InstaFlickr

Web applications DIT126
Gothenburg University

Thomé, Pontus — 870128-4898
pontusthome@gmail.com

Ottervad, Henry — 931123-6096
gussiihe@student.gu.se

Fritzon, Stefan — 860422-4611
stefan.fritzon@gmail.com

Högberg, Thomas — 931120-4458
blobber93@gmail.com

March 13, 2016

Contents

1 Overview

InstaFlickr, our web app, is inspired by two popular photo sharing apps: Instagram, a smartphone app; and Flickr, a web app. With InstaFlickr we wanted to combine select features from these applications. To use InstaFlickr, you have to create an account and set up a profile, which consists of your name, a profile description and a profile picture.

With InstaFlickr you can store and share single pictures, or you can store and share albums, which are groups of pictures. You can also explore what other users have uploaded to the site through a feed of pictures and albums, displayed in reverse chronological order. You may like and comment on other users pictures, and others can like and comment on your pictures.

If we were to further develop InstaFlickr, we would like to add more features, like the ability to follow users so that you only see their uploads in the feed, and functionality for collaborative albums multiple people can add pictures to. We would also like to features that would transform this into a real product, like email validation, improved security, admin control, ability to report offensive content, etc.

2 Use Cases

- **Create a profile**
  A user can create a profile with a unique user name and a password additional the user can give a email, name, description and a selected profile picture.
- **Edit a profile**
  In the *Settings* page a user can change their name, email, personal description and profile picture.
- **Sign in**
  The user can fill in a user name and connected password to sign in to the application, not if the user name is not registered or if the password is not correct.
- **Sign out**
  Once a user is signed in the *Sign Out* button in the menu can be clicked to sign the user out of the application.
- **Upload a picture**
  A signed in user can select a picture from the file system to upload it into the users profile.
- **Create an album**
  A signed in user can enter a unique album name to create a new album.
- **Upload a picture to an album**
  A signed in user can select a picture from the file system and then select an album to upload the picture into the album.
- **View uploaded pictures and albums**
  A signed in user can see all the pictures and albums that the user has uploaded in chronological order.

- **View other users uploaded pictures and albums**
  A signed in user can see all the pictures and albums that other user has uploaded in chronological order.
- **View the pictures of an album**
  A signed in user can see all pictures that belongs the album that the user clicked on.
- **View a single picture**
  A signed in user can click on a picture either on the *profile*, *feed* or *album* page to get to the *picture* page where the picture the user clicked on will be displayed in full size.
- **Like or remove like on a picture**
  A signed in user can click on the blue hearth button below the picture on the *picture* page to either like the picture or if the user has already liked the picture the like will be removed.
- **Comment on a picture**
  A signed in user can write a text to add a comment below the picture on the *picture* page.
- **Get help**
  A user can get help with using InstaFlickr by entering the *Help* page.

3 Technical Design

3.1 Model

The InstaFlickr model is written i Java. A class diagram of the model can be found in appendix 2.

The class **InstaFlickUser** is an entity that represents a registered user in our application. This entity contains the users sign in information, name, contact information and profile **Picture**. It also contains a list of the **Pictures** that the user uploads and a list of the **Albums** that the user creates. The **UserRegistry** class is a stateless DAO (Data Access Object) handles all the requests to the database that involves **InstaFlickUser** objects.

The class **Picture** is an entity that represents a image uploaded by a user. The image is stored on the file system of the web server, and the path the image location is stored in the **Picture** object together with the uploader, date, description. We choose to store the image on the file system instead of in the database because this would make the database very large, while the file system is made to handle storage of files like images. The **Picture** object also has a list of **Comments** that users has made on the image and **Likes** object that contains the users that has liked the image. The **PictureCatalogue** class is a stateless DAO handles all the requests to the database that involves **Picture** objects.

The class **Album** is an entity that represents an album created by a user. An album has a name, an uploader and a list of **Pictures** that a user has uploaded into the album. The **AlbumCatalogue** class is a stateless DAO that handles all the requests to the database that involves **Albums**.

The class **Comment** is an entity that represents an comment created by a user for an image. The comment contains the creator, the comment text and the time the comment was created.

The class **Likes** is an entity that is contained in a Picture and contains a list of the users that has liked the image that the **Picture** represents. The **LikesHandler** class is a stateless DAO that handles all the requests to the database that involves **Likes**.

The class **SessionHandler** handles the session for a user when they are sign in to the application.

The class **UserResource** provides RESTful web service for the model that involves **InstaFlickUsers** and **SessionHandler**.

The class **MediaResource** provides RESTful web service for the model that involves **Pictures**, **Albums**, **Comments** and **Likes**.

The class **InstaFlick** contains all the stateless DAOs.

3.2 Approach

InstaFlickr uses a RESTful approach, where access to the model and the database from the controllers of the view is provided through RESTful web services.

3.3 Physical Setup

The client, the Java model and the Apache Derby database runs on a GlassFish web server hosted by the local computer.

3.4 Components

3.4.1 Front-End

For the front-end we use AngularJS, Bootstrap and jQuery, as Bootstrap is dependent on jQuery. We chose this approach because we found it to be a fairly easy and flexible approach. With Boostrap we could easily design relatively good looking pages without spending a lot of time and with AngularJS we could get a pretty good decoupling from View and Controller, and get the very important connection to the back-end.

With AngularJS we created a different controller for each page that needed it. We then let the controller handle the functionality and data for that page. The controllers request information from our model through the RESTful web services that our model provides.

We extended AngularJS with two external libraries: ui-router, for more functionality regarding navigation and views; and ng-file-upload, for a more practical way to upload binary data.

3.4.2 Back-End

We ran our application on GlassFish 4.1 and used Derby as database. The database was generated with Java Persistance API (JPA). We have one Data Access Object for each of our entities that handle all the data traffic to, and from, our database. Testing our database was done using Arquillian.

Our RESTful web services were created in Jersey. We have two web service providers, one containing all services related to users, such as log in, log out, create profile and session handling, the other handles all the services related to media handling of our application such as getting pictures and albums and creating and getting likes and comments. As mentioned, these services are accessed by our front-end through AngularJS.

3.5 Modules

The back-end is divided into four sub sections: model, model.media, model.persistence and model.user.

3.5.1 model

This package contains two classes: ApplicationConfig and MainServlet. In ApplicationConfig we add the classes necessary for our back-end to function. MainServlet is used to force the database to be created.

3.5.2 model.media

The Media packages contains all the classes related to **Picture** which is the focus of the application. The class **Album** that can contain Pictures. And Picture itself contains **Comments** and **Likes**. **MediaResource** is also located in the media packages which supplies the RESTful web services related to the classes located in the media package.

3.5.3 model.persistence

This contains all the Data Access Objects (DAO), the object that handles all requests to the database. InstaFlickr has four DAOs **AlbumCatalogue**, **LikesHandler**, **PictureCatalogue** and **UserRegistry** each responsible for one entity. In the persistence module there is also the SessionHandler that saves the current session for logged in users.

3.5.4 model.user

This contains the User related classes InstaFlickUser, UserResource and UserWrapper. It is the classes responsible for the handling of users in the back-end, not counting the database. InstaFlickUser is the main class in the package. It is responsible for holding the information

about a user. Which is the information in its profile, the pictures, albums, likes and comments it made.

The UserResource is where the back-end receives the different queries from the front end. It handles the adding, removing and modifying of users. It handles the signing in and signing out of users and also verifies whether or not users are logged in or not.

UserWrapper was supposed to be used as a wrapper for the InstaFlickUser, to easily send the required information about users between the front-end and the back-end. In the end it was not needed but it still remains.

3.6 Layer View

Layered view can be found in the appendix: 4

We have 3 main layers, that is Presentation Layer, Logic Layer and Data layer. In the Presentation layer we have HTML with Bootstrap. The Logic layer we have a part from the Front-end which communicate with the back-end through REST. Also on the logical Layer we have a big part of the back-end. We have Web-resource, Model, Testing (Arquillian), persistence. On the Data Layer we have the databases and the server. The data is exchange through the persistence layer.

3.7 Domain Model

Domain model can be found in the appendix: 1 The Domain model is our initial diagram that contains our idea of the functionality and data of our application.

3.8 Database Diagram

Database diagram can be found in the appendix: 3

The main entities are Users, Pictures, Comments and Albums. The other entities are weak entities that are dependent on foreign keys.

User is the most dependent Entity because almost all entities are more or less depended on the User. Picture is the second most important entity. Likes, Comment and Album are depended on this entity. After this, Albums and Comments are important mostly because the weaker entities are depended on them.

Appendices

A Domain Model

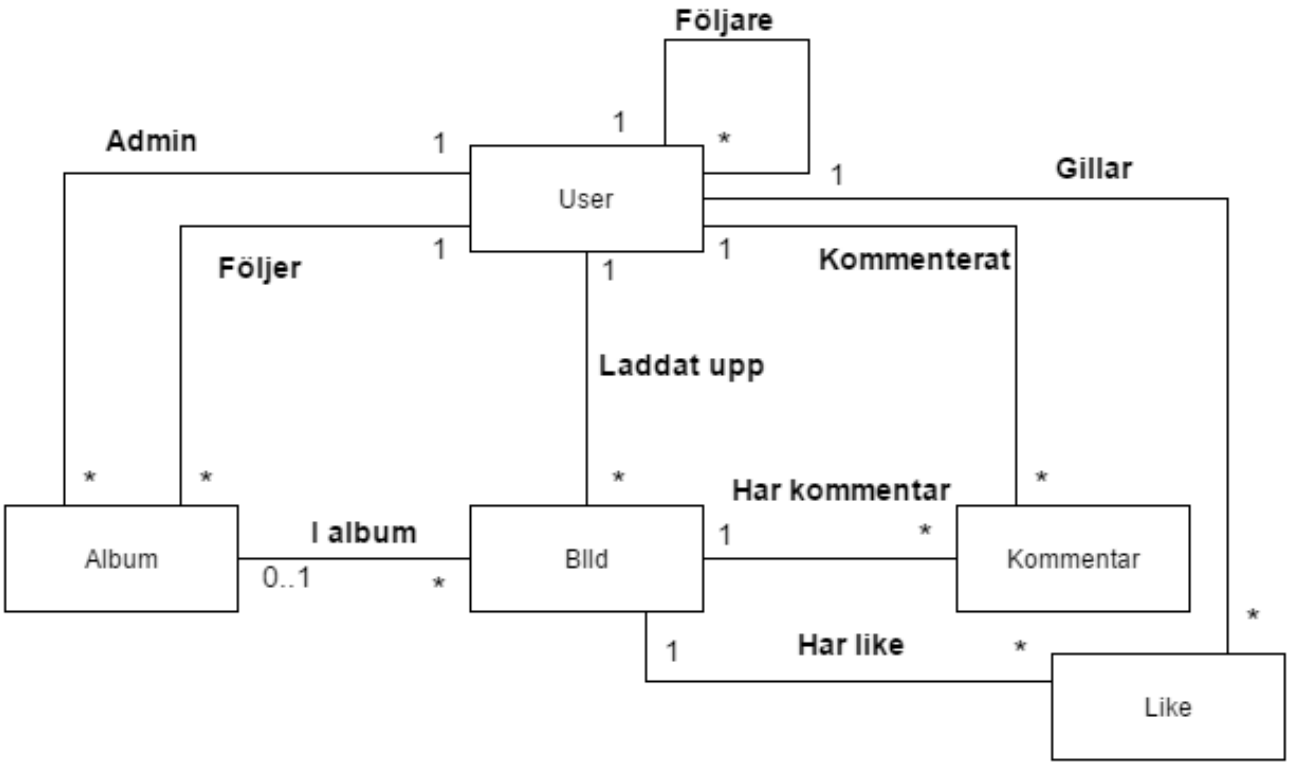

Figure 1: Domain model

B Class Diagram

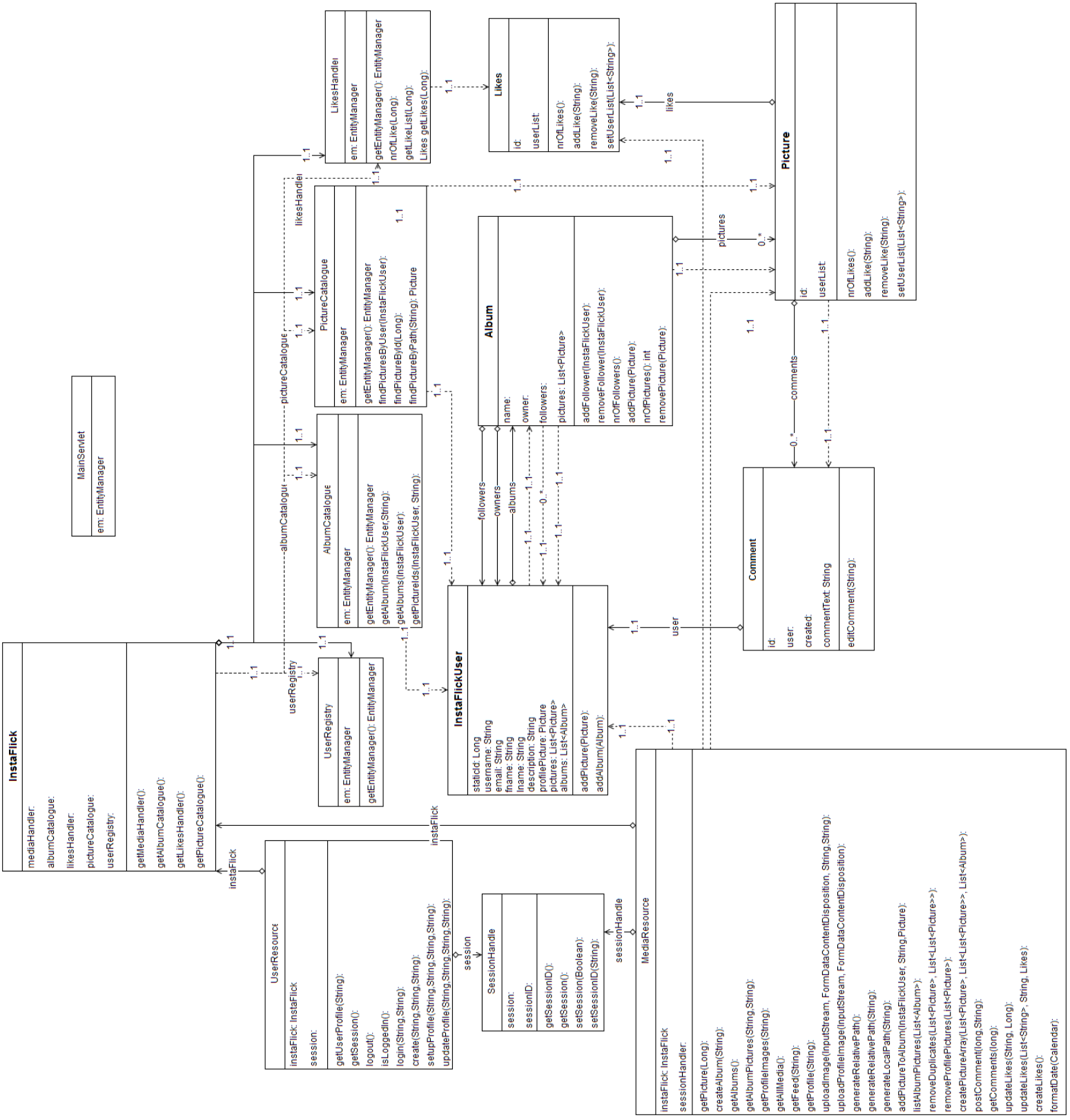

Figure 2: Class diagram

C Database Model

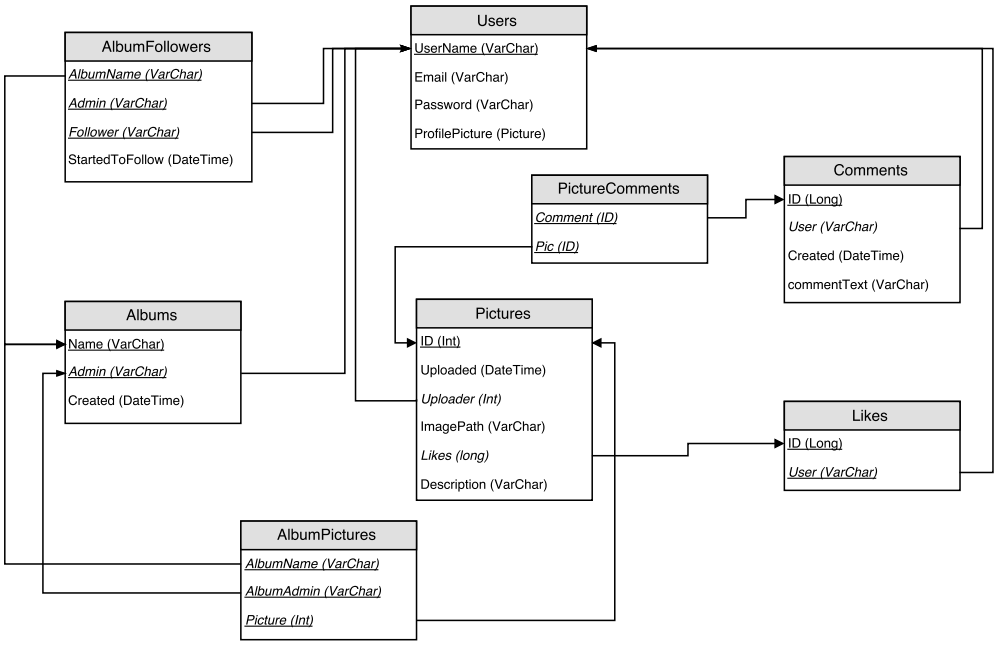

Figure 3: Database diagram

D Layer View

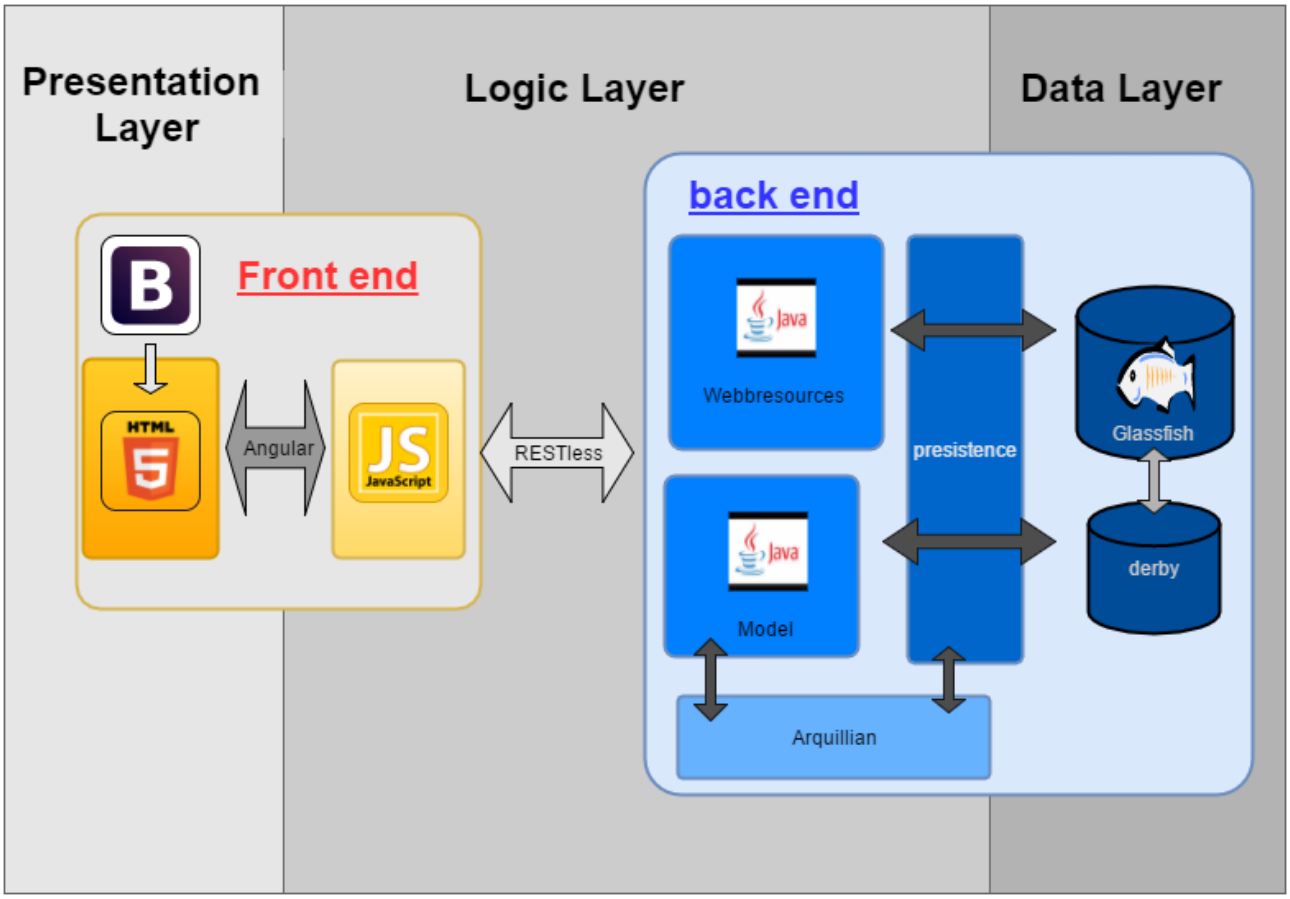

Figure 4: Layer View