες.

Λειτουργικές απαιτήσεις:

Οι λειτουργικές απαιτήσεις καθορίζουν τις συγκεκριμένες λειτουργίες που πρέπει να υποστηρίζει η εφαρμογή:

1. **Διαχείριση Χρηστών:** Η εφαρμογή πρέπει να επιτρέπει την εγγραφή νέων χρηστών, τον έλεγχο των στοιχείων τους (όπως email και username για αποφυγή διπλοεγγραφών), και την ανάθεση ρόλων (όπως admin και user).
2. **Διαχείριση Ταινιών:** Η εφαρμογή πρέπει να επιτρέπει στους διαχειριστές (admins) να προσθέτουν, να επεξεργάζονται και να διαγράφουν ταινίες. Κάθε ταινία πρέπει να περιλαμβάνει στοιχεία όπως τίτλο, ημερομηνία κυκλοφορίας, είδος, σκηνοθέτη, περιγραφή και URL αφίσας.
3. **Διαχείριση Ηθοποιών:** Η εφαρμογή πρέπει να υποστηρίζει τη δημιουργία, επεξεργασία και διαγραφή ηθοποιών. Τα δεδομένα των ηθοποιών περιλαμβάνουν το πλήρες όνομα και το URL της φωτογραφίας τους.
4. **Σχόλια και Βαθμολογίες:** Οι χρήστες πρέπει να μπορούν να υποβάλλουν σχόλια και βαθμολογίες για τις ταινίες. Αυτές οι κριτικές πρέπει να συνδέονται με τους λογαριασμούς χρηστών και τις ταινίες, επιτρέποντας την εμφάνιση σχολίων και μέσων όρων βαθμολογιών.
5. **Διασυνδέσεις Ταινιών-Ηθοποιών:** Η εφαρμογή πρέπει να επιτρέπει τη συσχέτιση ταινιών με ηθοποιούς μέσω μιας ενδιάμεσης σχέσης (junction) ταινιών-ηθοποιών (MoviesActors).

Μη Λειτουργικές Απαιτήσεις:

Οι μη λειτουργικές απαιτήσεις καθορίζουν τα πρότυπα ποιότητας και τις συνθήκες λειτουργίας της εφαρμογής:

1. **Απόδοση:** Η εφαρμογή πρέπει να ανταποκρίνεται γρήγορα στις αιτήσεις χρηστών, προσφέροντας ομαλή εμπειρία χρήστη χωρίς καθυστερήσεις. Αυτό απαιτεί αποτελεσματική διαχείριση βάσης δεδομένων και βελτιστοποίηση κώδικα.
2. **Ασφάλεια:** Πρέπει να εξασφαλιστεί η προστασία των δεδομένων των χρηστών και η ασφάλεια της εφαρμογής. Αυτό περιλαμβάνει την προστασία από επιθέσεις όπως ***SQL injection***, τη διασφάλιση της εμπιστευτικότητας των κωδικών πρόσβασης και τη διαχείριση των δικαιωμάτων πρόσβασης.
3. **Επεκτασιμότητα:** Η εφαρμογή πρέπει να σχεδιαστεί ώστε να μπορεί να κλιμακωθεί εύκολα για να εξυπηρετήσει αυξανόμενο αριθμό χρηστών και δεδομένων. Αυτό μπορεί να απαιτεί σχεδιασμό για την κατανομή φορτίου και την υποστήριξη πολλαπλών servers.
4. **Συντηρησιμότητα:** Ο κώδικας της εφαρμογής πρέπει να είναι κατανοητός και επεκτάσιμος. Αυτό περιλαμβάνει τη χρήση καλών πρακτικών προγραμματισμού, όπως σχολιασμός κώδικα, και την τήρηση αρχών του σχεδιασμού λογισμικού.

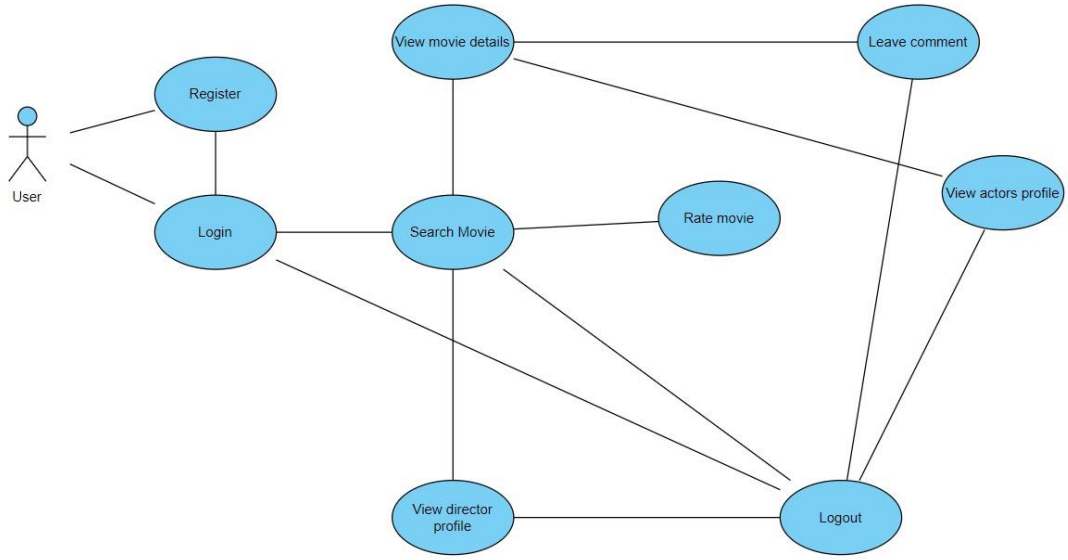
Η ανάλυση αυτών των απαιτήσεων βοηθάει στη δημιουργία μιας ισχυρής και λειτουργικής εφαρμογής που ανταποκρίνεται στις ανάγκες των χρηστών και μπορεί να συντηρηθεί και να επεκταθεί μελλοντικά. Η χρήση του ASP.NET MVC πλαισίου παρέχει τα εργαλεία και τη δομή για την υλοποίηση αυτών των απαιτήσεων, προσφέροντας ένα οργανωμένο και modular σύστημα ανάπτυξης.

2.1 Περιγραφή Λειτουργιών

Στο κεφάλαιο αυτό θα περιγράψουμε τις λειτουργίες που απαιτείται να εκτελεί το σύστημα ώστε να είναι χρήσιμο και ορθά κατασκευασμένο. Αρχικά, το σύστημά μας, όπως προείπαμε και παραπάνω χωρίζεται σε δύο μέρη, στο υποσύστημα των χρηστών, και στο υποσύστημα των διαχειριστών.

2.1.1 Υποσύστημα χρηστών

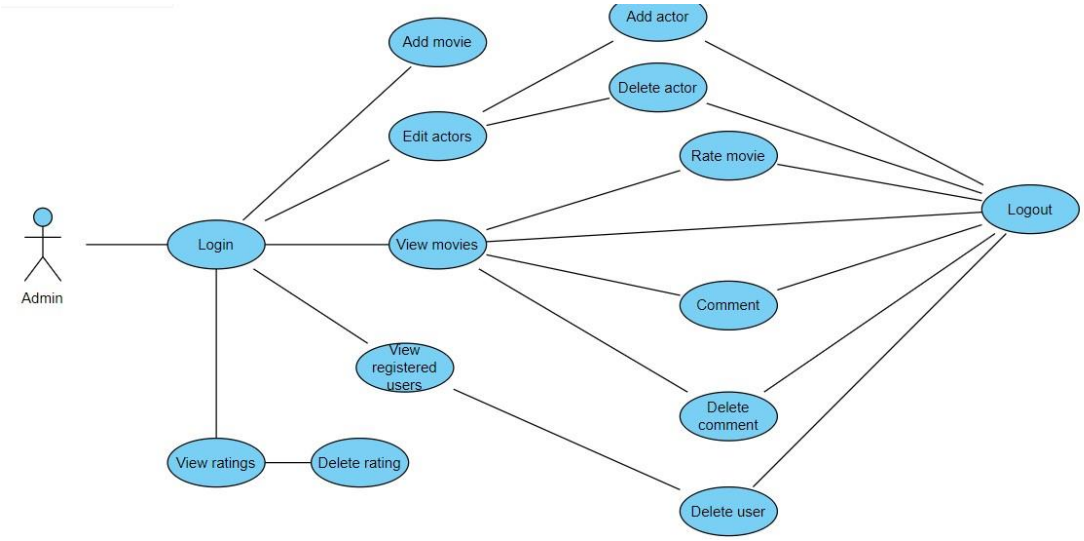
Στο υποσύστημα αυτό, οι χρήστες θα έχουν την δυνατότητα να εγγραφούν στο σύστημα, να συνδεθούν, να αναζητήσουν μια ταινία (με βάση τον τίτλο ή το είδος) , να δουν περισσότερες λεπτομέρειες και trailer της ταινίας, να την βαθμολογήσουν, να αφήσουν σχόλια, καθώς και να δουν το profile του σκηνοθέτη και των ηθοποιών.



Εικόνα 2.1 Διάγραμμα περιπτώσεων χρήστη

2.1.2 Υποσύστημα διαχειριστών

Ο διαχειριστής της εφαρμογής μπορεί να εκτελέσει όλες τις παραπάνω λειτουργίες χρηστών σε συνδυασμό με κάποιες επιπλέον. Συγκεκριμένα έχει τη δυνατότητα προσθήκης, επεξεργασίας καθώς και διαγραφής ταινιών, ηθοποιών και σκηνοθετών, προβολή εγγεγραμμένων χρηστών καθώς και διαγραφής τους. Επιπλέον μπορεί να διαγράψει σχόλια σε περίπτωση που είναι υβριστικά καθώς και να δει αναλυτικές πληροφορίες σχετικά με τις βαθμολογίες χρηστών.



Εικόνα 2.2 Διάγραμμα περιπτώσεων διαχειριστή

3. Τεχνολογίες που χρησιμοποιήθηκαν

3.1 Εισαγωγή στο ASP.NET MVC Framework

Λειτουργίες

Το **ASP.NET MVC (Model-View-Controller)** είναι ένα framework για την ανάπτυξη web εφαρμογών στην πλατφόρμα **.NET** της Microsoft. Χρησιμοποιεί το αρχιτεκτονικό μοντέλο **MVC**, το οποίο διαχωρίζει την εφαρμογή σε τρία κύρια συστατικά: το Model, το View, και το Controller. Αυτή η διάκριση βοηθάει στην καλύτερη οργάνωση του κώδικα, την επαναχρησιμοποίηση των κλάσεων, και την ευκολότερη συντήρηση και επέκταση της εφαρμογής.

Model (Μοντέλο)

Το Model αντιπροσωπεύει την επιχειρηματική λογική και την κατάσταση της εφαρμογής. Είναι υπεύθυνο για την ανάκτηση, αποθήκευση και διαχείριση των δεδομένων. Σε πολλές περιπτώσεις, τα Models συνδέονται με τη βάση δεδομένων και χειρίζονται τις λειτουργίες CRUD επεξεργασίας των πινάκων (Create, Read, Update, Delete).

View (Προβολή)

Το View είναι υπεύθυνο για την εμφάνιση των δεδομένων στον χρήστη. Αποτελείται από τα στοιχεία της διεπαφής χρήστη (UI) και παρουσιάζει τα δεδομένα που λαμβάνει από το Model μέσω του Controller. Το View δεν περιέχει επιχειρηματική λογική, αλλά απλά εμφανίζει τα δεδομένα με βάση τις οδηγίες του Controller.

Controller (Ελεγκτής)

Το Controller ενεργεί ως ενδιάμεσος μεταξύ του Model και του View. Διαχειρίζεται τα αιτήματα των χρηστών, επεξεργάζεται τα δεδομένα με το Model και επιστρέφει τα αποτελέσματα στο View. Το Controller είναι υπεύθυνο για την

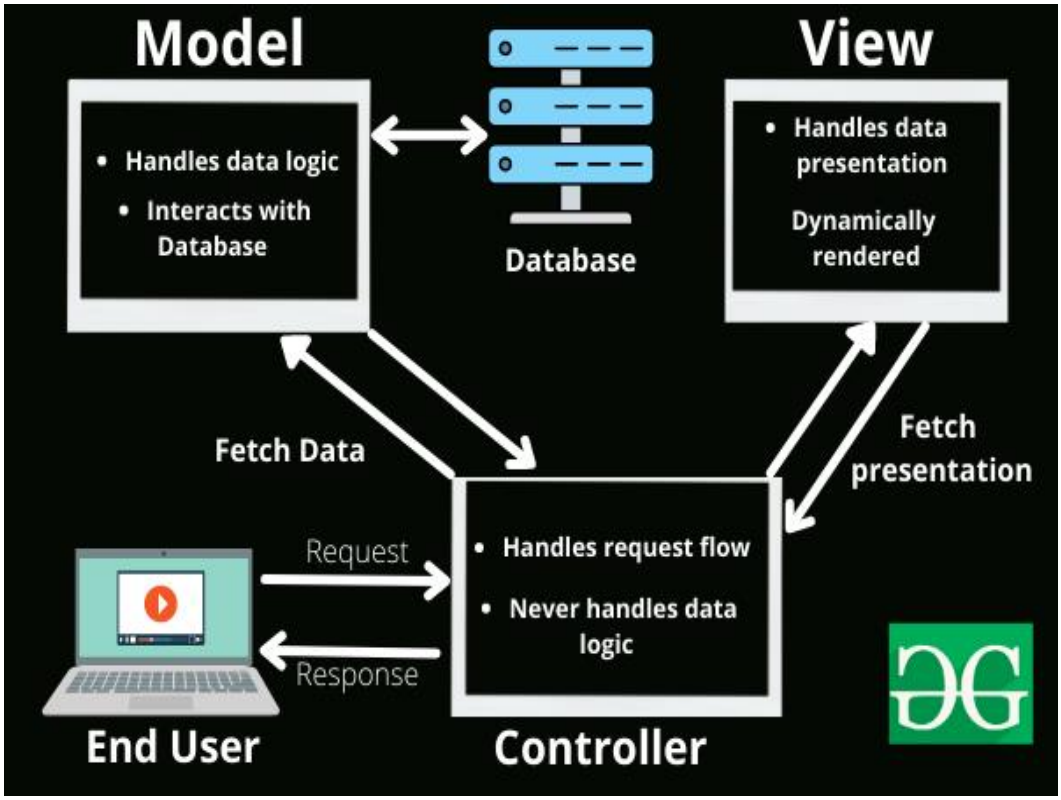
εκτέλεση των εντολών του χρήστη και τον καθορισμό της απάντησης που πρέπει να σταλεί πίσω.

Λειτουργία του ASP.NET MVC Framework

1. **Αίτηση Χρήστη (Request):**
    - Όταν ένας χρήστης κάνει ένα αίτημα μέσω του προγράμματος περιήγησης, αυτό το αίτημα φτάνει στον διακομιστή όπου φιλοξενείται η εφαρμογή ASP.NET MVC.
2. **Δρομολόγηση (Routing):**
    - Η δρομολόγηση (Routing) στο ASP.NET MVC καθορίζει ποιο Controller και ποια μέθοδος αυτού του Controller θα πρέπει να χειριστούν το αίτημα. Οι διαδρομές (Routes) καθορίζονται στο αρχείο RouteConfig.cs ή με χαρακτηριστικά (attributes) στα Controllers.

3. **Εκτέλεση της Μεθόδου του Controller:**
    - Το σύστημα δρομολόγησης χαρτογραφεί το αίτημα στον κατάλληλο Controller και στην αντίστοιχη μέθοδο δράσης (Action Method). Ο Controller επεξεργάζεται το αίτημα, αλληλεπιδρά με το Model για να πάρει ή να ενημερώσει δεδομένα, και επιλέγει το View που θα επιστρέψει στον χρήστη.
4. **Αλληλεπίδραση με το Model:**
    - Ο Controller καλεί τις απαραίτητες μεθόδους του Model για να εκτελέσει την επιχειρηματική λογική, όπως ανάκτηση δεδομένων από τη βάση.
5. **Δημιουργία του View:**
    - Το View λαμβάνει τα δεδομένα από τον Controller και τα προετοιμάζει για την εμφάνιση στον χρήστη. Το View είναι γραμμένο σε Razor syntax (συνήθως) που επιτρέπει την ενσωμάτωση C# κώδικα με HTML.
6. **Απόκριση Χρήστη (Response):**

- Τέλος, το View επιστρέφεται στον διακομιστή, το οποίο δημιουργεί το τελικό HTML και το στέλνει πίσω στο πρόγραμμα περιήγησης του χρήστη.



Εικόνα 3.1 Το μοντέλο MVC

Κύρια Πλεονεκτήματα του ASP.NET MVC

- **Διαχωρισμός ανησυχιών (Separation of Concerns):** Κάνει την εφαρμογή πιο οργανωμένη και ευκολότερη στη συντήρηση.
- **Ευκολία στη δοκιμή (Testability):** Επιτρέπει την εύκολη μονάδα ελέγχου (unit testing) της επιχειρηματικής λογικής.
- **Πλήρης έλεγχος του HTML και του URL:** Δίνει μεγαλύτερο έλεγχο στην κατασκευή του HTML και την δομή των URLs.
- **Ενσωμάτωση με το ASP.NET:** Συνδυάζεται με άλλα χαρακτηριστικά του ASP.NET όπως η ασφάλεια, η εξουσιοδότηση, και η διαχείριση καταστάσεων (state management).

Το ASP.NET MVC είναι ένα ισχυρό framework που επιτρέπει την ανάπτυξη ευέλικτων και επεκτάσιμων web εφαρμογών, παρέχοντας παράλληλα μια καθαρή διάκριση μεταξύ της επιχειρηματικής λογικής, της διεπαφής χρήστη, και της διαχείρισης των δεδομένων. Ας δούμε τώρα λίγο πιο αναλυτικά τα κύρια στοιχεία του.

3.1.1 Το μοντέλο

Το Model στο πλαίσιο του ASP.NET MVC framework είναι η καρδιά της εφαρμογής και έχει πολλές κρίσιμες λειτουργίες που σχετίζονται με την επιχειρηματική λογική και τη διαχείριση των δεδομένων. Αναλυτικά, το Model αντιπροσωπεύει τις **δομές των δεδομένων** και τις **επιχειρηματικές διαδικασίες** που διέπουν τη λειτουργία της εφαρμογής.

Η πρώτη βασική λειτουργία του Model είναι να παρέχει μια αναπαράσταση των δεδομένων που χειρίζεται η εφαρμογή. Αυτό επιτυγχάνεται μέσω της δημιουργίας κλάσεων που περιγράφουν τα αντικείμενα της εφαρμογής. Για παράδειγμα, στη συγκεκριμένη εφαρμογή περιήγησης ταινιών, το Model περιλαμβάνει μια κλάση Users που περιέχει ιδιότητες όπως username, email, password. Αυτές οι κλάσεις συχνά χαρτογραφούνται σε πίνακες μιας βάσης δεδομένων, επιτρέποντας έτσι την εύκολη αποθήκευση και ανάκτηση των δεδομένων.

Εκτός από τη δομή των δεδομένων, το Model είναι υπεύθυνο για την επιχειρηματική λογική της εφαρμογής. Αυτό σημαίνει ότι το Model περιέχει τους κανόνες και τις διαδικασίες που καθορίζουν πώς πρέπει να επεξεργάζονται τα δεδομένα. Αυτή η επιχειρηματική λογική μπορεί να περιλαμβάνει ελέγχους επικύρωσης (validation), όπως το να βεβαιωθεί ότι το email ενός χρήστη είναι έγκυρο (περιέχει @) ή ότι το password είναι μοναδικό. Μπορεί επίσης να περιλαμβάνει πολύπλοκους υπολογισμούς και άλλες διαδικασίες που είναι ζωτικής σημασίας για τη λειτουργία της εφαρμογής.

Μια άλλη σημαντική λειτουργία του Model είναι η αλληλεπίδραση με τη βάση δεδομένων. Στο ASP.NET MVC, αυτή η αλληλεπίδραση συχνά επιτυγχάνεται μέσω του **Entity Framework**, ένα **ORM (Object-Relational Mapping)** εργαλείο που επιτρέπει την εύκολη χαρτογράφηση των κλάσεων του Model σε πίνακες της βάσης δεδομένων. Το Entity Framework παρέχει μηχανισμούς για την εκτέλεση λειτουργιών **CRUD** (Create, Read, Update, Delete) χωρίς την ανάγκη για γραφή SQL

ερωτημάτων. Για παράδειγμα, για να ανακτηθεί μια λίστα χρηστών από τη βάση δεδομένων, μπορεί να χρησιμοποιηθεί μια μέθοδος όπως context.Users.ToList(), όπου context είναι το αντικείμενο του DbContext που διαχειρίζεται τη σύνδεση με τη βάση δεδομένων.

Το Model πρέπει επίσης να είναι ανεξάρτητο από το Controller και το View. Αυτό σημαίνει ότι η επιχειρηματική λογική και η διαχείριση των δεδομένων θα πρέπει να βρίσκονται εξ ολοκλήρου στο Model και όχι να εξαρτώνται από τον τρόπο με τον οποίο τα δεδομένα παρουσιάζονται ή πώς οι χρήστες αλληλεπιδρούν με την εφαρμογή. Αυτός ο διαχωρισμός καθιστά τον κώδικα πιο οργανωμένο και ευκολότερο στη συντήρηση, διευκολύνοντας την ανάπτυξη και τη δοκιμή της εφαρμογής.

Τα παραπάνω καθιστούν το μοντέλο ένα κρίσιμο συστατικό που χειρίζεται την αναπαράσταση, τη διαχείριση και την επεξεργασία των δεδομένων της εφαρμογής. Παρέχει τις κλάσεις που αντιπροσωπεύουν τα δεδομένα, περιέχει την επιχειρηματική λογική και αλληλεπιδρά με τη βάση δεδομένων, διασφαλίζοντας έτσι την ομαλή λειτουργία της εφαρμογής και επιτρέποντας την εύκολη διαχείριση και συντήρηση του κώδικα.

3.1.2 Η προβολή

Η προβολή (View) στο πλαίσιο του ASP.NET MVC είναι ένα θεμελιώδες συστατικό που **ασχολείται με την παρουσίαση των δεδομένων στον χρήστη**. Ενώ το Model ασχολείται με τη διαχείριση και την επεξεργασία των δεδομένων και το Controller με τη διαμεσολάβηση μεταξύ των δεδομένων και της προβολής τους, το View επικεντρώνεται αποκλειστικά στην εμφάνιση των δεδομένων με τρόπο που να είναι κατανοητός και χρήσιμος για τον χρήστη.

Όταν ένας χρήστης κάνει ένα αίτημα προς την εφαρμογή, το Controller λαμβάνει αυτό το αίτημα, επεξεργάζεται τα απαραίτητα δεδομένα μέσω του Model και στη συνέχεια καθορίζει ποιο View θα χρησιμοποιηθεί για την εμφάνιση των αποτελεσμάτων. Το View είναι υπεύθυνο για τη δημιουργία του HTML που θα σταλεί πίσω στον φυλλομετρητή του χρήστη. Αυτό σημαίνει ότι το View περιλαμβάνει τη δομή, τη μορφοποίηση και το περιεχόμενο της σελίδας που θα δει ο χρήστης. Στο ASP.NET MVC, η προβολή συνήθως δημιουργείται με τη χρήση του

Razor view engine. Το Razor είναι μια σύγχρονη και ευέλικτη συντακτική δομή που επιτρέπει την ενσωμάτωση κώδικα **C#** μέσα σε **HTML**. Αυτό επιτρέπει στους προγραμματιστές να δημιουργούν δυναμικές σελίδες που μπορούν να προσαρμόζονται στα δεδομένα που επιστρέφει το Controller. Για παράδειγμα, εάν το Controller επιστρέφει μια λίστα ταινιών, το View μπορεί να περιέχει έναν βρόχο foreach που διατρέχει αυτή τη λίστα και δημιουργεί τα απαραίτητα στοιχεία HTML για να εμφανίσει κάθε ταινία.

Η χρήση του Razor κάνει την ανάπτυξη των Views ευκολότερη και πιο αποτελεσματική, καθώς επιτρέπει τη χρήση της πλήρους ισχύος της γλώσσας C# για την επεξεργασία και την εμφάνιση των δεδομένων. Τα Views μπορούν επίσης να χρησιμοποιούν **μερικές προβολές** (**partial views**) για να διαχειρίζονται και να οργανώνουν καλύτερα τα διάφορα τμήματα μιας σελίδας. Οι μερικές προβολές είναι επαναχρησιμοποιήσιμα τμήματα μιας σελίδας που μπορούν να περιλαμβάνουν στοιχεία όπως φόρμες, λίστες ή άλλα κοινά UI στοιχεία, διευκολύνοντας έτσι την ανάπτυξη και τη συντήρηση της εφαρμογής.

Ένα άλλο σημαντικό χαρακτηριστικό του View είναι ότι μπορεί να περιλαμβάνει JavaScript, CSS, και άλλες τεχνολογίες web για να βελτιώσει την αλληλεπίδραση και την εμπειρία του χρήστη. Το View μπορεί να φορτώσει και να εκτελέσει scripts για την επεξεργασία των δεδομένων στην πλευρά του πελάτη, να εφαρμόσει στυλ για να βελτιώσει την εμφάνιση της σελίδας, και να χρησιμοποιήσει AJAX για να πραγματοποιήσει ασύγχρονες αιτήσεις προς τον διακομιστή, βελτιώνοντας έτσι την αλληλεπίδραση του χρήστη με την εφαρμογή.

Επίσης, το View πρέπει να διαχειρίζεται την ευκολία χρήσης (usability) και την προσβασιμότητα (accessibility). Αυτό σημαίνει ότι τα Views πρέπει να σχεδιάζονται με τρόπο που να είναι εύχρηστα για όλους τους χρήστες. Οι προγραμματιστές πρέπει να διασφαλίζουν ότι το HTML που δημιουργείται είναι συμμορφούμενο με τα πρότυπα προσβασιμότητας και ότι παρέχονται εναλλακτικά κείμενα για εικόνες, κατάλληλοι τίτλοι για φόρμες και άλλα στοιχεία που διευκολύνουν τη χρήση της εφαρμογής από όλους τους χρήστες.

3.1.2.1 Χρησιμότητα html και css στις προβολές

HTML (Hypertext Markup Language) και CSS (Cascading Style Sheets) είναι δύο βασικές τεχνολογίες που χρησιμοποιούνται για την κατασκευή και την παρουσίαση ιστοσελίδων. HTML χρησιμοποιείται για τη δημιουργία της δομής και του περιεχομένου μιας ιστοσελίδας, ενώ CSS χρησιμοποιείται για τη διαμόρφωση και την αισθητική της εμφάνιση.

HTML είναι η γλώσσα που επιτρέπει στους προγραμματιστές να καθορίζουν τη δομή των ιστοσελίδων χρησιμοποιώντας στοιχεία και ετικέτες. Κάθε ετικέτα HTML αντιπροσωπεύει ένα συγκεκριμένο στοιχείο της σελίδας, όπως παραγράφους, τίτλους, λίστες, συνδέσμους, εικόνες, και φόρμες. Για παράδειγμα, η ετικέτα <h1> χρησιμοποιείται για τίτλους πρώτου επιπέδου, ενώ η ετικέτα <p> χρησιμοποιείται για παραγράφους. Η HTML αποτελεί τη ραχοκοκαλιά κάθε ιστοσελίδας, καθορίζοντας πώς θα παρουσιαστεί το περιεχόμενο στον χρήστη.

CSS, από την άλλη πλευρά, χρησιμοποιείται για να καθορίσει το στυλ και την εμφάνιση των στοιχείων που έχουν οριστεί με HTML. Με CSS, οι προγραμματιστές μπορούν να εφαρμόσουν χρώματα, γραμματοσειρές, αποστάσεις, διαστάσεις, και άλλες μορφοποιήσεις στα στοιχεία της σελίδας. Για παράδειγμα, με την ιδιότητα color μπορούμε να αλλάξουμε το χρώμα του κειμένου, ενώ με την ιδιότητα margin μπορούμε να ορίσουμε τα περιθώρια γύρω από ένα στοιχείο. Η CSS επιτρέπει επίσης τη δημιουργία σύνθετων διατάξεων και την προσαρμογή της εμφάνισης της σελίδας ανάλογα με το μέγεθος της οθόνης ή άλλες παραμέτρους.

Στο πλαίσιο της ανάπτυξης web εφαρμογών με ASP.NET MVC, HTML και CSS χρησιμοποιούνται κυρίως στα Views για την κατασκευή και την παρουσίαση των σελίδων της εφαρμογής. Τα Views είναι υπεύθυνα για την απόδοση του περιεχομένου που αποστέλλεται στον χρήστη από τον διακομιστή. Με τη χρήση των Razor syntax και των βοηθητικών μεθόδων του ASP.NET, δημιουργούνται δυναμικά περιεχόμενα HTML που ενσωματώνουν δεδομένα από τα μοντέλα της εφαρμογής. Για παράδειγμα, σε ένα View που περιλαμβάνει φόρμα εγγραφής χρήστη, HTML χρησιμοποιείται για να δημιουργηθούν τα πεδία εισαγωγής, τα κουμπιά, και οι ετικέτες, ενώ CSS χρησιμοποιείται για να διαμορφωθεί η εμφάνιση της φόρμας, καθορίζοντας τα χρώματα, τα περιθώρια, και τις γραμματοσειρές. Επιπλέον, με τη χρήση των βοηθητικών μεθόδων του ASP.NET, όπως Html.TextBoxFor και

Html.ValidationMessageFor, μπορούμε να συνδέσουμε τα πεδία εισαγωγής με τις ιδιότητες του μοντέλου και να προσθέσουμε λειτουργίες επικύρωσης.

3.1.3 Η προβολή Layout

Το αρχείο **layout.cshtml** στο ASP.NET MVC αποτελεί ένα βασικό αρχείο διάταξης (layout file) το οποίο καθορίζει τη γενική δομή και το στυλ που θα χρησιμοποιηθεί σε όλες τις σελίδες της εφαρμογής. Το layout.cshtml χρησιμοποιείται για να καθορίσει τα κοινά στοιχεία της διεπαφής χρήστη, όπως την κεφαλίδα, την υποσέλιδα, τα μενού πλοήγησης και άλλες σταθερές ενότητες που εμφανίζονται σε κάθε σελίδα της εφαρμογής.

Το περιεχόμενο του αρχείου layout.cshtml συνήθως ξεκινά με τη δήλωση της διάταξης του HTML εγγράφου, περιλαμβάνοντας τις ετικέτες <html>, <head>, και <body>. Στην ενότητα <head>, περιλαμβάνονται μεταδεδομένα όπως η κωδικοποίηση του εγγράφου, ο τίτλος της σελίδας, καθώς και σύνδεσμοι προς εξωτερικά αρχεία CSS και JavaScript, που είναι απαραίτητα για το στυλ και τη λειτουργικότητα της εφαρμογής. Μέσα στο σώμα του HTML εγγράφου, το αρχείο layout.cshtml περιλαμβάνει την κεφαλίδα της σελίδας (header), η οποία μπορεί να περιέχει το λογότυπο της εφαρμογής και το κύριο μενού πλοήγησης. Αυτά τα στοιχεία εμφανίζονται σε κάθε σελίδα της εφαρμογής και διευκολύνουν τους χρήστες να πλοηγηθούν στα διάφορα μέρη της εφαρμογής. Το αρχείο χρησιμοποιεί επίσης το χαρακτηριστικό @RenderBody(), το οποίο είναι ένας placeholder που καθορίζει πού θα εισαχθεί το δυναμικό περιεχόμενο των επιμέρους σελίδων. Όταν μια συγκεκριμένη σελίδα προβληθεί, το περιεχόμενό της θα εισαχθεί στη θέση του @RenderBody() στο layout.cshtml, διατηρώντας έτσι τη συνοχή της διάταξης σε όλη την εφαρμογή.

Το υποσέλιδο (footer) της σελίδας περιλαμβάνεται συνήθως στο κάτω μέρος του αρχείου και μπορεί να περιέχει πρόσθετες πληροφορίες, όπως δικαιώματα πνευματικής ιδιοκτησίας, συνδέσμους προς σελίδες πολιτικής απορρήτου και όρους χρήσης. Επιπλέον, το αρχείο layout.cshtml μπορεί να περιέχει άλλα placeholder όπως το @RenderSection(), που επιτρέπουν την εισαγωγή προαιρετικού περιεχομένου σε συγκεκριμένες περιοχές της διάταξης. Για παράδειγμα, μια ενότητα μπορεί να

χρησιμοποιηθεί για την εισαγωγή πρόσθετων scripts ή στυλ που χρειάζονται μόνο σε συγκεκριμένες σελίδες.

Συνολικά, το layout.cshtml λειτουργεί ως το κεντρικό σημείο για τον καθορισμό της βασικής εμφάνισης και αίσθησης της εφαρμογής, διασφαλίζοντας ότι όλες οι σελίδες μοιράζονται μια συνεκτική δομή και στυλ, διευκολύνοντας έτσι τη συντήρηση και την αναβάθμιση της εφαρμογής. Ουσιαστικά το Layout καθορίζει την μορφοποίηση της σελίδας πάνω στην οποία στηρίζεται η προβολή για την εμφάνιση των δεδομένων στο χρήστη. Στην εφαρμογή αυτή χρησιμοποιήθηκαν δύο Layout. Το **'_Layout.cshtml'** που είναι το **αρχικό layout** που δημιουργείται αυτόματα κατά την εκκίνηση της εφαρμογής και πάνω σε αυτό εμφανίζονται οι προβολές των φορμών σύνδεσης και εγγραφής και το **'_LayoutLogin.cshtml'** που καθορίζει το **background** της κεντρικής σελίδας της εφαρμογής μετά την είσοδο.

3.1.4 Ο ελεγκτής

Ο **ελεγκτής (Controller)** στο πλαίσιο του ASP.NET MVC είναι το στοιχείο που συντονίζει τη διαχείριση των αιτημάτων των χρηστών και την παροχή των κατάλληλων απαντήσεων. Ο ελεγκτής λειτουργεί ως διαμεσολαβητής μεταξύ του Model και του View, λαμβάνοντας τα αιτήματα από τους χρήστες, επεξεργάζοντας τα δεδομένα μέσω του Model και καθορίζοντας ποια προβολή (View) θα χρησιμοποιηθεί για να παρουσιαστούν τα αποτελέσματα.

Όταν ένας χρήστης υποβάλλει ένα αίτημα, όπως το άνοιγμα μιας ιστοσελίδας ή την υποβολή μιας φόρμας, το αίτημα αυτό φτάνει στον διακομιστή και ανατίθεται σε έναν συγκεκριμένο ελεγκτή βάσει της διαδρομής URL που έχει οριστεί στο σύστημα δρομολόγησης (routing). Το routing σύστημα του ASP.NET MVC χρησιμοποιεί διαμορφώσεις (configuration) που χαρτογραφούν τις διαδρομές URL σε συγκεκριμένους ελεγκτές και μεθόδους δράσης (action methods). Οι ελεγκτές είναι κλάσεις C# που κληρονομούν από την κλάση Controller. Κάθε μέθοδος σε έναν ελεγκτή είναι μια action method που μπορεί να χειριστεί συγκεκριμένα αιτήματα. Όταν το αίτημα χαρτογραφείται σε μια action method, αυτή η μέθοδος εκτελείται. Οι action methods είναι υπεύθυνες για την εκτέλεση της επιχειρηματικής λογικής ή την

κλήση μεθόδων του Model για την αλληλεπίδραση με τη βάση δεδομένων ή άλλες πηγές δεδομένων.

Για παράδειγμα, αν ένας χρήστης ζητήσει μια λίστα ταινιών, ο ελεγκτής μπορεί να καλέσει το Model για να ανακτήσει τη λίστα των ταινιών από τη βάση δεδομένων. Αφού λάβει τα δεδομένα, ο ελεγκτής αποφασίζει ποιο View θα χρησιμοποιηθεί για την εμφάνιση της λίστας στον χρήστη. Η μέθοδος μπορεί να επιστρέψει μια προβολή χρησιμοποιώντας τη μέθοδο View(), παρέχοντας τα δεδομένα ως μοντέλο για την προβολή. Ο ελεγκτής μπορεί επίσης να επιστρέψει άλλα αποτελέσματα, όπως JSON δεδομένα για AJAX αιτήματα ή ανακατευθύνσεις σε άλλες διευθύνσεις URL.

Επιπλέον, οι ελεγκτές μπορούν να χειρίζονται διαφορετικούς τύπους αιτημάτων HTTP, όπως GET, POST, PUT και DELETE. Για παράδειγμα, μια μέθοδος που χειρίζεται ένα POST αίτημα μπορεί να επεξεργαστεί τα δεδομένα που υποβλήθηκαν από μια φόρμα, να επικυρώσει τα δεδομένα, να ενημερώσει τη βάση δεδομένων μέσω του Model και να επιστρέψει μια απάντηση, όπως την ανακατεύθυνση σε μια σελίδα επιβεβαίωσης ή την επιστροφή σελίδας με μηνύματα σφαλμάτων αν η επικύρωση αποτύχει.

Οι ελεγκτές επίσης διαχειρίζονται τις καταστάσεις εξαίρεσης και σφαλμάτων. Μπορούν να περιλαμβάνουν λογική χειρισμού εξαιρέσεων για να διασφαλίσουν ότι τα σφάλματα αντιμετωπίζονται καταλλήλως και ότι οι χρήστες λαμβάνουν κατανοητά μηνύματα σφάλματος όταν κάτι πάει στραβά. Ο σωστός χειρισμός εξαιρέσεων είναι κρίσιμος για τη διατήρηση της σταθερότητας και της ασφάλειας της εφαρμογής.

Τέλος, οι ελεγκτές μπορούν να χρησιμοποιούν χαρακτηριστικά (attributes) για να καθορίζουν πολιτικές ασφάλειας και εξουσιοδότησης. Χρησιμοποιώντας χαρακτηριστικά όπως [Authorize], οι ελεγκτές μπορούν να περιορίσουν την πρόσβαση σε συγκεκριμένες action methods μόνο σε εξουσιοδοτημένους χρήστες. Αυτό επιτρέπει την εφαρμογή συγκεκριμένων πολιτικών ασφάλειας σε ολόκληρη την εφαρμογή ή σε συγκεκριμένα τμήματα αυτής.

3.2 Λίγα λόγια για την C#

Η **C#** (προφέρεται **"C sharp"**) είναι μια σύγχρονη, αντικειμενοστραφής γλώσσα προγραμματισμού που αναπτύχθηκε από τη Microsoft ως μέρος της πλατφόρμας .NET. Από την εισαγωγή της στις αρχές της δεκαετίας του 2000, η C# έχει καταστεί

μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού για την ανάπτυξη εφαρμογών σε διάφορους τομείς, συμπεριλαμβανομένων των desktop εφαρμογών, των web εφαρμογών, των mobile εφαρμογών, και των παιχνιδιών. Ένα από τα κύρια χαρακτηριστικά της C# είναι η ευκολία χρήσης της σε συνδυασμό με την ισχύ και την ευελιξία που προσφέρει. Η γλώσσα παρέχει ισχυρή υποστήριξη για την αντικειμενοστραφή προγραμματιστική φιλοσοφία, γεγονός που την καθιστά ιδανική για την ανάπτυξη πολύπλοκων και μεγάλης κλίμακας εφαρμογών. Επιπλέον, η C# ενσωματώνει πολλές προηγμένες δυνατότητες, όπως γενικές κλάσεις (generics), ασύγχρονο προγραμματισμό (async/await), και σύνθετες εκφράσεις lambda.

Η C# είναι η βασική γλώσσα που χρησιμοποιείται για την ανάπτυξη των μοντέλων, των ελεγκτών, και των προβολών στο ASP.NET MVC framework. Η χρήση της C# σε συνδυασμό με το ASP.NET MVC framework παρέχει μια ισχυρή και ευέλικτη πλατφόρμα για την ανάπτυξη εφαρμογών. Η ισχυρή τυποποίηση της γλώσσας, μαζί με τις προηγμένες δυνατότητες του .NET framework, επιτρέπουν την ανάπτυξη ασφαλών, επεκτάσιμων, και συντηρήσιμων εφαρμογών. Επιπλέον, η υποστήριξη της Microsoft και η ενεργή κοινότητα προγραμματιστών συμβάλλουν στη συνεχή βελτίωση και την ενσωμάτωση των νεότερων τεχνολογιών και προτύπων στην ανάπτυξη web εφαρμογών.

4.Βάση δεδομένων

4.1 Χρησιμότητα

Η σημασία του να βασίζονται τα μοντέλα μιας εφαρμογής σε μια βάση δεδομένων είναι πολυδιάστατη και αποτελεί θεμελιώδες στοιχείο για την ανάπτυξη λειτουργικών και αποδοτικών εφαρμογών. Στην περίπτωση της εφαρμογής μας, η οποία διαχειρίζεται χρήστες, ταινίες, ηθοποιούς, σχόλια και αξιολογήσεις, η χρήση μιας βάσης δεδομένων προσφέρει μια σειρά από πλεονεκτήματα που διασφαλίζουν τη συνολική ποιότητα, απόδοση και συντήρηση.

Αρχικά, η βάση δεδομένων παρέχει ένα σταθερό και αξιόπιστο μέσο αποθήκευσης. Τα δεδομένα που σχετίζονται με χρήστες, ταινίες και άλλες οντότητες αποθηκεύονται σε πίνακες, οι οποίοι διασφαλίζουν την ακεραιότητα και τη συνοχή των πληροφοριών. Η χρήση πρωτευόντων και ξένων κλειδιών (***primary and foreign keys***) επιτρέπει τη δημιουργία σχέσεων μεταξύ των πινάκων, γεγονός που επιτρέπει τη σύνδεση και την προβολή δεδομένων μεταξύ διαφορετικών οντοτήτων. Για παράδειγμα, τα σχόλια και οι αξιολογήσεις συνδέονται με συγκεκριμένους χρήστες και ταινίες μέσω των ξένων κλειδιών.

Επιπροσθέτως, η βάση δεδομένων επιτρέπει την εύκολη διαχείριση μεγάλων όγκων δεδομένων. Καθώς η εφαρμογή αυξάνεται και προσελκύει περισσότερους χρήστες, ο όγκος των δεδομένων θα αυξάνεται αναλόγως. Η βάση δεδομένων είναι σχεδιασμένη να χειρίζεται μεγάλους όγκους δεδομένων αποτελεσματικά, παρέχοντας δυνατότητες όπως η ευρετηρίαση (***indexing***) και η βελτιστοποίηση ερωτημάτων (***query optimization***) για την ταχεία ανάκτηση των δεδομένων.

Επιπλέον, η χρήση μιας βάσης δεδομένων ενισχύει την ασφάλεια των δεδομένων. Οι βάσεις δεδομένων παρέχουν μηχανισμούς ελέγχου πρόσβασης (***access control***), οι οποίοι διασφαλίζουν ότι μόνο εξουσιοδοτημένοι χρήστες και διαδικασίες μπορούν να έχουν πρόσβαση ή να τροποποιούν τα δεδομένα. Για παράδειγμα, οι διαχειριστές μπορούν να έχουν δικαιώματα διαχείρισης της βάσης δεδομένων, ενώ οι απλοί χρήστες μπορούν να έχουν περιορισμένα δικαιώματα για την υποβολή σχολίων και αξιολογήσεων.

Η βάση δεδομένων προσφέρει επίσης δυνατότητες για την εξασφάλιση της ακεραιότητας των δεδομένων μέσω περιορισμών (***constraints***) και επικυρώσεων

(***validations***). Για παράδειγμα, οι έλεγχοι (CHECK constraints) που έχουν οριστεί στις στήλες των πινάκων που θα δούμε παρακάτω, όπως το role στον πίνακα Users και το rating στον πίνακα Rating, διασφαλίζουν ότι τα δεδομένα που εισάγονται είναι έγκυρα και συμβατά με τους κανόνες της εφαρμογής.

Η συντήρηση και η επέκταση της εφαρμογής βελτιώνονται επίσης όταν τα μοντέλα βασίζονται σε μια βάση. Καθώς η εφαρμογή εξελίσσεται, μπορεί να χρειαστεί να προστεθούν νέες λειτουργίες ή να τροποποιηθούν οι υπάρχουσες. Η δομή της βάσης δεδομένων μπορεί να επεκταθεί με την προσθήκη νέων πινάκων ή την τροποποίηση των υπαρχόντων, διασφαλίζοντας ότι η εφαρμογή μπορεί να προσαρμοστεί στις νέες απαιτήσεις χωρίς να επηρεάζονται τα υπάρχοντα δεδομένα.

Τέλος, η χρήση μιας βάσης δεδομένων διευκολύνει την ανάπτυξη και τη συντήρηση της εφαρμογής από την πλευρά του προγραμματιστή. Τα μοντέλα που βασίζονται στη βάση δεδομένων μπορούν να δημιουργηθούν αυτόματα μέσω εργαλείων όπως το ***scaffolding*** (μέσω αυτού δημιουργήθηκε η εφαρμογή), μειώνοντας τον χρόνο και την προσπάθεια που απαιτείται για την ανάπτυξη της εφαρμογής. Αυτό επιτρέπει στους προγραμματιστές να επικεντρωθούν περισσότερο στην επιχειρηματική λογική και τις λειτουργικές απαιτήσεις της εφαρμογής, αντί να ανησυχούν για τις λεπτομέρειες της διαχείρισης των δεδομένων.

4.2 Σχεδίαση της βάσης

Η εφαρμογή περιήγησης ταινιών δημιουργήθηκε με τη χρήση του **Visual studio** επιλέγοντας την ανάπτυξη κώδικα σε **ASP.NET MVC** από το μενού του launcher. Το project συνδέθηκε με βάση δεδομένων που αναπτύχτηκε στο εργαλείο **Microsoft Sql Management Studio**. Το Microsoft SQL Server Management Studio (SSMS) είναι ένα ολοκληρωμένο περιβάλλον διαχείρισης που παρέχεται από τη Microsoft για τη διαχείριση των SQL Server βάσεων δεδομένων. Το SSMS παρέχει μια εύχρηστη γραφική διεπαφή για την εκτέλεση διαφόρων εργασιών, όπως η δημιουργία και η διαχείριση βάσεων δεδομένων, η εκτέλεση ερωτημάτων SQL, η ρύθμιση ασφαλείας, και η δημιουργία αντιγράφων ασφαλείας και αποκατάστασης. Η συγκεκριμένη βάση δεδομένων περιλαμβάνει διάφορους πίνακες που σχετίζονται με τη διαχείριση χρηστών, ταινιών, ηθοποιών, σχολίων και αξιολογήσεων.

Δημιουργία Πίνακα Χρηστών (Users)

Ο πίνακας Users περιέχει πληροφορίες για τους χρήστες της πλατφόρμας. Οι στήλες περιλαμβάνουν το μοναδικό αναγνωριστικό του χρήστη (userid), το όνομα χρήστη (username), το email (email), τον κωδικό πρόσβασης (password) και τον ρόλο του χρήστη (role). Ο ρόλος του χρήστη μπορεί να είναι είτε 'admin' είτε 'user', όπως ορίζεται από τον έλεγχο (CHECK constraint) που εφαρμόζεται στη στήλη role.

```
CREATE TABLE Users (
    userid INT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(50) CHECK (role IN ('admin', 'user')) NOT NULL
);
```

Δημιουργία Πίνακα Ταινιών (Movies)

Ο πίνακας Movies αποθηκεύει πληροφορίες για τις ταινίες. Περιλαμβάνει το μοναδικό αναγνωριστικό της ταινίας (movieid), τον τίτλο (title), την ημερομηνία κυκλοφορίας (release_date), το είδος (genre), τον σκηνοθέτη (director), την περιγραφή (description) και το URL της αφίσας (posturl).

```
CREATE TABLE Movies (
    movieid INT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    release_date DATE NOT NULL,
    genre VARCHAR(100),
    director VARCHAR(255),
    description VARCHAR(255),
    posturl VARCHAR(255)
);
```

Δημιουργία Πίνακα Ηθοποιών (Actors)

Ο πίνακας Actors περιλαμβάνει πληροφορίες για τους ηθοποιούς. Περιέχει το μοναδικό αναγνωριστικό του ηθοποιού (actorid), το πλήρες όνομα (fullname) και το URL της φωτογραφίας του ηθοποιού (photourl). Αποθηκεύοντας το url στη βάση γίνεται ευκολότερη η ανάκτηση της φωτογραφίας του ηθοποιού μαζί με τα υπόλοιπα στοιχεία

```
CREATE TABLE Actors (
    actorid INT PRIMARY KEY,
    fullname VARCHAR(255) NOT NULL,
    photourl VARCHAR(255) NOT NULL

);
```

Δημιουργία Πίνακα Σύνδεσης Ταινιών και Ηθοποιών (MoviesActors)

Ο πίνακας MoviesActors λειτουργεί ως σύνδεσμος (junction table) μεταξύ των πινάκων Movies και Actors. Περιλαμβάνει το μοναδικό αναγνωριστικό (maid), το αναγνωριστικό της ταινίας (movieid) και το αναγνωριστικό του ηθοποιού (actorid). Οι στήλες movieid και actorid είναι ξένα κλειδιά που αναφέρονται στους πίνακες Movies και Actors αντίστοιχα. Έτσι μέσω των ξένων κλειδιών όταν γίνεται ανάκτηση μιας ταινίας, μέσω της επικοινωνίας των πινάκων θα γίνεται ανάκτηση και των ηθοποιών που έλαβαν μέρος.

```
CREATE TABLE MoviesActors (
    maid INT PRIMARY KEY,
    movieid INT FOREIGN KEY REFERENCES Movies(movieid),
    actorid INT FOREIGN KEY REFERENCES Actors(actorid),

);
```

Δημιουργία Πίνακα Σχολίων (Comments)

Ο πίνακας Comments αποθηκεύει σχόλια που αφήνουν οι χρήστες για τις ταινίες. Περιλαμβάνει το μοναδικό αναγνωριστικό του σχολίου (commentid), το αναγνωριστικό του χρήστη (userid), το αναγνωριστικό της ταινίας (movieid) και το κείμενο του σχολίου (comment). Οι στήλες userid και movieid είναι ξένα κλειδιά που αναφέρονται στους πίνακες Users και Movies.

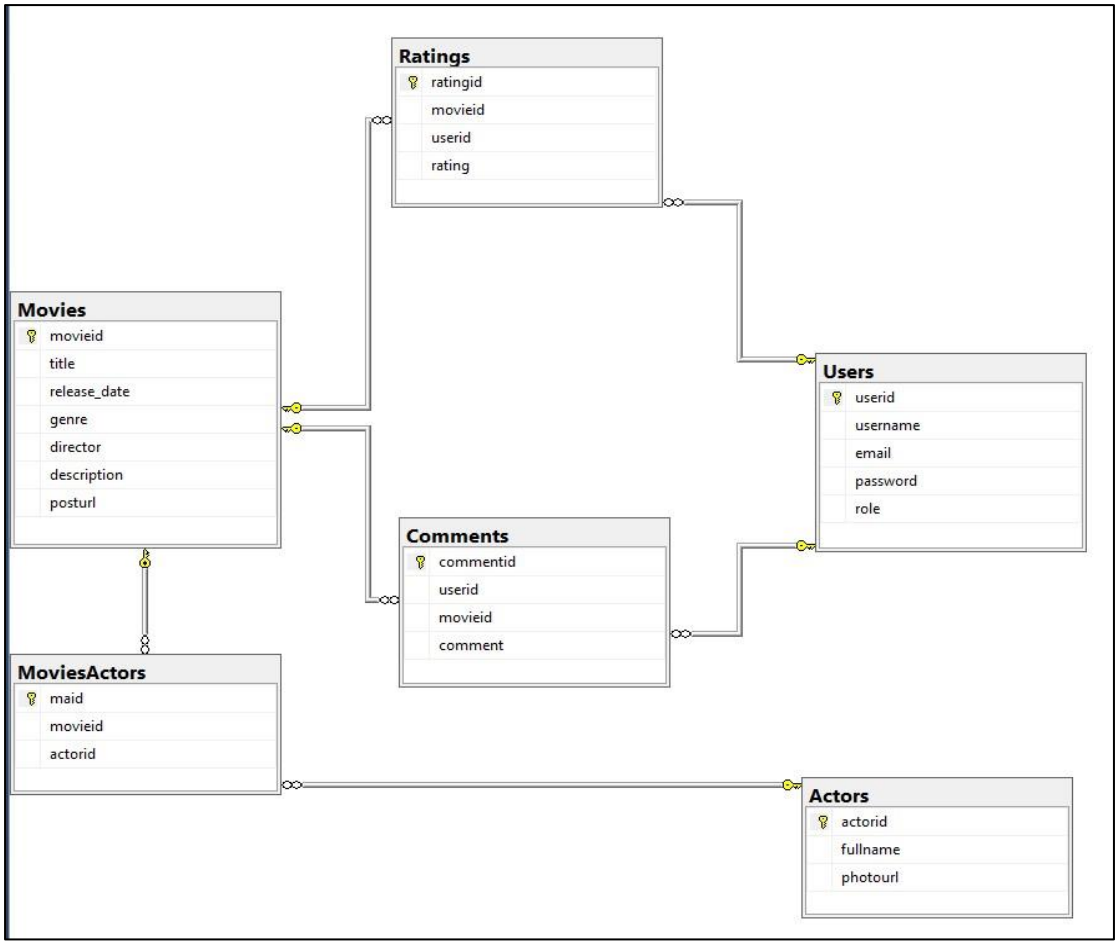
```
CREATE TABLE Comments (
    commentid INT PRIMARY KEY,
    userid INT FOREIGN KEY REFERENCES Users(userid),
    movieid INT FOREIGN KEY REFERENCES Movies(movieid),
    comment VARCHAR(MAX) NOT NULL
);
```

Δημιουργία Πίνακα Αξιολογήσεων (Ratings)

Ο πίνακας Rating αποθηκεύει τις αξιολογήσεις που δίνουν οι χρήστες στις ταινίες. Περιλαμβάνει το μοναδικό αναγνωριστικό της αξιολόγησης (ratingid), το αναγνωριστικό της ταινίας (movieid), το αναγνωριστικό του χρήστη (userid) και την αξιολόγηση (rating). Η στήλη rating έχει έναν έλεγχο (CHECK constraint) που διασφαλίζει ότι η αξιολόγηση είναι μεταξύ 1 και 10. Οι στήλες movieid και userid είναι ξένα κλειδιά που αναφέρονται στους πίνακες Movies και Users.

```
CREATE TABLE Rating (
    ratingid INT PRIMARY KEY,
    movieid INT FOREIGN KEY REFERENCES Movies(movieid),
    userid INT FOREIGN KEY REFERENCES Users(userid),
    rating INT CHECK (rating >= 1 AND rating <= 10) NOT NULL
);
```

Η δημιουργία αυτών των πινάκων γίνεται με την εκτέλεση των παραπάνω εντολών SQL στο Microsoft SQL Server Management Studio. Κάθε πίνακας έχει σχεδιαστεί ώστε να υποστηρίζει την αποθήκευση και τη διαχείριση συγκεκριμένων τύπων δεδομένων που απαιτούνται για τη λειτουργία της πλατφόρμας ταινιών. Οι σχέσεις μεταξύ των πινάκων ορίζονται μέσω των ξένων κλειδιών, επιτρέποντας την αναφορά και την ενσωμάτωση δεδομένων από διαφορετικούς πίνακες.



Εικόνα 4.1 Διάγραμμα σχέσεων των πινάκων της Βάσης

4.3 Σύνδεση βάσης με Visual Studio

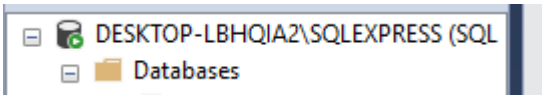
Η σύνδεση του ASP.NET MVC framework στο Visual studio με μια βάση δεδομένων Microsoft SQL Server και η αυτόματη ανάπτυξη του μοντέλου (Model), του ελεγκτή (Controller) και της προβολής (View) μπορεί να γίνει με τη χρήση του εργαλείου **scaffolding**. Αρχικά, πρέπει να διασφαλίσουμε ότι στο project ASP.NET MVC είναι εγκατεστημένο το πακέτο **Entity Framework Core**, το οποίο επιτρέπει τη σύνδεση με τη βάση δεδομένων και την αλληλεπίδραση με αυτήν. Αυτό μπορεί να γίνει μέσω του **nuGet package manager** επιλέγοντας να κατέβει το πακέτο **Microsoft.EntityFrameWork.SqlServer**. Στη συνέχεια, για να συνδεθούμε με τη βάση δεδομένων Microsoft SQL Server που δημιουργήσαμε στο Microsoft SQL Management Studio, χρησιμοποιούμε την εντολή Scaffold-DbContext. Αυτή η εντολή αναλαμβάνει να δημιουργήσει τις κλάσεις του μοντέλου (model classes) από τη βάση δεδομένων, καθώς και το πλαίσιο δεδομένων (DbContext class) που επιτρέπει την

αλληλεπίδραση με τη βάση δεδομένων. Η εντολή αυτή τρέχει από την κονσόλα του Visual Studio.

```
Scaffold-DbContext "Server=DESKTOP-LBHQIA2\SQLEXPRESS;Database=MoviesCentralDb;Trusted_Connection=True;Trust Server Certificate=True"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Context "MoviesCentralDBContext" -DataAnnotations
```

Αναλύοντας την εντολή:

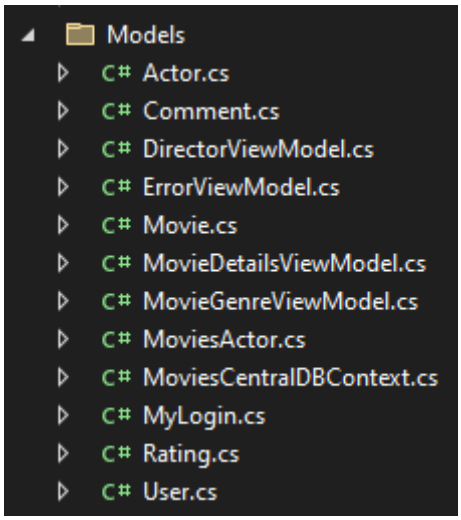
- Η παράμετρος ‘Server=DESKTOP-LBHQIA2\SQLEXPRESS’ καθορίζει τον διακομιστή της βάσης δεδομένων. Η τιμή αυτή φαίνεται μόλις γίνει εκκίνηση του SSMS



- Η παράμετρος ‘Database=MoviesCentralDb’ καθορίζει το όνομα της βάσης δεδομένων που χρησιμοποιούμε.
- Η παράμετρος ‘Trusted_Connection=True’ χρησιμοποιείται για να αναφέρει ότι χρησιμοποιούμε τα διαπιστευτήρια των Windows για τη σύνδεση.
- Η παράμετρος ‘Trust Server Certificate=True’ χρησιμοποιείται για να παρακάμπτει τα ζητήματα πιστοποιητικών του διακομιστή κατά τη σύνδεση.
- Το ‘Microsoft.EntityFrameworkCore.SqlServer’ είναι το πακέτο που χρησιμοποιούμε για τη σύνδεση με τον SQL Server.
- Η παράμετρος ‘-OutputDir Models’ καθορίζει τον φάκελο όπου θα αποθηκευτούν οι κλάσεις του μοντέλου.
- Η παράμετρος ‘-Context "MoviesCentralDBContext"’ ορίζει το όνομα της κλάσης DbContext που θα δημιουργηθεί.
- Η παράμετρος ‘-DataAnnotations’ καθορίζει ότι θα χρησιμοποιηθούν data annotations στις κλάσεις του μοντέλου για επικύρωση καθώς και σχέσεις κλειδιών (ξένα κλειδιά).

Μετά την εκτέλεση της εντολής, το εργαλείο scaffolding θα δημιουργήσει αυτόματα τις κλάσεις μοντέλων που αντιπροσωπεύουν τους πίνακες της βάσης δεδομένων, όπως οι User, Movie, Actor, Comment και Rating. Κάθε κλάση θα περιέχει ιδιότητες που αντιστοιχούν στις στήλες των πινάκων της βάσης δεδομένων. Για παράδειγμα, η κλάση Movie έχει ιδιότητες όπως MovieId, Title, ReleaseDate, Genre, Director, Description και PostUrl. Επιπλέον, θα δημιουργηθεί η κλάση

MoviesCentralDBContext, η οποία κληρονομεί από την κλάση DbContext του Entity Framework. Αυτή η κλάση θα περιέχει σύνολα δεδομένων (DbSet) που αντιπροσωπεύουν τους πίνακες της βάσης δεδομένων.



Εικόνα 4.2

Το DbContext είναι μια κεντρική κλάση στο Entity Framework Core που συνδέει την εφαρμογή με τη βάση δεδομένων και επιτρέπει την αλληλεπίδραση με τα δεδομένα μέσω μοντέλων. Αρχικά, έχουμε δύο constructors για την κλάση MoviesCentralDBContext. Ο πρώτος constructor είναι χωρίς παραμέτρους και χρησιμοποιείται για γενική αρχικοποίηση. Ο δεύτερος constructor δέχεται ένα DbContextOptions<MoviesCentralDBContext> αντικείμενο, το οποίο περιέχει τις επιλογές διαμόρφωσης για το DbContext. Αυτός ο constructor καλεί τον αντίστοιχο constructor της βασικής κλάσης DbContext, περνώντας τις επιλογές αυτές.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Actor>(entity =>
    {
        entity.HasKey(e => e.Actorid).HasName("PK__Actors__83335D33E58D6E85");

        entity.Property(e => e.Actorid).ValueGeneratedNever();
    });
```

Η κλάση περιλαμβάνει επίσης ιδιότητες για κάθε σύνολο δεδομένων (DbSet) που θα διαχειρίζεται. Αυτές οι ιδιότητες αντιπροσωπεύουν τους πίνακες στη βάση

δεδομένων και επιτρέπουν την εκτέλεση CRUD (Create, Read, Update, Delete) λειτουργιών πάνω σε αυτές. Οι ιδιότητες περιλαμβάνουν Actors, Comments, Movies, MoviesActors, Ratings και Users, καθεμία από τις οποίες είναι τύπου DbSet<T>, όπου T είναι το αντίστοιχο μοντέλο δεδομένων.

```
public virtual DbSet<Actor> Actors { get; set; }

– references
public virtual DbSet<Comment> Comments { get; set; }

– references
public virtual DbSet<Movie> Movies { get; set; }

– references
public virtual DbSet<MoviesActor> MoviesActors { get; set; }

8 references
public virtual DbSet<Rating> Ratings { get; set; }

19 references
public virtual DbSet<User> Users { get; set; }
```

Η OnConfiguring μέθοδος χρησιμοποιείται για τη διαμόρφωση του DbContext όταν δεν έχει ήδη παρασχεθεί μια διαμόρφωση μέσω των επιλογών. Σε αυτή τη μέθοδο, χρησιμοποιείται η UseSqlServer μέθοδος για να καθοριστεί η συμβολοσειρά σύνδεσης (connection string) με τη βάση δεδομένων SQL Server.

```
0 references
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
rning To protect potentially sensitive information in your connection string, you should move it out of source code.
    => optionsBuilder.UseSqlServer("Server=2EKATHIN15\\SQLEXPRESS;Database=MoviesCentralDb;Trusted_Connection=True;
```

Η OnModelCreating μέθοδος χρησιμοποιείται για τη διαμόρφωση του μοντέλου χρησιμοποιώντας το ModelBuilder. Αυτή η μέθοδος επιτρέπει τη ρύθμιση περιορισμών, σχέσεων και άλλων ρυθμίσεων για τις οντότητες. Κάθε οντότητα διαμορφώνεται ξεχωριστά:

- Για την οντότητα Actor, ορίζεται ως πρωτεύον κλειδί (HasKey) η ιδιότητα Actorid και καθορίζεται ότι δεν θα γίνεται αυτόματη αύξηση (ValueGeneratedNever).

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Actor>(entity =>
    {
        entity.HasKey(e => e.Actorid).HasName("PK__Actors__83335D33E58D6E85");

        entity.Property(e => e.Actorid).ValueGeneratedNever();
    });
```

- Για την οντότητα Comment, ορίζεται ως πρωτεύον κλειδί η ιδιότητα Commentid και καθορίζονται οι σχέσεις με την οντότητα Movie και User μέσω ξένων κλειδιών.

```
modelBuilder.Entity<Comment>(entity =>
{
    entity.HasKey(e => e.Commentid).HasName("PK__Comments__CDA84BC5D8F8BFCF");

    entity.Property(e => e.Commentid).ValueGeneratedNever();

    entity.HasOne(d => d.Movie).WithMany(p => p.Comments).HasConstraintName("FK__Comments__moviei__2F10007B");

    entity.HasOne(d => d.User).WithMany(p => p.Comments).HasConstraintName("FK__Comments__userid__2E1BDC42");
});
```

- Για την οντότητα Movie, ορίζεται ως πρωτεύον κλειδί η ιδιότητα Movieid και καθορίζεται ότι δεν θα γίνεται αυτόματη αύξηση.

```
modelBuilder.Entity<Movie>(entity =>
{
    entity.HasKey(e => e.Movieid).HasName("PK__Movies__42EACB66D2A9532C");

    entity.Property(e => e.Movieid).ValueGeneratedNever();
});
```

- Για την οντότητα MoviesActor, ορίζεται ως πρωτεύον κλειδί η ιδιότητα Maid και καθορίζονται οι σχέσεις με τις οντότητες Actor και Movie μέσω ξένων κλειδιών.

```
modelBuilder.Entity<MoviesActor>(entity =>
{
    entity.HasKey(e => e.Maid).HasName("PK__MoviesAc__7A21293D2539DE1F");

    entity.Property(e => e.Maid).ValueGeneratedNever();

    entity.HasOne(d => d.Actor).WithMany(p => p.MoviesActors).HasConstraintName("FK__MoviesAct__actor__2B3F6F97");

    entity.HasOne(d => d.Movie).WithMany(p => p.MoviesActors).HasConstraintName("FK__MoviesAct__movie__2A4B4B5E");
});
```

- Για την οντότητα Rating, ορίζεται ως πρωτεύον κλειδί η ιδιότητα Ratingid και καθορίζονται οι σχέσεις με τις οντότητες Movie και User μέσω ξένων κλειδιών.

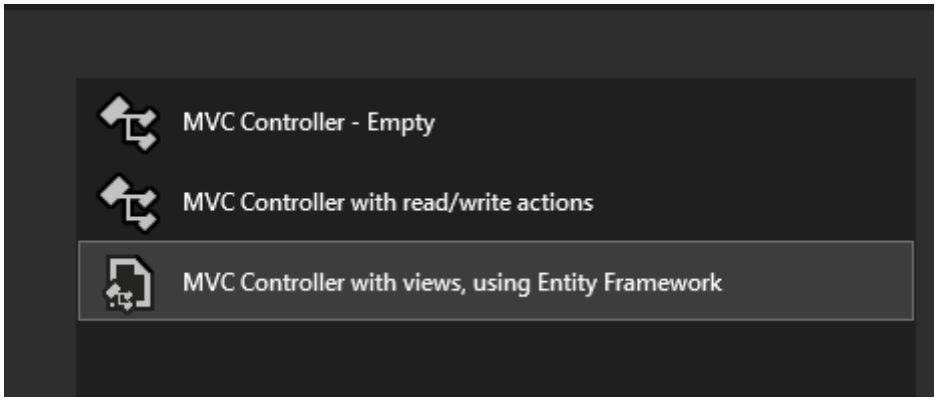
```
modelBuilder.Entity<Rating>(entity =>
{
    entity.HasKey(e => e.Ratingid).HasName("PK__Ratings__2D2E08C1DFC6B698");

    entity.Property(e => e.Ratingid).ValueGeneratedNever();

    entity.HasOne(d => d.Movie).WithMany(p => p.Ratings).HasConstraintName("FK__Ratings__movieid__31EC6D26");

    entity.HasOne(d => d.User).WithMany(p => p.Ratings).HasConstraintName("FK__Ratings__userid__32E0915F");
});
```

- Για την οντότητα User, ορίζεται ως πρωτεύον κλειδί η ιδιότητα Userid και καθορίζεται ότι δεν θα γίνεται αυτόματη αύξηση.

```
modelBuilder.Entity<User>(entity =>
{
    entity.HasKey(e => e.Userid).HasName("PK__Users__CBA1B2571CFF1022");

    entity.Property(e => e.Userid).ValueGeneratedNever();
});

OnModelCreatingPartial(modelBuilder);
```

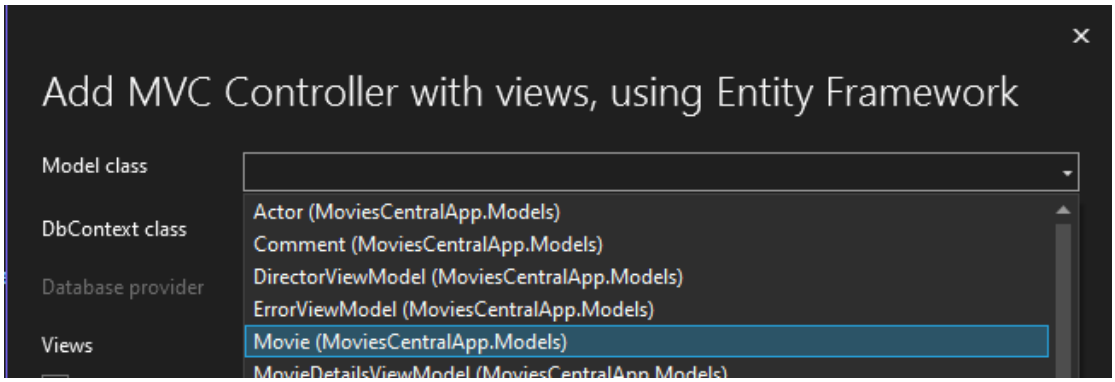
Τέλος, υπάρχει μια μερική μέθοδος OnModelCreatingPartial, η οποία καλείται στο τέλος της OnModelCreating μεθόδου για να επιτρέψει πρόσθετες διαμορφώσεις στο μοντέλο, εφόσον έχουν οριστεί σε κάποιο άλλο μέρος του κώδικα.

Ο κώδικας συνολικά διαμορφώνει το DbContext για να αλληλεπιδρά με τη βάση δεδομένων και να διαχειρίζεται τα δεδομένα της εφαρμογής σύμφωνα με τους ορισμούς των μοντέλων και των σχέσεων μεταξύ αυτών. Αφού δημιουργηθούν τα μοντέλα και το πλαίσιο δεδομένων, μπορούμε να προχωρήσουμε στη δημιουργία των ελεγκτών (controllers) και των προβολών (views) χρησιμοποιώντας το εργαλείο scaffolding στο Visual Studio. Για να δημιουργήσουμε έναν ελεγκτή και τις αντίστοιχες προβολές για τον πίνακα Movies, θα χρησιμοποιήσουμε την ενσωματωμένη λειτουργία scaffolding του Visual Studio. Κάνοντας δεξί κλικ στο φάκελο Controllers, επιλέγουμε "Add" -> "Controller" και στη συνέχεια επιλέγουμε "MVC Controller with views, using Entity Framework".



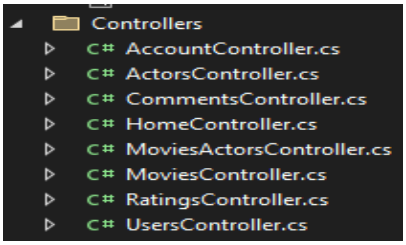
Εικόνα 4.3

Στον οδηγό που εμφανίζεται, επιλέγουμε την κλάση Movie ως το μοντέλο και την κλάση MoviesCentralDBContext ως το πλαίσιο δεδομένων. Το εργαλείο scaffolding θα δημιουργήσει αυτόματα έναν ελεγκτή MoviesController και τις αντίστοιχες προβολές για τις λειτουργίες CRUD για την επεξεργασία του πίνακα (Create, Read, Update, Delete).



Εικόνα 4.4

Επαναλαμβάνοντας το ίδιο δημιουργήθηκαν ελεγκτές και προβολές για όλα τα μοντέλα της εφαρμογής.



Εικόνα 4.5

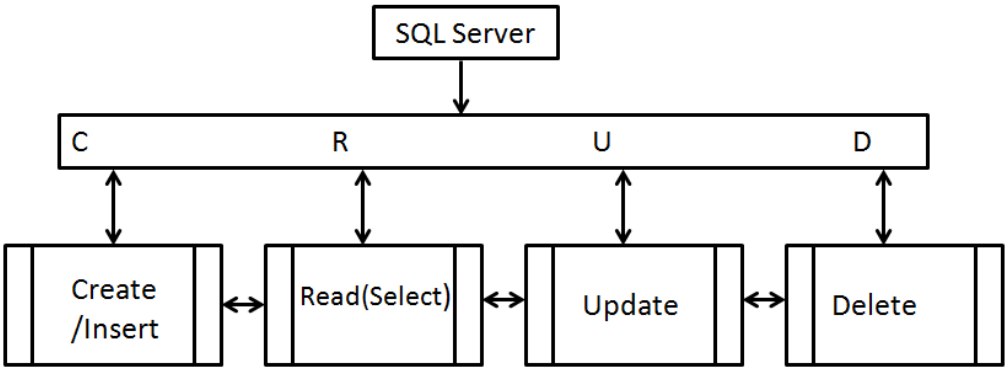
4.4 Λειτουργίες GRUD

Οι λειτουργίες CRUD αποτελούν το θεμέλιο για τη διαχείριση δεδομένων σε κάθε βάση δεδομένων και εφαρμογή λογισμικού. Το ακρωνύμιο CRUD αντιπροσωπεύει τις τέσσερις βασικές λειτουργίες που εκτελούνται σε δεδομένα: **Create (Δημιουργία)**, **Read (Ανάγνωση)**, **Update (Ενημέρωση)** και **Delete (Διαγραφή)**.

1. **Create (Δημιουργία)**: Αυτή η λειτουργία επιτρέπει την προσθήκη νέων δεδομένων σε μια βάση δεδομένων. Στο πλαίσιο μιας εφαρμογής, αυτή η λειτουργία συνήθως περιλαμβάνει την αποστολή ενός αιτήματος για την προσθήκη ενός νέου αντικειμένου δεδομένων στη βάση, όπως η δημιουργία ενός νέου χρήστη ή η καταχώρηση μιας νέας ταινίας.
2. **Read (Ανάγνωση)**: Η λειτουργία ανάγνωσης επιτρέπει την ανάκτηση δεδομένων από τη βάση. Αυτή η λειτουργία είναι ζωτικής σημασίας για την παρουσίαση δεδομένων στον χρήστη. Για παράδειγμα, η προβολή της λίστας χρηστών ή η ανάκτηση των λεπτομερειών μιας συγκεκριμένης ταινίας είναι λειτουργίες ανάγνωσης.
3. **Update (Ενημέρωση)**: Η ενημέρωση επιτρέπει την τροποποίηση των υπαρχόντων δεδομένων στη βάση. Αυτή η λειτουργία χρησιμοποιείται όταν χρειάζεται να αλλάξουμε πληροφορίες ενός αντικειμένου, όπως η ενημέρωση των στοιχείων ενός χρήστη ή η επεξεργασία των λεπτομερειών μιας ταινίας.
4. **Delete (Διαγραφή)**: Η διαγραφή επιτρέπει την αφαίρεση δεδομένων από τη βάση. Αυτή η λειτουργία χρησιμοποιείται για την αφαίρεση ανεπιθύμητων ή παρωχημένων αντικειμένων, όπως η διαγραφή ενός χρήστη ή η αφαίρεση μιας ταινίας από τη βάση δεδομένων.

Οι λειτουργίες CRUD είναι θεμελιώδεις για τη διαχείριση δεδομένων και είναι απαραίτητες σε κάθε εφαρμογή που αλληλεπιδρά με μια βάση. Αυτές οι λειτουργίες διασφαλίζουν ότι τα δεδομένα μπορούν να προστεθούν, να ανακτηθούν, να ενημερωθούν και να διαγραφούν με οργανωμένο και αποδοτικό τρόπο, επιτρέποντας στις εφαρμογές να είναι δυναμικές και ευέλικτες στη χρήση και διαχείριση των

δεδομένων. Ας μεταβούμε τώρα στον UsersController για να δούμε πως δημιουργήθηκαν.



Εικόνα 4.6 Λειτουργίες Grud σε SQL Server

4.4.1 Create

```
public IActionResult Create()
{
    return View();
}

// POST: Users/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create([Bind("Userid,Username,Email,Password,Role")] User user)
{
    if (ModelState.IsValid)
    {
        _context.Add(user);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(user);
}
```

Η πρώτη μέθοδος Create είναι μια HTTP GET μέθοδος. Όταν καλείται, επιστρέφει την προβολή (view) που σχετίζεται με τη δημιουργία ενός νέου χρήστη. Αυτή η μέθοδος δεν δέχεται παραμέτρους και απλώς επιστρέφει την προβολή χωρίς να εκτελεί κάποια επιπλέον λογική. Αυτό επιτρέπει στον διαχειριστή να δει τη φόρμα δημιουργίας ενός νέου χρήστη.

Η δεύτερη μέθοδος Create είναι μια HTTP POST μέθοδος και εκτελείται όταν ο χρήστης υποβάλει τη φόρμα για τη δημιουργία λογαριασμού. Αυτή η μέθοδος έχει διάφορα χαρακτηριστικά και παραμέτρους. Το HttpPost χαρακτηριστικό υποδηλώνει ότι αυτή η μέθοδος θα ανταποκριθεί μόνο σε HTTP POST αιτήματα. Το ValidateAntiForgeryToken χαρακτηριστικό προσθέτει ένα επίπεδο ασφαλείας για την προστασία κατά των επιθέσεων CSRF (Cross-Site Request Forgery). Η μέθοδος δέχεται ως παράμετρο ένα αντικείμενο User, το οποίο δημιουργείται αυτόματα από τα δεδομένα της φόρμας λόγω του χαρακτηριστικού Bind. Το Bind καθορίζει ποιες ιδιότητες του User αντικειμένου πρέπει να δεσμευτούν από τα δεδομένα της φόρμας για την αποτροπή επιθέσεων υπερβολικής ανάρτησης δεδομένων (overposting). Η μέθοδος ελέγχει αν η κατάσταση του μοντέλου (ModelState) είναι έγκυρη, δηλαδή αν τα δεδομένα που έχει εισαγάγει ο χρήστης είναι σύμφωνα με τους κανόνες επικύρωσης που έχουν οριστεί για το μοντέλο User. Αν η κατάσταση του μοντέλου είναι έγκυρη, το νέο User αντικείμενο προστίθεται στο DbContext και οι αλλαγές αποθηκεύονται στη βάση δεδομένων ασύγχρονα με την χρήση της μεθόδου SaveChangesAsync. Μετά την επιτυχή αποθήκευση, ο χρήστης ανακατευθύνεται στη μέθοδο Index του controller, η οποία συνήθως εμφανίζει τη λίστα των χρηστών. Η Index μαζί με την αντίστοιχη προβολή δημιουργείται αυτόματα με τις άλλες GRUD λειτουργίες και συνήθως εμφανίζει μια φόρμα προβολής όλων των στοιχείων του πίνακα. Αν η κατάσταση του μοντέλου δεν είναι έγκυρη, η μέθοδος επιστρέφει την ίδια προβολή με τα δεδομένα του χρήστη, επιτρέποντάς του να διορθώσει τυχόν σφάλματα και να υποβάλει ξανά τη φόρμα.

Η αντίστοιχη προβολή αυτής της μεθόδου περιλαμβάνει μια φόρμα για την προσθήκη των στοιχείων του χρήστη στη βάση δεδομένων.

```
<h4>User</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Userid" class="control-label"></label>
                <input asp-for="Userid" class="form-control" />
                <span asp-validation-for="Userid" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Username" class="control-label"></label>
                <input asp-for="Username" class="form-control" />
                <span asp-validation-for="Username" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password" class="control-label"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Role" class="control-label"></label>
                <input asp-for="Role" class="form-control" />
                <span asp-validation-for="Role" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
```

4.4.2 Details

```
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var user = await _context.Users
        .FirstOrDefaultAsync(m => m.Userid == id);
    if (user == null)
    {
        return NotFound();
    }

    return View(user);
}
```

Η μέθοδος Details υλοποιεί τη λογική για την εμφάνιση των λεπτομερειών ενός συγκεκριμένου χρήστη (User) μέσω μιας ασύγχρονης μεθόδου Details.
Αρχικά, η μέθοδος Details δέχεται ένα προαιρετικό ακέραιο όρισμα id, το οποίο αντιπροσωπεύει το αναγνωριστικό (ID) του χρήστη του οποίου οι λεπτομέρειες ζητούνται. Εάν το id είναι null, δηλαδή δεν έχει παρασχεθεί, η μέθοδος επιστρέφει ένα αποτέλεσμα NotFound(), το οποίο υποδηλώνει ότι η σελίδα δεν βρέθηκε. Στη συνέχεια, η μέθοδος προσπαθεί να ανακτήσει το χρήστη από τη βάση δεδομένων χρησιμοποιώντας το Entity Framework. Χρησιμοποιεί την ασύγχρονη μέθοδο FirstOrDefaultAsync για να βρει τον πρώτο χρήστη (ή τον μοναδικό χρήστη, δεδομένου ότι το Userid είναι πρωτεύον κλειδί) που έχει το Userid ίσο με την τιμή του παραμέτρου id. Η αναζήτηση γίνεται στη συλλογή Users του DbContext. Αν δεν βρεθεί κανένας χρήστης με το συγκεκριμένο id, δηλαδή αν το user είναι null, η μέθοδος επιστρέφει ξανά ένα αποτέλεσμα NotFound(), υποδηλώνοντας ότι ο συγκεκριμένος χρήστης δεν υπάρχει στη βάση δεδομένων. Αν όμως βρεθεί ο χρήστης, η μέθοδος επιστρέφει την προβολή (view) View(user), που εμφανίζει τις λεπτομέρειες του συγκεκριμένου χρήστη. Το αντικείμενο user περιέχει όλες τις πληροφορίες για τον χρήστη και περνιέται στην προβολή για να παρουσιαστούν. Συνολικά, η μέθοδος Details παρέχει έναν ασφαλή και αποτελεσματικό τρόπο για την αναζήτηση και εμφάνιση των λεπτομερειών ενός χρήστη από τη βάση δεδομένων, και ταυτόχρονα διαχειρίζεται καταστάσεις όπου το id δεν είναι έγκυρο ή δεν αντιστοιχεί σε κανέναν χρήστη.

4.4.3 Edit

```
public async Task<IActionResult> Edit(int id, [Bind("Userid,Username,Email,Password,Role")] User user)
{
    if (id != user.Userid)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(user);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!UserExists(user.Userid))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(user);
}
```

Αρχικά υπάρχει μια μέθοδος Edit που είναι μια ασύγχρονη HTTP GET μέθοδος που δέχεται ένα προαιρετικό ακέραιο όρισμα id, το οποίο αντιπροσωπεύει το αναγνωριστικό του χρήστη που πρόκειται να επεξεργαστεί ακριβώς όπως το Details. Ελέγχει αν το id είναι null. Αν είναι null, επιστρέφει ένα αποτέλεσμα NotFound(), υποδηλώνοντας ότι δεν βρέθηκε η σελίδα επεξεργασίας λόγω μη παροχής του αναγνωριστικού. Στη συνέχεια, η μέθοδος προσπαθεί να βρει τον χρήστη με το συγκεκριμένο id στη βάση δεδομένων χρησιμοποιώντας την ασύγχρονη μέθοδο FindAsync του DbContext. Αν δεν βρεθεί ο χρήστης, η μέθοδος επιστρέφει και πάλι ένα αποτέλεσμα NotFound(). Αν βρεθεί, η μέθοδος επιστρέφει την προβολή View(user), η οποία παρουσιάζει τη φόρμα επεξεργασίας με τα δεδομένα του χρήστη.

Η δεύτερη μέθοδος Edit είναι μια ασύγχρονη HTTP POST μέθοδος που καλείται όταν ο διαχειριστής υποβάλει τη φόρμα επεξεργασίας. Η μέθοδος αυτή διασφαλίζει την αποτροπή επιθέσεων CSRF (Cross-Site Request Forgery) μέσω του χαρακτηριστικού ValidateAntiForgeryToken. Δέχεται ως παραμέτρους το id και ένα αντικείμενο User, το οποίο δεσμεύει τις ιδιότητες Userid, Username, Email, Password και Role μέσω του χαρακτηριστικού Bind. Η μέθοδος ξεκινά με τον έλεγχο αν το id

που παρέχεται στο URL ταιριάζει με το Userid του αντικειμένου user. Αν δεν ταιριάζουν, επιστρέφει ένα αποτέλεσμα NotFound(), υποδεικνύοντας ότι υπάρχει ασυμφωνία μεταξύ των αναγνωριστικών. Στη συνέχεια, η μέθοδος ελέγχει την εγκυρότητα της κατάστασης του μοντέλου (ModelState). Αν η κατάσταση του μοντέλου είναι έγκυρη, προσπαθεί να ενημερώσει τον χρήστη στη βάση δεδομένων. Η ενημέρωση γίνεται με την κλήση της μεθόδου Update του DbContext, ακολουθούμενη από την ασύγχρονη αποθήκευση των αλλαγών με τη μέθοδο SaveChangesAsync. Αν κατά την ενημέρωση προκύψει κάποιο πρόβλημα ταυτόχρονης τροποποίησης δεδομένων (DbUpdateConcurrencyException), η μέθοδος ελέγχει αν ο χρήστης εξακολουθεί να υπάρχει στη βάση δεδομένων καλώντας τη μέθοδο UserExists. Αν ο χρήστης δεν υπάρχει, επιστρέφει ένα αποτέλεσμα NotFound(). Αν ο χρήστης υπάρχει, η εξαίρεση επανεντάσσεται για περαιτέρω επεξεργασία.

Αν η ενημέρωση ολοκληρωθεί επιτυχώς, ο διαχειριστής ανακατευθύνεται στη μέθοδο Index, η οποία συνήθως εμφανίζει τη λίστα των χρηστών. Αν η κατάσταση του μοντέλου δεν είναι έγκυρη, η μέθοδος επιστρέφει την ίδια προβολή με τα δεδομένα του χρήστη, επιτρέποντας στον διαχειριστή να διορθώσει τα σφάλματα και να υποβάλει ξανά τη φόρμα. Συνολικά, ο κώδικας διασφαλίζει μια ασφαλή και αποτελεσματική διαδικασία επεξεργασίας χρηστών στην εφαρμογή, ελέγχοντας περιπτώσεις όπου τα δεδομένα δεν είναι έγκυρα ή οι χρήστες δεν βρίσκονται στη βάση δεδομένων.

4.4.4 Delete

```
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var user = await _context.Users
        .FirstOrDefaultAsync(m => m.Userid == id);
    if (user == null)
    {
        return NotFound();
    }

    return View(user);
}

// POST: Users/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var user = await _context.Users.FindAsync(id);
    if (user != null)
    {
        _context.Users.Remove(user);
    }

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
```

Η μέθοδος Delete χειρίζεται τη διαγραφή ενός χρήστη από τη βάση δεδομένων. Η διαδικασία διαγραφής περιλαμβάνει δύο μεθόδους: μία GET και μία POST.

Η πρώτη μέθοδος Delete είναι μια ασύγχρονη HTTP GET μέθοδος που δέχεται ένα προαιρετικό ακέραιο όρισμα id, το οποίο αντιπροσωπεύει το αναγνωριστικό του χρήστη που πρόκειται να διαγραφεί. Αρχικά, η μέθοδος ελέγχει αν το id είναι null. Αν το id είναι null, επιστρέφει ένα αποτέλεσμα NotFound(), το οποίο υποδεικνύει ότι η σελίδα δεν βρέθηκε λόγω της μη παροχής του αναγνωριστικού. Στη συνέχεια, η μέθοδος προσπαθεί να βρει τον χρήστη με το συγκεκριμένο id στη βάση δεδομένων χρησιμοποιώντας την ασύγχρονη μέθοδο FirstOrDefaultAsync του DbContext. Αν δεν βρεθεί ο χρήστης, η μέθοδος επιστρέφει ένα αποτέλεσμα NotFound(). Αν βρεθεί,

επιστρέφει την προβολή View(user), η οποία παρουσιάζει τα δεδομένα του χρήστη και επιβεβαιώνει την πρόθεση διαγραφής του.

Η δεύτερη μέθοδος DeleteConfirmed είναι μια ασύγχρονη HTTP POST μέθοδος που εκτελείται όταν ο διαχειριστής επιβεβαιώνει τη διαγραφή στη φόρμα. Η μέθοδος φέρει το χαρακτηριστικό ActionName("Delete"), το οποίο επιτρέπει να χρησιμοποιηθεί η ίδια URL διαδρομή όπως η GET μέθοδος Delete. Επίσης, το χαρακτηριστικό ValidateAntiForgeryToken προσθέτει ένα επίπεδο ασφαλείας για την αποτροπή επιθέσεων CSRF (Cross-Site Request Forgery). Η μέθοδος δέχεται ένα ακέραιο id, το οποίο αντιπροσωπεύει το αναγνωριστικό του χρήστη που πρόκειται να διαγραφεί. Αρχικά, προσπαθεί να βρει τον χρήστη στη βάση δεδομένων χρησιμοποιώντας την ασύγχρονη μέθοδο FindAsync του DbContext. Αν βρεθεί ο χρήστης, αφαιρείται από τη συλλογή Users του DbContext με τη μέθοδο Remove. Μετά την αφαίρεση του χρήστη, οι αλλαγές αποθηκεύονται ασύγχρονα στη βάση δεδομένων με τη μέθοδο SaveChangesAsync. Τέλος, η μέθοδος ανακατευθύνει τον χρήστη στη μέθοδο Index. Η βοηθητική μέθοδος UserExists είναι μια ιδιωτική μέθοδος που επιστρέφει ένα boolean αποτέλεσμα, υποδεικνύοντας αν υπάρχει κάποιος χρήστης με το συγκεκριμένο id στη βάση δεδομένων. Χρησιμοποιεί τη μέθοδο Any του DbContext για να ελέγξει την ύπαρξη του χρήστη. Συνολικά, ο κώδικας αυτός παρέχει έναν ασφαλή και αποτελεσματικό τρόπο για τη διαγραφή χρηστών από τη βάση δεδομένων, διασφαλίζοντας ότι η διαδικασία διαγραφής γίνεται μόνο αν ο χρήστης υπάρχει και επιβεβαιώνει τη διαγραφή πριν την εκτέλεση της.

4.4.5 Index

Η μέθοδος Index μαζί με την αντίστοιχη προβολή της δημιουργείται και αυτή αυτόματα και χειρίζεται την προβολή όλων των εγγραφών ενός πίνακα της βάσης. Ο τρόπος που εμφανίζει τις εγγραφές είναι με τη βοήθεια του ενσωματωμένου πακέτου ‘HtmlHelper’. Συγκεκριμένα δημιουργείται ένας πίνακας (table) με επικεφαλίδες για κάθε εγγραφή του πίνακα και κάτω από αυτό η τιμή του πεδίου. Ας πάρουμε για παράδειγμα τον πίνακα με τους χρήστες (Users).

```
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Username)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Email)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Password)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Role)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Username)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Email)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Password)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Role)
            </td>
```

Στο '<thead>' κομμάτι του κώδικα, για την εμφάνιση των ετικετών του πίνακα χρησιμοποιείται η βοηθητική μέθοδος '@Html.DisplayNameFor', η οποία παίρνει μια έκφραση lambda που καθορίζει την ιδιότητα του μοντέλου της οποίας η ετικέτα πρέπει να εμφανιστεί. Για παράδειγμα η γραμμή '@Html.DisplayFor(model => model.Username)' δηλώνει ότι για το μοντέλο Uses (που βρισκόμαστε τώρα) εμφάνισε το 'Username' ως επικεφαλίδα του αντίστοιχου πεδίου. Στο '<tbody>' κομμάτι του κώδικα χρησιμοποιείται η ίδια λογική με την διαφορά ότι η εντολή γίνεται '@Html.DisplayFor' για να εμφανίσει αυτή τη φορά την τιμή του πεδίου όχι την ετικέτα. Αυτό το κομμάτι είναι τοποθετημένο μέσα σε ένα βρόχο έτσι ώστε η προβολή των στοιχείων να επαναληφθεί για όλες τις εγγραφές του πίνακα χρηστών.

5.Υλοποίηση την εφαρμογής

5.1 Είσοδος και δημιουργία λογαριασμού

Η πρώτη και από τις πιο βασικές λειτουργίες της εφαρμογής είναι η είσοδος των χρηστών καθώς και η δημιουργία λογαριασμού για να μπορούν να εισέλθουν. Προκειμένου να γίνει αυτό αρχικά δημιουργήθηκε το μοντέλο 'MyLogin', κάνοντας δεξί κλικ στον φάκελο μοντέλα (Models) και δημιουργία κλάσης στο μενού που εμφανίζεται, το οποίο θα ελέγχει την ορθότητα των στοιχείων που πληκτρολόγησε ο χρήστης στη φόρμα 'Login'. Συγκεκριμένα, θα αποθηκεύεται το email και ο κωδικός που πληκτρολογήθηκαν όπως φαίνεται παρακάτω.

```
public class MyLogin
{
    [Key]
    4 references
    public string Email { get; set; }

    4 references
    public string Password { get; set; }
}
```

Στη συνέχεια δημιουργήθηκε στο φάκελο των Controller ο 'AccountController' ο οποίος θα είναι υπεύθυνος για την είσοδο του χρήστη στην εφαρμογή, εφόσον αυτός διαθέτει λογαριασμό, καθώς και για την δημιουργία νέου. Όσον αφορά την είσοδο στην εφαρμογή με υπάρχοντα λογαριασμό, θα γίνεται ως εξής:
Αρχικά ο χρήστης εισάγει το email και τον κωδικό του λογαριασμού του σε μια φόρμα εισόδου. Έπειτα με τη χρήση ενός ερωτήματος (querry) πραγματοποιείται αναζήτηση στο στιγμιότυπο της βάσης δεδομένων όπου γίνεται έλεγχος αν τα στοιχεία που καταχωρήθηκαν υπάρχουν στο table Users. Αν ταυτοποιηθούν αποθηκεύονται στη μεταβλητή querry αλλιώς μένει null.

```
public IActionResult Login(MyLogin myLogin)
{
    var querry = dbContext.Users.SingleOrDefault(m => m.Email == myLogin.Email && m.Password == myLogin.Password);
```

Σε περίπτωση που γίνει ταυτοποίηση και το περιεχόμενο της μεταβλητής querry δεν είναι κενό (null) πραγματοποιείται αποθήκευση των στοιχείων του χρήστη στο

session προκειμένου να είναι συνδεδεμένος καθ όλη τη διάρκεια που είναι ανοιχτή η εφαρμογή και οδηγείται στην κεντρική σελίδα της (LoginView). Αν τα στοιχεία δεν ταυτοποιηθούν και το περιεχόμενο της μεταβλητής είναι κενό (null) εμφανίζεται μήνυμα στο χρήστη ότι η είσοδος ήταν ανεπιτυχής.

```
if(querry != null)
{
    HttpContext.Session.SetInt32("UserId", querry.Userid);


    TempData["Message"] = "Login successful";

    if (querry.Role == "admin")
    {
        return View("~/Views/AdminView.cshtml");
    }
    else
    {
        return View("~/Views/LoginView.cshtml");
    }


}
else
{

    TempData["Message"] = "Login failed";
    return View("~/Views/Account/Login.cshtml");
}
```

Σε περίπτωση που ο χρήστης δεν διαθέτει λογαριασμό πρέπει να δημιουργήσει. Για το σκοπό αυτό υλοποιήθηκε η μέθοδος ‘Register’ στον ‘AccountController’ όπου χειρίζεται την εγγραφή των χρηστών με δύο μεθόδους, μια HTTP Get και μία HTTP Post. Η πρώτη μέθοδος Register είναι μια HTTP GET μέθοδος που απλά επιστρέφει την προβολή (View) για την εγγραφή. Όταν ένας χρήστης επισκέπτεται τη σελίδα εγγραφής, αυτή η μέθοδος καλείται και αποδίδει την αντίστοιχη φόρμα.

Η δεύτερη μέθοδος Register είναι μια HTTP POST μέθοδος που χειρίζεται την υποβολή της φόρμας εγγραφής. Η μέθοδος αυτή δέχεται ένα αντικείμενο User ως παράμετρο, το οποίο περιέχει τα δεδομένα που εισήγαγε ο χρήστης στη φόρμα. Η μέθοδος ξεκινάει με τον έλεγχο της εγκυρότητας του μοντέλου χρησιμοποιώντας το ModelState.IsValid. Αν τα δεδομένα της φόρμας είναι έγκυρα, η μέθοδος συνεχίζει

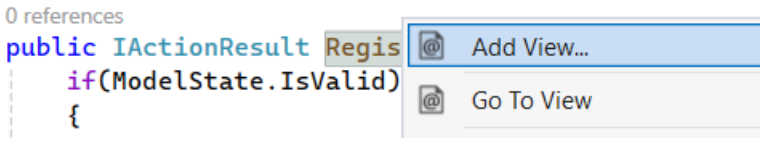
με περαιτέρω ελέγχους. Αρχικά, ελέγχει αν υπάρχει ήδη καταχωρημένος χρήστης με το ίδιο email στη βάση δεδομένων, χρησιμοποιώντας την μέθοδο Any του dbContext.Users. Αν υπάρχει χρήστης με το ίδιο email, η μέθοδος θέτει το μήνυμα "Email already registered" στο ViewBag.Message και επιστρέφει την ίδια προβολή, ώστε ο χρήστης να δει το μήνυμα σφάλματος. Αν δεν υπάρχει χρήστης με το ίδιο email, η μέθοδος ελέγχει αν υπάρχει χρήστης με το ίδιο όνομα (username) στη βάση δεδομένων με παρόμοιο τρόπο. Αν υπάρχει με το ίδιο όνομα, η μέθοδος θέτει το μήνυμα "Username already registered" στο ViewBag.Message και επιστρέφει την ίδια προβολή. Αν δεν υπάρχουν ούτε χρήστες με το ίδιο email ούτε με το ίδιο όνομα, η μέθοδος προχωράει με την καταχώρηση του νέου χρήστη. Ορίζει την ιδιότητα Role του χρήστη σε "user", προσθέτει τον χρήστη στη συλλογή Users του dbContext και αποθηκεύει τις αλλαγές στη βάση δεδομένων με την μέθοδο SaveChanges. Τέλος, επιστρέφει την προβολή για την σελίδα σύνδεσης (Login), υποδεικνύοντας ότι η εγγραφή ήταν επιτυχής και ο χρήστης μπορεί να συνδεθεί.

```
public IActionResult Register()
{
    return View();
}
[HttpPost]

0 references
public IActionResult Register(User user) {
    if(ModelState.IsValid)
    {
        if(dbContext.Users.Any(x => x.Email == user.Email))
        {
            ViewBag.Message = "Email already registered";
            return View();
        }else if (dbContext.Users.Any(x => x.Username == user.Username))
        {
            ViewBag.Message = "Username already registered";
            return View();
        }
        else
        {

            user.Role = "user";
            dbContext.Users.Add(user);
            dbContext.SaveChanges();
            return View("~/Views/Account/Login.cshtml");
        }

    }
    return View();

}
```

5.2 Φόρμες Εγγραφής και Σύνδεσης.

5.2.1 Φόρμα σύνδεσης

Μαζί με τις μεθόδους αναπτύχθηκαν και οι αντίστοιχες προβολές με τις φόρμες εγγραφής και σύνδεσης. Κάνοντας δεξί κλικ στην αντίστοιχη μέθοδο, επιλέγουμε προσθήκη προβολής (add view) και στη συνέχεια από το μενού που εμφανίζεται επιλέγουμε για πρότυπο (template) δημιουργία (Create) και η αντίστοιχη προβολή με τα πεδία εισαγωγής για τα στοιχεία του χρήστη δημιουργείται αυτόματα με html.



Εικόνα 5.1

```
<div class="login-container">
    <div class="login-form">
        <h1 class="login-title">Login</h1>
        <hr />
        <form asp-action="Login">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" type="email" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password" class="control-label"></label>
                <input asp-for="Password" class="form-control" type="password" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Login" class="btn btn-login btn-block" />
            </div>
        </form>
        <h8>Dont have an account?</h8>
        <div>
            <a asp-area="" asp-controller="Account" asp-action="Register">Register Here</a>
        </div>
        @if (TempData["Message"] != null)
        {
            <div class="alert alert-info">
                @TempData["Message"]
            </div>
        }
    </div>
</div>
```

Συγκεκριμένα:

- Το div με κλάση login-container περιέχει τη φόρμα σύνδεσης και την κεντράρει στη σελίδα.
- Μέσα σε αυτό, το div με κλάση login-form περιέχει την ίδια τη φόρμα.
- Το h1 στοιχείο με κλάση login-title εμφανίζει τον τίτλο της φόρμας ("Login").
- Η οριζόντια γραμμή (<hr />) προσθέτει ένα οπτικό διαχωριστικό κάτω από τον τίτλο.
- Το form με asp-action="Login" ορίζει την ενέργεια που θα εκτελεστεί όταν υποβληθεί η φόρμα (δηλαδή, η μέθοδος Login στον αντίστοιχο controller).
- Το div με asp-validation-summary="ModelOnly" εμφανίζει μηνύματα σφαλμάτων επικύρωσης μόνο για το μοντέλο.
- Τα div στοιχεία με class="form-group" περιέχουν τις ετικέτες (label), τα πεδία εισαγωγής (input) και τα μηνύματα επικύρωσης (span) για το email και τον κωδικό πρόσβασης.
- Το κουμπί υποβολής (input type="submit") έχει στυλ btn btn-login btn-block για να εφαρμόσει το προκαθορισμένο στυλ του κουμπιού.
- Υπάρχει ένα μήνυμα που ενθαρρύνει τους χρήστες χωρίς λογαριασμό να εγγραφούν και ένας σύνδεσμος που οδηγεί στη σελίδα εγγραφής (asp-controller="Account" asp-action="Register").
- Τέλος, εάν υπάρχει κάποιο μήνυμα στην TempData, αυτό εμφανίζεται σε ένα div με κλάση alert alert-info, το οποίο μπορεί να χρησιμοποιηθεί για την εμφάνιση πληροφοριακών μηνυμάτων προς τον χρήστη (π.χ. επιβεβαίωση επιτυχούς εγγραφής).

Επιπλέον έχει δημιουργηθεί πεδίο style όπου χρησιμοποιεί css για την μορφοποίηση και εμφάνιση κάποιων πεδίων ανάλογα με την κλάση τους.

```
<style>
    .login-container {
        display: flex;
        justify-content: center;
        align-items: center;
        height: 100vh;
    }

    .login-form {
        max-width: 400px;
        width: 100%;
        padding: 20px;
        border: 1px solid #ccc;
        border-radius: 5px;
        background-color: #121212;
        color: #ffffff;
    }

    .login-title {
        text-align: center;
    }

    .btn-login {
        background-color: #f5c518; /* IMDb yellow */
        color: #000; /* Black text */
    }
</style>
```

Συγκεκριμένα, το style login-container ορίζει έναν container με flex διάταξη που κεντράρει το περιεχόμενο οριζόντια και κατακόρυφα, με ύψος ίσο με το ύψος του παραθύρου προβολής σε όσα αντικείμενα έχουν για κλάση το login-container. Αντίστοιχα, το login-form καθορίζει τις διαστάσεις και το στυλ της φόρμας σύνδεσης, περιλαμβάνει μέγιστο πλάτος 400px, padding 20px, περίγραμμα, στρογγυλεμένες γωνίες, και σκούρο φόντο με λευκό κείμενο. Το login-title κεντράρει τον τίτλο της φόρμας σύνδεσης και το btn-login καθορίζει το χρώμα του κουμπιού σύνδεσης, χρησιμοποιώντας το χαρακτηριστικό κίτρινο χρώμα του IMDb και μαύρο χρώμα κειμένου.

5.2.2 Φόρμα εγγραφής

```
<div class="register-container">
    <div class="register-form">
        <h1 class="register-title">Register</h1>
        <hr />
        <form asp-action="Register">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Userid" id="useridInput" />
            <div class="form-group">
                <label asp-for="Username" class="control-label"></label>
                <input asp-for="Username" class="form-control" placeholder="Username" />
                <span asp-validation-for="Username" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" type="email" placeholder="Email" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password" class="control-label"></label>
                <input asp-for="Password" class="form-control" type="password" placeholder="Password" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Register" class="btn btn-register btn-block" />
            </div>
        </form>
        <h8>Already have an account?</h8>
        <div>
            <a asp-area="" asp-controller="Account" asp-action="Login">Login Here</a>
        </div>
```

Προχωρώντας στη φόρμα Register, δημιουργήθηκε με αντίστοιχο τρόπο. Χρησιμοποιήθηκε html για την φόρμα καταχώρησης των στοιχείων (username, email, password) και κάθε πεδίο ανήκει σε μια κλάση ώστε να είναι στοιχισμένο στο χώρο και να έχει το απαραίτητο styling. Η φόρμα (<form>) χρησιμοποιεί το Razor βοηθητικό tag asp-action="Register", υποδεικνύοντας ότι θα υποβληθεί στη μέθοδο Register του αντίστοιχου controller. Στην αρχή της φόρμας, υπάρχει ένα div με το βοηθητικό tag asp-validation-summary="ModelOnly", το οποίο εμφανίζει σφάλματα επικύρωσης, αν υπάρχουν, με την κλάση text-danger για να εμφανίζονται με κόκκινο χρώμα. Ένα κρυφό πεδίο (<input type="hidden">) με το βοηθητικό tag asp-for="Userid" βρίσκεται στην αρχή της φόρμας για την αποθήκευση του Userid, και έχει σαν αναγνωριστικό το "useridInput". Το πεδίο αυτό συμπληρώνεται αυτόματα κατά την εγγραφή του χρήστη με χρήση Javascript.

```
<script>
    // Generate a random unique Userid within the limits of a signed 32-bit integer
    function generateRandomUserId() {
        var randomUserId;
        do {
            // Generate random positive integer within the range of a signed 32-bit integer
            randomUserId = Math.floor(Math.random() * (Math.pow(2, 31) - 1));
        } while (document.getElementById("useridInput").value == randomUserId);

        document.getElementById("useridInput").value = randomUserId;
    }

    // Call the function on page load
    window.onload = function () {
        generateRandomUserId();
    };
</script>
```

Η συνάρτηση generateRandomUserId παράγει έναν τυχαίο θετικό ακέραιο αριθμό εντός των ορίων ενός 32-bit ακεραίου, και αποθηκεύει αυτή την τιμή στο κρυφό πεδίο εισαγωγής με id "useridInput". Αυτή η συνάρτηση καλείται κατά τη φόρτωση της σελίδας, διασφαλίζοντας ότι κάθε φορά που ο χρήστης επισκέπτεται τη σελίδα εγγραφής, ένα νέο τυχαίο Userid θα δημιουργηθεί και θα συμπληρωθεί αυτόματα.

Η φόρμα περιέχει διάφορα πεδία εισαγωγής για τα στοιχεία του χρήστη. Κάθε πεδίο εισαγωγής περιλαμβάνεται σε ένα div με κλάση form-group για τη διάταξη και περιλαμβάνει τα εξής:

1. Ένα label με το βοηθητικό tag asp-for, το οποίο δημιουργεί τη σωστή ετικέτα για την ιδιότητα του μοντέλου χρήστη (User).
2. Ένα input με το βοηθητικό tag asp-for, που δέχεται την τιμή της αντίστοιχης ιδιότητας.
3. Ένα span με το βοηθητικό tag asp-validation-for, το οποίο εμφανίζει μηνύματα επικύρωσης σφαλμάτων για την αντίστοιχη ιδιότητα, αν υπάρχουν σφάλματα.

Τα πεδία εισαγωγής περιλαμβάνουν το Username, Email (με τύπο email για να διασφαλιστεί ότι εισάγεται έγκυρο email), και Password (με τύπο password για να αποκρύπτει τον κωδικό). Κάθε πεδίο έχει μια προτροπή (placeholder) για να καθοδηγεί τον χρήστη σχετικά με το τι πρέπει να εισάγει. Στο τέλος της φόρμας υπάρχει ένα κουμπί υποβολής (<input type="submit">) με την ένδειξη "Register" για να υποβάλει ο χρήστης την φόρμα. Κάτω από τα πεδία αυτά, υπάρχει ένα μήνυμα που

ρωτάει αν ο χρήστης έχει ήδη λογαριασμό και ένας σύνδεσμος που οδηγεί στη φόρμα σύνδεσης (Login) του λογαριασμού.

Οι παραπάνω φόρμες όπως αναφέρθηκε προηγουμένως εμφανίζονται πάνω στο '_Layout.cshtml' το οποίο δημιουργείται αυτόματα και ορίζεται από το Visual Studio ως το βασικό layout της εφαρμογής. Προκειμένου το layout να ταιριάζει με τη μορφοποίηση και το στυλ της εφαρμογής χρησιμοποιήθηκε css.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - MoviesCentralApp</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/MoviesCentralApp.styles.css" asp-append-version="true" />

    <style>
        /* Custom IMDb-style CSS */
        body {
            background-color: #121212; /* Dark gray */
            color: #ffffff; /* White text */
            overflow-y: auto; /* Prevents scrolling from revealing the background */
            margin: 0; /* Remove default margin */
            padding: 0; /* Remove default padding */
        }

        /* Container for background image */
        .background-container {
            position: fixed; /* Fixes the container in place */
            top: 0;
            left: 0;
```

Επίσης δημιουργήθηκαν οι απαραίτητες επικεφαλίδες για να είναι ευκολότερη η κατεύθυνση των χρηστών στις φόρμες εγγραφής και σύνδεσης αντίστοιχα.

```
<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Account" asp-action="Register">Register</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Account" asp-action="Login">Login</a>
        </li>
```

Επιπλέον όπως αναφέρθηκε προηγουμένως το layout διαμορφώνει τη σελίδα πάνω στην οποία κάποια προβολή εμφανίζει τα δεδομένα στο χρήστη. Στη προκειμένη περίπτωση όπως δημιουργήθηκε αυτόματα το αρχικό layout της εφαρμογής έτσι δημιουργήθηκε και η Index προβολή που του αντιστοιχεί. Έτσι έγινε επεξεργασία της προβολής αυτής έτσι ώστε να καλωσορίζει τον χρήστη στην

εφαρμογή. Συγκεκριμένα αποθηκεύτηκε μια εικόνα ποπ κορν στο φάκελο Images του project κάτω από το wwwroot. Με τη χρήση της ετικέτας '<img>' html και δίνοντας ως τιμή την διαδρομή από τον διακομιστή μέχρι το φάκελο εμφανίζεται στη φόρμα.

```
<div class="text-center">
    <h1 class="display-4">Welcome to Movies Central!</h1>
    <img src="~/Images/popcorngif.gif" alt="Popcorn GIF" class="img-fluid mt-4" style="max-width: 200px;">
```

Επίσης προστέθηκαν επιπλέον σύνδεσμοι για τις φόρμες εγγραφής και σύνδεσης.

```
<p>Please <a asp-action="Login" asp-controller="Account" class="login-link">login</a> or
iv>

<a asp-action="Register" asp-controller="Account" class="register-link">register</a> to continue</p>
```

5.3. Κεντρική Σελίδα

Η κεντρική σελίδα της εφαρμογής είναι το κύριο μέρος της εργασίας καθώς σε αυτήν ο χρήστης μπορεί να πραγματοποιήσει τις περισσότερες ενέργειες. Σε αυτήν παρουσιάζεται για κάθε ταινία ο τίτλος της, το poster, η μέση βαθμολογία όπως έχει διαμορφωθεί από τους χρήστες, ο σκηνοθέτης, η ημερομηνία κυκλοφορίας καθώς και το είδος στο οποίο ανήκει. Η προβολή της κεντρικής σελίδας καθώς και όλες οι λειτουργίες της γίνονται στο Index View του πίνακα MoviesActors.

5.3.1 Αναζήτηση βάση βαθμολογίας

Στο πάνω μέρος αριστερά της σελίδας έχει τοποθετηθεί το κουμπί με όνομα μεγαλύτερη βαθμολογία (highest rating). Με την επιλογή αυτή πραγματοποιείται κατάταξη των ταινιών από τις ταινίες με τη μεγαλύτερη μέση βαθμολογία σε αυτές με την μικρότερη και εμφανίζονται στον χρήστη. Για τη λειτουργία αυτή αρχικά τοποθετήθηκε στο Index View του MoviesActors ένα κουμπί με όνομα sortOrder και για τιμή δόθηκε το όνομα μεγαλύτερη βαθμολογία (highest rating).

```
 
<input type="submit" name="sortOrder" value="Highest Rating" class="btn" />
```

Έπειτα την αντίστοιχη μέθοδο Index στον ελεγκτή του MoviesActors δόθηκε σαν όρισμα η μεταβλητή sortOrder όπου θα λαμβάνει την τιμή Highest Rating όταν ο

χρήστης πατάει το κουμπί, έτσι όπως έχει οριστεί στο πεδίο value του κουμπιού. Στη συνέχεια αποθηκεύουμε, με τη χρήση ερωτήματος στη βάση δεδομένων, όλα τα στοιχεία των πινάκων ταινιών και ηθοποιών.

```
var moviesCentralDBContext = _context.MoviesActors.Include(m => m.Actor).Include(m => m.Movie).Include(m => m.Movie.Ratings);
```

Συγκεκριμένα, ο context στο '_context.MoviesActors' είναι το αντικείμενο του DbContext (του στιγμιότυπου της βάσης δεδομένων) που αντιπροσωπεύει τη βάση δεδομένων. Το MoviesActors (junction table) είναι η συλλογή που αντιπροσωπεύει τον πίνακα ή το σύνολο δεδομένων που περιέχει τις σχέσεις μεταξύ ταινιών (Movies) και ηθοποιών (Actors) δηλαδή το id της κάθε ταινίας και το id του κάθε ηθοποιού. Όμως πρέπει να γίνει συλλογή και των υπόλοιπων στοιχειών και όχι μόνο των id, επομένως χρησιμοποιείται η include μέθοδος.

Η μέθοδος Include χρησιμοποιείται για την φόρτωση συνδεδεμένων οντοτήτων. Στην περίπτωση του 'Include(m=>m.Actor) ', φορτώνονται οι ηθοποιοί (Actor) που σχετίζονται με κάθε εγγραφή στον πίνακα MoviesActors βάση του id τους που λειτουργεί σαν ξένο κλειδί (foreign key) για την επικοινωνία των 2 πινάκων. Η δεύτερη κλήση της Include φορτώνει τις ταινίες (Movie) που σχετίζονται με κάθε εγγραφή στον πίνακα MoviesActors. Η τρίτη κλήση της Include πηγαίνει ένα επίπεδο πιο βαθιά και φορτώνει τις βαθμολογίες (Ratings) που σχετίζονται με κάθε ταινία. Αυτό σημαίνει ότι όχι μόνο θα φορτωθούν οι ταινίες που σχετίζονται με τις εγγραφές του πίνακα MoviesActors, αλλά και οι βαθμολογίες αυτών των ταινιών.

Συνοψίζοντας ο παραπάνω κώδικας δημιουργεί ένα ερώτημα που ανακτά δεδομένα από τη βάση δεδομένων και φορτώνει τις σχετικές οντότητες Actor, Movie και Ratings για κάθε εγγραφή στον πίνακα MoviesActors. Με αυτόν τον τρόπο, εξασφαλίζεται ότι τα δεδομένα που σχετίζονται μεταξύ τους φορτώνονται μαζί, αποφεύγοντας έτσι επιπλέον ερωτήματα στη βάση δεδομένων όταν τα δεδομένα χρησιμοποιηθούν αργότερα στον κώδικα. Αποθηκεύουμε το αποτέλεσμα αυτό με τη χρήση ερωτήματος ελέγχοντας ότι δεν έχει αποθηκευτεί κενό πεδίο στον πίνακα ταινιών.

```
var movies = from m in moviesCentralDBContext.Include(m => m.Movie)
             where m.Movie != null
             select m;
```

Έπειτα γίνεται έλεγχος αν ο χρήστης έχει πατήσει το κουμπί προβολής ταινιών με τη μεγαλύτερη βαθμολογία και αν ναι τότε τρέχει νέο ερώτημα στη βάση.

```
if (sortOrder == "Highest Rating")
{
movies = movies.OrderByDescending(m => m.Movie.Ratings.Any() ? m.Movie.Ratings.Average(r => (double)r.Rating1) : 0);
}
```

Συγκεκριμένα, ταξινομείται η συλλογή movies σε φθίνουσα σειρά με βάση την μέση βαθμολογία κάθε ταινίας. Εάν η ταινία δεν έχει βαθμολογίες, θεωρείται ότι έχει μέση βαθμολογία 0. Αυτό γίνεται χρησιμοποιώντας το OrderByDescending και το Any για να ελέγξει αν υπάρχουν βαθμολογίες, και το Average για τον υπολογισμό της μέσης βαθμολογίας όπου εμφανίζεται σαν αριθμός με 2 δεκαδικά ψηφία. Τέλος η συλλογή αυτή στέλνεται στην Index προβολή του MoviesActors για να προβληθούν τα στοιχεία της. Αν ο χρήστης δεν έχει πατήσει το κουμπί, επιστρέφεται η αρχική συλλογή χωρίς την ταξινόμιση.

```
return View(await movies.ToListAsync());
```

5.3.2 Αναζήτηση βάση είδους και τίτλου

Εκτός από την αναζήτηση ταινιών με βάση τη μεγαλύτερη βαθμολογία, οι χρήστες έχουν την δυνατότητα να αναζητήσουν την ταινία που επιθυμούν ανάλογα με το είδος στο οποίο ανήκει (genre) ή και τον τίτλο της.

Στο πάνω μέρος της σελίδας υπάρχουν ένα drop-down μενού όπου γίνεται επιλογή του είδους και ένα πεδίο όπου εισάγεται ο τίτλος της ταινίας για να γίνει αναζήτηση.

Όσον αφορά την αναζήτηση με βάση το είδος αρχικά δημιουργήθηκε το drop-down μενού, στην Index προβολή του MoviesActors, με επικεφαλίδα genre και id=’movieGenre’ και σαν επιλογές τα είδη ταινιών (δράσης, δράμα κτλ.).

```
<form asp-controller="MoviesActors" asp-action="Index" method="get">
    <p class="text-center">
        <label for="movieGenre">Genre:</label>
        <select name="movieGenre" id="movieGenre" onchange="saveSelectedGenre()">
            <option value="">Default</option>
            <option value="Action">Action</option>
            <option value="Adventure">Adventure</option>
            <option value="Biography">Biography</option>
            <option value="Crime">Crime</option>
            <option value="Drama">Drama</option>
            <option value="Sci-Fi">Sci-Fi</option>
            <option value="Romance">Romance</option>
            <option value="Western">Western</option>
        </select>
        <input type="submit" value="Apply" class="btn" />
```

Στη συνέχεια στην Index μέθοδο του MoviesActors Controller προστέθηκε ως παράμετρος η μεταβλητή movieGenre όπου εκεί θα αποθηκευτεί το είδος της ταινίας που θα επιλέξει ο χρήστης από το μενού.

```
public async Task<IActionResult> Index(string movieGenre,
```

Με βάση αυτήν την παράμετρο γίνεται έλεγχος εάν ο χρήστης έχει επιλέξει είδος ταινίας από το μενού. Αυτό γίνεται με τη χρήση της μεθόδου IsNullOrEmpty όπου ελέγχει αν η μεταβλητή είναι κενή (null). Εφόσον δεν είναι κενή εκτελείται ερώτημα όπου επεξεργάζεται στη συλλογή movies που δημιουργήθηκε προηγουμένως και αποθηκεύει σε αυτή τις ταινίες με είδος το είδος που επέλεξε από το μενού ο χρήστης και αποθηκεύτηκε στη μεταβλητή. Επιπλέον μέσα σε αυτό τον έλεγχο έχει τοποθετηθεί και ο έλεγχος του sortOrder που αναφέρθηκε προηγουμένως, έτσι ώστε αν ο χρήστης επιλέξει είδος ταινίας και στη συνέχεια πατήσει το κουμπί Highest Rating, να ταξινομηθούν μόνο οι ταινίες με το είδος αυτό.

```
if (!string.IsNullOrEmpty(movieGenre))
{
    movies = from m in moviesCentralDBContext.Include(m => m.Movie)
             where m.Movie != null && m.Movie.Genre == movieGenre
             select m;

    if (sortOrder == "Highest Rating")
    {
        movies = movies.OrderByDescending(m => m.Movie.Ratings.Any() ? m.Movie.Ratings.Average(r => (double)r.Rating1) : 0);
    }

}
```

Προχωρώντας στην αναζήτηση ταινίας με βάση τον τίτλο της δημιουργήθηκε ένα πεδίο με επικεφαλίδα ‘Title’ και όνομα ‘searchString’, στην προβολή Index του MoviesActors για να πληκτρολογήσει ο χρήστης τον τίτλο της ταινίας και ένα κουμπί με επικεφαλίδα ‘Search’ για να ξεκινήσει η αναζήτηση.

```
Title: <input type="text" name="searchString" />
<input type="submit" value="Search" class="btn" />
```

Στην αντίστοιχη μέθοδο τώρα Index του MoviesActors ελεγκτή έχει οριστεί σαν παράμετρος η μεταβλητή searchString όπου θα δέχεται ως είσοδο τον τίτλο της ταινίας που πληκτρολόγησε στο πεδίο ο χρήστης.

```
public async Task<IActionResult> Index(string movieGenre, string searchString,
```

Έπειτα αφού γίνουν αποθήκευση στην movies τα δεδομένα των ταινιών με τη χρήση ερωτήματος που αναφέρθηκε προηγουμένως, γίνεται έλεγχος εάν η παράμετρος searcString είναι κενή (δηλαδή ο χρήστης δεν έχει πληκτρολογήσει τίτλο) με τη μέθοδο IsNullOrEmpty και αν δεν είναι τότε με τη χρήση ερωτήματος αποθηκεύουμε στην movies τα δεδομένα της ταινίας με τίτλο αυτό που εισήγαγε ο χρήστης. Αυτό γίνεται με την χρήση της μεθόδου Contains (περιέχει) όπου ελέγχει αν το string που πληκτρολόγησε ο χρήστης αντιστοιχεί σε τίτλο ταινίας. Επίσης πρέπει να σημειωθεί ότι αν ο χρήστης εισάγει ένα κομμάτι του τίτλου (πχ. αντί για Inception εισάγει Ince) γίνεται αυτόματη διόρθωση. Σε περίπτωση που αυτό το κομμάτι τίτλου περιέχεται και σε άλλες ταινίες τότε θα αποθηκευτούν και τα δεδομένα αυτών (πχ. αν γίνει πληκτρολόγηση της λέξης the θα αποθηκευτούν τα δεδομένα όλων των ταινιών που περιέχουν το the στον τίτλο τους και θα εμφανιστούν σε αυτόν).

```
if (!string.IsNullOrEmpty(searchString))
{
        movies = from m in moviesCentralDBContext.Include(m => m.Movie)
                 where m.Movie != null && m.Movie.Title.Contains(searchString)
                 select m;

        if (sortOrder == "Highest Rating")
        {
        movies = movies.OrderByDescending(m => m.Movie.Ratings.Any() ? m.Movie.Ratings.Average(r => (double)r.Rating1) : 0);
        }


}
```

Έτσι όπως έγινε και προηγουμένως, μέσα στον έλεγχο για τίτλο προστέθηκε και ο έλεγχος για sortOrder έτσι ώστε ο χρήστης να μπορεί να συνδυάσει την προβολή των ταινιών βάση βαθμολογίας με τον τίτλο. Επιπροσθέτως σε περίπτωση που γίνει επιλογή είδους και τίτλου, στην movies αποθηκεύεται η ταινία ή οι ταινίες που πληρούν και τις 2 προϋποθέσεις ταυτόχρονα ή και τις 3 ταυτόχρονα.

```
if(!string.IsNullOrEmpty(movieGenre) && !string.IsNullOrEmpty(searchString))
{
    movies = from m in moviesCentralDBContext.Include(m => m.Movie)
             where m.Movie != null && m.Movie.Title.Contains(searchString) && m.Movie.Genre == movieGenre
             select m;

    if (sortOrder == "Highest Rating")
    {
        movies = movies.OrderByDescending(m => m.Movie.Ratings.Any() ? m.Movie.Ratings.Average(r => (double)r.Rating1) : 0);
    }
}
```

Έτσι η movies που θα επιστραφεί στο View για να προβάλει τα δεδομένα των ταινιών θα είναι διαφορετική κάθε φορά ανάλογα με τις επιλογές του χρήστη.

```
return View(await movies.ToListAsync());
```

Η προβολή όλων των παραπάνω δεδομένων που αναφέρθηκαν, λαμβάνει μέρος στο Index View του μοντέλου MoviesActors. Το View αυτό συντάσσεται σε cshtml δίνοντας την δυνατότητα ανάπτυξης κώδικα C# μαζί με html για την διαμόρφωση της σελίδας. Στο πάνω μέρος του View έχει δημιουργηθεί το μενού για αναζήτηση ταινίας με βάση το είδος καθώς και το πεδίο για αναζήτηση βάση τίτλου που περιγράφτηκαν προηγουμένως. Όπως είδαμε η Index προβολή του κάθε μοντέλου δημιουργείται αυτόματα με την δημιουργία του κάθε ελεγκτή. Έτσι προκειμένου να προβληθούν τα στοιχεία που θέλουμε μπορούμε να αλλάξουμε κάποιες εντολές χωρίς να χρειαστεί να γραφτεί από την αρχή ο κώδικας. Η προβολή αυτόματα είχε δημιουργήσει ένα πίνακα όπου εμφάνιζε τα πεδία του πίνακα MoviesActors (maid,movieid,actorid) με τη χρήση της εντολής '@Html.DisplayNameFor' για την επικεφαλίδα του πεδίου και '@Html.DisplayFor' για την τιμή του πεδίου. Μέσω των ξένων κλειδιών, επικοινωνώντας με τους πίνακες Movies και Actors, μπορούμε να προβάλουμε στη θέση τους στοιχεία από αυτούς τους πίνακες. Για παράδειγμα αλλάζοντας την εντολή '@Html.DisplayNameFor(model => model.MovieId)' σε '@Html.DisplayNameFor(model => model.Movie.Title)' γίνεται navigate μέσω του κλειδιού στον πίνακα των ταινιών και εμφανίζεται ο τίτλος της ταινίας. Με τον ίδιο τρόπο γίνεται επεξεργασία των υπόλοιπων πεδίων για να εμφανιστούν ο σκηνοθέτης, η ημερομηνία κυκλοφορίας και το είδος της ταινίας. Για την επικεφαλίδα της φωτογραφίας της ταινίας και της βαθμολογίας, ορίστηκαν οι τίτλοι 'Poster' και 'Rating' αντίστοιχα. Έτσι επιστρέφοντας μια συγκεκριμένη συλλογή (movies σε αυτήν την περίπτωση) θα εμφανίσει τα πεδία που ορίσαμε από την συγκεκριμένη

συλλογή και όχι από όλη τη βάση. Αντίθετα αν δεν επιστρεφόταν παράμετρος θα εμφάνιζε τα παρακάτω πεδία για όλες τις ταινίες της βάσης.

```
<table class="table">
    <thead>
        <tr class="details-row">
            <th>
                @Html.DisplayNameFor(model => model.Movie.Title)
            </th>
            <th>
                Poster
            </th>
            <th>
                Rating
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movie.Director)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movie.ReleaseDate)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Movie.Genre)
            </th>
            <th></th>
        </tr>
    </thead>
```

Όσον αφορά την ετικέτα ‘tbody’ αυτή περιέχει το σώμα του πίνακα. Εκεί εμφανίζονται τα δεδομένα των ταινιών κάτω από το αντίστοιχο όνομα στήλης. Αρχικά προκειμένου να εξασφαλιστεί ότι δεν θα εμφανίζονται τα δεδομένα μιας ταινίας παραπάνω από μια φορά, ορίστηκε η μεταβλητή ‘encounteredMovieIds’ όπου περιέχει τη λίστα ‘HashSet<int>()’ (λίστα ακεραίων) που εκεί αποθηκεύονται τα id των ταινιών των οποίων τα δεδομένα έχουν προβληθεί ήδη στον πίνακα.

```
@{
    var encounteredMovieIds = new HashSet<int>();
}
```

Στη συνέχεια δημιουργήθηκε foreach επανάληψη που θα τρέχει για κάθε ένα αντικείμενο του μοντέλου του πίνακα MoviesActors.

```
@foreach (var item in Model) {
```

Έπειτα γίνεται έλεγχος αν το id του αντικειμένου του μοντέλου MoviesActors (δηλαδή το id της ταινίας), του οποίου θα εμφανιστούν τα στοιχεία, υπάρχει στη μεταβλητή ‘encounteredMovieIds’, δηλαδή αν τα στοιχεία του έχουν ήδη προβληθεί μια φορά. Αν το id δεν υπάρχει στη λίστα τότε προστίθεται και στη συνέχεια

προβάλλονται τα δεδομένα αλλιώς διακόπτεται η επανάληψη για το συγκεκριμένο αντικείμενο και συνεχίζεται στο επόμενο.

```
(var item in Model) {
   if (!encounteredMovieIds.Contains(item.Movie.Movieid))
   {
      encounteredMovieIds.Add(item.Movie.Movieid);
```

Ο έλεγχος γίνεται με τη βοήθεια της συνάρτησης 'Contains' (περιέχει) και συγκεκριμένα ελέγχει αν το id του αντικειμένου περιέχεται στη λίστα. Το αποτέλεσμα είναι αληθές (true) η ψευδές (false) για αυτό χρησιμοποιείται το σημείο στίξης '!' που λειτουργεί σαν 'διάφορο'. Έτσι αν το αποτέλεσμα είναι ψευδές, δηλαδή το id δεν υπάρχει στη λίστα, το αποτέλεσμα με το '!' γίνεται αληθές για να συνεχιστεί η επανάληψη. Για την προβολή των δεδομένων δημιουργείται νέα γραμμή στον πίνακα για κάθε αντικείμενο με τη χρήση της ετικέτας 'tr'. Κάθε γραμμή περιλαμβάνει πολλές στήλες (<td>), καθεμία από τις οποίες περιέχει διαφορετικές πληροφορίες για την ταινία.

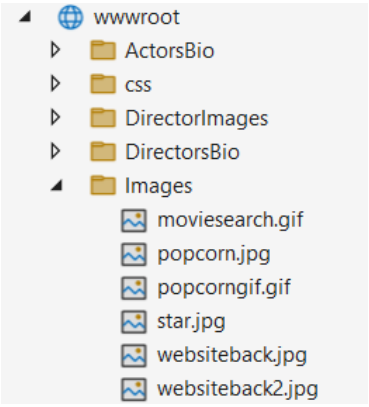
Η πρώτη στήλη εμφανίζει τον τίτλο της ταινίας χρησιμοποιώντας την μέθοδο '@HtmlDisplayFor(modelItem => item.Movie.Title). Με τη μέθοδο αυτή αποδίδεται η τιμή ενός αντικειμένου του μοντέλου Movies, στη συγκεκριμένη περίπτωση ο τίτλος. Το 'Html.DisplayFor' είναι μια μέθοδος το 'HtmlHelper' που παρέχεται από το ASP.NET MVC framework και χρησιμοποιείται για αυτή τη λειτουργία. Συγκεκριμένα δημιουργεί κώδικα html για την εμφάνιση της τιμής του αντικειμένου. Το 'modelItem => model.Movies.Title' είναι μια έκφραση 'lambda' που υποδεικνύει ποια ιδιότητα του μοντέλου πρέπει να αποδοθεί. Σε αυτήν την περίπτωση, το 'modelItem' είναι μια παράμετρος που αντιπροσωπεύει κάθε στοιχείο της συλλογής του μοντέλου (δηλαδή κάθε ταινία), και το 'item.Movie.Title' είναι η ιδιότητα 'Title' της ταινίας που θέλουμε να εμφανίσουμε.

```
<td>
    @Html.DisplayFor(modelItem => item.Movie.Title)
</td>
```

Η δεύτερη στήλη εμφανίζει την εικόνα της ταινίας με την ετικέτα '<img>' και το URL για τη φόρτωση της λαμβάνεται από την ιδιότητα Posturl. Οι τιμές των στοιχείων στο style ρυθμίζουν το ύψος, το πλάτος και τον τρόπο που θα προβληθεί η εικόνα αντίστοιχα.

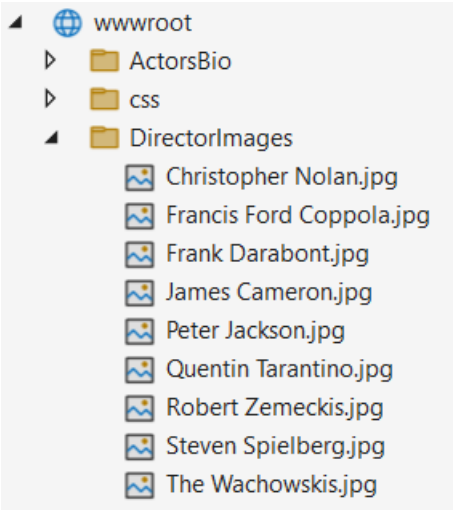
```
<td>
    <img src="@item.Movie.Posturl" alt="@item.Movie.Title" style="max-height: 300px; max-width: 300px;
</td>
```

Η τρίτη στήλη υπολογίζει τη μέση βαθμολογία κάθε ταινίας από τη συλλογή Ratings και την εμφανίζει μέσω της μεταβλητής 'averageRating' με δύο δεκαδικά ψηφία. Επίσης εμφανίζεται και μια εικόνα αστεριού δίπλα από τη βαθμολογία. Η εικόνα του αστεριού έχει αποθηκευτεί ως 'star.jpg' στο φάκελο 'Images' κάτω από το wwwroot στο Solution explorer του Visual studio και ανακτάται με τη βοήθεια της ετικέτας 'img' εισάγοντας ως 'src' το μονοπάτι μέχρι αυτήν.




```
@{
    double averageRating = item.Movie.Ratings.Any() ? item.Movie.Ratings.Average(r => (double)r.Rating1) : 0;
}
@averageRating.ToString("F2")
<img src="~/Images/star.jpg" style="width: 40px; height: 30px;" />
```

Η τέταρτη στήλη εμφανίζει το όνομα του σκηνοθέτη της ταινίας με την χρήση του '@HtmlDisplayFor' που αναφέρθηκε προηγουμένως. Κάτω από το όνομα του σκηνοθέτη εμφανίζεται μια φωτογραφία του με τη χρήση της ετικέτας 'img'. Η φωτογραφία του κάθε σκηνοθέτη έχει αποθηκευτεί στον φάκελο 'DirectorImages' κάτω από το 'wwwroot' με όνομα το όνομα το σκηνοθέτη '.jpg'.



Για την ανάκτησή της δίνεται σαν 'src' η διαδρομή μέχρι το 'DirectorImages' και όνομα αρχείου το αποτέλεσμα του 'DispalyFor' που είναι το όνομα του σκηνοθέτη συν '.jpg'. Έτσι εμφανίζεται η φωτογραφία του συγκεκριμένου καθώς η φωτογραφία κάθε σκηνοθέτη έχει αποθηκευτεί ως το όνομά του.

```
<img src="~/DirectorImages/@(item.Movie.Director).jpg" alt="@item.Movie.Director" style="width: 100px; height: auto;
```

Η γραμμή αυτή κώδικα έχει τοποθετηθεί μέσα στην ετικέτα 'ahref' ,δηλαδή έναν υπερσύνδεσμο. Ο σύνδεσμος επιτρέπει στον χρήστη να κάνει κλικ πάνω στη φωτογραφία του σκηνοθέτη και να οδηγηθεί σε ένα νέο View το 'DirectorDetails' όπου εκεί εμφανίζεται ένα σύντομο βιογραφικό για αυτόν καθώς και οι ταινίες που έχει σκηνοθετήσει. Οι τελευταίες 2 στήλες εμφανίζουν την ημερομηνία κυκλοφορίας και το είδος της ταινίας.

5.3.3 Προβολή βιογραφικού σκηνοθέτη

Για την προβολή του βιογραφικού του σκηνοθέτη μέσω της κεντρικής σελίδας αρχικά δημιουργήθηκε η μέθοδος 'DirectorDetails' στον MoviesActors Controller. Η μέθοδος αυτή παίρνει σαν παράμετρο το 'directorName' που έχει το όνομα του σκηνοθέτη της φωτογραφίας που επέλεξε ο χρήστης. Η μεταβλητή 'directorName' λαμβάνει το όνομα του σκηνοθέτη μέσω της 'HtmlDisplayFor' και δίνοντας ως είσοδο στην 'ahref' το κατάλληλο μονοπάτι, μεταφέρεται στην 'DirectorDetails' σαν παράμετρος.

```
<a href="/MoviesActors/DirectorDetails?directorName=@item.Movie.Director" class="director-link">
```

```
0 references
public IActionResult DirectorDetails(string? directorName)
```

Αφού αποθηκευτεί το όνομα στη συνέχεια ορίζουμε την μεταβλητή '_webHostEnvironment' της ιδιότητας 'IWebHostEnvironment' στον constructor του MoviesActors Controller. Το webhost είναι πακέτο που θα κάνει ευκολότερη την περιήγηση σε διαδρομές αρχείων.

```
public class MoviesActorsController : Controller
{
    private readonly MoviesCentralDBContext _context;

    private readonly IWebHostEnvironment _webHostEnvironment;


    0 references
    public MoviesActorsController(MoviesCentralDBContext context, IWebHostEnvironment webHostEnvironment)
    {
        _context = context;

        _webHostEnvironment = webHostEnvironment;
    }
```

Έπειτα πρέπει να ανακτήσουμε το αρχείο που περιέχει το βιογραφικό του σκηνοθέτη που θα προβληθεί. Το βιογραφικό έχει αποθηκευτεί ως αρχείο κειμένου στο φάκελο 'DirectorsBio' κάτω από το 'wwwroot' με όνομα το όνομα του σκηνοθέτη όπως και με την φωτογραφία προηγουμένως. Για την ανάκτηση χρησιμοποιήθηκε το webhost.

Αρχικά, ορίστηκε η μεταβλητή 'webrootPath' που αποθηκεύει τη διαδρομή του root φακέλου της εφαρμογής web. Η ιδιότητα WebRootPath της IWebHostEnvironment περιέχει τη φυσική διαδρομή στον διακομιστή όπου αποθηκεύονται τα στατικά αρχεία της εφαρμογής, όπως αρχεία HTML, CSS, JavaScript και εικόνες. Έπειτα δημιουργήθηκε η μεταβλητή 'bioFilePath' που αποθηκεύει την πλήρη διαδρομή του αρχείου του βιογραφικού του σκηνοθέτη. Η μέθοδος Path.Combine συνδυάζει τη διαδρομή του web root (webRootPath), το όνομα του φακέλου "DirectorsBio" και το όνομα του αρχείου του βιογραφικού το οποίο προκύπτει από το όνομα του σκηνοθέτη (directorName) με την επέκταση ".txt". Έτσι, δημιουργείται η πλήρης διαδρομή για το αρχείο βιογραφικού μέσα στον φάκελο

"DirectorsBio". Τέλος η μέθοδος System.IO.File.ReadAllText διαβάζει το περιεχόμενο του αρχείου που βρίσκεται στη διαδρομή bioFilePath και το αποθηκεύει στη μεταβλητή bio. Η μέθοδος αυτή ανοίγει το αρχείο, διαβάζει όλο το περιεχόμενό του ως κείμενο και κλείνει το αρχείο μετά την ολοκλήρωση της ανάγνωσης. Αν το αρχείο υπάρχει και είναι προσβάσιμο, το περιεχόμενό του θα αποθηκευτεί στη μεταβλητή bio ως μια συμβολοσειρά (string). Αν το αρχείο δεν υπάρχει ή δεν είναι προσβάσιμο για κάποιο λόγο (π.χ., αν δεν υπάρχουν τα κατάλληλα δικαιώματα πρόσβασης), θα προκληθεί εξαίρεση (exception).

```
// Retrieve the director's bio from the text file based on the director's name
string webRootPath = _webHostEnvironment.WebRootPath;
string bioFilePath = Path.Combine(webRootPath, "DirectorsBio", $"{directorName}.txt");
string bio = System.IO.File.ReadAllText(bioFilePath);
```

Οι παραπάνω μεταβλητές αποθηκεύονται σε ViewBag. Με την χρήση των μεταβλητών αυτών μπορούμε να μεταφέρουμε τις τιμές τους από τον Controller στο αντίστοιχο View.

```
ViewBag.DirectorsName = directorName;
ViewBag.PicturePath = $"~/DirectorImages/{directorName}.jpg";
ViewBag.Bio = bio;
```

Τέλος για να εμφανίζονται στο 'DirectorsDetails' View οι ταινίες του σκηνοθέτη κάτω από το βιογραφικό του, θα τρέξουμε ένα ερώτημα στον πίνακα Movies ώστε να ανακτήσουμε τις ταινίες όπου το πεδίο 'Director' έχει το όνομα που έχει αποθηκευτεί στο 'directorName' και το αποτέλεσμα θα αποθηκευτεί σε ViewBag για να το χρησιμοποιήσουμε στο View.

```
var movies = _context.Movies.Where(m => m.Director == directorName).ToList();

ViewBag.Movies = movies;
```

Με δεξί κλικ στο όνομα της μεθόδου όπως αναφέρθηκε προηγουμένως, επιλέγουμε προσθήκη προβολής (add view) στο μενού που εμφανίζεται και στη συνέχεια στο νέο μενού κενή προβολή (empty view). Στο 'DirectorDetails' View

λοιπόν, αρχικά με χρήση html και τις ετικέτας ‘img’ εμφανίζουμε την εικόνα του σκηνοθέτη που είχε αποθηκευτεί στο ViewBag στον Controller. Η ετικέτα τοποθετείται μέσα σε ένα container για καλύτερη στοίχηση.

```
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-4">
            <img src="@Url.Content(ViewBag.PicturePath)" alt="Director Image" style="width: 300px; height: auto;" />
        </div>
    </div>
```

Έπειτα δημιουργήθηκε ένα κομμάτι κώδικα για την στοίχιση και καλύτερη εμφάνιση του βιογραφικού του σκηνοθέτη. Αρχικά ορίστηκαν δύο μεταβλητές ο ‘paragraphBuilder’ που χρησιμοποιείται για τη συσσώρευση των παραγράφων που θα δημιουργηθούν και στο τέλος θα περιέχει το πλήρες μορφοποιημένο κείμενο και ο ‘currentParagraph’ που χρησιμοποιείται για τη συσσώρευση των λέξεων της τρέχουσας παραγράφου. Το βιογραφικό που είχε αποθηκευτεί στο ViewBag μεταφέρεται στη μεταβλητή bio, το ‘wordsPerParagraph’ αρχικοποιείται στο 100 και το ‘wordcount’ στο 0.

```
@{
    string bio = ViewBag.Bio;
    int wordsPerParagraph = 100;
    int wordsCount = 0;

    StringBuilder paragraphBuilder = new StringBuilder();
    StringBuilder currentParagraph = new StringBuilder();
```

Η λογική που ακολουθήθηκε είναι ότι θα δημιουργείται παράγραφος στο κείμενο κάθε 100 λέξεις εκτός αν η 100^{η} λέξη είναι στη μέση της πρότασης επομένως η παράγραφος συνεχίζεται μέχρι να βρεθεί τελία. Στη συνέχεια το βιογραφικό χωρίζεται σε λέξεις με βάση τα κενά διαστήματα και κάθε λέξη που προκύπτει προστίθεται στην ‘CurrentParagraph’ και αυξάνεται ο μετρητής λέξεων ‘wordcount’.

```
foreach (string word in bio.Split(' '))
{
    currentParagraph.Append(word);
    currentParagraph.Append(" ");
    wordsCount++;
```

Έτσι όταν το ‘wordcount’ γίνει 100 δηλαδή έχουν διαβαστεί 100 λέξεις τότε γίνεται έλεγχος αν υπάρχει τελεία στην τελευταία λέξη και αν ναι τότε προστίθεται η τρέχουσα παράγραφος στον paragraphBuilder ως HTML παράγραφος (<p>) και η ‘currentParagraph’ καθαρίζεται και το ‘wordcount’ γίνεται μηδέν. Αλλιώς αν αν ο αριθμός των λέξεων είναι τουλάχιστον ίσος με το wordsPerParagraph (100) αλλά δεν περιέχει τελεία τότε αναζητείται η τελευταία τελεία στην τρέχουσα παράγραφο με τη χρήση του ‘LastIndexOf(.)’. Αν βρεθεί τότε δημιουργείται παράγραφος μέχρι εκεί και προστίθεται στο ‘paragraphBuilder’ και η επεξεργασμένη λέξη αφαιρείται από το ‘currentParagraph’ και ο μετρητής λέξεων μηδενίζεται.

```
if (wordsCount >= wordsPerParagraph && word.Contains('.'))
{
    paragraphBuilder.Append("<p>");
    paragraphBuilder.Append(currentParagraph);
    paragraphBuilder.Append("</p>");
    currentParagraph.Clear();
    wordsCount = 0;
}
else if (wordsCount >= wordsPerParagraph)
{
    // Find the index of the last period in the current paragraph
    int lastPeriodIndex = currentParagraph.ToString().LastIndexOf('.');
    if (lastPeriodIndex != -1)
    {
        string paragraphText = currentParagraph.ToString().Substring(0, lastPeriodIndex + 1);
        paragraphBuilder.Append("<p>");
        paragraphBuilder.Append(paragraphText);
        paragraphBuilder.Append("</p>");

        // Remove the processed part from the current paragraph
        currentParagraph.Remove(0, lastPeriodIndex + 1);
        wordsCount = 0;
    }
```

Αν υπάρχει υπόλοιπο κείμενο στην currentParagraph μετά την επανάληψη, προστίθεται ως η τελευταία HTML παράγραφος.

```
// Add the final paragraph
if (currentParagraph.Length > 0)
{
    paragraphBuilder.Append("<p>");
    paragraphBuilder.Append(currentParagraph);
    paragraphBuilder.Append("</p>");
}

string formattedBio = paragraphBuilder.ToString();
@Html.Raw(formattedBio)
```

Τέλος το περιεχόμενο του paragraphBuilder μετατρέπεται σε συμβολοσειρά (formattedBio). Χρησιμοποιείται η μέθοδος @Html.Raw για την απόδοση του

formattedBio ως HTML στην προβολή. Η Html.Raw χρησιμοποιείται για να μην εκτελείται HTML κωδικοποίηση στην έξοδο, επιτρέποντας την απευθείας εμφάνιση των HTML στοιχείων.

Για την προβολή των ταινιών του σκηνοθέτη, γίνεται επανάληψη με foreach βρόχο για κάθε ταινία που έχει αποθηκευτεί στο ViewBag και εμφανίζεται το poster της ταινίας με την χρήση της ετικέτας 'img'. Η ετικέτα αυτή έχει τοποθετηθεί μέσα σε έναν υπερσύνδεσμο 'ahref' έτσι ώστε ο χρήστης να έχει τη δυνατότητα να επισκεφτεί την σελίδα της ταινίας πατώντας πάνω στο poster της.

```
<h2>Movies by @ViewBag.DirectorsName:</h2>
<div class="movie-posters mt-5">
    @foreach (var movie in ViewBag.Movies)
    {
        <div class="movie-poster">
            <!-- Movie poster as a clickable link -->

            <a href="@Url.Action("Index", "MoviesActors", new { searchString = movie.Title })">
                <img src="@movie.Posturl" alt="@movie.Title" style="width: 150px; height: auto;">
            </a>

        </div>
    }
</div>
```

Για την προβολή της σελίδας της ταινίας, χρησιμοποιήθηκε το '@Url.Action' μέσω του οποίου ορίζουμε στην παράμετρο 'searchString' της μεθόδου Index του MoviesActors Controller τον τίτλο της ταινίας έτσι ώστε να προβληθούν μόνο αυτά τα δεδομένα και όχι όλων των ταινιών.

5.4 Βαθμολόγηση ταινίας

Μια σημαντική λειτουργία σε εφαρμογή περιήγησης ταινιών είναι η απόδοση βαθμολογίας στις ταινίες. Η λειτουργία αυτή είναι σημαντική για τους εξής λόγους:

Α) Οδηγός για τους θεατές: Οι βαθμολογίες παρέχουν μια γρήγορη και εύκολη αναφορά για την ποιότητα μιας ταινίας ή μιας σειράς, βοηθώντας τους θεατές να αποφασίσουν τι να παρακολουθήσουν.

Β) Αντικειμενική Αξιολόγηση: Η συλλογή και ανάλυση των βαθμολογιών από ένα μεγάλο πλήθος χρηστών προσφέρει μια πιο αντικειμενική εικόνα για την αξία ενός έργου, σε αντίθεση με την υποκειμενική κριτική ενός ατόμου.

Γ) Ανταγωνιστικότητα: Οι δημιουργοί και παραγωγοί ταινιών/σειρών χρησιμοποιούν τις βαθμολογίες για να αξιολογήσουν την ανταπόκριση του κοινού και να βελτιώσουν μελλοντικά έργα τους.

Δ) Δημιουργία Κοινοτήτων: Οι βαθμολογίες προωθούν τη συμμετοχή και την αλληλεπίδραση των χρηστών, δημιουργώντας κοινότητες που μοιράζονται παρόμοια γούστα και απόψεις.

Ε) Προώθηση Καλύτερου Περιεχομένου: Τέλος, οι υψηλές βαθμολογίες βοηθούν να αναδειχθούν αξιόλογες ταινίες και σειρές που μπορεί να μην έχουν μεγάλο μπάτζετ διαφήμισης, αλλά αξίζουν την προσοχή του κοινού.

Έτσι δημιουργήθηκε ένα σύστημα βαθμολογιών όπου κάθε χρήστης βαθμολογεί μια ταινία από 1-10 και η μέση βαθμολογίας μιας ταινίας ,έτσι όπως έχει προκύψει από τους χρήστες, εμφανίζεται στον πίνακα της κεντρικής σελίδας μαζί με τα υπόλοιπα δεδομένα (τίτλος, ημερομηνία κυκλοφορίας κτλ.). Προκειμένου ο χρήστης να βαθμολογήσει μια ταινία, επιλέγει το κουμπί ‘Rate Movie’ δεξιά από τα δεδομένα κάθε ταινίας. Το κουμπί αυτό με τη χρήση του ‘@Url.Action’ στέλνει ως παράμετρο στη μέθοδο ‘Create’ του ‘Ratings Controller’ το id της ταινίας που επέλεξε να κάνει rate ο χρήστης μέσω της μεταβλητής movieId.

```
<a href="@Url.Action("Create", "Ratings", new { movieId = item.Movie.Movieid })" class="btn">Rate Movie</a>
 
```

Προχωρώντας στην μέθοδο ‘Create’ αρχικά γίνεται ανάκτηση του id του χρήστη που βαθμολογεί την ταινία από το session που ήταν αποθηκευμένο αμέσως μετά την είσοδό του στην εφαρμογή και αποθηκεύεται σε Viewbag.

```
int userId = HttpContext.Session.GetInt32("UserId") ?? 0;

// Pass the user ID to the view
ViewBag.UserId = userId;
```

Έπειτα μέσω επανάληψης δημιουργείται ένα unique id για την καταχώρηση της βαθμολογίας στην βάση δεδομένων. Συγκεκριμένα μέσω της μεθόδου ‘Random’ γίνεται generate ένας τυχαίος ακέραιος από το 500 μέχρι τον μεγαλύτερο 32-bit ακέραιο. Στη συνέχεια μέσω του ‘_context’,μέσα σε ‘do-while’ βρόχο, τρέχει ερώτημα στον πίνακα ‘Ratings’ έτσι ώστε ο ακέραιος που προέκυψε να είναι

μοναδικός. Αν δεν είναι τότε συνεχίζει η επανάληψη μέχρι να βρεθεί μοναδικός. Μόλις βρεθεί αποθηκεύεται σε ViewBag.

```
Random rnd = new Random();
int minRatingId = 500;
int maxRatingId = 2147483647; // Maximum signed 32-bit integer value
int ratingId;

// Generate a unique ratingId
do
{
    ratingId = rnd.Next(minRatingId, maxRatingId);
}
while (_context.Ratings.Any(r => r.Ratingid == ratingId));

ViewBag.RatingId = ratingId;
```

Προχωρώντας τώρα στο ‘Create’ View του μοντέλου Ratings έχει δημιουργηθεί μια φόρμα για την καταχώρηση της βαθμολογίας. Το πρώτο πεδίο της φόρμας αφορά το id της βαθμολογίας που θα αποθηκευτεί στη βάση. Το πεδίο αυτό είναι κρυφό στο χρήστη καθώς και συμπληρώνεται αυτόματα με το id της βαθμολογίας που είχε αποθηκευτεί στο ViewBag προηγουμένως, με την χρήση των ιδιοτήτων type και value αντίστοιχα στην γραμμή του input.

```
<div class="form-group" style="display: none;">
    <label asp-for="Ratingid" class="control-label"></label>
    <input id="ratingIdInput" asp-for="Ratingid" class="form-control" readonly="readonly" value="@ViewBag.RatingId" type="hidden" />
    <span asp-validation-for="Ratingid" class="text-danger"></span>
</div>
```

Το δεύτερο πεδίο αφορά το id της ταινίας που γίνεται rate. Για την ανάκτησή του δημιουργήθηκε ένα script κάτω από την html φόρμα. Αρχικά γίνεται ανάκτηση του id από το url της εφαρμογής και αποθηκεύεται στην μεταβλητή ‘movieid’.

```
// Get the movieId parameter from the URL
const urlParams = new URLSearchParams(window.location.search);
const movieId = urlParams.get('movieId');
```

Έπειτα ορίστηκε id για το πεδίο της φόρμας movieId και με βάση αυτό συμπληρώνεται αυτόματα με την τιμή που ανακτήθηκε. Το πεδίο είναι και αυτό κρυφό στο χρήστη με την ιδιότητα ‘type’.

```
<div class="form-group" style="display: none;">
    <label asp-for="Movieid" class="control-label"></label>
    <input id="movieIdInput" asp-for="Movieid" class="form-control" readonly="readonly" type="hidden" />
    <span asp-validation-for="Movieid" class="text-danger"></span>
</div>
```

```
// Set the movieId value in the input field
document.getElementById('movieIdInput').value = movieId;
```

Το τρίτο πεδίο όπως και τα δύο προηγούμενα είναι και αυτό κρυφό προς τον χρήστη και αφορά το id του χρήστη που βαθμολογεί την ταινία. Συμπληρώνεται και αυτό αυτόματα με script. Ακολουθήθηκε η προηγούμενη προσέγγιση, δηλαδή ορίστηκε id για το πεδίο UserId και αφού έγινε ανάκτηση του id του χρήστη μέσω του ViewBag που είχε χρησιμοποιηθεί προηγουμένως και αναζητώντας το id του πεδίου συμπληρώθηκε.

```
<div class="form-group" style="display:none;">
    <label asp-for="Userid" class="control-label"></label>
    <input id="Userid" asp-for="Userid" class="form-control" readonly type="hidden" />
</div>
```

```
const userId = '@ViewBag.UserId';

// Set the movieId value in the input field
document.getElementById('movieIdInput').value = movieId;
document.getElementById('Userid').value = userId;
```

Συγκεκριμένα η γραμμή κώδικα ‘document.getElementById().value’, που χρησιμοποιήθηκε για καταχώρηση id στα πεδία ταινίας και χρήστη , αναζητά το πεδίο με id αυτό που έχει δοθεί στην παρένθεση και μέσω του ‘.value’ το συμπληρώνει με την τιμή που δόθηκε δεξιά του ‘=’. Τέλος το τέταρτο και πιο σημαντικό πεδίο της φόρμας είναι το μοναδικό πεδίο που είναι ορατό στον χρήστη. Εκεί θα εισάγει την βαθμολογία της ταινίας. Προκειμένου να μην είναι ένα κενό πεδίο όπου θα πληκτρολογείται απλά η βαθμολογία, αναπτύχτηκε κώδικας σε Javascript και σε html για να είναι εμφανισιακά καλύτερο.

```
<div class="form-group" id="starRating">
    <label class="control-label"></label>
    <div class="stars" data-rating="0">
        <span class="star" data-value="1">&#9733;</span>
        <span class="star" data-value="2">&#9733;</span>
        <span class="star" data-value="3">&#9733;</span>
        <span class="star" data-value="4">&#9733;</span>
        <span class="star" data-value="5">&#9733;</span>
        <span class="star" data-value="6">&#9733;</span>
        <span class="star" data-value="7">&#9733;</span>
        <span class="star" data-value="8">&#9733;</span>
        <span class="star" data-value="9">&#9733;</span>
        <span class="star" data-value="10">&#9733;</span>
    </div>
    <input asp-for="Rating1" id="ratingValue" class="form-control" type="hidden" />
    <span asp-validation-for="Rating1" class="text-danger"></span>
</div>
<div class="form-group text-center">
    <input type="submit" value="Submit" class="btn btn-primary" />
</div>
```

Α) Container: Το είναι το container που περιέχει το σύστημα βαθμολόγησης.

Β) Label: Το <label class="control-label"></label> είναι μια ετικέτα για το σύστημα βαθμολόγησης.

Γ) Stars: Το περιέχει τα 10 αστέρια. Κάθε αστέρι είναι ένα με κλάση star και ένα data-value που δείχνει την τιμή του αστεριού (1 έως 10). Το σύμβολο ★ είναι το σύμβολο του αστεριού (★).

Δ) Hidden Input: Το <input asp-for="Rating1" id="ratingValue" class="form-control" type="hidden" /> είναι ένα κρυφό πεδίο εισόδου που αποθηκεύει την επιλεγμένη βαθμολογία.

Ε) Validation: Το είναι ένα στοιχείο για την εμφάνιση μηνυμάτων επικύρωσης.

Έτσι όταν ο χρήστης επιλέγει να βαθμολογήσει μια ταινία θα εμφανίζονται στην οθόνη του δέκα αστέρια και θα επιλέγει τόσα όσα η βαθμολογία που θέλει από 1 έως 10. Για την λειτουργικότητα των αστεριών αναπτύχτηκε κώδικας Javascript.

```
document.addEventListener('DOMContentLoaded', function () {
    const stars = document.querySelectorAll('.star');
    const ratingValue = document.getElementById('ratingValue');

    stars.forEach((star) => {
        star.addEventListener('click', () => {
            const value = parseInt(star.getAttribute('data-value'));
            ratingValue.value = value; // Set the rating value to the hidden input
            stars.forEach((s, i) => {
                if (i < value) {
                    s.style.color = 'yellow'; // Change color to yellow for selected stars
                } else {
                    s.style.color = ''; // Reset color for unselected stars
                }
            });
        });
    });
});
```

Α) DOMContentLoaded: Το script εκτελείται όταν το DOM φορτωθεί πλήρως.

Β) Αναφορά στα Αστέρια: Το const stars = document.querySelectorAll('.star'); συλλέγει όλα τα στοιχεία με την κλάση star.

Γ) Αναφορά στο Hidden Input: Το const ratingValue = document.getElementById('ratingValue'); αναφέρεται στο κρυφό πεδίο εισόδου.

Δ) Event Listener: Για κάθε αστέρι, προστίθεται ένα click event listener δηλαδή μια αντίδραση όταν πατιέται από τον χρήστη.

Ε) Ανάγνωση της Τιμής: Το const value = parseInt(star.getAttribute('data-value')); διαβάζει την τιμή του αστεριού.

Ζ) Ορισμός Τιμής: Το ratingValue.value = value; ορίζει την τιμή στο κρυφό πεδίο.

Στ) Αλλαγή Χρώματος: Το script αλλάζει το χρώμα των επιλεγμένων αστεριών σε κίτρινο και επαναφέρει το χρώμα των μη επιλεγμένων αστεριών.

Αυτός ο κώδικας δημιουργεί μια διαδραστική εμπειρία για τους χρήστες, επιτρέποντάς τους να βαθμολογούν μια ταινία ή ένα προϊόν εύκολα με το πάτημα ενός αστεριού, και να βλέπουν οπτικά την επιλογή τους. Μόλις επιλέξει τα αστέρια ο χρήστης κάνει κλικ στο κουμπί ‘Submit’ και η βαθμολογία του καταχωρείται στη βάση.

5.5 Κεντρική σελίδα ταινίας

Μόλις ο χρήστης αναζητήσει την ταινία που επιθυμεί, προκειμένου να δει περισσότερες πληροφορίες για αυτήν επιλέγει το κουμπί ‘Details’. Αυτό θα τον

οδηγήσει στην κεντρική σελίδα της ταινίας. Τα δεδομένα που παρουσιάζονται και η μορφοποίησή τους έχει γίνει με html. Εκεί το πρώτο στοιχείο που παρουσιάζεται είναι ο τίτλος της ταινίας. Η προβολή του τίτλου γίνεται με τη χρήση του '@Html.DisplayFor' και λάμδα έκφρασης. Μέσω του μοντέλου MoviesActors και του movieid που λειτουργεί ως ξένο κλειδί που συνδέει το μοντέλο αυτό με το μοντέλο Movies γίνεται ανάκτηση του τίτλου. Η γραμμή αυτή κώδικα τοποθετείται μέσα σε επικεφαλίδα '<h1>'. Το '<h1>' καθορίζει το μέγεθος της επικεφαλίδας.

```
<h1>@Html.DisplayFor(model => model.Movie.Title)</h1>
```

Για την καλύτερη μορφοποίηση των υπόλοιπων στοιχείων δημιουργήθηκε ένα container που θα τα περιέχει.

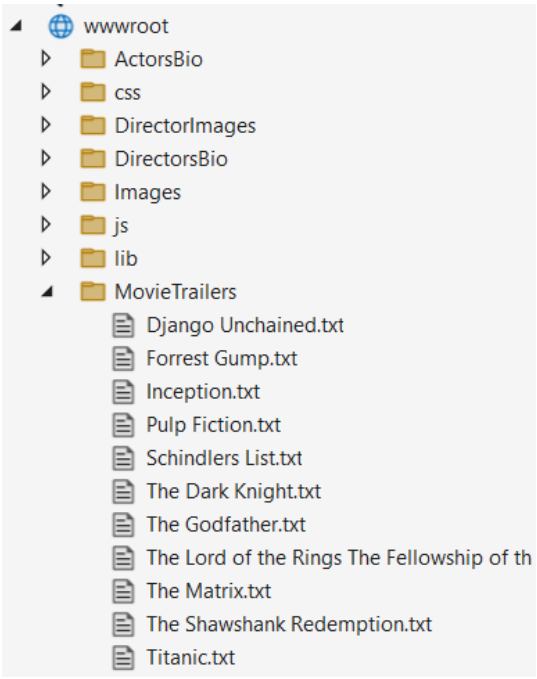
```
<div class="container movie-details-container">
```

Αριστερά του container εμφανίζεται το poster της ταινίας. Η ανάκτησή του έγινε με τον ίδιο τρόπο με τον τίτλο της ταινίας δηλαδή με την χρήση του '@Html.DisplayFor' και την επικοινωνία των μοντέλων MoviesActors και Movies μέσω του ξένου κλειδιού movieid. Συγκεκριμένα ανακτήθηκε το πεδίο 'PostUrl' από τον πίνακα Movies που περιέχει το url της φωτογραφίας. Έτσι για να εμφανιστεί δόθηκε σαν τιμή στο 'src' στην ετικέτα 'img'.

```
<img src="@Model.Movie.Posturl" alt="@Model.Movie.Title" class="img-fluid" style="max-width: 300px; height: auto;" />
```

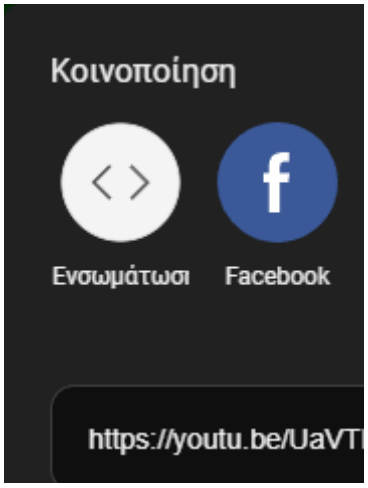
Δεξιά του container εμφανίζεται το trailer της ταινίας. Η λειτουργία του έγινε ως εξής:

Αρχικά δημιουργήθηκε ο φάκελος με όνομα 'MovieTrailers' κάτω από το wwwroot. Στο φάκελο αυτό δημιουργήθηκε ένα αρχείο κειμένου για κάθε ταινία στη βάση, με όνομα τον τίτλο της κάθε ταινίας, και μέσα σε αυτό αποθηκεύτηκε ο σύνδεσμος του trailer της ταινίας στο youtube.



Εικόνα 5.2

Συγκεκριμένα πηγαίνοντας στο trailer που θέλουμε να αποθηκεύσουμε επιλέγουμε κοινοποίηση και στο επόμενο μενού που εμφανίζεται ενσωμάτωση.



Εικόνα 5.3

Τέλος μετά την ενσωμάτωση παρουσιάζεται κώδικας html όπου αποθηκεύουμε την τιμή του ‘src’ στο αρχείο κειμένου.

```
src="https://www.youtube.com/embed/
UaVTIH8mujA?si=VVDXB1A1f9b7RgSt"
```

Προκειμένου να διαβαστεί το περιεχόμενο αυτό από το αρχείο δημιουργήθηκε αντίστοιχος κώδικας στην Details μέθοδο του MoviesActors controller. Η μέθοδος δέχεται σαν παράμετρο το αναγνωριστικό της ταινίας στην οποία πάτησε το κουμπί 'Details' ο χρήστης. Επιπλέον, μέσω ενός ερωτήματος και της σύνδεσης των πινάκων MoviesActors, Movies και Actors γίνεται ανάκτηση των δεδομένων των ηθοποιών και ταινιών που σχετίζονται με την ταινία με το συγκεκριμένο αναγνωριστικό και αποθηκεύονται στην 'moviesActors'.

```
0 references
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }



    var moviesActor = await _context.MoviesActors
        .Include(m => m.Actor)
        .Include(m => m.Movie)
        .FirstOrDefaultAsync(m => m.Maid == id);
    if (moviesActor == null)
    {
        return NotFound();
    }
```

Στη συνέχεια γίνεται ανάθεση της τιμής του τίτλου της ταινίας στη μεταβλητή movieTitle. Η τιμή αυτή λαμβάνεται από την ιδιότητα Title του αντικειμένου Movie το οποίο επικοινωνεί με το MoviesActors με την βοήθεια του ξένου κλειδιού 'movieid'. Στη συνέχεια, δημιουργείται η μεταβλητή directoryPath η οποία καθορίζει τη διαδρομή του φακέλου που περιέχει τα τρέιλερ ταινιών. Η διαδρομή αυτή σχηματίζεται με τη χρήση της μεθόδου Path.Combine, η οποία συνδυάζει τη ρίζα της ιστοσελίδας, που βρίσκεται στην ιδιότητα WebRootPath του αντικειμένου _webHostEnvironment, με τον φάκελο "MovieTrailers".

```
string movieTitle = moviesActor.Movie.Title;
string directoryPath = Path.Combine(_webHostEnvironment.WebRootPath, "MovieTrailers");
string[] trailerFiles = Directory.GetFiles(directoryPath, "*.txt");
```

Μετά, δημιουργείται ένας πίνακας από συμβολοσειρές, ο οποίος ονομάζεται trailerFiles και περιέχει τα ονόματα όλων των αρχείων με επέκταση ".txt" που

βρίσκονται στη διαδρομή που ορίζεται από τη μεταβλητή directoryPath. Αυτή η λίστα των αρχείων λαμβάνεται με τη χρήση της μεθόδου Directory.GetFiles. Στη συνέχεια, δημιουργείται μια μεταβλητή trailerUrl και αρχικοποιείται με την τιμή null.
Ακολουθεί μια επανάληψη foreach, η οποία διατρέχει κάθε αρχείο στον πίνακα trailerFiles. Για κάθε αρχείο, αφαιρείται η επέκταση του αρχείου χρησιμοποιώντας τη μέθοδο Path.GetFileNameWithoutExtension και το αποτέλεσμα αποθηκεύεται στη μεταβλητή fileName. Μέσα στην επανάληψη, γίνεται σύγκριση του fileName με τον τίτλο της ταινίας movieTitle, χρησιμοποιώντας την μέθοδο Equals με την παράμετρο StringComparison.OrdinalIgnoreCase για να εξασφαλιστεί η ευαισθησία σε πεζά-κεφαλαία.

```
string trailerUrl = null;
foreach (var file in trailerFiles)
{
    string fileName = Path.GetFileNameWithoutExtension(file);
    if (fileName.Equals(movieTitle, StringComparison.OrdinalIgnoreCase))
    {
        // Found the matching file, read the trailer URL
        trailerUrl = System.IO.File.ReadAllText(file);
        break;
    }
}
```

Εάν βρεθεί ένα αρχείο με όνομα που ταιριάζει με τον τίτλο της ταινίας, η διεύθυνση URL του τρέιλερ διαβάζεται από το αρχείο χρησιμοποιώντας τη μέθοδο System.IO.File.ReadAllText και αποθηκεύεται στη μεταβλητή trailerUrl. Αμέσως μετά, η επανάληψη διακόπτεται με τη χρήση της εντολής break. Η μεταβλητή αυτή αποθηκεύεται σε ένα ViewBag για να χρησιμοποιηθεί η τιμή της στο Details View. Ο κώδικας αυτός επιτυγχάνει την αναζήτηση του URL του τρέιλερ μιας ταινίας μέσα από μια συλλογή αρχείων κειμένου, βασιζόμενος στον τίτλο της ταινίας και επιστρέφοντας το περιεχόμενο του αντίστοιχου αρχείου.

Στο Details View τώρα για να προβληθεί το trailer και να είναι λειτουργικό, χρησιμοποιήθηκε η ετικέτα ‘<iframe>’ και στο στοιχείο ‘src’ για τιμή δόθηκε η Viewbag μεταβλητή που περιείχε το σύνδεσμο για το trailer.

```
<div class="embed-responsive embed-responsive-16by9">
    <iframe class="embed-responsive-item" src="@ViewBag.TrailerUrl" allowfullscreen></iframe>
</div>
```

Κάτω από το trailer παρουσιάζεται μια σύντομη περιγραφή του θέματος της ταινίας το οποίο ανακτήθηκε με την βοήθεια του '@Html.DisplayFor' από το πεδίο 'Description' του πίνακα Movies.

5.6 Προβολή ηθοποιών

Μια από τις πιο σημαντικές πληροφορίες που οφείλουν να αναφέρονται για μια ταινία είναι οι ηθοποιοί που έλαβαν μέρος σε αυτή. Τα ονοματεπώνυμα των ηθοποιών είναι αποθηκευμένα στο πεδίο 'fullname' στον πίνακα Actors. Στον πίνακα MoviesActors όμως καθώς κάθε εγγραφή έχει ξεχωριστό id (maid) αν ανήκουν παραπάνω από ένας ηθοποιοί σε μια ταινία θα έχουν διαφορετικό maid όμως ίδιο 'movieid'. Επομένως απευθείας στο View μπορεί να γίνει ανάκτηση μόνο του ενός από αυτούς (του πρωταγωνιστή). Για να ανακτηθούν όλοι οι ηθοποιοί της ταινίας αναπτύχτηκε κώδικας στην Details μέθοδο στον MoviesActors controller.

Ο κώδικας ξεκινά με την αναζήτηση μιας συγκεκριμένης εγγραφής από τη βάση δεδομένων χρησιμοποιώντας το πλαίσιο Entity Framework. Η μεταβλητή moviesActor παίρνει την τιμή από μια ασύγχρονη κλήση στο _context.MoviesActors, όπου περιλαμβάνονται τα σχετικά δεδομένα του ηθοποιού στον πίνακα Actors και της ταινίας στον πίνακα Movies. Η μέθοδος FirstOrDefaultAsync χρησιμοποιείται για να βρεθεί η πρώτη εγγραφή που έχει το Maid ίσο με την τιμή της παραμέτρου id. Αν η εγγραφή moviesActor είναι null, δηλαδή αν δεν βρεθεί καμία αντίστοιχη εγγραφή στη βάση δεδομένων, επιστρέφεται το αποτέλεσμα NotFound(), δηλώνοντας ότι δεν βρέθηκε η ζητούμενη εγγραφή. Στη συνέχεια, γίνεται μια δεύτερη ασύγχρονη κλήση στη βάση δεδομένων για να βρεθούν όλοι οι άλλοι ηθοποιοί που συμμετέχουν στην ίδια ταινία εκτός του πρωταγωνιστή που όπως αναφέρθηκε μπορεί να προβληθεί απευθείας στο View. Αυτό επιτυγχάνεται με την κλήση _context.MoviesActors, όπου χρησιμοποιείται η μέθοδος Where για να φιλτραριστούν οι εγγραφές που έχουν το ίδιο Movieid με την ταινία του moviesActor, αλλά διαφορετικό Maid από το moviesActor. Η μέθοδος Select χρησιμοποιείται για να επιλεγούν μόνο τα δεδομένα του ηθοποιού (Actor), και τα αποτελέσματα μετατρέπονται σε λίστα χρησιμοποιώντας τη μέθοδο ToListAsync.

```
// Get all actors for the same movie
var otherActors = await _context.MoviesActors
    .Where(ma => ma.Movieid == moviesActor.Movieid && ma.Maid != moviesActor.Maid)
    .Select(ma => ma.Actor)
    .ToListAsync();

// Pass the list of other actors to the view
ViewBag.OtherActors = otherActors;
```

Η λίστα των άλλων ηθοποιών που βρέθηκε αποθηκεύεται στη μεταβλητή otherActors, η οποία στη συνέχεια ανατίθεται στο ViewBag.OtherActors. Αυτό επιτρέπει την πρόσβαση στη λίστα των άλλων ηθοποιών από τη θέα (view) που θα χρησιμοποιηθεί για την παρουσίαση των δεδομένων στον χρήστη. Στο Details View τώρα δημιουργήθηκε μια λίστα html στοιχείων για την προβολή των δεδομένων που αναφέρθηκαν. Αρχικά ο πρωταγωνιστής της ταινίας, ο τίτλος και ο σκηνοθέτης προβάλλονται με την χρήση του '@Html.DisplayFor'. Ο πρωταγωνιστής μέσω του 'actorid' που λειτουργεί ως ξένο κλειδί για την επικοινωνία των μοντέλων Actors και MovieActors, ο τίτλος μέσω του 'movieid' που είναι ξένο κλειδί και συνδέει τους πίνακες MoviesActors και Movies και ο σκηνοθέτης μέσω του 'movieid' και αυτός.

```
>@Html.DisplayFor(model => model.Actor.Fullname)·
```

Έπειτα δημιουργήθηκε κώδικας cshtml για την προβολή των υπόλοιπων ηθοποιών πλην του πρωταγωνιστή που συμμετείχαν στην ταινία. Ο κώδικας αρχίζει με έναν έλεγχο στην μεταβλητή ViewBag.OtherActors για να γίνει έλεγχος ότι δεν είναι κενή και εάν περιέχει τουλάχιστον έναν ηθοποιό. Εάν η συνθήκη αυτή ισχύει, δημιουργείται ένας τίτλος επιπέδου h4 με το κείμενο "Other Actors in the Movie". Στη συνέχεια, δημιουργείται ένα div με τις κλάσεις d-flex και flex-wrap για να οριστεί μια διάταξη με ευέλικτη στοίχιση και περιτύλιξη των στοιχείων. Μέσα σε αυτό το div, χρησιμοποιείται ένας βρόχος foreach για να διατρέξει κάθε ηθοποιό στην ιδιότητα ViewBag.OtherActors. Για κάθε ηθοποιό, δημιουργείται ένα div με την κλάση m-2 για προσθήκη περιθωρίου και text-center για κεντράρισμα του κειμένου και εμφανίζεται η φωτογραφία του με τη χρήση της ετικέτας '<img>' και σαν τιμή στο 'src' το πεδίο 'PhotoUrl' από τον πίνακα Actors καθώς και το όνομά του από το πεδίο 'Fullname'.

```
@if (ViewBag.OtherActors != null && ViewBag.OtherActors.Count > 0)
{
    <h4>Other Actors in the Movie</h4>
    <div class="d-flex flex-wrap">
        @foreach (var actor in ViewBag.OtherActors)
        {
            <div class="m-2 text-center">
                <img src="@actor.Photourl" alt="@actor.Fullname" class="rounded-circle" style="width: 120px; height: 150px;">
                <div><a href="/MoviesActors/ActorDetails?actorName=@actor.Fullname" class="director-link">@actor.Fullname</a>
            </div>
        }
    </div>

}
```

Όπως φαίνεται παραπάνω το όνομα του ηθοποιού δεν προβάλλεται απλά στην φόρμα αλλά δίνεται και ως παράμετρος στην μέθοδο ActorDetails στον MoviesActors controller μέσω της χρήσης ετικέτας συνδέσμου '<aherf>'. Με τη χρήση αυτής της μεθόδου ο χρήστης έχει την δυνατότητα πατώντας το όνομα του ηθοποιού να κατευθυνθεί σε καινούργια σελίδα όπου παρουσιάζεται ένα σύντομο βιογραφικό για τον ηθοποιό αυτό καθώς και οι ταινίες στις οποίες λαμβάνει μέρος.

5.7 Βιογραφικό ηθοποιού

Συγκεκριμένα η μέθοδος ActorDetails δέχεται ως παράμετρο το όνομα του ηθοποιού και μέσω αυτού γίνεται ανάκτηση του βιογραφικού του, της φωτογραφίας του και των ταινιών που λαμβάνει μέρος. Το βιογραφικό του κάθε ηθοποιού έχει αποθηκευτεί ως αρχείο κειμένου με όνομα το όνομά του ηθοποιού με την επέκταση '.txt' στο φάκελο 'ActorsBio' κάτω από το wwwroot. Αρχικά, προσδιορίζεται η ρίζα του δικτυακού τόπου (wwwroot) μέσω της ιδιότητας _webHostEnvironment.WebRootPath, και στη συνέχεια κατασκευάζεται το μονοπάτι του αρχείου του βιογραφικού του ηθοποιού συνδυάζοντας αυτή τη ρίζα με το φάκελο "ActorsBio" και το όνομα του ηθοποιού μαζί με την κατάληξη ".txt". Το περιεχόμενο του αρχείου βιογραφίας διαβάζεται και αποθηκεύεται στη μεταβλητή bio.

```
public IActionResult ActorDetails(string? actorName)
{
    string webRootPath = _webHostEnvironment.WebRootPath;
    string bioFilePath = Path.Combine(webRootPath, "ActorsBio", $"{actorName}.txt");
    string bio = System.IO.File.ReadAllText(bioFilePath);
```

Στη συνέχεια, το όνομα του ηθοποιού (actorName) αποθηκεύεται στο ViewBag για να είναι διαθέσιμο στη θέαση (view). Έπειτα, αναζητείται στη βάση δεδομένων ο ηθοποιός που αντιστοιχεί στο όνομα που δόθηκε. Εάν βρεθεί, αποθηκεύεται στη

μεταβλητή actor και λαμβάνεται το αναγνωριστικό του (actorId). Με βάση το αναγνωριστικό του ηθοποιού, αναζητούνται όλες οι ταινίες στις οποίες έχει συμμετάσχει από τον πίνακα MoviesActors, και δημιουργείται μια λίστα με τα στοιχεία τίτλου της ταινίας και το URL της αφίσας της. Τα δεδομένα για τις ταινίες στις οποίες έχει συμμετάσχει ο ηθοποιός αποθηκεύονται στο ViewBag.Movies. Το URL της φωτογραφίας του ηθοποιού καθορίζεται από την ιδιότητα Photourl του αντικειμένου actor και αποθηκεύεται στο ViewBag.PhotoUrl. Εάν δεν βρεθεί ηθοποιός, αποθηκεύεται μια κενή συμβολοσειρά. Τέλος, το περιεχόμενο της βιογραφίας αποθηκεύεται στο ViewBag.ActorBio και η μέθοδος επιστρέφει τη θέαση (view).

```
ViewBag.ActorName = actorName;


// Get the actor's photo URL based on the actor's name
var actor = _context.Actors.FirstOrDefault(a => a.Fullname == actorName);

// Get actor's ID
int actorId = actor.Actorid;

// Get movies featuring the actor
var moviesFeaturingActor = _context.MoviesActors
    .Where(ma => ma.Actorid == actorId)
    .Select(ma => new { ma.Movie.Title, ma.Movie.Posturl })
    .ToList();


ViewBag.Movies = moviesFeaturingActor;

string photoUrl = actor != null ? actor.Photourl : string.Empty;

ViewBag.PhotoUrl = photoUrl;

ViewBag.ActorBio = bio;

return View();
```

Στο αντίστοιχο View της μεθόδου ActorDetails δημιουργήθηκε κώδικας cshtml για την προβολή των παραπάνω στοιχείων στο χρήστη. Αρχικά, μέσα σε ένα div στοιχείο με κλάση "row justify-content-center", υπάρχει ένα div στοιχείο με κλάση "col-md-4 text-center" για να κεντραριστεί το περιεχόμενο του. Εδώ, εμφανίζεται η εικόνα του ηθοποιού, η οποία προέρχεται από το URL που βρίσκεται στο ViewBag.PhotoUrl. Η εικόνα έχει επίσης κλάση "img-fluid" και στυλ που καθορίζει

το πλάτος της στα 300px ενώ το ύψος της είναι αυτόματο. Στη συνέχεια, υπάρχει ένα div στοιχείο με κλάση "bio-container", όπου γίνεται η μορφοποίηση της βιογραφίας του ηθοποιού. Ο κώδικας για τη ρύθμιση των παραγράφων της βιογραφίας είναι ο ίδιος που χρησιμοποιήθηκε για την βιογραφία του σκηνοθέτη. Η βιογραφία λαμβάνεται από το ViewBag.ActorBio και αποθηκεύεται σε μια μεταβλητή bio. Ορίζεται επίσης ότι κάθε παράγραφος θα περιέχει περίπου 100 λέξεις (wordsPerParagraph). Δημιουργούνται δύο StringBuilder αντικείμενα: το ένα (paragraphBuilder) για να συγκεντρώνει τις τελικές παραγράφους και το άλλο (currentParagraph) για να κατασκευάζει την τρέχουσα παράγραφο. Ο κώδικας διασπά τη βιογραφία σε λέξεις και προσθέτει κάθε λέξη στην τρέχουσα παράγραφο. Όταν η παράγραφος περιέχει τον καθορισμένο αριθμό λέξεων και η τελευταία λέξη περιέχει τελεία, προστίθεται η παράγραφος στο paragraphBuilder. Εάν η παράγραφος δεν περιέχει τελεία, αναζητείται η τελευταία τελεία στην τρέχουσα παράγραφο, και η παράγραφος κόβεται σε αυτό το σημείο. Το υπόλοιπο κείμενο μένει για την επόμενη παράγραφο. Τέλος, εάν υπάρχει υπόλοιπο κείμενο μετά την επεξεργασία όλων των λέξεων, αυτό προστίθεται ως τελευταία παράγραφος. Η μορφοποιημένη βιογραφία προστίθεται στο formattedBio και στη συνέχεια αποδίδεται χρησιμοποιώντας Html.Raw.

```
string bio = ViewBag.ActorBio;
int wordsPerParagraph = 100;
int wordsCount = 0;

StringBuilder paragraphBuilder = new StringBuilder();
StringBuilder currentParagraph = new StringBuilder();

foreach (string word in bio.Split(' '))
{
    currentParagraph.Append(word);
    currentParagraph.Append(" ");
    wordsCount++;

    if (wordsCount >= wordsPerParagraph && word.Contains('.'))
    {
        paragraphBuilder.Append("<p>");
        paragraphBuilder.Append(currentParagraph);
        paragraphBuilder.Append("</p>");
        currentParagraph.Clear();
        wordsCount = 0;
    }
    else if (wordsCount >= wordsPerParagraph)
    {
        // Find the index of the last period in the current paragraph
        int lastPeriodIndex = currentParagraph.ToString().LastIndexOf('.');
        if (lastPeriodIndex != -1)
        {
            string paragraphText = currentParagraph.ToString().Substring(0, lastPeriodIndex + 1);
            paragraphBuilder.Append("<p>");
            paragraphBuilder.Append(paragraphText);
            paragraphBuilder.Append("</p>");
```

```
                // Remove the processed part from the current paragraph
                currentParagraph.Remove(0, lastPeriodIndex + 1);
                wordsCount = 0;
            }
        }
    }

    // Add the final paragraph
    if (currentParagraph.Length > 0)
    {
        paragraphBuilder.Append("<p>");
        paragraphBuilder.Append(currentParagraph);
        paragraphBuilder.Append("</p>");
    }

    string formattedBio = paragraphBuilder.ToString();
    @Html.Raw(formattedBio)
```

Ακολουθεί ένας τίτλος επιπέδου h2 που αναφέρει τις ταινίες στις οποίες έχει συμμετάσχει ο ηθοποιός, με το όνομά του να λαμβάνεται από το ViewBag.ActorName. Τέλος, ένα div στοιχείο με κλάση "movies-list" περιέχει μια λίστα από div στοιχεία, το καθένα από τα οποία αναπαριστά μια ταινία. Για κάθε ταινία, υπάρχει ένας υπερσύνδεσμος (a tag) που περιέχει την εικόνα της ταινίας (img tag). Ο υπερσύνδεσμος οδηγεί στη σελίδα αναζήτησης ταινιών με βάση τον τίτλο της ταινίας χρησιμοποιώντας το Url.Action με τις αντίστοιχες παραμέτρους. Δηλαδή επιλέγοντας την εικόνα της ταινίας ο χρήστη μεταφέρεται στη σελίδα της. Με τη χρήση του βρόχου ‘foreach’ η διαδικασία αυτή επαναλαμβάνεται για όσες ταινίες υπάρχουν στην ‘ViewBag.Movies’.

```
<h2>Movies featuring @ViewBag.ActorName:</h2>
<div class="movies-list">
    @foreach (var movie in ViewBag.Movies)
    {
        <div class="movie">
            <a href="@Url.Action("Index", "MoviesActors", new { searchString = movie.Title })">
                <img src="@movie.Posturl" alt="@movie.Title" style="border-radius: 0;" />
            </a>
        </div>
    }
</div>
```

5.8 Προτεινόμενες ταινίες

Προκειμένου κάθε χρήστης να μπορεί να πλοηγηθεί ευκολότερα στην εφαρμογή και να βρει τις ταινίες που επιθυμεί αναπτύχτηκε λειτουργία προτεινόμενων ταινιών. Έτσι μόλις ο χρήστης μεταβεί στην κεντρική σελίδα μιας ταινίας (όπου προβάλλονται οι ηθοποιοί, το trailer κτλ) στο κάτω μέρος θα εμφανίζεται μια λίστα από ταινίες που ανήκουν στο ίδιο είδος(genre). Πατώντας τη εικόνα μιας από τις ταινίες της λίστας, γίνεται ανακατεύθυνση στην αντίστοιχη κεντρική σελίδα. Αρχικά όπως αναφέρθηκε προηγουμένως, μόλις ο χρήστης μεταβεί στην κεντρική σελίδα μιας ταινίας το id της (movieid) στέλνεται σαν παράμετρος στην μέθοδο Details του MovieActors controller έτσι ώστε τα στοιχεία της να προβληθούν στο αντίστοιχο View (Details View). Στη μεταβλητή 'moviesActor' μέσω του id αποθηκεύονται όλα τα στοιχεία της ταινίας (τιτλος, σκηνοθέτης κτλ.).

```
var moviesActor = await _context.MoviesActors
    .Include(m => m.Actor)
    .Include(m => m.Movie)
    .FirstOrDefaultAsync(m => m.Maid == id);
if (moviesActor == null)
{
    return NotFound();
}
```

Έπειτα τρέχει ένα ερώτημα στη βάση όπου σαρώνει όλες τις ταινίες από τον πίνακα Movies και αποθηκεύει στη μεταβλητή 'recommendedMovies' ως λίστα αυτές που ανήκουν στο ίδιο είδος με την ταινία που το id της δόθηκε για παράμετρος, δηλαδή της ταινίας που ο χρήστης βρίσκεται στην κεντρική της σελίδα. Έτσι προκειμένου να μην σαρωθεί και η ίδια δεύτερη φορά, γίνεται έλεγχος έτσι ώστε αν βρεθεί κάποια με το ίδιο είδος να έχει διαφορετικό id. Τέλος η 'recommendedMovies' ορίζεται σε ένα ViewBag για να χρησιμοποιηθεί στο View.

```
// Get recommended movies with the same genre
var recommendedMovies = _context.Movies.Where(m => m.Genre == moviesActor.Movie.Genre && m.Movieid != moviesActor.Movie.Movieid).ToList
ViewBag.RecommendedMovies = recommendedMovies;
```

Περνώντας τώρα στο αντίστοιχο View αναπτύχτηκε κώδικας για την προβολή της λίστας των προτεινόμενων ταινιών. Στην αρχή, υπάρχει ένας τίτλος επιπέδου h4 με το κείμενο "Recommendation" και μια οριζόντια γραμμή (hr) για διαχωρισμό.

Έπειτα, χρησιμοποιώντας την Razor syntax, γίνεται έλεγχος αν το ViewBag.RecommendedMovies δεν είναι null και περιέχει ταινίες (δηλαδή αν ο αριθμός των ταινιών είναι μεγαλύτερος από μηδέν). Αν υπάρχουν προτεινόμενες ταινίες, εμφανίζεται ένα μήνυμα που λέει "Here's other movies in the same genre:" και δημιουργείται ένα div στοιχείο με κλάση "recommended-movies". Μέσα σε αυτό το div, γίνεται μια επανάληψη (foreach) σε κάθε ταινία στη λίστα ViewBag.RecommendedMovies. Για κάθε μια, δημιουργείται ένα div στοιχείο με κλάση "movie-item". Μέσα σε αυτό το div, υπάρχει ένας υπερσύνδεσμος (a tag) που οδηγεί στη σελίδα αναζήτησης ταινιών με βάση τον τίτλο της ταινίας χρησιμοποιώντας το Url.Action με τις αντίστοιχες παραμέτρους. Μέσα στον υπερσύνδεσμο, υπάρχει μια εικόνα (img tag) της ταινίας, με το URL της εικόνας να προέρχεται από την ιδιότητα Posturl της ταινίας. Η εικόνα έχει την κλάση "img-fluid" και στυλ που καθορίζει το μέγιστο πλάτος της στα 100px και το ύψος της να είναι αυτόματο. Αν δεν υπάρχουν προτεινόμενες ταινίες, εμφανίζεται το μήνυμα "No recommendation available."

```
<div class="row mt-4">
    <div class="col-md-12">
        <h4>Recommendation</h4>
        <hr />
        @if (ViewBag.RecommendedMovies != null && ViewBag.RecommendedMovies.Count > 0)
        {
            <p>Here's other movies in the same genre:</p>
            <div class="recommended-movies">
                @foreach (var movie in ViewBag.RecommendedMovies)
                {
                    <div class="movie-item">
                        <a href="@Url.Action("Index", "MoviesActors", new { searchString = movie.Title })"><img src="@movie.Posturl"
                    </div>
                }
            </div>
        }
        else
        {
            <p>No recommendation available.</p>
        }
```

5.9 Σχόλια

Κάτω από το trailer της ταινίας υπάρχει το κουμπί ‘comment’ όπου πατώντας το ο χρήστης μπορεί να αφήσει σχόλιο στην ταινία. Η δυνατότητα για έναν χρήστη να αφήνει σχόλια σε μια ταινία σε μια εφαρμογή περιήγησης ταινιών είναι σημαντική για διάφορους λόγους:

1. **Αλληλεπίδραση και Κοινότητα**: Τα σχόλια επιτρέπουν στους χρήστες να αλληλεπιδρούν μεταξύ τους, δημιουργώντας μια κοινότητα γύρω από την αγάπη για τον κινηματογράφο. Οι χρήστες μπορούν να μοιράζονται απόψεις,

να συζητούν πλοκές και να ανταλλάσσουν προτάσεις, καθιστώντας την εμπειρία πιο κοινωνική και διαδραστική.

2. **Αυθεντική Ανατροφοδότηση**: Τα σχόλια παρέχουν αυθεντική και ποικιλόμορφη ανατροφοδότηση που μπορεί να είναι πιο αξιόπιστη από τις επίσημες κριτικές. Οι χρήστες μπορούν να διαβάσουν τις εμπειρίες και τις απόψεις άλλων θεατών που έχουν παρόμοια ενδιαφέροντα, βοηθώντας τους να αποφασίσουν αν μια ταινία αξίζει τον χρόνο τους.
3. **Βελτίωση Υπηρεσιών**: Οι προγραμματιστές και οι διαχειριστές της εφαρμογής μπορούν να χρησιμοποιούν τα σχόλια για να κατανοήσουν καλύτερα τις προτιμήσεις και τις ανάγκες των χρηστών τους. Αυτή η πληροφορία είναι πολύτιμη για τη βελτίωση των λειτουργιών της εφαρμογής και την προσαρμογή της στις απαιτήσεις του κοινού.
4. **Δημιουργία Ιστορικού**: Τα σχόλια δημιουργούν ένα αρχείο απόψεων που μπορεί να εξελιχθεί με τον χρόνο. Αυτό το ιστορικό μπορεί να είναι χρήσιμο για μελλοντικούς χρήστες που θέλουν να δουν πώς μια ταινία έχει αξιολογηθεί και συζητηθεί σε βάθος χρόνου.
5. **Προώθηση και Διαφήμιση**: Οι θετικές κριτικές και τα σχόλια μπορούν να λειτουργήσουν ως ανεπίσημη διαφήμιση για τις ταινίες, ενθαρρύνοντας περισσότερους ανθρώπους να τις παρακολουθήσουν. Αντίστοιχα, οι αρνητικές κριτικές μπορούν να δώσουν στους χρήστες μια πιο ολοκληρωμένη εικόνα πριν κάνουν την επιλογή τους.
6. **Εκπαιδευτικός Χαρακτήρας**: Μέσα από τα σχόλια, οι χρήστες μπορούν να μάθουν περισσότερα για τα διάφορα στοιχεία της κινηματογραφικής παραγωγής, όπως σκηνοθεσία, σενάριο, ερμηνείες κ.λπ. Οι πιο έμπειροι χρήστες μπορούν να προσφέρουν αναλύσεις και λεπτομέρειες που ίσως δεν είναι εμφανείς σε όλους.

Συνολικά, η δυνατότητα σχολιασμού ενισχύει την εμπειρία χρήσης, προάγοντας την αλληλεπίδραση και την ανταλλαγή γνώσεων και απόψεων μέσα σε μια κοινότητα κινηματογραφόφιλων. Όσον αφορά την υλοποίηση αυτής της λειτουργικότητας, αρχικά ο χρήστης μεταφέρεται στο Index View του μοντέλου Comments μέσω ‘asp action’ εντολής στο κάτω μέρος της Details προβολής του MoviesActors.

```
<div>
    <a asp-action="Index" asp-controller="Comments" asp-route-movieId="@Model.Movie.Movieid" class="btn">Comments</a>
```

Η γραμμή αυτή κώδικα δημιουργεί έναν υπερσύνδεσμο που, όταν κλικάρεται, κατευθύνει τον χρήστη στην ενέργεια "Index" του controller "Comments" με την παράμετρο movieId που αντιστοιχεί στην ταυτότητα της ταινίας από το μοντέλο δεδομένων. Ο σύνδεσμος εμφανίζεται σαν κουμπί λόγω της κλάσης "btn". Μεταφέροντας το id της ταινίας εξασφαλίζεται ότι το σχόλιο θα αναφέρεται στην συγκεκριμένη και θα αποθηκευτεί στη βάση μαζί με το id της.

Στο Index View τώρα έχει δημιουργηθεί ένα τμήμα div με την κλάση table-responsive που περιέχει έναν πίνακα. Ο πίνακας έχει τις κλάσεις table table-dark table-striped, οι οποίες του προσδίδουν συγκεκριμένη μορφοποίηση και στυλ. Ο πίνακας περιέχει μια επικεφαλίδα (<thead>) που ορίζει τρεις στήλες: "User", "Comment" και μια κενή στήλη για τις ενέργειες. Το σώμα του πίνακα (<tbody>) δημιουργείται δυναμικά με βάση τα δεδομένα από το μοντέλο. Για κάθε αντικείμενο στο μοντέλο, δημιουργείται μια γραμμή (<tr>). Κάθε γραμμή περιέχει τρία κελιά (<td>): το πρώτο κελί εμφανίζει το όνομα του χρήστη που αφήσε σχόλιο (item.User.Username), το δεύτερο εμφανίζει το σχόλιο του (item.Comment1). Έτσι προβάλλονται όλα τα σχόλια που άφησαν άλλοι χρήστες στην ταινία.

```
<div class="table-responsive">
    <table class="table table-dark table-striped">
        <thead>
            <tr>
                <th scope="col">User</th>
                <th scope="col">Comment</th>
                <th scope="col"></th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>@Html.DisplayFor(modelItem => item.User.Username)</td>
                    <td>@Html.DisplayFor(modelItem => item.Comment1)</td>
                    <td>
```

Προκειμένου να αφήσει σχόλιο ο χρήστης, δημιουργείται μια παράγραφος (<p>) που περιέχει ένα σύνδεσμο (<a>). Αυτός ο σύνδεσμος οδηγεί στην ενέργεια "Create" του controller "Comments" και περιλαμβάνει την παράμετρο movieId, η οποία παίρνει την τιμή της πρώτης ταινίας από το μοντέλο δεδομένων (@Model.FirstOrDefault()?.Movie.Movieid). Ο σύνδεσμος εμφανίζεται σαν κουμπί με την κλάση CSS btn btn-primary και έχει το κείμενο "Add Comment".

```
<h1>Comments</h1>

<p>
    <a asp-action="Create" asp-controller="Comments" asp-route-movieId="@Model.FirstOrDefault()?.Movie.Movieid" class="btn btn-primary">
</p>
```

Στο Create View τώρα δημιουργήθηκε μια φόρμα όπου περιέχει ένα textbox για να αφήσει ο χρήστης το σχόλιό του στην ταινία.

Η φόρμα περιλαμβάνει πολλά div στοιχεία για τη διάταξη των πεδίων της:

1. Ένα div με την ιδιότητα asp-validation-summary για την εμφάνιση των μηνυμάτων σφάλματος μόνο από το μοντέλο, με κόκκινο χρώμα κειμένου (text-danger).
2. Τρία κρυφά πεδία στο χρήστη (input elements με type="hidden") για την ταυτότητα του σχολίου (Commentid), την ταυτότητα του χρήστη (Userid), και την ταυτότητα της ταινίας (Movieid) που συμπληρώνονται αυτόματα όταν ο χρήστης υποβάλει το σχόλιό του. Το id του ανακτήθηκε από το session που ήταν αποθηκευμένο κατά την είσοδό του στην εφαρμογή. Το movieid είχε σταλθεί μέσω του url με την asp action που αναφέρθηκε προηγουμένως και η ανάκτησή του από εκεί γίνεται με javascript.

```
<script>
    // Get the movieId from the URL
    const urlParams = new URLSearchParams(window.location.search);
    const movieId = urlParams.get('movieId');

    // Set the value of the Movieid input field to the movieId
    document.getElementById('Movieid').value = movieId;
</script>
```

Το id του comment γίνεται αυτόματα generate μέσω συνάρτησης. Συγκεκριμένα με τη μέθοδο ‘random’ ορίζεται ένας τυχαίος ακέραιος από το 400 μέχρι τον μεγαλύτερο 32-bit ακέραιο. Τα όρια αυτά καθορίζονται από τις μεταβλητές ‘minCommentId’ και ‘maxCommentId’. Έπειτα με έναν do-while βρόχο γίνεται έλεγχος ότι δεν έχει καταχωρηθεί σχόλιο με ίδιο αναγνωριστικό. Αν ναι τότε η διαδικασία επαναλαμβάνεται μέχρι να είναι

ξεχωριστό. Στη συνέχεια αποθηκεύεται σε ViewBag και ανακτάται στο View για να συμπληρωθεί το αντίστοιχο πεδίο.

```
Random rnd = new Random();
int minCommentId = 400;
int maxCommentId = 2147483647; // Maximum signed 32-bit integer value
int commentId;

do
{
    commentId = rnd.Next(minCommentId, maxCommentId);
} while (_context.Comments.Any(c => c.Commentid == commentId));

ViewBag.CommentId = commentId;
```

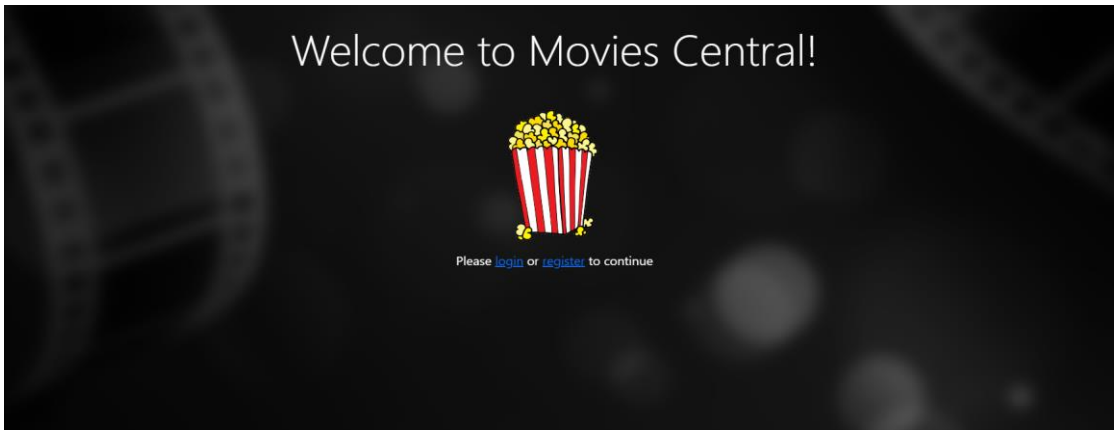
3. Ένα πεδίο κειμένου (textarea) για την εισαγωγή του σχολίου (Comment1) με κλάση form-control για τη μορφοποίηση και ένα στοιχείο span για την εμφάνιση μηνυμάτων σφάλματος σε περίπτωση σφαλμάτων κατά την εισαγωγή του σχολίου.
4. Ένα κουμπί υποβολής (input με type="submit") που εμφανίζεται κεντραρισμένο και έχει την κλάση btn btn-primary, με την ετικέτα "Submit" για την υποβολή του σχολίου

```
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-6">
            <h4 class="text-center">Leave Comment</h4>
            <hr />
            <form asp-action="Create">
                <div asp-validation-summary="ModelOnly" class="text-danger"></div>
                <div class="form-group" style="display: none;">
                    <label asp-for="Commentid" class="control-label"></label>
                    <input id="commentIdInput" asp-for="Commentid" class="form-control" readonly="readonly" value="@ViewBag.CommentId" t
                    <span asp-validation-for="Commentid" class="text-danger"></span>
                </div>
                <div class="form-group" style="display: none;">
                    <label asp-for="Userid" class="control-label"></label>
                    <input id="Userid" asp-for="Userid" class="form-control" readonly value="@((int)Context.Session.GetInt32("UserId"))'
                </div>
                <div class="form-group" style="display: none;">
                    <label asp-for="Movieid" class="control-label"></label>
                    <input id="Movieid" asp-for="Movieid" class="form-control" readonly type="hidden" />
                </div>
                <div class="form-group">
                    <textarea asp-for="Comment1" class="form-control" rows="4"></textarea>
                    <span asp-validation-for="Comment1" class="text-danger"></span>
                </div>
                <div class="form-group text-center">
                    <input type="submit" value="Submit" class="btn btn-primary" />
                </div>
            </form>
        </div>
    </div>
</div>
```

6. Παρουσίαση Web εφαρμογής

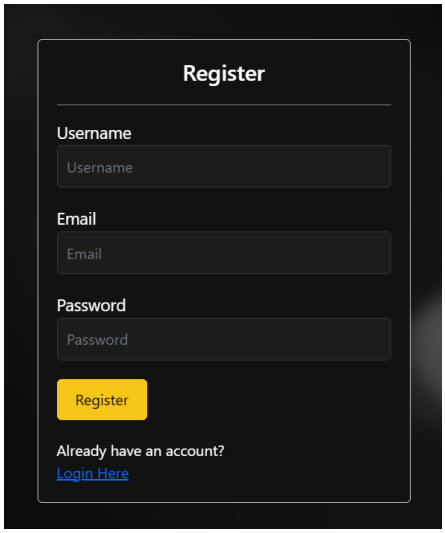
6.1 Χρήστης

Με την έναρξη της εφαρμογής, ο χρήστης καλωσορίζεται και του εμφανίζονται δύο σύνδεσμοι. Ο ένας αφορά τη σύνδεσή του στην εφαρμογή εφόσον διαθέτει λογαριασμό, ενώ επιλέγοντας τον άλλο σύνδεσμο οδηγείται στη φόρμα εγγραφής.



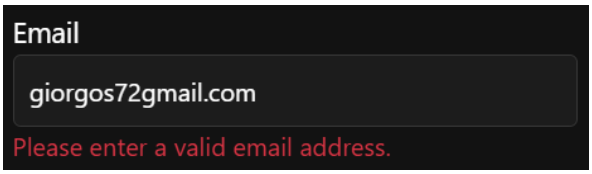
Εικόνα 6.1

Αν επιλέξει εγγραφή τότε του εμφανίζεται η αντίστοιχη φόρμα για να καταχωρήσει όνομα, email και κωδικό. Μόλις υποβάλει την εγγραφή του και γίνει έλεγχος ότι τα στοιχεία που εισήγαγε δεν είναι καταλυμένα από άλλο χρήστη οδηγείται αυτόματα στην φόρμα σύνδεσης.



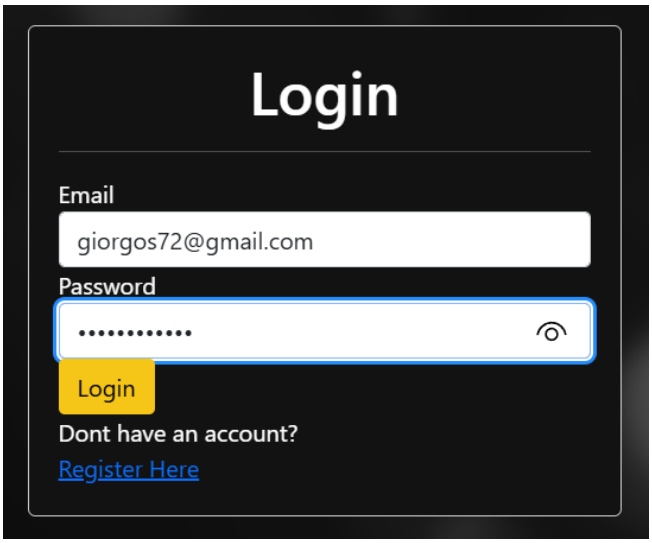
Εικόνα 6.2

Επίσης, σε περίπτωση που εισάγει λάθος email εμφανίζεται κατάλληλο μήνυμα για να διορθώσει το λάθος του.



Εικόνα 6.3

Μόλις εισάγει σωστά στοιχεία οδηγείται στη φόρμα σύνδεσης και εισέρχεται στην εφαρμογή.

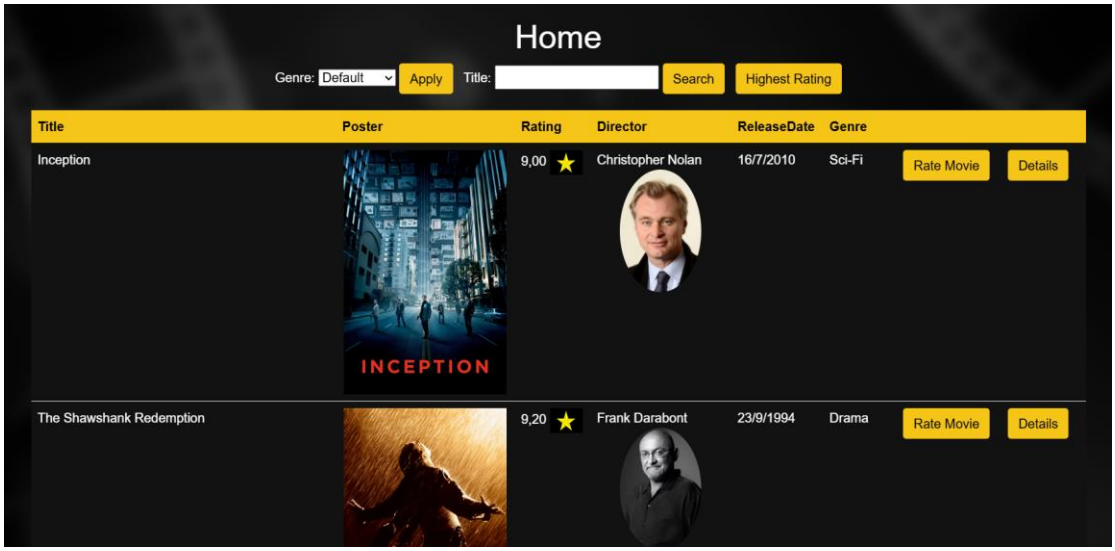


Εικόνα 6.4

Η εφαρμογή τον καλωσορίζει και παρέχει σύνδεσμο για την κεντρική σελίδα της.

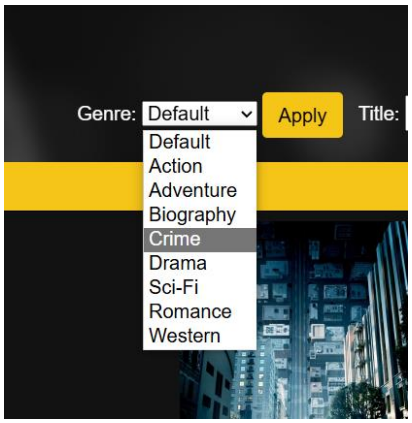
Εικόνα 6.5

Μόλις ο χρήστης εισέλθει στην κεντρική σελίδα, εμφανίζονται οι ταινίες της βάσης και κάποιες βασικές πληροφορίες για αυτές όπως ο τίτλος, ο σκηνοθέτης, η μέση βαθμολογία, η ημερομηνία κυκλοφορίας καθώς και η επιλογή βαθμολόγησης και κουμπί που οδηγεί στη κεντρική σελίδα της συγκεκριμένης ταινίας.

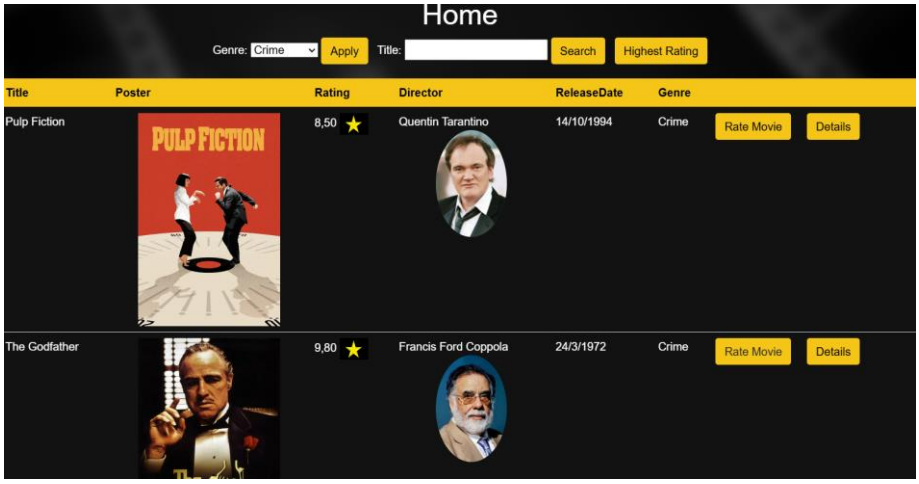


Εικόνα 6.6

Στην κορυφή της σελίδας υπάρχουν **φίλτρα** για να διευκολύνουν την αναζήτηση ταινιών του χρήστη. Το πρώτο φίλτρο αφορά την αναζήτηση ταινίας *με βάση το είδος της*. Πατώντας πάνω του εμφανίζεται ένα μενού όπου ο χρήστης επιλέγει το είδος που θέλει και πατώντας Apply εμφανίζονται μόνο ταινίες του είδους αυτού.



Εικόνα 6.7

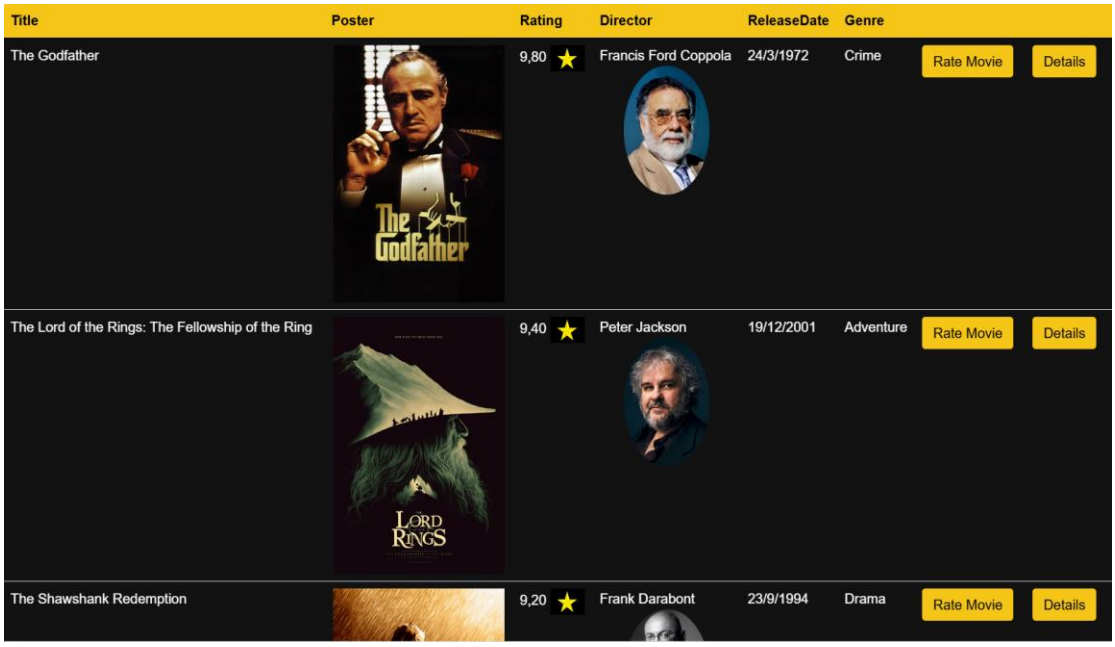


Εικόνα 6.8

Το δεύτερο φίλτρο αφορά την αναζήτηση ταινίας *με βάση τον τί*τλο. Ο χρήστης πληκτρολογεί τον τίτλο της ταινίας που επιθυμεί και πατώντας αναζήτηση του εμφανίζεται.

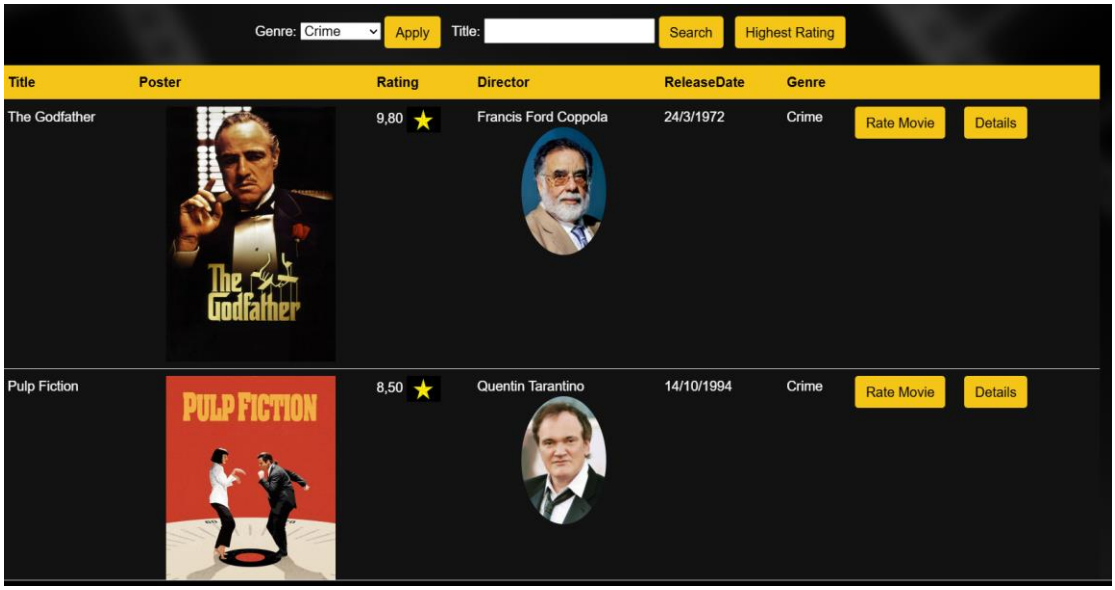
Εικόνα 6.9

Με τη χρήση του τρίτου φίλτρου ο χρήστης έχει τη δυνατότητα να αναζητήσει ταινίες *με βάση τη μέση βαθμολογία τους*. Συγκεκριμένα επιλέγοντας το κουπί υψηλότερης βαθμολογίας (**Highest Rating**) εμφανίζονται στο χρήστη ταινίες ξεκινώντας από αυτές με τη μεγαλύτερη βαθμολογία μέχρι αυτές με την μικρότερη.



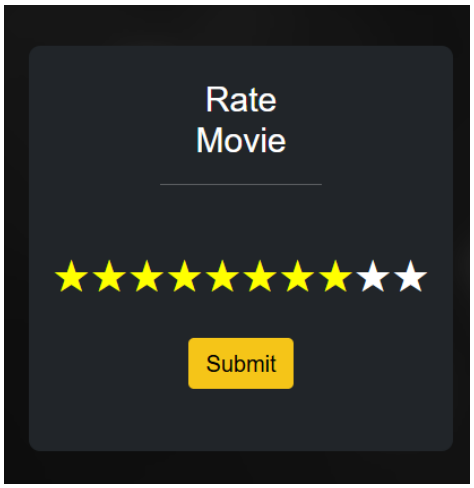
Εικόνα 6.10

Επιπλέον *υπάρχει η δυνατότητα συνδυασμού των παραπάνω φίλτρων*. Συγκεκριμένα, ο χρήστης μπορεί να επιλέξει είδος ταινίας από το μενού (για παράδειγμα crime) και αφού του προβληθούν οι ταινίες αυτού του είδους, να επιλέξει το κουμπί υψηλότερης βαθμολογίας για να ταξινομηθούν.



Εικόνα 6.11

Επιπροσθέτως μια από τις πιο σημαντικές λειτουργίες της εφαρμογής είναι η βαθμολόγηση ταινιών. Δίπλα από τις πληροφορίες κάθε ταινίας υπάρχει το κουμπί βαθμολόγησης **(Rate Movie**) όπου πατώντας το ο χρήστης μπορεί να βαθμολογήσει την ταινία. Η βαθμολόγηση είναι σε μορφή αστεριών (1-10 αστέρια) και ο χρήστης επιλέγει τα αντίστοιχα αστέρια με τον βαθμό που θέλει.

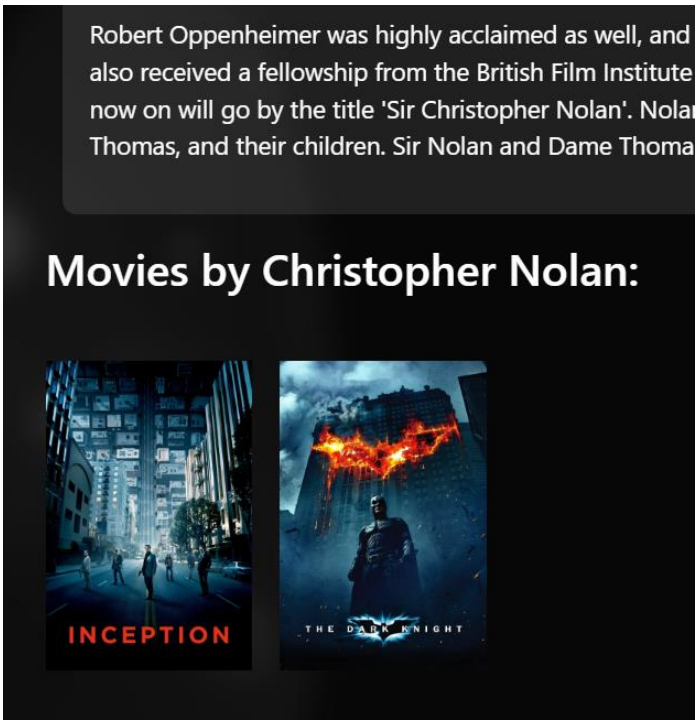


Εικόνα 6.12

Τέλος όσον αφορά την κεντρική σελίδα **Home** (Εικόνα 6.6), πατώντας πάνω στη φωτογραφία του σκηνοθέτη της κάθε ταινίας, παρουσιάζεται *ένα σύντομο βιογραφικό* για αυτόν.

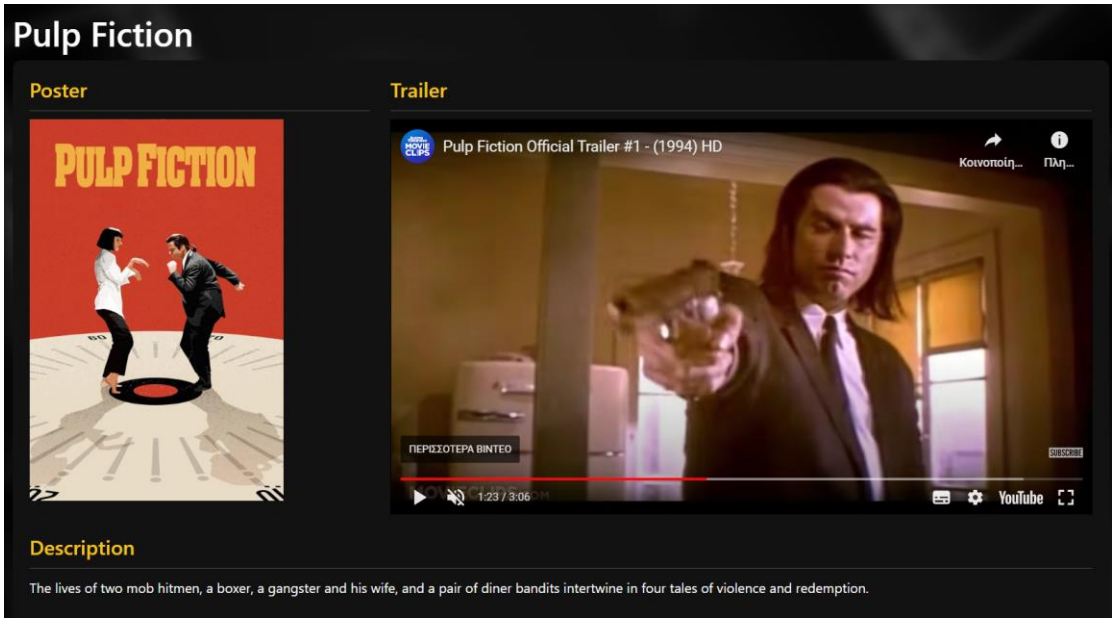
Εικόνα 6.13

Κάτω από το βιογραφικό του επίσης, εμφανίζονται οι ταινίες που έχει σκηνοθετήσει και πατώντας κάποια από αυτές ο χρήστης κατευθύνεται στην κεντρική σελίδα της.



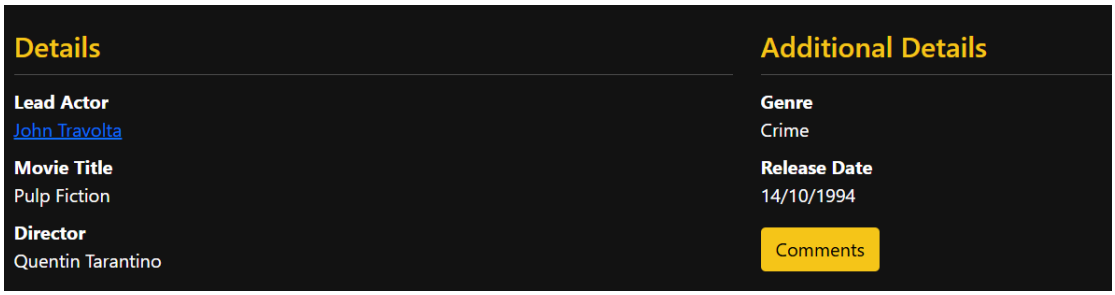
Εικόνα 6.14

Για κάθε ταινία υπάρχει κουμπί με όνομα πληροφορίες (**Details**) όπου οδηγεί στην κεντρική σελίδα της αντίστοιχης ταινίας. Εκεί παρουσιάζονται παραπάνω πληροφορίες για αυτήν όπως μια βασική περιγραφή, το trailer, οι ηθοποιοί που έλαβαν μέρος καθώς και η δυνατότητα ο χρήστης να αφήσει κάποιο σχόλιο για την ταινία.



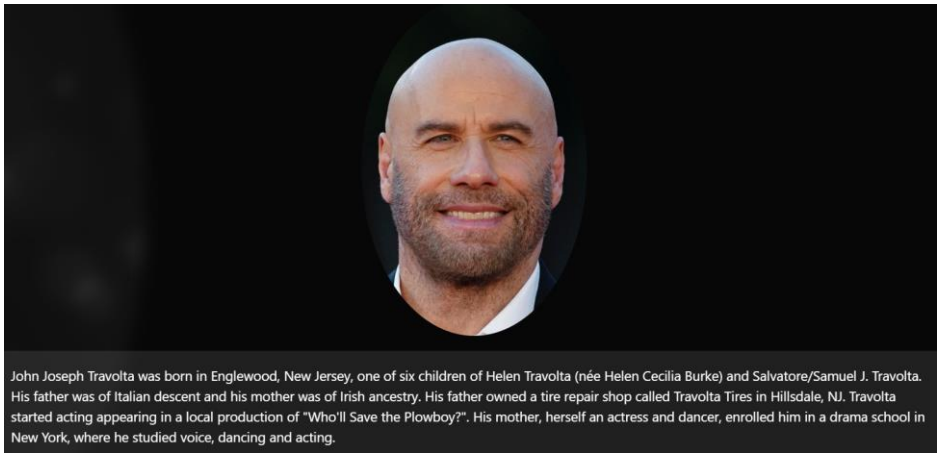
Εικόνα 6.15

Κάτω από την περιγραφή εμφανίζονται κάποιες σημαντικές πληροφορίες και την ταινία καθώς και ο πρωταγωνιστής της.



Εικόνα 6.16

Πατώντας το όνομα του παρουσιάζεται ένα σύντομο βιογραφικό για αυτόν και όπως και στον σκηνοθέτη στο κάτω μέρος του βιογραφικού εμφανίζονται οι ταινίες που έλαβε μέρος.



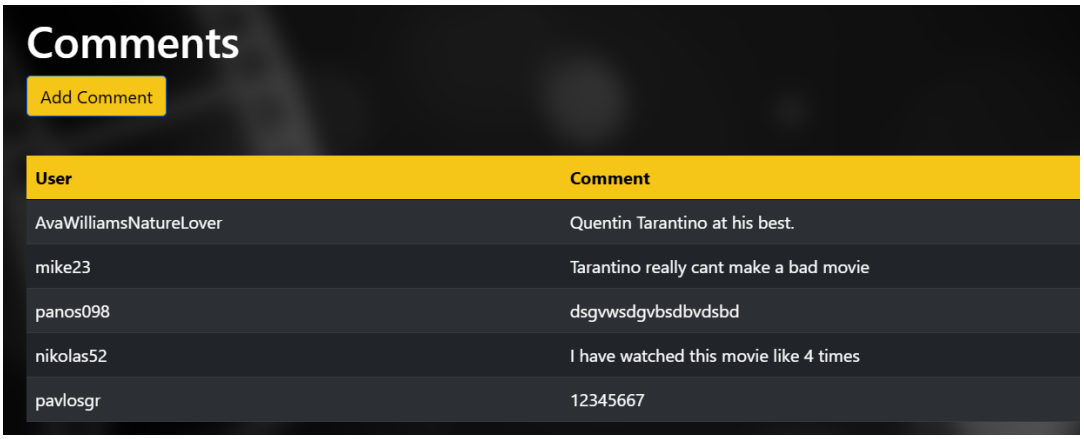
Εικόνα 6.17

Εικόνα 6.18

Επιστρέφοντας τώρα στην κεντρική σελίδα της ταινίας, κάτω από τις βασικές πληροφορίες εμφανίζονται και οι υπόλοιποι ηθοποιοί της ταινίας εκτός του πρωταγωνιστή και πατώντας το όνομά κάποιου από αυτούς, ο χρήστης οδηγείται σε αντίστοιχη σελίδα με τον πρωταγωνιστεί όπου παρουσιάζεται ένα σύντομο βιογραφικό και οι ταινίες που έχει λάβει μέρος.

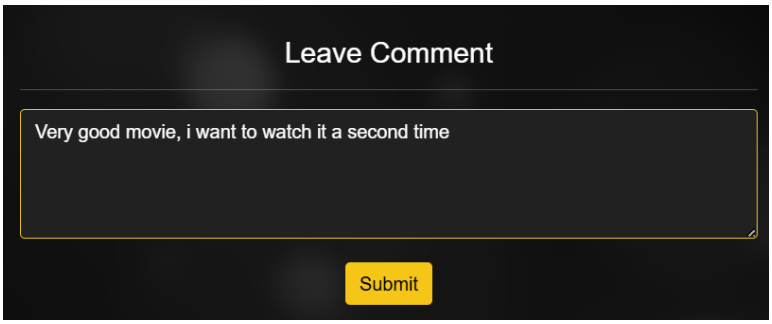
Εικόνα 6.19

Δεξιά από τις πληροφορίες της ταινίας υπάρχει κουμπί με όνομα σχόλια (**Comments**) όπου πατώντας το ο χρήστης μπορεί να δει σχόλια άλλων χρηστών για την ταινία καθώς και να σχολιάσει ο ίδιος.

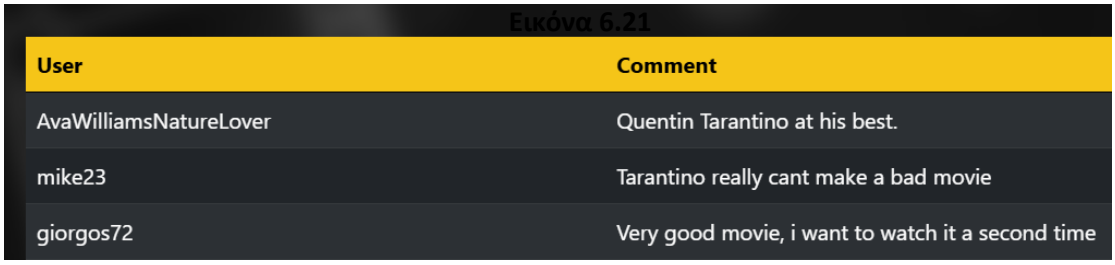


Εικόνα 6.20

Πατώντας το κουμπί προσθήκη σχόλιου ο χρήστης αφήνει σχόλιο για την ταινία.



Εικόνα 6.21



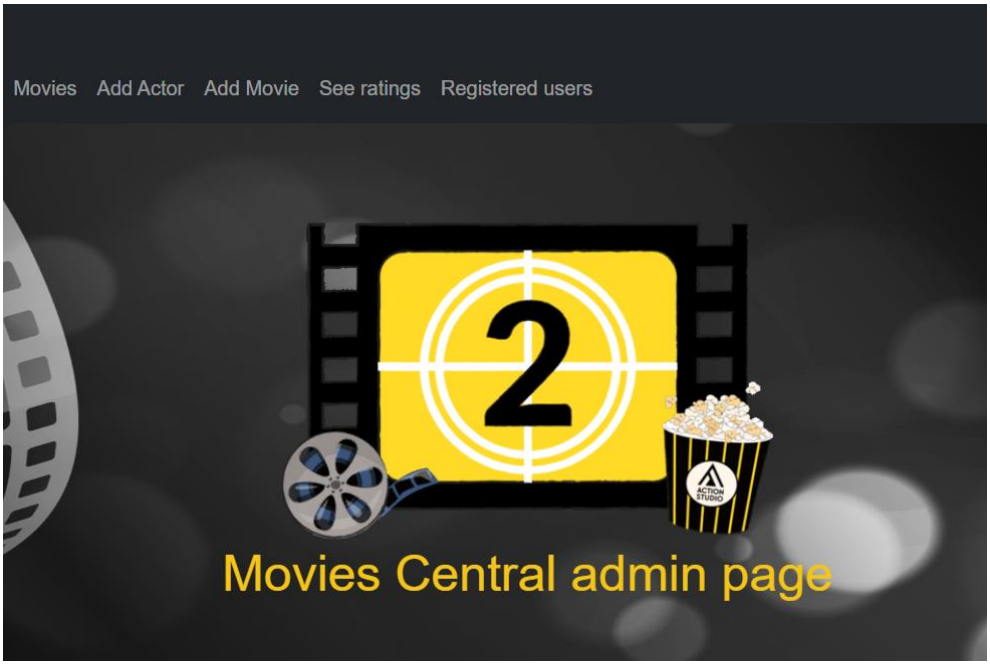
Εικόνα 6.22

Τέλος στο κάτω μέρος της σελίδας *προτείνονται στο χρήστη ταινίες που ανήκουν στο ίδιο είδος* και πατώντας πάνω τους ο χρήστης οδηγείται στην αντίστοιχη κεντρική σελίδα.

Εικόνα 6.23

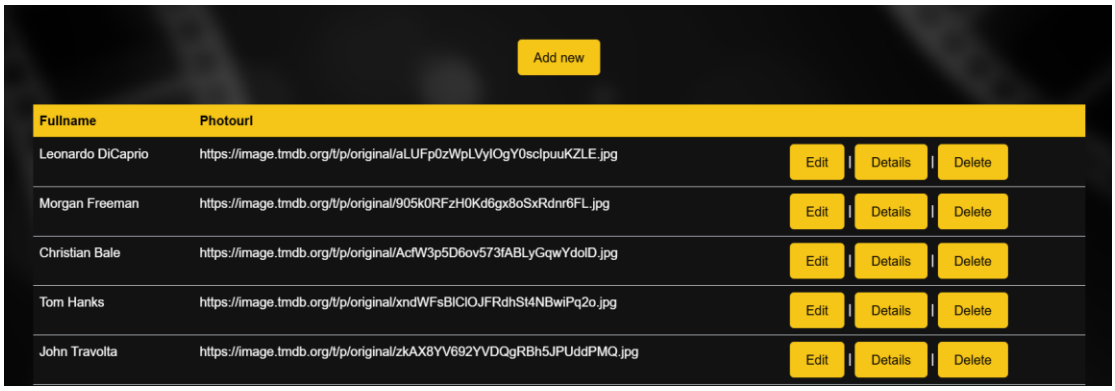
6.2 Διαχειριστής

Ο διαχειριστής έχει τη δυνατότητα να πραγματοποιήσει όλες τις παραπάνω λειτουργίες χρήστη σε συνδυασμό με κάποιες επιπλέον. Αρχικά αφού συνδεθεί με τον ίδιο τρόπο με τον χρήστη οδηγείται σε διαφορετική κεντρική σελίδα. Εκεί μπορεί να πραγματοποιήσει αρκετές επιπλέον λειτουργίες όπως να προσθέσει ταινία στην εφαρμογή, να προσθέσει ηθοποιούς, να δει τους εγγεγραμμένους χρήστες και τα στοιχεία τους καθώς και τις βαθμολογίες κάθε χρήστη.



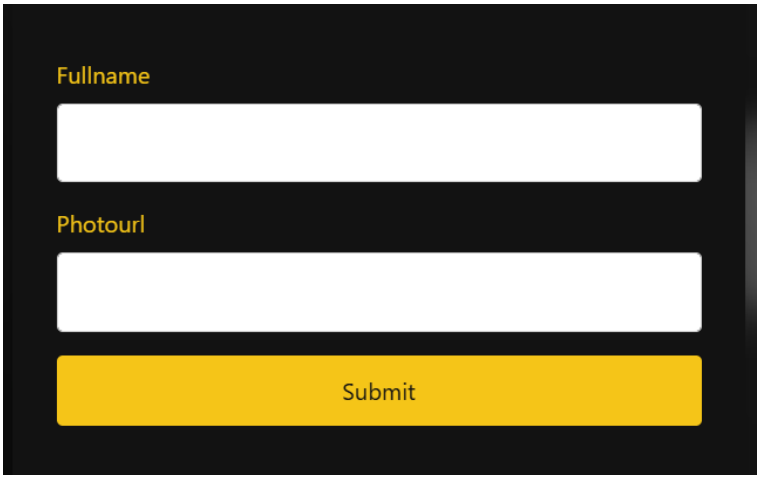
Εικόνα 6.24

Αρχικά επιλέγοντας προσθήκη ηθοποιού (**Add actor**) ο διαχειριστής οδηγείται στην κεντρική σελίδα με τα στοιχεία των ηθοποιών της εφαρμογής. Εκεί έχει την δυνατότητα είτε να επεξεργαστεί τα στοιχεία ενός υπάρχον ηθοποιού είτε να προσθέσει νέο στη βάση.



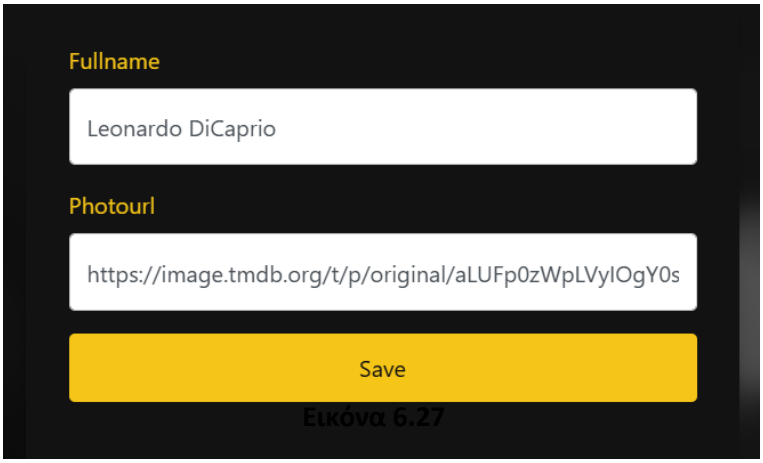
Εικόνα 6.25

Πατώντας προσθήκη νέου (**Add new**) ο διαχειριστής εισάγει τα στοιχεία του ηθοποιού που θέλει να προσθέσει στη βάση.



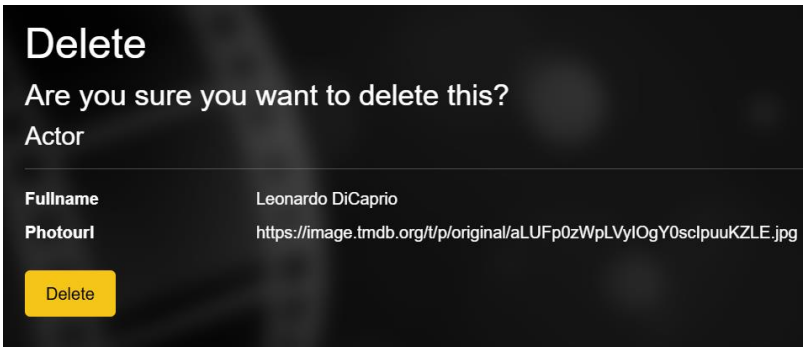
Εικόνα 6.26

Αν θέλει να επεξεργαστεί τα στοιχεία ενός ηθοποιού πατάει το κουμπί επεξεργασία (**Edit**) του αντίστοιχου ηθοποιού και αλλάζει τα στοιχεία.



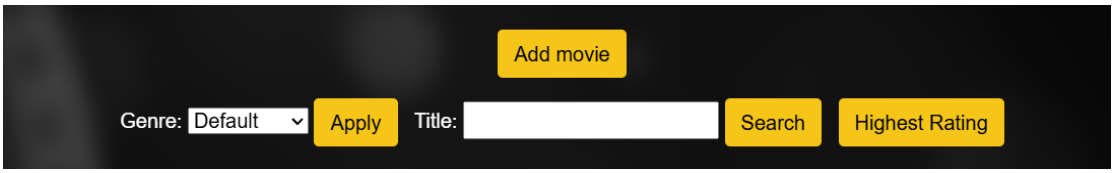
Εικόνα 6.27

Αντίστοιχα αν θέλει να διαγράψει κάποιον επιλέγει το κουμπί διαγραφής (**Delete**).



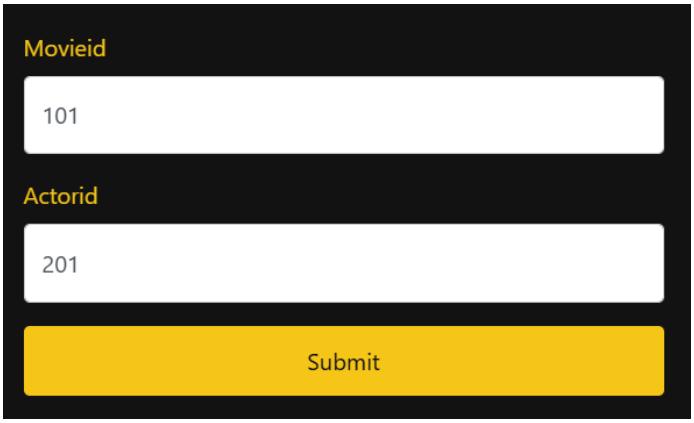
Εικόνα 6.28

Σε περίπτωση που θέλει να αντιστοιχήσει ένα νέο ηθοποιό που πρόσθεσε στην ταινία που λαμβάνει μέρος, πηγαίνει πίσω στην αρχική σελίδα (Εικόνα 6.24) και επιλέγει το κουμπί ταινίες (**Movies**) και κατευθύνεται στην κεντρική σελίδα της εφαρμογής. Εκεί πάνω από τα φίλτρα αναζήτησης επιλέγει προσθήκη ταινίας (**Add movie**).



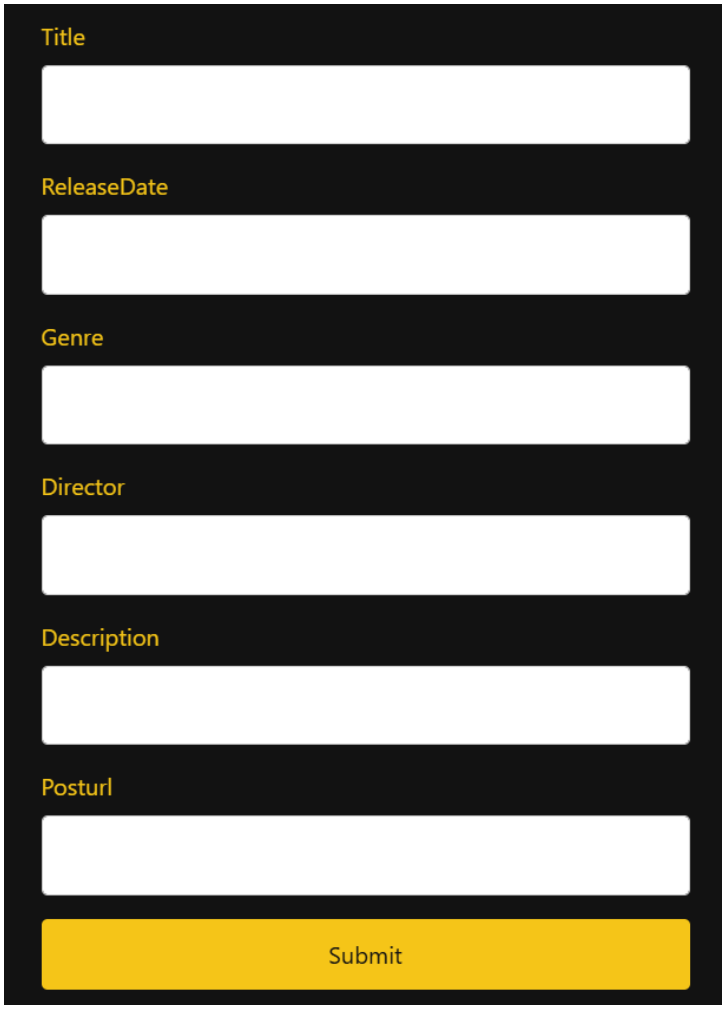
Εικόνα 6.29

Στη συνέχεια εισάγει το αναγνωριστικό του ηθοποιού που πρόσθεσε στη βάση και το αναγνωριστικό της ταινίας που θέλει να τον προσθέσει.



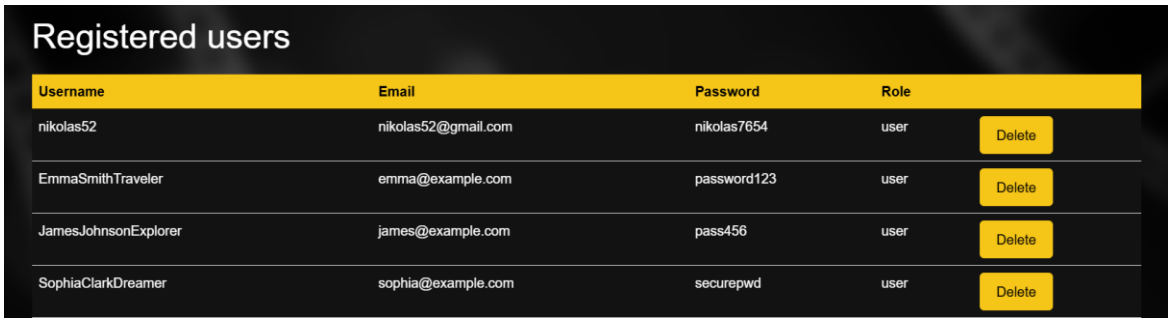
Εικόνα 6.30

Μια άλλη λειτουργία που εκτελεί ο διαχειριστής είναι η προσθήκη νέων ταινιών στην εφαρμογή. Αρχικά μεταβαίνει στην κεντρική σελίδα του (Εικόνα 6.24) και επιλέγει προσθήκη ταινίας (**Add Movie**). Έπειτα εισάγει τα στοιχεία της ταινίας και καταχωρείται στη βάση.



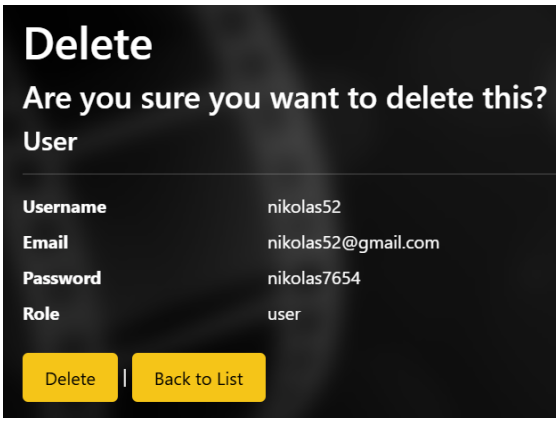
Εικόνα 6.31

Για την προβολή των εγγεγραμμένων χρηστών τώρα ο διαχειριστής από την κεντρική του σελίδα (Εικόνα 6.24) επιλέγει εγγεγραμμένοι χρήστες (**Registered users**) και οδηγείται σε σελίδα που εμφανίζονται τα στοιχεία των χρηστών.



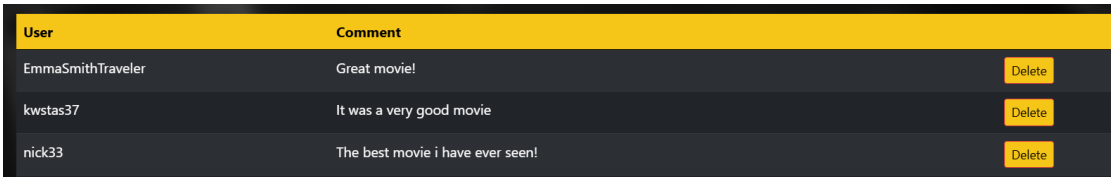
Εικόνα 6.32

Με αντίστοιχο κουμπί διαγραφής (**Delete**) που υπάρχει δίπλα από τα στοιχεία κάθε χρήστη, ο διαχειριστής τον διαγράφει από την εφαρμογή.



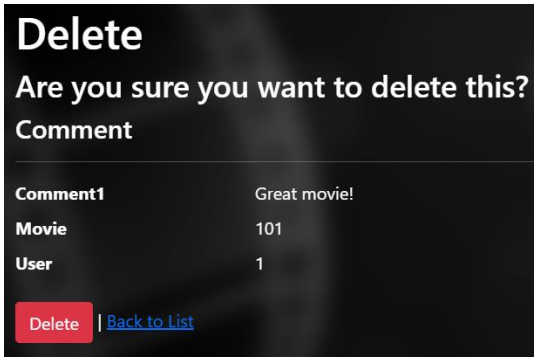
Εικόνα 6.33

Επίσης μεταβαίνοντας στην κεντρική σελίδα μιας ταινίας, με τον ίδιο τρόπο με τους χρήστες, επιλέγει το κουμπί σχόλια και εμφανίζονται τα σχόλια της ταινίας και η δυνατότητα διαγραφής.



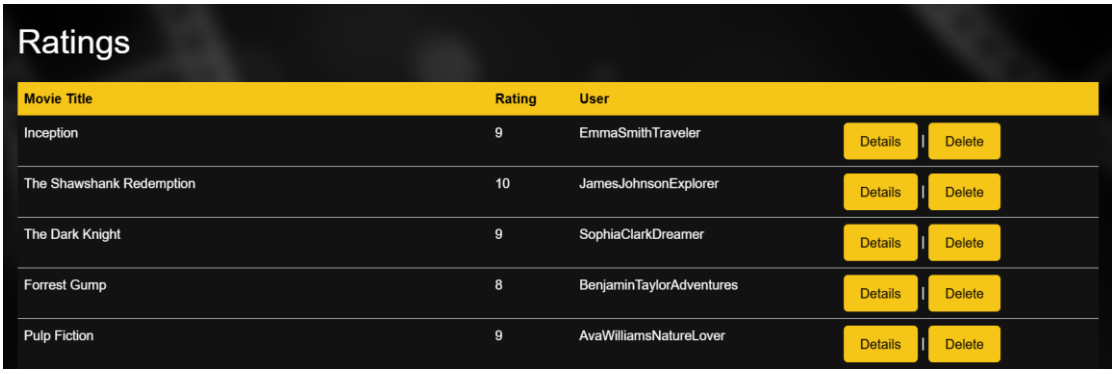
Εικόνα 6.34

Πατώντας διαγραφή (**Delete**) το σχόλιο σβήνεται.



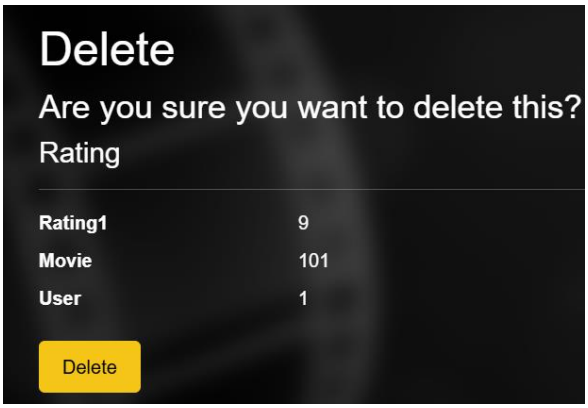
Εικόνα 6.35

Τέλος, από την κεντρική του σελίδα (Εικόνα 6.24) ο διαχειριστής έχει την δυνατότητα να δει αναλυτικά τις βαθμολογίες κάθε χρήστη για κάποια ταινία. Από την κορυφή της κεντρικής σελίδας επιλέγει προβολή βαθμολογιών (**See ratings**) και εμφανίζεται λίστα με αναλυτικές βαθμολογίες κάθε χρήστη.



Εικόνα 6.36

Πατώντας διαγραφή ο διαχειριστής μπορεί να σβήσει τη βαθμολογία.



Εικόνα 6.37

7. Επίλογος

Κατά την ανάπτυξη της παρούσας εφαρμογής, αντιμετωπίστηκαν διάφορες δυσκολίες και προκλήσεις που απαιτούσαν προσεκτική προσέγγιση και επίλυση σύνθετων προβλημάτων. Αυτές οι δυσκολίες, αν και αναμενόμενες σε ένα τέτοιο έργο, αποτέλεσαν πολύτιμες ευκαιρίες για μάθηση και βελτίωση των δεξιοτήτων. Μία από τις πρώτες και πιο σημαντικές δυσκολίες ήταν η σωστή σχεδίαση της βάσης δεδομένων. Έπρεπε να εξασφαλιστεί ότι οι πίνακες και οι σχέσεις μεταξύ τους ήταν σωστά καθορισμένες ώστε να υποστηρίζουν τις απαιτούμενες λειτουργίες της εφαρμογής χωρίς να δημιουργούνται προβλήματα απόδοσης ή ακεραιότητας των δεδομένων. Η διαδικασία αυτή απαιτούσε λεπτομερή ανάλυση των απαιτήσεων της εφαρμογής και προσεκτικό σχεδιασμό για τη σωστή λειτουργία και επεκτασιμότητα του συστήματος. Επιπλέον, η δημιουργία ενός φιλικού και εύχρηστου περιβάλλοντος για τον τελικό χρήστη αποτέλεσε μια συνεχή πρόκληση. Έπρεπε να σχεδιαστεί μια διεπαφή χρήστη που να είναι εύκολα κατανοητή και να επιτρέπει στους χρήστες να ολοκληρώνουν τις ενέργειές τους χωρίς δυσκολίες. Η δημιουργία φόρμας εγγραφής και σύνδεσης χρηστών, καθώς και η υλοποίηση λειτουργιών διαχείρισης για τους διαχειριστές της εφαρμογής, απαιτούσε λεπτομερή σχεδιασμό και συνεχή δοκιμή.

Συμπερασματικά, η παρούσα εργασία ανέδειξε τις δυνατότητες και τις προκλήσεις της ανάπτυξης μιας web εφαρμογής χρησιμοποιώντας το ASP.NET MVC framework. Μέσα από τη διαδικασία υλοποίησης της εφαρμογής διαχείρισης ταινιών, ηθοποιών και χρηστών, κατανοήθηκαν σε βάθος οι αρχές της αρχιτεκτονικής Model-View-Controller για πιο αποδοτική διαχείριση δεδομένων. Η χρήση του Microsoft SQL Management Studio για τη διαχείριση της βάσης δεδομένων σε συνδυασμό με το Entity Framework για τη σύνδεση της εφαρμογής με τη βάση δεδομένων, απέδειξε την αξία αυτών των εργαλείων στην ανάπτυξη ολοκληρωμένων και ευέλικτων λύσεων. Η εφαρμογή επιτυγχάνει να προσφέρει στους χρήστες μια φιλική και λειτουργική εμπειρία, επιτρέποντάς τους να εγγράφονται, να συνδέονται, να προσθέτουν σχόλια και βαθμολογίες σε ταινίες, γεγονός που ενισχύει τη διαδραστικότητα και την αλληλεπίδραση μεταξύ τους.

Τέλος η εργασία αυτή κατέδειξε τη σημασία της σωστής σχεδίασης και της οργανωμένης ανάπτυξης λογισμικού, καθώς και τον κρίσιμο ρόλο που παίζουν τα

σύγχρονα εργαλεία και τεχνολογίες στην επίτευξη αυτών των στόχων. Κλείνοντας, το ASP.NET MVC framework αποτελεί ένα ισχυρό εργαλείο για την ανάπτυξη web εφαρμογών, παρέχοντας την ευελιξία και τις δυνατότητες που απαιτούνται για την υλοποίηση σύνθετων και απαιτητικών έργων.

8. Βιβλιογραφία

[1] Συστήματα Βάσεων Δεδομένων 6η έκδοση, Εκδόσεις: Μ. Γκιούρδας

[2] Συστήματα Βάσεων Δεδομένων 7η έκδοση, Εκδόσεις: Μ. Γκιούρδας

[3] Το πέρασμα από τη Java στη C#, Ε. Αλέπης, Ι-Χ Παναγιωτόπουλος

[4] https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-8.0

[5] https://thales.cs.unipi.gr/

[6] Learn to Code Html & Css, Shay Howe

[7] ASP.NET MVC Succinctly, Nick Harrison

[8] Τεχνολογίες και Προγραμματισμός στον Παγκόσμιο Ιστό, Δουληγέρης Χρήστος, Μαυροπόδι Ρόζα, Κοπανάκι Εύη, Καραλής Απόστολος, Εκδόσεις: Νέων Τεχνολογιών

[9] Programming ASP.NET MVC 5 A Problem Solution Approach, Nimit Joshi, Εκδόσεις: C# Corner

[10] JavaScript Simplified, Taye Abidakun, Εκδόσεις: Smashwords