MovieFinder

A web application for the course DAT076 (2014)

Members

Peter Eliasson - 930322-2997, peter-eliasson@hotmail.com
Carl Jansson - 911108-4878, carjan91@gmail.com
Anton Hallin - 931012-3618, antonhallin3@gmail.com
John Johansson - 910404-0754, Johanssonjohn91@gmail.com

Overview

MovieFinder is a demo (as in not yet fully finished) web application for keeping track of and displaying local movie files. The application is built as a Spring Framework REST API with an AngularJS based application as the frontend.

The application watches some specified folders on the local computer, and stores media files in these folders in a database. The information is also extended by parsing the file names and querying third party media APIs for additional information about the media. Such as title, runtime, imdb rating, etc. The information is then displayed so that one can more easily see what movies one has available on the computer. Since the application exposes a web interface, one can also access these movies from different computers, or let any guests browse through your collection.

Permissions

Since the movies one has on ones computer might be sensitive information the application requires any user to sign in. We currently have two roles implemented;

- VIEWER - Allowed access to most read functionality of the application, such as browsing through movies/series on the local computer.
- ADMIN - Can do anything VIEWER can, and can also add and remove file paths that the application searches for media files in.

Use cases

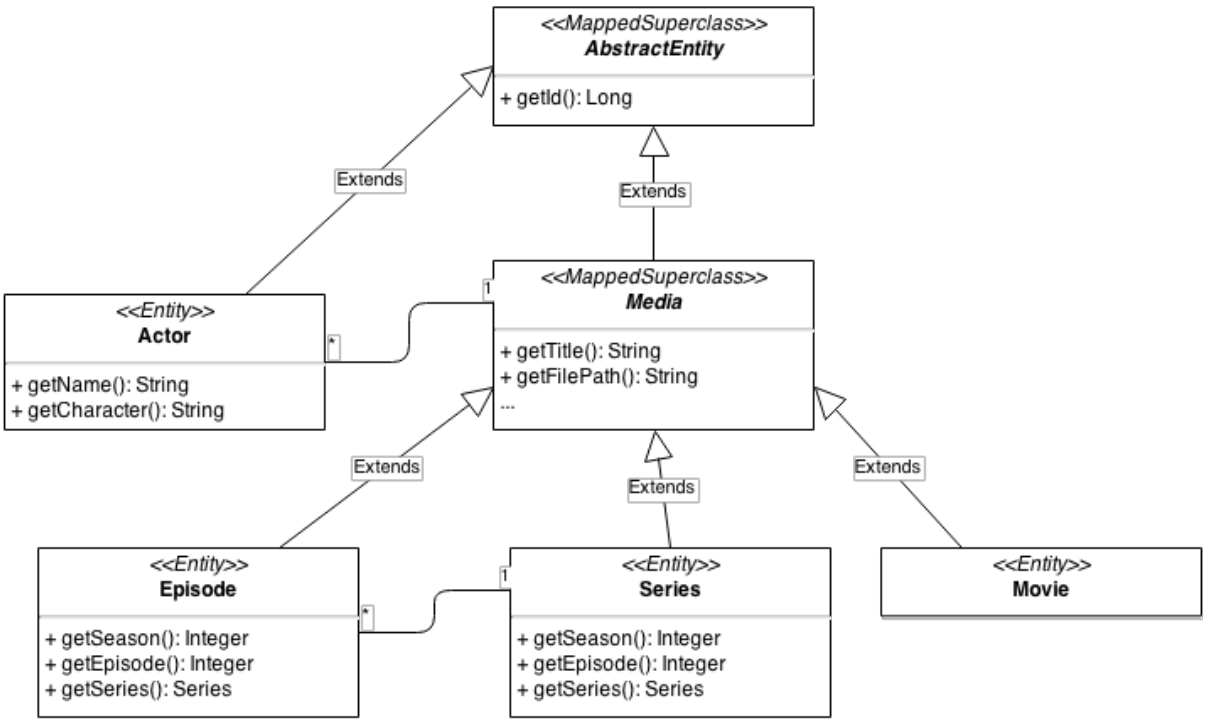
The user can;

- Sign in and sign out of the application.
- Sign in as different roles. (VIEWER, ADMIN)
- Get additional information about his/her media files. (file names are parsed and additional information fetched from media API).
- See a list of his/her movies. (currently limited to 25, paging not fully implemented)
- See a list of his/her series.
- See a list of episodes for a specified series.
- See a detailed view of movies and episodes, where additional information is presented.
- Stream some movie files directly from the details view, mp4 files should work.
- Filter the list of movies/series based on some criteria.
- Add additional file paths to scan, if logged in as ADMIN.
- Remove file paths added, if logged in as ADMIN.

Technical design

OO Model

We based our model around JPA entities and Spring Data repositories. The abstract Media class holds all shared properties between Movies, Series and Episodes. For the sake of readability not all its getters are listed in the model below.



Selected approach

As previously mentioned our app is designed using a service based approach. The backend exposes an API to the frontend. The API uses JSON for transferring data, and the routes are designed to try and follow REST best practises.

Physical Setup (Tiers)

Our application seems to be following the Three-tier architecture;

- Presentation tier
- Business Tier
- Data Tier

Components

Firstly we have a web server for serving our static files, we have mostly been using Apache Tomcat but we have also frequently tested our application running on Glassfish.

Then we are using Servlets to dynamically add some data to the index.html file used as the starting page for our angular application.

Below the Servlets we are using the Spring web framework and its various components to build the business tier of our application. Spring handles most of the communication between layers, for example it handles the mapping between controllers and their views.

We are also using a “plugin” called Spring-Security for some additional security related features (such as csrf-protection and security related http headers). Furthermore Jackson is used to provide Spring with a java to JSON mapper.

Within the Spring application we have a couple of services of our own that uses various third party libraries. Our media API fetcher uses Google’s gson library for parsing the responses, and Apache’s http libraries for making the requests.

For communication with the database we are using Spring-Data, which is a further abstraction of database access. It allows for using multiple types of databases (as in both standard SQL, NoSQL, graph based, etc.). It abstracts away most of the query building by automatically implementing our repository interfaces by parsing their method names. Below Spring-Data is Spring-Data-JPA which, as its name implies, is a specialisation of Spring-Data for using JPA based Entity classes. Finally we are using Hibernate as our JPA provider.

The database (server) we have been using is mostly DerbyDB, either as embedded or remote. We are also using HSQLDB in embedded mode for our tests.

Modules (packages)

All our modules are more or less part of the Spring application, as Spring handles most of the components already for us. Our packages;

- config
  - Configuration settings for Spring, Spring-Security, Spring-Data.
- controller
  - Spring handled controllers. Spring handles both the calling of and displaying the returned data.
- filter
  - Spring handled filters. Spring applies these filters based on the request url.
- initializer
  - Spring bootstrapping. Starts Spring and Spring-Data using the appropriate configurations.
- listener
- model
  - Models not connected to the database. Mostly data-only objects used for JSON to java mapping.
- persistence
  - Database entities and repositories related to these entities.
- service
  - Spring handled services.
- util
  - Generic helper classes.

Layered view of the application

As already stated, Spring makes up most of the Business Tier and also handles the connections to the other tiers. Below is a rough estimate of how some the various components are connected. The first two diagrams represents HTTP requests. The first for the index page resulting in the dynamically creation of the index.html view. The second for an API resource that relies on data from the database. The second diagram also shows the file system service, and how new media files are added to the database.