MovieRama

Introduction

The assignment for the developer position was to create a web application named *MovieRama* for sharing movies online. To this end, we leveraged the **Ruby-on-Rails** MVC framework to develop the forementioned application. The MovieRama database schema consists of the following three entities:

1. **User**: a user should be able to sign up and log in into the application to upload and express their opinions about movies. Users are not able to rate their own movies.
2. **Movie**: the movies that are uploaded to the system and rated by users.
3. **Reaction**: the reaction of a specific user to a specific movie, can be either "like" or "hate".

Database Schema Design

In order to implement the concept, we designed the database as depicted in the following schema:

- **Users**: have a one-to-many relationship to Movies, thus each movie has a foreign_key **'user_id'** to the Users table. Users have unique emails. The User model was generated through the use of the '***devise gem***' and not from scratch.
- **Movies**: each movie belongs to exactly one user. Furthermore, Movies have a one-to-many relationship to the Reactions table. Movies have unique **titles**.
- **Reactions**: each reaction belongs to exactly one user, and to exactly one movie. A reaction is uniquely defined by the ***(user_id, movie_id) tuple***, since a movie can have only a unique reaction type from a specific user.

The decision to store movie likes and hates in a single table or in two seperate tables (e.g., **Likes** table and **Hates** table) depends on many factors. We decided to use a single table named **Reactions** to have a more simplified schema and be more flexible since we store the reaction type of a user to a movie via the string field **reaction_type**. If we decide to introduce more reaction types to the application we can easily extend the corresponding enumerate that defines the supported reaction types to the application, without having to create new tables or modify the database schema. Also, we manage to avoid database joins since the information is in a single table. However, if our target were to display better performance for specific queries (e.g., queries related to likes or dislikes), seperate tables may be the better alternative.

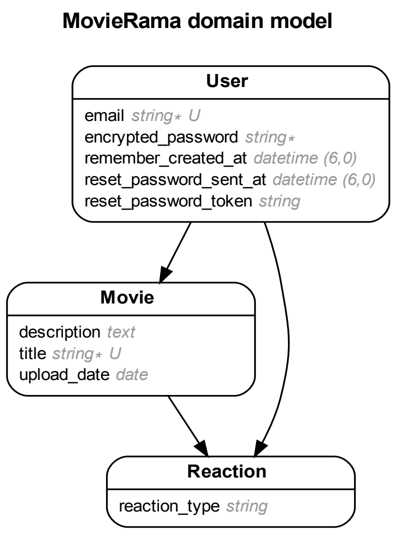
MovieRama domain model
User
email string* U
encrypted_password string*
remember_created_at datetime (6,0)
reset_password_sent_at datetime (6,0)
reset_password_token string
Movie
description text
title string* U
upload_date date
Reaction
reaction_type string

Figure: Entity-Relationship Diagram generated by ***erd gem.***

Running MovieRama

Our application was developed using Ruby on Rails version '7.1.3' and Ruby version '3.0.0' on **Ubuntu 18.04.6 LTS**. All dependencies of the application are listed in its **Gemfile**. Initially, the application was setup using the **Postgres** database, however during the end of the development process, we decided to switch to the **Sqlite3** database which is supported by default by Rails in order to simplify the delivery of the current assessment. The application was running successfully with Postgres prior to switching to Sqlite3.

(PROPOSED) To further simplify the delivery and testing of the application, we also developed the **Dockerfile.rails** docker file. In order to run the application through the Dockerfile, we run the following commands:

1. Open a terminal and change to '***movie_rama***' directory which contains the web application code and the Dockerfile
2. docker build -t movie_rama_web -f Dcokerfile.rails
3. docker run -it -p 3000:3000 movie_rama_web

Then, by opening a browser you visit localhost:3000 and the MovieRama application should be now visible in the browser.

(NOT PROPOSED) In case you do not wish to use the provided Dockerfile, all system dependencies must be installed manually (e.g., rails, ruby, npm, etc.). After installing al necessary dependencies, you need to run the following commands to start the application:

1. rake db:drop
2. rails db:create
3. rake db:empty
4. rails db:seed
5. bundle exec rspec spec
6. rails s

Notes:

- The '*rake db:empty*' command executes the code of ***'empty_database.rake'*** file which is is responsible for cleaning up the whole database.
- The *'rails db:seed'* command is responsible for populating the database with initial data for testing purposes.

Testing

As part of the implementation we include a number of tests regarding the application **models**, and **controllers**. In order to develop the tests we mainly relied on '***gem rspec-rails***' for setting up our application tests. The tests are located in the **spec** folder and are seperated into two folder, the **models** folder that contains validation tests about our models, and the **controllers folder** that contain code which tests the main functionalities of the MoviesController and ReactionsController. The tests are executed by running ***'bundle exec rspec spec'***.

P.S. A total of 18 tests were implemented. We did not follow a **Test Driven Development** process during the implementation of the current assignment, and implemented as many tests as possible during the time that was left until delivering the application.

Github: the application code is accessible through Github on https://github.com/vagosdim/MovieRama. In order to clone the repository you execute *'git clone https://github.com/vagosdim/MovieRama.git'*. For the purpose of this assignment only the main branch was used.