**Code Structure – Cinema Web Application**

Martin Krisko (240314)

Miroslav Fratric (240160)

**August 2018**

IT-SEP-X-A18

Table of Contents

[1. Back end code structure 3](#_Toc522822397)

[1.1. Controllers Folder 3](#_Toc522822398)

[1.2. Models folder 3](#_Toc522822399)

[1.3. Startup.cs 4](#_Toc522822400)

[2. Front end code structure 4](#_Toc522822401)

[2.1. app folder 5](#_Toc522822402)

[2.2. \_guard and \_helper folder 5](#_Toc522822403)

[2.3.\_models folder 5](#_Toc522822404)

[2.4.\_services folder 5](#_Toc522822405)

[2.5. Component folders 6](#_Toc522822406)

# Back end code structure

Our project on the back end is WEB.API project template from ASP.NET. Here except config files and nugget packages you can find folders Controllers and Models as well as file Startup.cs. These are the most important for our project.

# Controllers Folder

Inside this folder there are WEB API controllers, each controller represents part of logic and can be accessed by calling “/api/[controller]”. The API endpoints name is name of the controller without the “controller” part. Therefore, endpoint for MovieController will be “/api/movie”.

Each controller contains methods for HTTP requests relevant to its purpose. Within these controllers we are also processing data and storing them to the database which is not the best practice. Reason why we did it this way is the short period for this project, so it was convenient for us to have everything at one place.

Special place between controllers belongs to MovieController. Because we are using in memory database we need to create database every time we are running the program. Once you start the server it will open browser with URL to API call to movie (.../api/movie). This action will call the constructor of this controller where we call private function fillDatabase(), this function then set the context of the database.

# Models folder

Models are folder where we are storing our models. In this folder there are three types of files.

* Database models: These models are used for generating the Entity Framework database tables. These models are Movie, MovieSeat, User and ParkingPlace. Within these models there are special fields with annotation [NotMapped] which tells Entity Framework not to put them into the database.
* Database context file: This file is called CinemaContext.cs and it is responsible for creating tables from database models with the Entity framework.
* Data transfer models: The rest of the models are specific models for handling HTTP requests from front end. They were created because HTTP methods can send only one object.

# Startup.cs

In this file we are adding services which are used in our project.

* Setting database: by calling AddDbContext on CinemaContext.cs. Also, there is a commented service for switching to SQLServer.
* Enabling CORS: for every call from localhost:4200, for every method, header and credentials. This is required for running this application on localhost.
* Serializer settings: for our database, to help Entity Framework handle table reference in the database.
* JWT token authentication: In early stage of development we wanted to use JWT authentication. If you would like to use it uncomment this code, it was tested on back end and it’s working,

# Front end code structure

Front end is bit more complicated regarding structure. First, we look into configuration elements.

* Angular.json: is a configuration file for a whole project.
* Package.json: it contains all dependencies required for running an application, also there are scripts which can be used from the terminal. We modified script start (npm start) to use a proxy file.
* Proxyconfig.json: here we are setting up simple proxy to helps us avoid problems with CORS.

Then most important part is in app folder where is the actual application.

# app folder

App folder contains the top level of our application as well as several folders which we will look into later.

* App.module.ts: here we instantiate all components, importing third party modules and setting up providers so they are accessible through the whole application.
* App-routing.module.ts: here we are setting up routes to each component so we can then navigate to them through router.
* App.component.\*: is the top component we have there first level of navigation and getting user information from localStorage if it has any.

# \_guard and \_helper folder

In general, these folders contain TypeScript files which were created to handle JWT authentication. But because we did not have time to finish implementation of this in the front end, they are never called. We are keeping these files inside to show the way how they meant to be implemented.

Only exception is handler.ts in \_helper folder. We are using this for handling errors from HTTP response.

# 2.3.\_models folder

In this folder we are keeping our models. These models are TypeScript equivalent of back end models.

# 2.4.\_services folder

Here we are storing our services. We have three types of services in our application:

* Services handling API calls: these services are equivalent of back end controllers. Sending requests to the server and getting the responses.
* Toast service: for calling toast in the component upon success or fail.
* Data service: this service store data we are getting from back end. These data are immutable therefore it can be only set or get.

# Component folders

The rest of the folders are component folders. They include html, CSS and TypeScript files for each component.