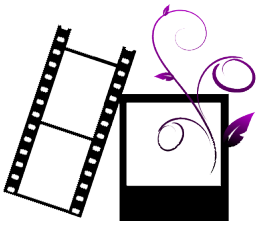
Coursera

The Hong Kong University of Science and Technology

Full Stack Web Development Specialization

Capstone Project



MyMovies

Author:
Tiago Justino

Supervisor:
Jogesh Muppala

April 30, 2017

Contents

1 Introduction **2**
1.1 Expected List of Features 2

2 Design and Implementation **3**
2.1 The REST API Specification 3
2.2 Front-end Architecture Design 6
2.3 Database Schemas, Design and Structure 7
2.4 Communication 9

3 Conclusions **13**

1 Introduction

MyMovies is a social network for cinema fans. It keeps track of movies you've watched, want to watch and don't want to watch. By leveraging the IMDB database it allows you to add custom pieces of information, such as, rating and comments. Also, the social network aspect brings you the possibility of discovering new movies you want to watch and new points of view on movies you've watched at the same time that it allows you to share good moments with people you love.

As a bonus, by keeping track of people's movie watching habits MyMovies is able to help the cinema industry on targeting their customer more precisely.

1.1 Expected List of Features

Inside the scope of this project we plan on achieving the following features:

- The user will be able to create an account and login to the system with or without Facebook. The user will also be able to link or unlink their MyMovies account to their Facebook account;
- The user will be able to invite other users to be their friends on MyMovie. The invited user will be able to accept or decline the invite;
- The user will be able to invite their Facebook friends to join MyMovies.
- The user will be able to follow other users. Users added as friends are followed by default. The user will also be able unfollow friend without unfriending them;
- The user will be able to mark a movie as watching, watched, want to watch or don't want to watch;
- For movies previously marked as watched, the user will be able to favorite and to add rating and comments;
- The user will be able to *fan* names, such as actors, actresses, directors and producers;
- Movies marked as watching are automatically marked as watched after the movie duration (known from imdb api);
- When marking a movie as watching or watched the user will be able to mark a friend as "watching with";

- The user will be able to add a date (day, month or year) for movies watched in the past. Date is not mandatory;
- The user will be able to mark a movie as watched more than once;
- The user will be able to access a timeline page where they can see the activities of whom they follow. The timeline will show activities such as watching and watched movies, and comments;
- The user will be able to like or comment on friend's activities,
- The user will be able to mark their comment (or part of it) as spoiler,
- Comments marked as spoiler will appear with a warning message on a user timeline if they haven't watched that movie,
- The user will be able to create and join groups (private and public), such as "Movie Club at Work" or "Stanley Kubrick Fans",
- The user will be able to recommend a movie to a friend.

2 Design and Implementation

2.1 The REST API Specification

.

Here we describe all the endpoints and the supported operations. Wikipedia[1] was used as reference.

/users

POST Registers a user.

GET List registered users.

/users/:id

GET Get user details.

PUT Update user details.

DELETE Delete a user account.

/invitation

POST When a loggedin user invites another user to be friends.

GET Lists friendship invitations.

/invitation/:id

DELETE Cancel a friendship invitation.

/follow

POST When a loggedin user start following another user.

GET Lists people you follow.

/follow/:id

DELETE Stop following someone.

/friendship

GET Lists friends.

/friendship/:id

DELETE Stops being friend to someone.

/friendshiprequest

GET Lists friendship requests.

/friendshiprequest/:id

GET Get friendship request details.

POST Accepts a friendship request.

DELETE Reject a friendship request.

/watching

GET Lists movies currently marked as watching.

POST Marks a movie as watching.

/watching/:id

DELETE Removes a movie from watching.

PUT Change a watching event (add people to watching with).

/watched

GET Lists movies marked as watched.

POST Adds a movie to watched list

/watched/:id

GET Gets details of a watching event.

PUT Change a watch event (adds favorite, rating or comments). The user can also change date they watched the movie, times, and with whom.

DELETE Remove a movie from the watched list.

/wanttowatch

GET Lists movies marked as want to watch.

POST Adds a movie to want to watch list

/wanttowatch/:id

DELETE Remove a movie from the want to watch list.

/dontwanttowatch

GET Lists movies marked as don't want to watch.

POST Adds a movie to don't want to watch list

/dontwanttowatch/:id

DELETE Remove a movie from the don't want to watch list.

/group

GET Lists created groups.

POST Creates a group.

/group/:id

GET Get a group details.

PUT Change group details. Add people to group.

DELETE Deletes a group.

/groupjoin

POST Asks to join a group.

GET Lists pending requests to join groups.

/group/:id

DELETE Cancels a pending request to join a group.

2.2 Front-end Architecture Design

The client side, for both the web application and the mobile hybrid application, will have the following structure:

/index.html The SPA landing page.

/images Images storage directory.

/views View directory.

/styles CSS storeage directory.

/scripts Javascript files directory.

/scripts/app.js The main application file.

/scripts/controllers.js Controllers definition.

/scripts/services.js Services definition.

/views/register.html User registration page.

/views/login.html Login page.

/views/header.html Website header.

/views/footer.html Website footer.

/views/aboutus.html About us page.

/views/timeline.html The timeline page component.

/views/profile.html The users profile page.

/views/stats.html User statistics page component.

/views/watched.html List of watched movies.

/views/wanttowatch.html List of want to watch movies.

/views/dontwanttowatch.html List of don't want to watch movies.

/views/movie.html Movie details page.

/views/group.html Group details page.

2.3 Database Schemas, Design and Structure

The following database Schemas will be used:

```
var User = new Schema({
  username: String,
  password: String,
  OauthId: String,
  OauthToken: String,
  firstname: {
    type: String,
    default: ''
  },
  lastname: {
    type: String,
    default: ''
  },
  picture: {
    type: String,
    default: ''
  },
  friends: [User],
  following: [User],
  watching: {
    type: WatchingEvent,
    required: false
  },
  watched: [WatchedEvent],
  wanttowatch: [Movie],
  dontwanttowatch: [Movie]
});
```

Listing 1: User Schema

```
var Movie = new Schema({
  imdbId: {
    type: String,
    required: true
  }
});
```

Listing 2: Movie Schema

```
var FriendshipRequest = new Schema({
  requester: {
    type: User,
    required: true
  },
  requestee: {
    type: User,
    required: true
```

```
  }
});
```

Listing 3: FriendshipRequest Schema

```
var WatchingEvent = new Schema({
  user: {
    type: User,
    required: true
  },
  movie: {
    type: Movie,
    required: true
  }
  friends: [User],
  datetime {
    type: String,
    required: true
  }
});
```

Listing 4: WachingEvent Schema

```
var WatchedEvent = new Schema({
  user: {
    type: User,
    required: true
  },
  movie: {
    type: Movie,
    required: true
  }
  friends: [User],
  datetime {
    type: String,
    required: true
  }
});
```

Listing 5: WachedEvent Schema

```
var Group = new Schema({
  name: {
    type: String,
    required: true
  },
  admin: {
    type: User,
    required: true
  },
```

```
  users: [User]
});
```

Listing 6: Group Schema

```
var GroupJoinRequest = new Schema({
  requester: {
    type: User,
    required: true
  },
  group: {
    type: Group,
    required: true
  }
});
```

Listing 7: GroupJoinRequest Schema

2.4 Communication

Here we describe the messages for each one of the operations listed in Section 2.1.

/users

POST

username The username.
password The password.
firstname First name.
lastname Last name.

GET

name String to search for users first and last names.

/users/:id

GET

(empty)

PUT

username The username.
password The password.
firstname First name.
lastname Last name.

DELETE

(empty)

/invitation

POST

userid The user id.

GET

(empty)

/invitation/:id

DELETE

(empty)

/follow

POST

userid The user id.

GET

(empty)

/follow/:id

DELETE

(empty)

/friendship

GET

(empty)

/friendship

DELETE

(empty)

/friendshiprequest

userid The user id.

GET

(empty)

/friendshiprequest/:id

GET

(empty)

POST

accept Set to true.

DELETE

(empty)

/watching

GET

(empty)

POST

movieid The movie id.

friends A list of user ids.

/watching/:id

DELETE

(empty)

PUT

friends A list of user ids.

/watched

GET

(empty)

POST

movieid The movie id.

friends A list of user ids.

datetime When that movie was watched.

/watched/:id

GET

(empty)

PUT

friends A list of user ids.

datetime When that movie was watched.

DELETE

(empty)

/wanttowatch

GET

(empty)

POST

movieid The movie id.

/wanttowatch/:id

DELETE

(empty)

/dontwanttowatch

GET

(empty)

POST

movieid The movie id.

/dontwanttowatch/:id

DELETE

(empty)

/group

GET

name A name for searching for group names.

POST

name The group name.

/group/:id

GET

(empty)

PUT

name The group name.

DELETE

(empty)

/groupjoin

POST

groupid The group id.

GET

(empty)

/group/:id

DELETE

(empty)

3 Conclusions

This document described important architectural aspects in order to support the implementation of MyMovies website. In Section 2.1 we described all the endpoints on the server side, responsible for providing and receiving data from the client. In Section 2.2 we described the file structure for the client side web application and the mobile hybrid application. In Section 2.3 we present the database Schemas. Finally, Section 2.4 specified the messages exchanged between client side and server side.

References

[1] Wikipedia - rest. `https://en.wikipedia.org/wiki/Representational_state_transfer`. Accessed: 2017-04-29.