Universidade de Coimbra
Faculdade de Ciências e Tecnologias
Departamento de Engenharia Informática

Enterprise Application Integration

Webflix

Francisco José Rodrigues dos Santos 2015238068

Novembro, 2018

Index

1 Introduction

The main goals of this assignment is to gain familiarity with the web development of a three tier architecture, the utilization of an Object Relationship Mapping and the use of logging tools.

The secondary goal of this assignment is to develop Webflix, a that competitor the will outperform Netflix. Alike Netflix, Webflix users will pay a monthly subscription fee to watch multimedia content, and managers of the application, can manage the content available.

In this report, I will go through all the layers and explain the development proccess and the decisions made in each layer. The first section will explain the presentation tier, focusing on the webpage structure and password storage. The second section will approach the business tier and all the services implemented. The third section will explain the data tier, the entities created and their respective relationships. The last section will focus on the project structure and the external packages utilized in the development proccess.

2 Presentation Tier

The presentation tier in Ruby on Rails is the also called the view layer. The user will interact with this layer, generally, by clicking buttons and filling form fields that will trigger actions in another layer.

2.1 Webpage Structure

In this project, this layer is composed by a main template (*application.html.erb*) that will render every other template upon user request, in the form of routing. Some of the pages also implement partial templates, which are isolated components that can be included into any other page, such as forms or the navigation bar.

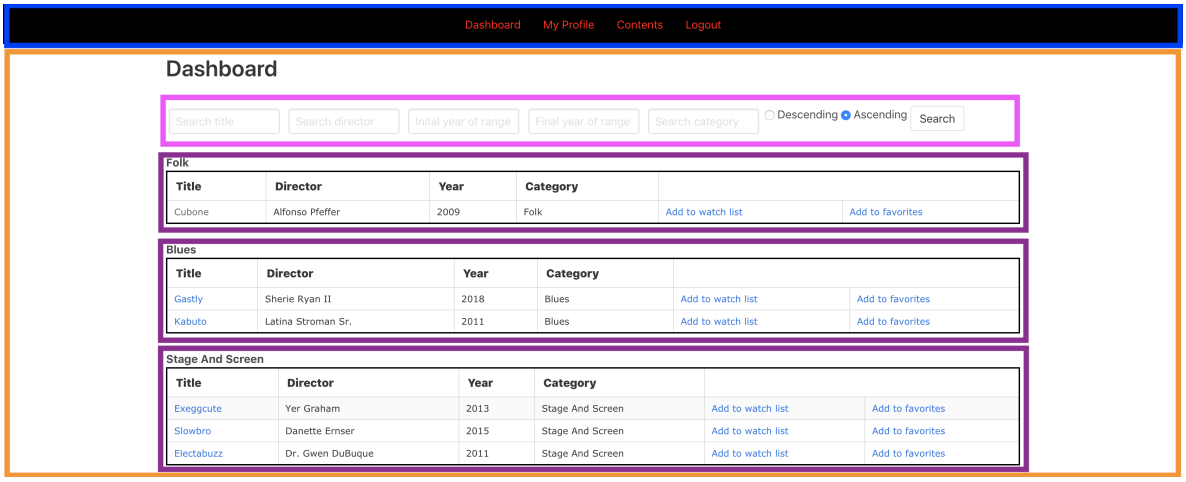

Figure 1: Webpage Structure Example

As you can see, this specific page, the dashboard, is divided into four separate components, one being the navbar, the other being the search bar, the third being the partial used to render content tables and the last component is the page content itself.

2.2 Password Storage

When it comes to the presentation layer, the user will also never be able to see his password anywhere on the web application. This means that the password data itself is never transferred to the presentation layer so it can never be a target to attacks listening in on the requests done by the user, except, of course, when the user inputs the password for the first time and every time it tries to change it.

As for the business layer, Ruby on Rails, with the help of an external package, knows to encode every password that it receives, technically speaking the package utilizes a *bcrypt* algorithm with the help of password salting. After that every password comparison is done with the hashed passwords and never with their raw text versions.

3 Business Tier

The business tier is where all business logic is implemented and all the business rules are established. In this project, the rules established concern entity management, meaning entity creation, entity updates, entity reading and entity destruction.

All of the services implemented, where done in the form of controllers, that will serve their data, gathered from the data layer, to the presentation layer.

3.1 Pages Controller

This controller acts as a general pages manager. Simply put, this controller manages the main pages for unregistered users, and the main page for registered users. It guards the access to the dashboard view from unregistered users, and fetches the data for the dashboard if the user is registered.

The dashboard method of this controller is used for search functionality in the application. When calling this method, if there are parameters in the URL, it will call the search method implemented in the content model of the data layer, making it query the database and return all of the objects it can find.

3.2 User Controller

This controller implements all the user CRUD methods while also protecting the user from accessing routes or casting methods for which he does not have access. It also implements method for changing the user's role, making him a manager or a regular user depending on the case.

3.3 Content Controller

This controller, alike the user controller, implements all the CRUD methods for the multimedia content in the application. It also implements a function that guards non-manager users from calling actions that affect the content on the website.

The index method of this controller also acts as the search functionality for the application, if upon redirecting to this route, there exist parameters in the URL, the method will call the search method from the data layer, making the respective layer query and return the objects found in the database.

3.4 User Content Controller

This controller acts as a manager for the relationship between the user and the content. It implements methods for adding and removing a movie to and from the watch list, and methods for adding and removing to and from favorite list.

3.5 Notes

All of these controller implement helper methods that guard the parameters being received by each request. Simply put, if a request is done with invalid parameters the method calling the helper will stop executing.

4 Data Tier

The data tier is the layer where access is provided to a persistent data storage (i.e entity-relational database) in order to query tables from said data storage. It is in this tier that Object-Relational Mapping (ORM) tools provide their functionalities.

In this project the ORM used is called Active Record, and follows the active record pattern architecture, that stores in-memory object data to relational databases.

4.1 Entity Relationship

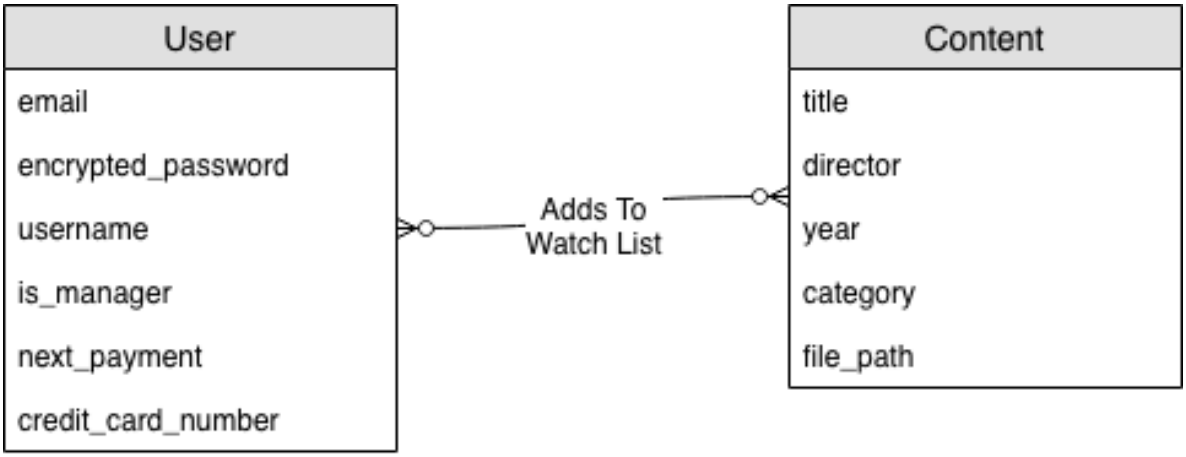

Figure 2: Entity Relationship Diagram

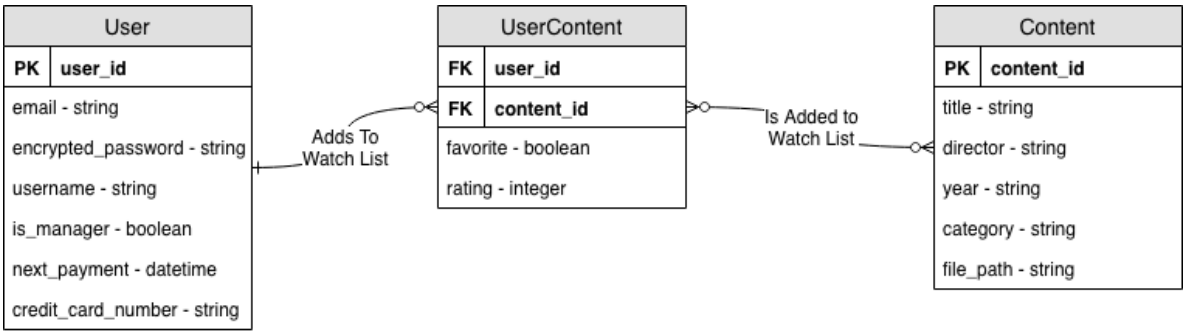

Figure 3: Physical Data Diagram

4.2 Entities

In the projects there are two entities, that take the name of models. The user model and the content model. These models are created and managed by Active Record, a Ruby on Rails default package, which acts an Object Relationship Mapping tool.

These two entities establish a Many-To-Many relationship between them, given that a user can add several multimedia content to its watch list, and that same content can be added by other users to their respective watch lists. This relationship types mandates that a "third" entity is created, the user_content. This new entity will manage all aspects of the relationship, including it's CRUD methods.

4.3 Notes

There was the possibility of using another entity called Manager, instead of the *is_manager* field actually in use, and make that class inherit from the User class. Said possibility wasn't implemented because even tough Ruby is an Object Oriented Programming Language, said paradigm isn't used when working with web technologies.

It is highly unlikely to find a Ruby on Rails project that makes use of polymorphism, and STI to define it's classes. Single Table Inheritance (STI), as it's described in Rails, is a very unorthodox way of defining tables, such that it's a common pattern to ignore this concept of implementation as it brings a lot of problems with it.

5 Project Management and Packaging

In this section we'll look at how the project is structured and what packages were used during the development process of this application.

The project structure closely resembles an *MVC* architecture, that's mostly due to the fact that Ruby on Rails follows the basic principles of said architecture and *RESTful* design. Although the assignment focus on the implementation of a three tier architecture, this architecture can be parallel to that. The models relate to the data tier, the controllers relate to the business tier and the views relate to the presentation tier.

The implementation also tries to closely follow the concept of Separation of Concerns to closely mimic a three tier architecture. This means that our models focus mostly on executing queries, our controllers

focus solely on serving our views, and our views focus only on user interaction and data visualization.

5.1 Project Structure

In the project root folder there are two folders that deserve attention. The first being the **db** folder, where all the migrations are stored and where you can edit said migrations.

The second folder worth mentioning is the **app** folder, where the main source code belongs. This folder has various sub-directories regarding the different concerns of the application. It has a directory for the views which concern the presentation layer, it has a directory for controllers which concern the business and a directory for models which concern the data layer of our application.

5.2 External Packages

In the project root folder there is a special file, the *Gemfile*, used to manage all the externals packages required by ruby to use rails. There a few packages that were used to ease the development process or implement some extra features. Out of all of the packages, there are some worth mentioning:

- **Devise** - This gem is used as a session manager, it provides all the helpers needed to login and logout a user, and mantain a logged in status during the use of the application.
- **FactoryBot** - This gem was used to fill the database with seeds, and provide class builders for the testing environment.
- **Faker** - This gem provides fake data to use in the class builders specifiedd above.
- **Byebug** - This gem was used in the development to add breakpoints to the code and help debug some issues.
- **Whenever** - This gem was used in the rake task that runs everyday in order to charge the users for their subscription.
- **Rspec** - This gem was used to implement unit testing on the models.

6 Conclusion

At first, this project gave me a broader understanding of three tier architecture, the work required by each layer, and the importance of the Separation of Concerns between each layer.

To conclude, this project allowed me to practice my understanding of Ruby on Rails, and, more specifically, learn good programming patterns within the Web Development paradigm. It allowed me to learn how to develop an application with the Separation of Concerns concept in mind to more thoroughly structure the architecture of my application.