

Movies-TAB/TAB

Abstract

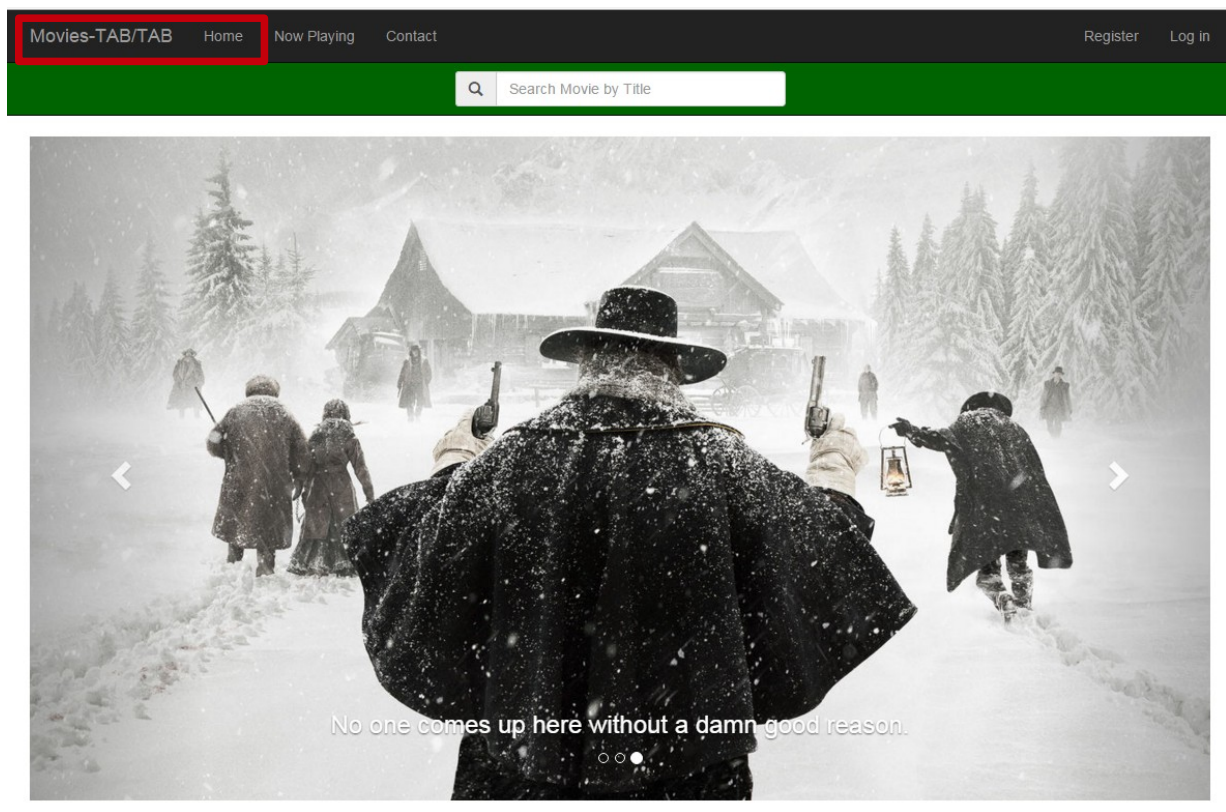
Movies-TAB/TAB is an Asp.Net Web Application designed with the MVC design pattern, using Visual Studio 2015, Entity Framework and the Code First workflow.

It is a basic fictional online movie tickets booking app in which a registered user can purchase tickets for a movie show of his choice within the catalog of the movie theater. A user who has been granted as “admin” has extended functionalities, giving him possibilities like managing the movies catalog as well as the screenings schedule.

In this document, will be shown the main pages of the website and their respective roles, then we will see the model built for the database, and finally we will take a tour on functionalities from a code-behind point of view.

I – Web Pages

1. Home Page



The Home page is pretty neat. Below the “Search” navbar, a carousel displays the three most popular movies of the moment within our catalog. By clicking on the picture, the user, registered or not, will be redirected to the “Details” page of the corresponding movie, which we will see later on.

2. Now Playing

Movies-TAB/TAB

Home


Now Playing

Contact

Register

Log in

Now Playing




Bridge of Spies - 141 min.
Release date: 10/16/2015

Director: Steven Spielberg
In the shadow of war, one man showed the world what we stood for.

[Watch Trailer!](#) [Book Now!](#)

★6.8/10 (339 votes)




The Martian - 141 min.
Release date: 10/2/2015

Director: Ridley Scott
Bring Him Home

[Watch Trailer!](#) [Book Now!](#)

★7.7/10 (1742 votes)



Spectre - 148 min.
Release date: 11/6/2015

Director: Sam Mendes

[Watch Trailer!](#)

★6.3/10 (1426 votes)

This is the movie “catalog” page of the website. Basic information like run time, release date, score... is displayed. For every title, a “Watch Trailer” button is available. When clicked, a modal appears and displays the movie trailer.

Movies-TAB/TAB

Home


Now Playing

Contact

Register

Log in

Now Playing




Bridge of Spies
Release date: 10/16/2015

Director: Steven Spielberg
In the shadow of war, one man showed the world what we stood for.

[Watch Trailer!](#)

★6.8/10 (339 votes)

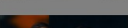


The Martian
Release date: 10/2/2015

Director: Ridley Scott
Bring Him Home

[Watch Trailer!](#) [Book Now!](#)

★7.7/10 (1742 votes)



Spectre
Release date: 11/6/2015

Director: Sam Mendes

[Watch Trailer!](#)

★6.3/10 (1426 votes)

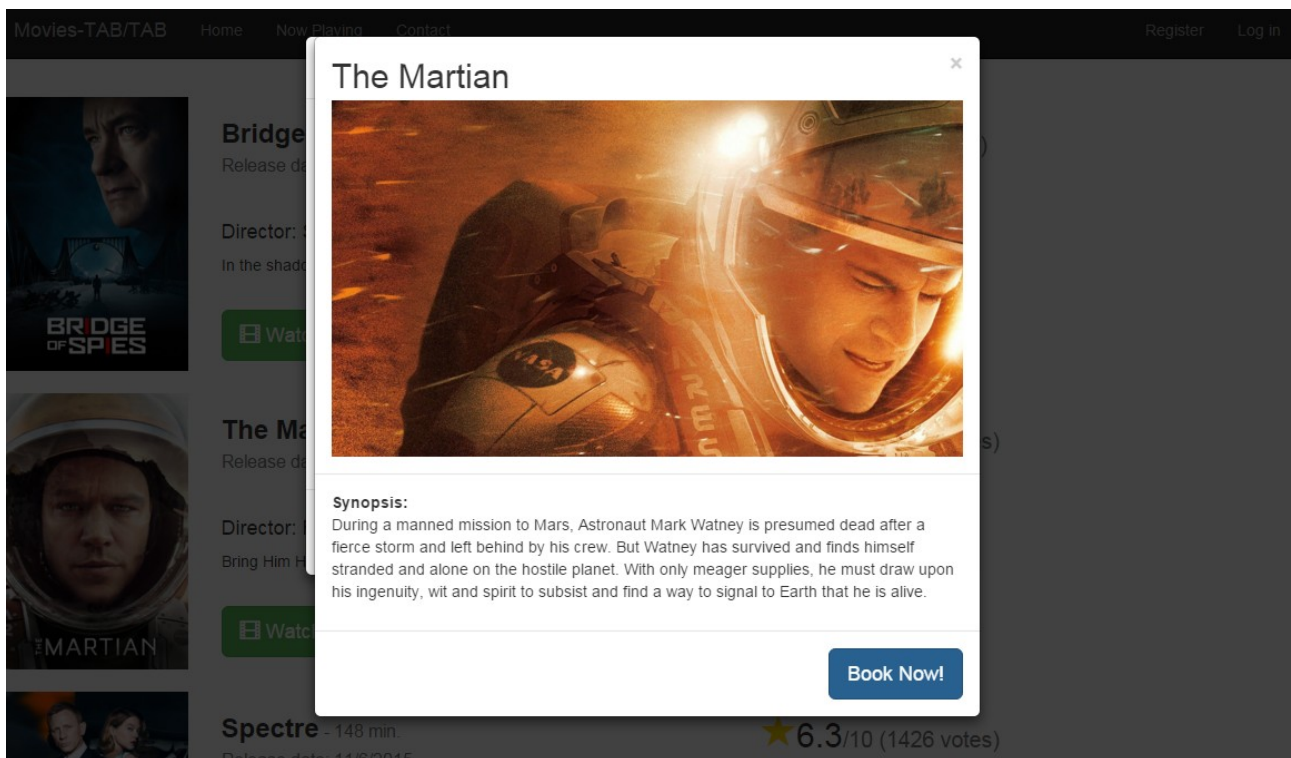
The Martian - Trailer

The Martian Official Trailer 1 HD



[Synopsis](#) [Book Now!](#)

As you can see, there is in the modal window a Synopsis button which opens an other modal window as in the following figure:

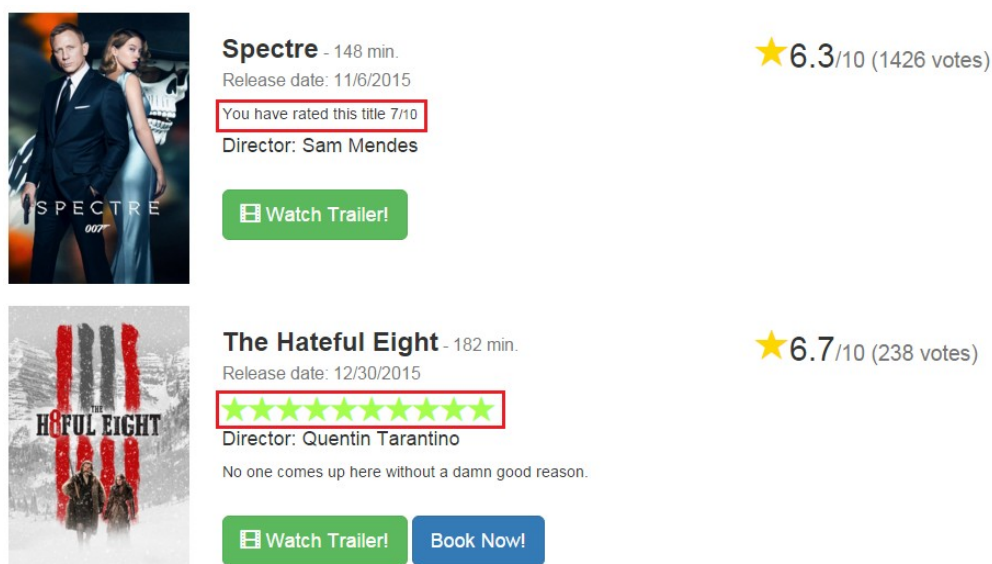


The “Synopsis Modal” can also be opened by clicking on the movie poster or title.

You probably noticed the “Book Now!” button, available almost everywhere except if there are no upcoming screenings scheduled for the title, like “Spectre” in the first figure of this section.

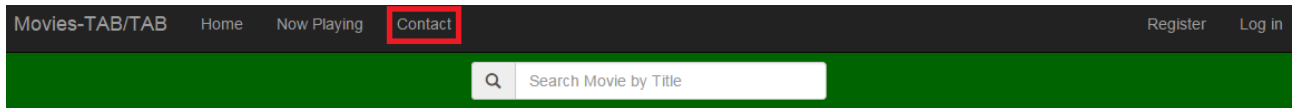
You probably guessed that the “Book Now!” button will redirect the user to the booking page, where he will have to choose a show time and the number of tickets he desires to purchase. In the case the user isn't logged in, he will be invited to do so or to register if he doesn't have an account yet.

If the user is logged in, he will have the possibility to rate a movie once. Then his rate will be displayed as in the following figure:



3. Contact

The contact page gives the user the address of the company and an e-mail address for support purposes.



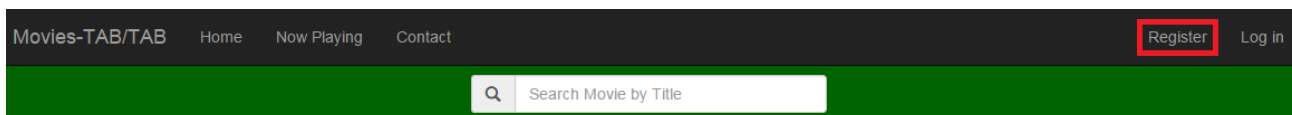
Contact Us.

Movies-TAB/TAB
Somewhere on Earth 🌐

Support: moviestabtab@gmail.com

© 2016 - Movies-TAB/TAB

4. Register



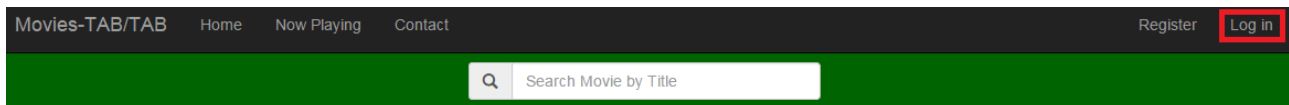
Register.

Create a new account.

First Name	<input type="text"/>
Last Name	<input type="text"/>
ID Number	<input type="text"/>
Nickname	<input type="text"/>
Email	<input type="text"/>
Password	<input type="password"/>
Confirm password	<input type="password"/>
<input type="button" value="Register"/>	

In the register page, the user fills in the form in order to create an account. We were asked to add a credit card number field, but I chose to not add it because I would not like as a potential customer, to give my credit card number at the same time as I register.

5. Log in

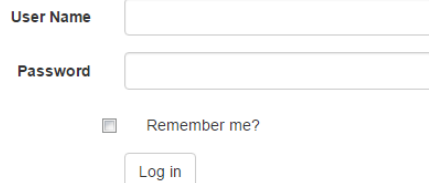


The screenshot shows the top navigation bar of the 'Movies-TAB/TAB' website. The bar has a dark background with white text for 'Movies-TAB/TAB', 'Home', 'Now Playing', and 'Contact'. On the right side, there are links for 'Register' and 'Log in', with the 'Log in' link highlighted by a red box. Below the navigation bar is a green search bar with a magnifying glass icon and the placeholder text 'Search Movie by Title'.

Log in.

Use a local account to log in.

Use another service to log in.



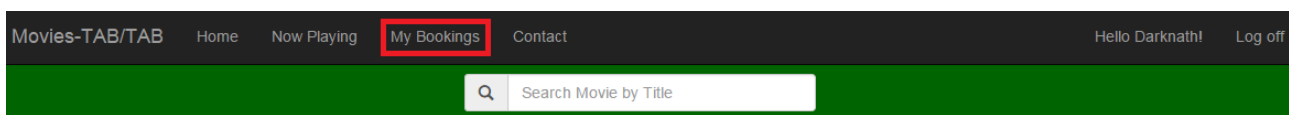
The screenshot shows the local login form. It has two input fields: 'User Name' and 'Password'. Below the 'Password' field is a checkbox labeled 'Remember me?'. At the bottom of the form is a 'Log in' button.

Facebook

[Register as a new user](#)

Here the user can log in the classic way with user name and password, or via a social network.

6. My Bookings (for logged in users)



The screenshot shows the top navigation bar of the 'Movies-TAB/TAB' website. The bar has a dark background with white text for 'Movies-TAB/TAB', 'Home', 'Now Playing', 'My Bookings', and 'Contact'. The 'My Bookings' link is highlighted by a red box. On the right side, there are links for 'Hello Darknath!' and 'Log off'. Below the navigation bar is a green search bar with a magnifying glass icon and the placeholder text 'Search Movie by Title'.

MyBookings

Title	Show Time	Hall Name	Number of Tickets
Star Wars: The Force Awakens	1/11/2016 9:30:00 PM	A	5
Bridge of Spies	1/11/2016 5:00:00 PM	B	2

© 2016 - Movies-TAB/TAB

The top navbar menu will be substantially different according to the user's permissions. A “customer” user has in addition a link to a page on which is displayed his booking history.

7. Manage Movies Catalog (for users in role of “admin”)

An admin has full access to the movie catalog. He can create, update and delete movies from it.

You can see in the following figure that there is a search bar in the “Create” page. Indeed, instead of filling all the form manually, the admin has the possibility to look for the title he wishes to add to the catalog in an API granted by themoviedb.org (for further information follow the link below: <https://www.themoviedb.org/documentation/api>). Once he chooses the title from the results drop-down list, the different fields are filled in automatically in accordance to the data provided by the API.

Movies-TAB/TAB
Home
Now Playing
Manage Movies Catalog
Manage Screenings
My Bookings
Contact
Hello admin!
Log off

Create

Movie

the revenant

Search!

See the Results ▾

The Revenant-2015-12-16

The Revenant-2009-08-16

The Dead and the Deadly-1982-01-01

The Revenants-

Original Title

Runtime

Release Date

mm/dd/yyyy

Tagline

8. Manage Screenings (for users in role of “admin”)

Similar to “Manage Movie Catalog”. The admin is able to manage screenings for the different movies.

Movies-TAB/TAB
Home
Now Playing
Manage Movies Catalog
Manage Screenings
My Bookings
Contact
Hello admin!
Log off

Q

Search Movie by Title

Create

Screening

Title

The Hateful Eight ▾

Hall

A ▾

Available Seats

100

Show Time

01/11/2016 7:00 PM

The hall is not available at this time. Check the schedules and choose an other day/hour.

Create

[Back to List](#)


9. Booking (for logged in users)

Movies-TAB/TAB
Home
Now Playing
My Bookings
Contact
Hello Darknath!
Log off

Q

Search Movie by Title

Booking - Bridge of Spies



Show

1/11/2016 5:00:00 PM ▾

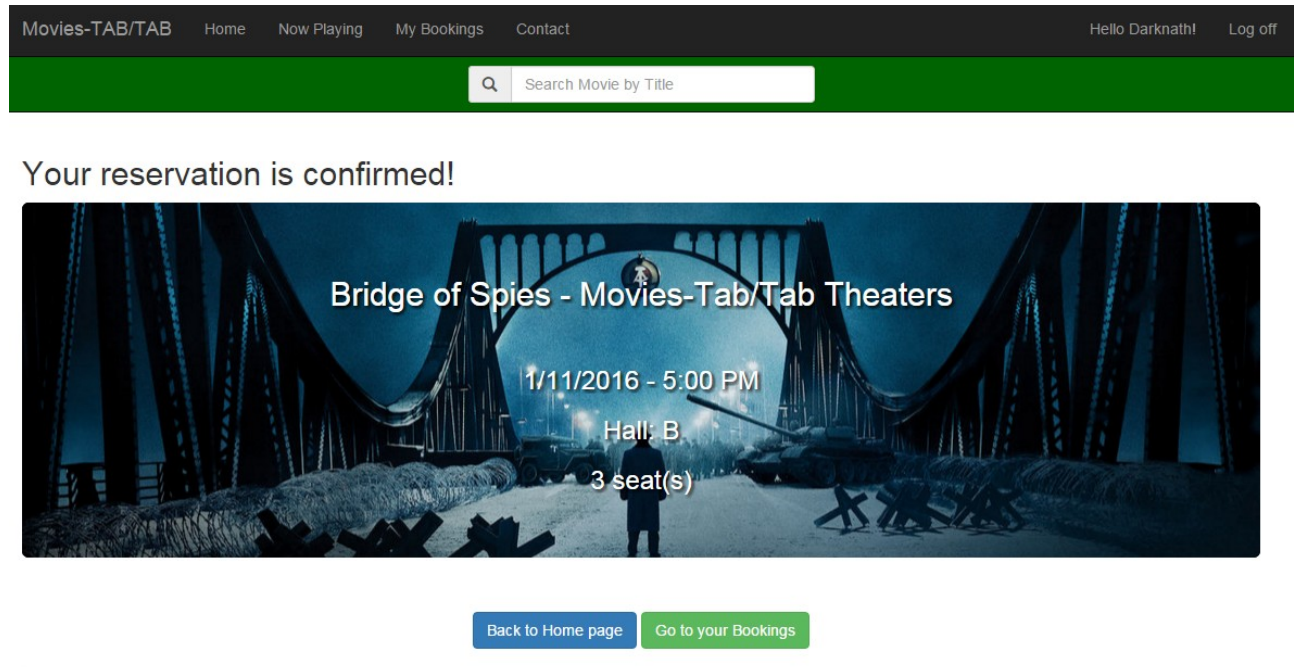
Number of Tickets

3| ▴ ▾

Submit

When the user clicks on the “Book Now!” button of a movie, he is redirected to the booking page of this one. He then has to choose a show time and a number of tickets in order to submit the form.

In the “real world”, I would have of course added a “payment step” before validating the booking, but it was not the purpose in this project. So when the form is submitted and validated, the user is redirected to a “booking confirmation” page and receives also an email.



There are at the bottom of the page a “Back to Home Page” and a “Go to your Bookings” button, so the user is invited to check his bookings immediately, especially if an error occurred and he didn't receive the confirmation email.

II – Model and Database structure

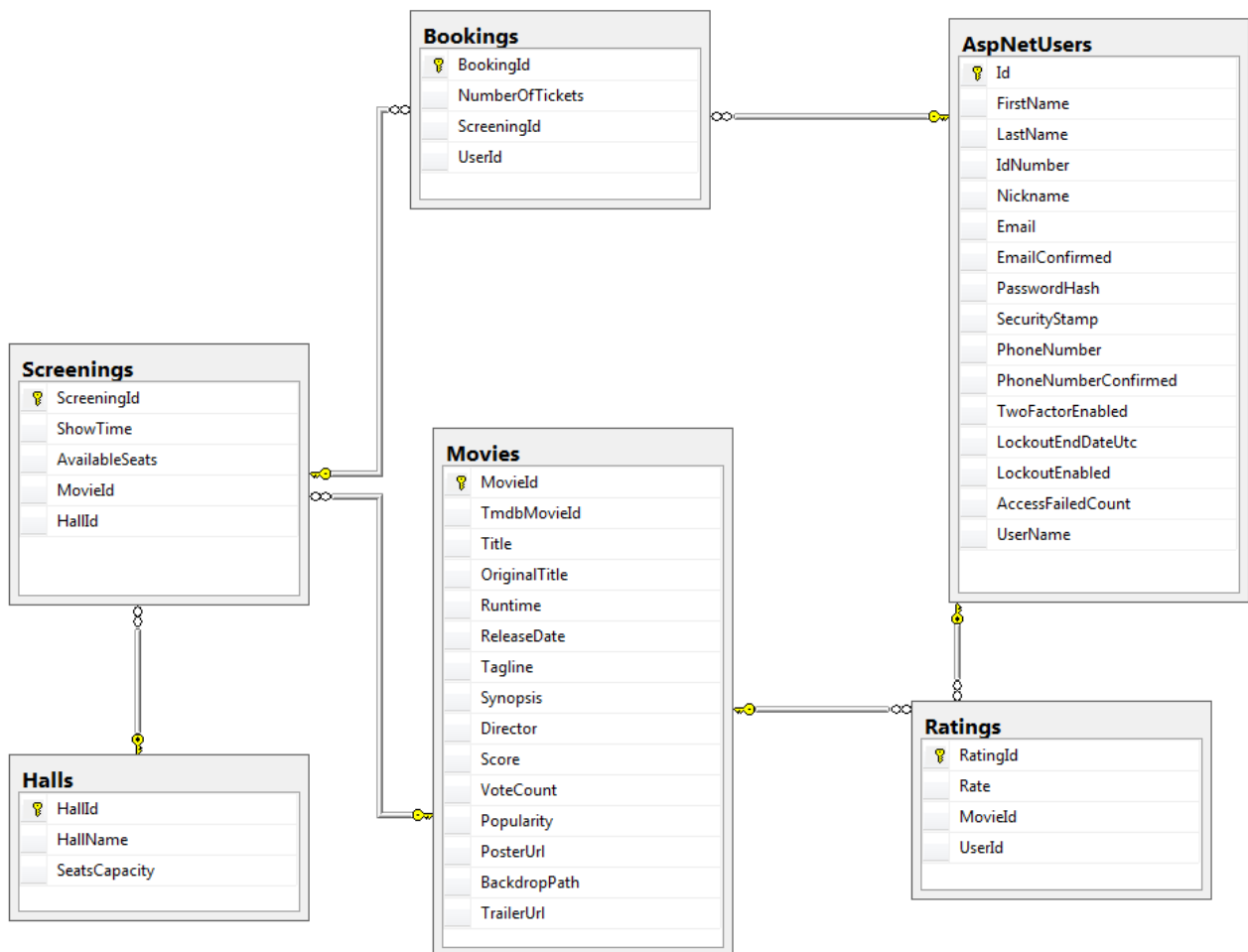
In this section, we will see the different entities the data model is made of and the relationships between them.

First, we will take a look at the database diagram, in order to have a general view of its structure, then we will zoom into each entity, and see their role within the general “mechanics”.

1. Database Diagram

The following figure represents the database diagram of the entities we will discuss here.

The “AspNetUserRoles” and “AspNetRoles” tables could have been added but in worries of readability, I didn't include them to this diagram.



2. The Movie Entity

```

public class Movie
{
    [Key]
    public int MovieId { get; set; }

    public string TmdbMovieId { get; set; }

    [Required]
    public string Title { get; set; }

    [Display(Name = "Original Title")]
    public string OriginalTitle { get; set; }

    public short? Runtime { get; set; }

    [DataType(DataType.Date)]
    [Display(Name = "Release Date")]
    public DateTime ReleaseDate { get; set; }

    public string Tagline { get; set; }

    public string Synopsis { get; set; }

    public string Director { get; set; }

    [Range(0.0d, 10.0d)]
    public double? Score { get; set; }
}
  
```



```

[Display(Name = "Vote Count")]
public int? VoteCount { get; set; }

public double? Popularity { get; set; }

public string PosterUrl { get; set; }

public string BackdropPath { get; set; }

public string TrailerUrl { get; set; }

public virtual ICollection<Screening> Screenings { get; set; }

public virtual ICollection<Rating> Ratings { get; set; }

public Movie()
{
    this.Screenings = new HashSet<Screening>();
    this.Ratings = new HashSet<Rating>();
}
}

```

A movie, in addition of properties we can logically expect like a Title, Release Date, Director..., has two – one to many relationships. One movie has a collection of screenings, and a collection of ratings given by the different users.

The “TmdbMovieId” is the Id of the title in themoviedb Api. I needed to include it because I don't use the tmdb api only for searching a title I wish to add to the movie catalog. There are other requests made to the api in order to update for each movie, its score, number of votes and popularity, which gives some “life” to the app with fresh data on an everyday basis. Also, when a registered user gives a score to a movie, this one is sent to the api, but we will discuss those two points further.

2. The Screening Entity

```

public class Screening
{
    [Key]
    public int ScreeningId { get; set; }

    [Required]
    [DataType(DataType.DateTime)]
    [Display(Name = "Show Time")]
    public DateTime ShowTime { get; set; }

    [Display(Name = "Available Seats")]
    public byte? AvailableSeats { get; set; }

    [ForeignKey("Movie")]
    public int MovieId { get; set; }

    [ForeignKey("Hall")]
    public int HallId { get; set; }

    public virtual Movie Movie { get; set; }

    public virtual Hall Hall { get; set; }
}

```

```

    public ICollection<Booking> Bookings { get; set; }

    public Screening()
    {
        this.Bookings = new HashSet<Booking>();
    }
}

```

A screening must have a time, a place (the Hall), a number of available seats (by default the capacity of the hall), and of course the Movie projected. It has a one to many relationship to a collection of bookings.

3. The Hall Entity

```

public class Hall
{
    [Key]
    public int HallId { get; set; }

    [Required]
    [Display(Name = "Hall Name")]
    public string HallName { get; set; }

    [Display(Name = "Capacity")]
    public byte? SeatsCapacity { get; set; }

    public virtual ICollection<Screening> Screenings { get; set; }

    public Hall()
    {
        Screenings = new HashSet<Screening>();
    }
}

```

In a Hall a movie is screened. It has a Name, a number of seats, and a collection of screenings. N.b: I didn't implement CRUD functions in the Halls Controller. In my opinion, the number of halls a movie theater has, doesn't evolve often to do so, and I have added them via the Seed method in the Configuration class.

4. The Rating Entity

```

public class Rating
{
    [Key]
    public int RatingId { get; set; }

    [Range(0.0d, 10.0d)]
    public double Rate { get; set; }

    [ForeignKey("Movie")]
    public int MovieId { get; set; }

    [ForeignKey("User")]
    public string UserId { get; set; }

    public virtual Movie Movie { get; set; }

    public virtual ApplicationUser User { get; set; }
}

```

Each user can rate each movie once.

5. The Booking Entity

```
public class Booking
{
    [Key]
    public int BookingId { get; set; }

    [Required]
    [Display(Name = "Number of Tickets")]
    public byte NumberOfTickets { get; set; }

    [ForeignKey("Screening")]
    public int ScreeningId { get; set; }

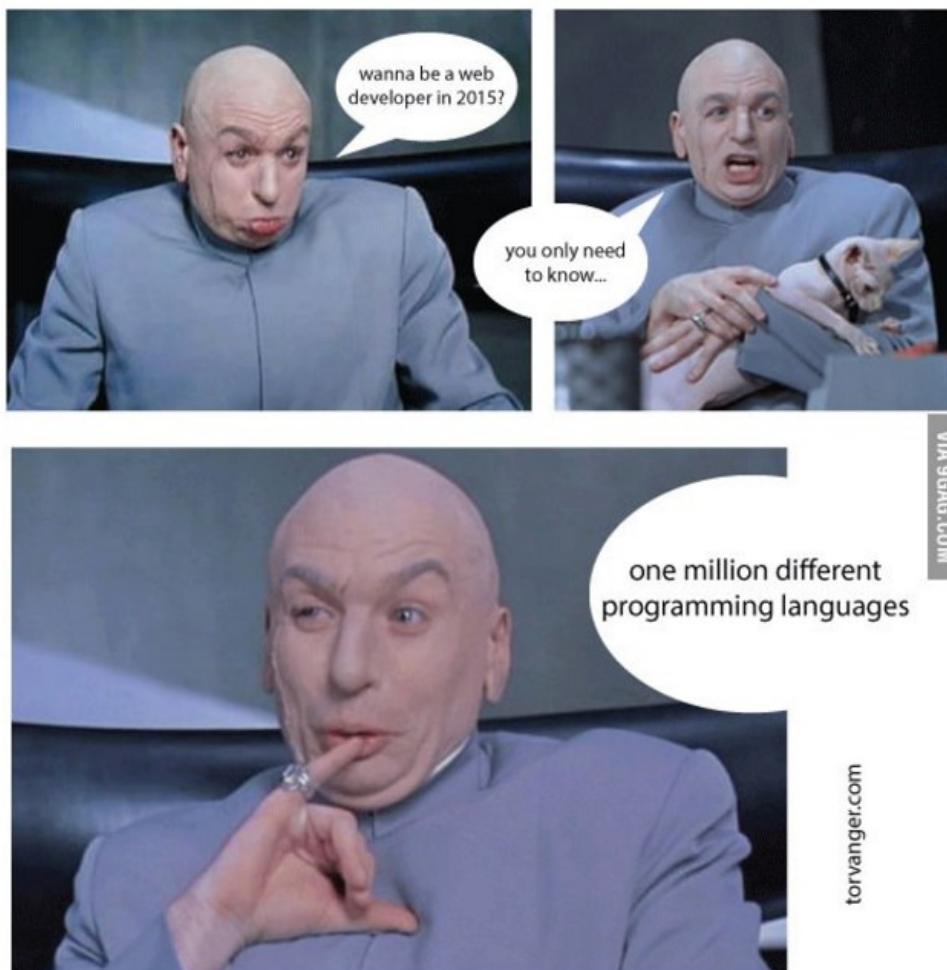
    [ForeignKey("User")]
    public string UserId { get; set; }

    public virtual Screening Screening { get; set; }

    public virtual ApplicationUser User { get; set; }
}
```

A booking is like we know, when a user chooses a show and purchases tickets for this one.

III – Functionalities/Behind the scenes



1. User's Roles

For the purpose of this project, I needed to create two different roles: “customer” and “admin”. As talked before, a “customer” will be able to rate once every movie, buy tickets for show and see his purchases history in “My Bookings”. An admin can also do so, but in addition, and after that's his job, will be able to manage the movies catalog and their screenings.

So after having enabled migrations, in the Seed method of the Configuration class, we check if those two roles exists and if not, they are added to the “AspNetRoles” table, and an “admin” user is added to the “AspNetUsers” table as in the following block of code:

```
var roleStore = new RoleStore<IdentityRole>(context);
var roleManager = new RoleManager<IdentityRole>(roleStore);

string[] roles = { "admin", "customer" };

foreach (var role in roles)
{
    if (!roleManager.RoleExists(role))
    {
        roleManager.Create(new IdentityRole(role));
    }
}

var userStore = new UserStore<ApplicationUser>(context);
var userManager = new UserManager<ApplicationUser>(userStore);

var adminUser = new ApplicationUser() { UserName = "admin", Email = "moviestabtab@gmail.com" };
userManager.Create(adminUser, "P@ssw0rd");
userManager.AddToRole(adminUser.Id, "admin");
```

By default, every new user who registers by filling the form or by external log in gets automatically the “customer” role: in the “AccountController”, in the “Register” and “ExternalLoginConfirmation” tasks, I added:

```
var result = await UserManager.CreateAsync(user, model.Password);
if (result.Succeeded)
{
    UserManager.AddToRole(user.Id, "customer");
    await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);
}
```

2. Using API provided by themoviedb.org

a. Add/Edit movies from the catalog

As explained before, in order to give some “life” and real data to the project, some requests are made to this API in different purposes.

At the beginning, I wanted to offer our admin guy the possibility to easily add new titles to the movies catalog. To do so, I've added a search bar in the “Create” action of the Movies Controller.

The admin then enters the title he is looking for, and once we get the response, a dropdown list appears in which there are the titles and their release date, in the case there are homonyms.

Once the desired title is selected, the fields of the form are filled in automatically and the admin just has to click on the submit button. If an error occurred or the search result is null, he can always fill in the form by himself.

The script is in the “Scripts” folder, TmdbApiSearch.js

b. Movie Rating

In the pages where movies are displayed like the “Now Playing” page or the “Details” page of a title, a logged in user has the possibility to give a rate, and if done, the score he gave is displayed instead of the widget.

I saw in the Api documentation that there is a function to rate movies, so I thought it would be nice to share with them the rates my users give to the different titles (see the `_MovieRating` view in `~Views/Shared` in the solution).

In order to rate a movie in the Api, a guest session id is needed. It's valid for 24 hours but in the script I wrote, every time a user rates a movie, a new request for it is done. The Api accepts from its users (users with an api key, not guest sessions) up to 40 calls every 10 sec, so in the limits of our fictional movie theater website it works just fine. Therefore, an important update in my eyes would be to give each user a guest session id to the Api on a session start level so it would reduce drastically the number of useless calls in the case `Movies-TAB/TAB` would be deployed.

Furthermore, the global score and number of voters displayed are the ones gotten from the Api at the time the movies were added to the database. My first thought was to use this data as an initialization and from it, when a user rates a movie, the movie's score average would have been calculated, and the number of voters increased by one. On a second thought, if the rates given by the users are sent to the Api, why not getting the updated data back from it? I read in the Api documentation that the movies statistics are updated once a day. So in `Global.asax`, I've added a function which updates those stats on a 24hrs basis as if the app was deployed: a variable that holds a `DateTime` indicates when the task last ran, and is initiated to the current time when the application starts. I assumed that the web app gets enough visitors that `Session_Start()` will be fired often enough to be the trigger of the update function. My only regret is that when a user rates a movie, he will not see his vote counted immediately, but in an educational point of view, it was more interesting to set up all the rating functionalities this way.

3. Bookings

When a user books a movie show, he is invited to choose a screening and a number of tickets. I've handled the case the user chooses less than one ticket and the case he wishes to buy more tickets than there are available seats for the screening.

When the booking is confirmed, the number of available seats for the show is decreased by the number of tickets just sold, and an email is sent to the user asynchronously so he will not

experience some freezing, especially if the smtp server is down or something.

4. Hall

As said previously, I've added the halls via the Seed() method in Configuration.cs.

However, in the Hall Controller, there is a function which returns in Json the seats capacity of the hall it takes as parameter. I needed this function to make the admin's life easier. When he is adding a new screening, the number of available seats field is automatically set with the capacity of the selected hall. If for a reason or an other he needs to change the value, he can always do it manually.

5. Screenings

In order to schedule a new screening, the admin has to choose a movie, a hall/available seats, and a show time. Here, a lot of logic can be taken in consideration to validate or not the form, and I think that the screenings management deserves its own interface. The least I could do for our admin guy was to provide him a “datetime-picker” and to tell him if the hall is available or not according to the movie and the time he picks up.

Create

Screening

Title	<input type="text" value="Bridge of Spies"/>
Hall	<input type="text" value="B"/>
Available Seats	<input type="text" value="80"/>
Show Time	<input type="text" value="01/11/2016 4:30 PM"/> <div>The hall is not available at this time. Check the schedules and choose an other day/hour.</div>
	<input type="button" value="Create"/>

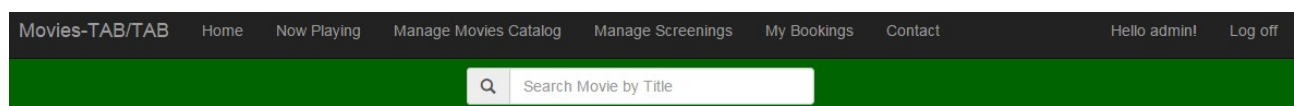
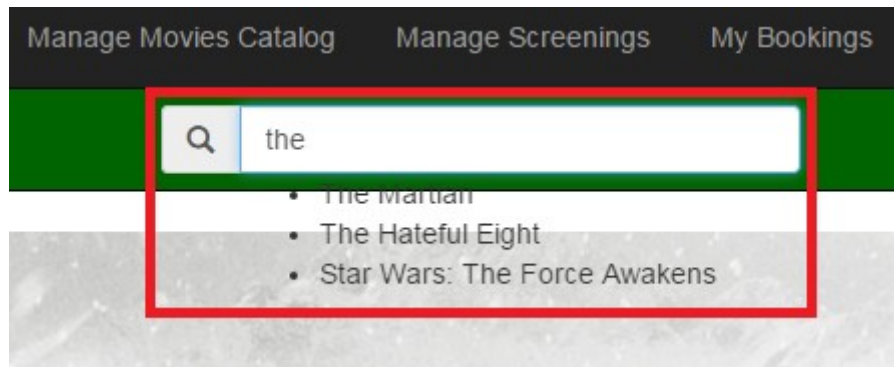
The logic I introduced in this validation is divided into four cases. You can check the Cases() method at the bottom of the Screenings Controller, but globally, in order to validate a show time, the hall must be available 15mins after the end of the previous show, and 15mins before the beginning of the next one.

6. Search-bar with autocomplete

I didn't mention it yet but bellow the main navigation-bar, there is a search-bar on which a user can look for a movie by title.

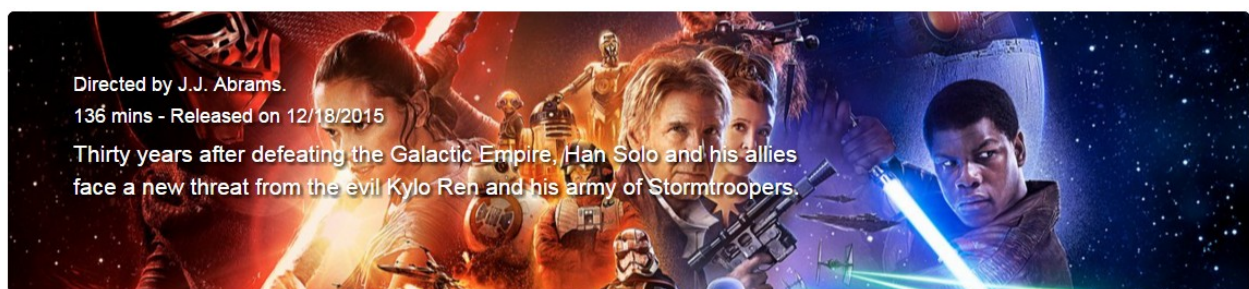
The widget is an implementation of the jqueryui-autocomplete, and when he clicks on the desired

title from the results, he is redirected to the “Details” page of the corresponding movie:



Star Wars: The Force Awakens

★ 8/10 (1877 votes)



Edit this Title

Watch Trailer!

Book Now!

Back to Catalog

In this figure the user is “admin” so he has a button link to edit the title. If a user not in this role is in this page, the “Edit this Title” button doesn't appear.

Despite the fact that I didn't manage to change the style of the suggestions, jqueryui-autocomplete gave me some hard times. In the Search Controller, a function gets a research term as parameter and returns a list of movies whose titles contains the research term. As long as it was returning a string array of titles, it worked fine, but I needed to send also the MovieId in order to be able to redirect the user to the Details page. So in the query, instead of selecting the title, I used to select the Movie objects themselves and to serialize the list of objects with the json() function. This way, everything worked for a few days, then strangely, I systematically got a status 500. At this point, I still don't understand why it used to work perfectly and suddenly not. I thought that the Json() function maybe encountered some difficulties when serializing Movie objects even though there were no exceptions thrown. So in the controller, I have created a small class called MovieHelper which has only two properties: Title, and Id; and instead of returning heavy json with hole Movie objects, it serializes a List of MovieHelper objects containing just a title and an id, and since then it works again.