Bookies App

Documentation

Table of Contents

[Bookies App 2](#_Toc445124528)

[Running the Project 2](#_Toc445124529)

[Setting up the environment 2](#_Toc445124530)

[Building the project 2](#_Toc445124531)

# Bookies App

Technologies used:

Back-end(For more info check build.gradle file):

* [Java](https://www.oracle.com/java/index.html)
* [Spring Boot](http://projects.spring.io/spring-boot/)
* [Hibernate](http://hibernate.org/)
* [MySql Database](https://www.mysql.com/)
* [Gradle](http://gradle.org/)
* [Jetty](https://eclipse.org/jetty/)

Front-end:

* [AngularJs](https://angularjs.org/)
* [Jquery](https://jquery.com/)
* [NodeJs](https://nodejs.org/en/)
* [GulpJs](http://gulpjs.com/)

# Running the Project

Before building and running the project we will have to set up the environment with the tools we’ll be utilizing.

## Setting up the environment

* [Install Java JDK 1.8](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)
* [Install Gradle](https://docs.gradle.org/current/userguide/installation.html)
* [Install NodeJs](https://nodejs.org/en/download/)
  + [Install GulpJs](https://github.com/gulpjs/gulp/blob/master/docs/getting-started.md)
* Setup a MySQL server and run the data.sql script provided in the DB folder of the project. This will instantiate the database and put in some data.

Recommended choices for quick set up:

* + Install MySQL server through [xampp](https://www.apachefriends.org/index.html) and manage the Database through [MySQL workbench](https://www.mysql.com/products/workbench/)

## Building/Running the project

Now that hopefully every dependency of the project has been downloaded it’s time to try running the project.

1. We’ll need to run the MySQL server. The settings for the MySQL database can be found in the **./src/main/resources/application.properties** file.

More specifically:

spring.datasource.url= jdbc:mysql://localhost:3306/books

spring.datasource.username=root

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

The above are the default DB(database) settings.

The **spring.datasource.url** specifies the JDBC driver which is mysql and the url of the database schema we’ll be using. Do not change the JDBC driver part, if you have to change something, change the default url to your database’s url.

The **spring.datasource.username** and **spring.datasource.password** are referring to the MySQL database’s user and his password.

Finally **spring.jpa.hibernate.ddl-auto** represents how we want the project to interact with the DB. The options are:

* validate: validate the schema, makes no changes to the database.
* update: update the schema.
* create: creates the schema, destroying previous data.
* create-drop: drop the schema at the end of the session.

1. We now have to run the **gradle bootRun** command through the command line, on the root folder of the application. If the settings are ok, it should run the application on an embedded instance of the Jetty server. Now we have the server of our application up and running, so what we still need is to build and run our front-end.
2. For the front-end what we need is:
   1. To run the **npm install** command from the root folder of the application
   2. And then to open command line as ADMIN and run the **gulp watch** command from the root folder of the application

(It’s important for this command to run as admin if running on windows!!!)

# More in depth analysis

Here we are going to explain how everything comes together and how the components and technologies interact with one another so it’s important to delve a little deeper on the technologies used in this project before reading about this part.

## Back-end

To better understand the server side structure of the app, we’ll have to start with the way Gradle, spring boot, hibernate and the database interact with one another.

(The entirety of the Back-end files,except for the build.gradle file, are located inside the **./src** folder)

### Gradle

1. Gradle is the building tool in charge of downloading all external libraries, compiling the code and serving the resulting output to an embedded Jetty server.

The code responsible for the build of the back-end is in the build.gradle file, located on the root folder of the app. Here spring boot related libraries are downloaded together with the hibernate library and other key libraries like the MySQL JDBC driver library.

### Spring Boot

1. Spring Boot is a java framework with a huge variety of extra plugins, with its use of the MVC model it helps keep your java code well organized and with it’s various plugins and preconfigured settings, it makes possible running the application with very little code needed on our part.

The java root folder is **./src/main/java/com/books**. Inside the **books** folder the subfolders are organized by the type and use of containing java files. Also the **books** folder has two java files, essential for running a spring boot app, these are **ServletInitializer.java** and **SpringBooksApplication.java**.

The subfolders with their respective roles are:

1. **Controllers (./src/main/java/com/books/controllers)**

The **MainController.java** makes sure the user is being served the correct page. It is a very minimal controller for most of the redirecting is being done by the front-end Javascript side, through angularJs, the only redirect the main controller needs to do is the first one, when a user opens the application in a browser.

In the **Rest (./src/main/java/com/books/controllers/rest)** subfolder is where the controllers, that the [RESTFUL API](http://stackoverflow.com/questions/671118/what-exactly-is-restful-programming) consists of, are located. These are:

1. The **UserController.java** is in charge of all the HTTP requests that have to do the user. In more detail, it deals with:
   1. Creating, logging in and retrieving **users**
   2. Creating, retrieving **userBooks**
   3. Creating, retrieving, update **userFriends**
   4. Creating, Rejecting and retrieving **BorrowedBooks**
   5. Retrieving all the user’s notifications(friend requests, borrow request)
2. The **BookController.java** is in charge of all the HTTP requests that have to do with the books, whether that means inserting new books in the database or getting the list of all the books.
3. The **BorrowedBookController.java** is mostly for keeping statistics, it’s not supposed to be used by the users, it retrieves the full history of **borrowedBooks**
4. **Entities (./src/main/java/com/books/entity)**

The **entities** folder contains java classes that through the use of hibernate annotations are being mapped to the database tables with the same name. These classes are called hibernate persistence entities and you can identify them by the [@Entity](http://www.tutorialspoint.com/hibernate/hibernate_annotations.htm) annotation above the class declaration. All the **entities** classes extend the **StartEntity.java** class which provides some default fields that every table has(The **StartEntity.java** is in the **helpers** subfolder). The **entities** are:

1. **Book.java**
2. **BookOfUser.java**
3. **BorrowedBook.java**
4. **User.java**
5. **UserFriend.java**
6. **Exceptions (./src/main/java/com/books/exceptions)**

The exceptions folder contains some exceptions that will be thrown by the app when certain database requests are resolved unsuccessfully.

1. **Repo (./src/main/java/com/books/repo)**

The repo folder contains classes that provide a variety of methods for retrieving/inserting or deleting data from the database tables.

1. **Security (./src/main/java/com/books/security)**

Though not active, here you can see some sample security settings that are going to be useful for the continuation of the project.

### MySQL Database

1. The MySQL is one of the most influential relational sql databases. We get data from the database with the help of hibernate as was mentioned previously. The database tables are as follows:
   1. **Book** holds all the data that correspond to the application’s books. It has a 1-N connection with **BookOfUser**.
   2. **BookOfUser** holds all the data that have to do with a specific instance of a book that a user owns. It has a N-1 connection with **User** table and N-1 with the **Book** table.
   3. **BorrowedBook** holds the data that have to do with the books a user has borrowed to or from other users. It has 2 N-1 connections with the **User** table(1 for the **owner** and 1 for the **borrower**) and a N-1 connection with the **Book** table.
   4. **User** holds the data regarding the applications users.
   5. **UserFriend** holds the data regarding the friend requests and the friends a user has.

