Bookify

[Image of Home Page]

Created By: Konstantinos Arkoulis, Georgios-Alexandros Kostas

For the purposes of Internet Application Technologies Class

University of Athens, 2022-23

Contents

[Installation Guide 3](#_Toc145293306)

[Run in the Cloud 3](#_Toc145293307)

[Run locally from pre-built files 3](#_Toc145293308)

[Build and Run Locally 5](#_Toc145293309)

[Run Setup Scripts 5](#_Toc145293310)

[Application Features 6](#_Toc145293311)

[Architecture Breakdown 6](#_Toc145293312)

[Database 6](#_Toc145293313)

[Backend/Server 6](#_Toc145293314)

[General Architecture 6](#_Toc145293315)

[Authentication 6](#_Toc145293316)

[Users 6](#_Toc145293317)

[Rooms 6](#_Toc145293318)

[Booking and Availability 6](#_Toc145293319)

[Search 6](#_Toc145293320)

[Reviews 6](#_Toc145293321)

[Recommendation System 6](#_Toc145293322)

[Administrator 6](#_Toc145293323)

[Images 6](#_Toc145293324)

[Messages 6](#_Toc145293325)

[Utility and Configuration 6](#_Toc145293326)

[Frontend/Website 7](#_Toc145293327)

[API Documentation 7](#_Toc145293328)

[Attributions 7](#_Toc145293329)

# Installation Guide

The following installation instructions assume a Linux Debian-based operating system. However, installing and running the application in a Windows environment should be a similar process. The main differences concern the installation of various programs which will require you to download and run the installer from the vendor’s official website. We will be trying to cover all the information required to install and run the application no matter what operating system you are on.

There are various ways to run the application:

1. From the provided link running in our own virtual machine in the cloud.
2. Downloading the pre-built .jar file and running the app from there.
3. Downloading the source code and building it locally.

In the following sections we will be providing detailed instructions for each of those options. If you are running the server locally, please refer to [this](#_Run_Setup_Scripts) subsection to check how you can use the provided setup scripts to preload some useful data to the database.

## Run in the Cloud

The easiest way to quickly and effortlessly test our application is by using this link: <https://bookify.duckdns.org>. This will be the same as running any other online application from the browser and - latency aside - you will be able to use it as if it was running on your local machine. Please note that the application is running on our own rented virtual machine in the cloud which has very limited resources. Therefore, we can not guarantee uptime or response time when using this method. We also ask that you do not overload the server with a lot of big image files, as storage space is already limited. However, this method should be enough to demonstrate all the functionality required by the assignment.

## Run locally from pre-built files

Another option is to use the provided pre-compiled .jar file to run the server locally. However, this option will require significantly more effort on your part as it is necessary to download various components and set up the database. Please note that you will need to have administrator rights to the machine you are using. **Please try to avoid copying and pasting the Linux commands directly on a terminal as the pdf format will probably affect some of the special characters.**

1. If you are using Linux, make sure the package manager is up to date with this command:

*sudo apt update*

1. Install the Java Development Kit (JDK) if not already installed. For this you can either use the installer provided in the official website or on Linux you can run the following command:

*sudo apt install openjdk-18-jdk*

1. Install MySQL Server either via the official website or with this command if you are on Linux:

*sudo apt install mysql-server*

1. To set up the database you need to open a MySQL console if you are on Windows or execute the following command on Linux: *sudo mysql.* You will then need to execute the following commands (same in both operating systems):
   1. *create database db\_bookify;*
   2. *create user ‘admin’@’localhost’ identified by ‘[password]’;* where *[password]* will be the password you want to set for the database user.
   3. *grant all privileges on \*.\* to ‘admin’@’localhost’;*
2. You can now close the MySQL console and use the terminal of your operating system to navigate to the build folder of the deliverable we provided. An example, that still requires you to provide the appropriate path, is this command: *cd C:/Users/[USER]/Documents/*bookify/build.
3. After you have **correctly** executed **all the steps** described above you can run the executable using this command:

java -jar bookify-0.0.1-SNAPSHOT.jar --upload.directory.root= *C:/Users/[USER]/Documents/*bookify --spring.datasource.password=[password]

The provided options are crucial to the correct execution of the application. --upload.directory.root defines a path to the parent folder the application can use to store the required data such as images, recommendation results etc. If the given path does not exist, the application will create it **if and only if** it has the required permissions to do so. As a result, please make sure you only provide a path the application will actually have access to write to.

The --spring.datasource.password option defines the password of the database user as created in step 4b. The app assumes the password ‘1234’ for the database user, so if your password is different than that you will need to set it using this option. Other options that may be useful are

* spring.datasource.username which defines the username of the database user (default: admin).
* server.port=8443 which defines the port the server will be listening to (default: 8443).

After successfully executing those instructions, you should have a Spring Boot application up and running in your terminal. It will take between 5-20 seconds to startup. Immediately after, it will start running the recommendation algorithm in the background if any rooms and users are available.

We have set up server to provide all the pages of the website to the browser when requested. Therefore, it is not required to create any additional static server to serve the website content. The only thing that should be required to use the app once the server is running is to open your browser and type the following into the address bar: <https://localhost:8443>.

**IMPORTANT:** due to our SSL certificate being self-signed most of the browsers will reject those requests. It is therefore necessary to add an exception to the browser for our website. If the browser shows you a message rejecting the request, click on the advanced options button and then click on proceed/add exception. If you do not manage to get the exception done this way, you will need to manually set it via your browser’s security settings.

## Build and Run Locally

It is also possible to compile and run the code yourself on your local machine. The steps are similar to those of the previous section, but you will now need to execute the following steps instead of step 5 to build the project locally:

1. Install Maven (the build tool). In Linux this can be done with this command: *sudo apt install maven.*
2. In the terminal of your operating system, navigate to the bookifySystem subfolder in the provided deliverable. An example of this command, that still requires you to provide the correct path, is the following: *cd C:/Users/[USER]/Documents/*bookify/bookifySystem.
3. Use the following command to build and run the project: *mvn spring-boot:run.* Please note that the first time this command is executed, Maven may need some time to download all the required dependencies and then compile and run the project. When it is done you should have a Spring Application running in your terminal. Refer to the previous section for instructions on how to use the app from your browser.
4. Other useful Maven commands include:
   1. *mvn clean:* cleans up all the build files.
   2. *mvn package:* packages the application into a .jar file that can be easily distributed, deployed or moved around.

## Run Setup Scripts

When running the application locally on your machine for the first time, the database will contain no useful data such as rooms and users, apart from the preinstalled admin user. We have created a python script to fill in the database with reviews, users and rooms from the dataset that came with the assignment.

**CAUTION: This script will delete all existing users, rooms, reviews and availability-related information from the database. Make sure to use this script BEFORE you start using the app to avoid loss of data!**

To use this script, you will need to follow these instructions:

1. Install Python 3 via the official website or using this Linux command: *sudo apt install python3-pip.* Make sure that Python is also added to PATH if you are using Windows.
2. Install the required package to allow python to connect to the database with this command:

*pip install mysql-connector-python*

1. Open the populateDatabase.py script in any text editor and scroll down to the main function. At the start of this function, you will need to edit the database credentials to match those you set up when creating the database. The default host is “localhost”, the default username “admin”, default password is “1234” and the database username is “db\_bookify”. If any of these are different to your settings, you will need to change them in the script and save any modifications.
2. Run the server application locally at least once following the instructions in the previous sections so that the spring application will create all the necessary tables in the database. Wait for the startup process to finish before you run the script. The tables are created the first time the application runs, so if you have already started the server, you can skip this step.
3. In the terminal of your operating system navigate to the scripts subfolder containing the python script along with the required datasets. Execute the script with the following command:

*python populateDatabase.py*

Depending on the python version you installed, you may need to start the command with *python3* instead of *python*.

Depending on how fast your machine is, this script should need a couple of minutes to load all the necessary data. When it is done, you will find that the website now includes about 2500 rooms and 32000 reviews associated with them.

# Application Features

# Architecture Breakdown

## Database

## Backend/Server

In the following subsections we will be briefly presenting the various subsystems of the backend part of our application along with the main design considerations that led us to those specific implementations. We will also be exploring the limitations of various components of the backend.

### General Architecture

The backend code can be found in the /bookifySystem subfolder of the deliverable and is organized into packages. Each main subsystem or feature is contained in its own package, so that all the classes and code related to a given feature are logically organized and also close together. There are 4 main types of classes that can be found in our codebase:

* Data Classes and DTOs: these classes define various entities for our database that are automatically converted to database tables and relations using the Hibernate Object Relational Mapping (ORM) Framework. They may also define classes called Data Transfer Objects (DTOs) that are used to conveniently pass a lot of data (such as the registration information of the user) around or capture this data from incoming HTTP requests.
* Repositories: our repositories are extensions of the classes provided by Hibernate and are used to provide Create, Read, Update and Delete (CRUD) operations on the database tables to the rest of the program, thus removing the need to write SQL calls for most of the required operations. However, they can easily be extended to run custom queries in the database in an SQL-like language (HQL) for more complex operations. Each database entity needs to have its own repository associated with it.
* Services: these are the classes that handle the business logic of the application. They are responsible for carrying out all the steps required to perform a given operation. Such steps include but are not limited to doing necessary checks on the data, asking the repositories to perform some subset of the CRUD operations, handling errors and much more.
* Controllers: these classes provide access to the application’s API. They are responsible for defining the various API endpoints, catching incoming requests, passing the request for a given operation down to the appropriate service and returning the result in the form of an HTTP message. They also catch various exceptions thrown by the services and return the appropriate HTTP response code for each. Essentially the controllers define the interface of our app to the outside world.

We make use of Spring Boot’s automatic dependency injection system to make the setup of our application as effortless as possible. The process of handling any given request can briefly be described as follows: the controller corresponding to the endpoint we hit catches the incoming HTTP requests along with any query parameters, path variables or body arguments. It passes all the necessary information down to the corresponding service responsible for handling this operation. The service will make all the required checks to ensure the operation can be performed and that the user has the required access to perform said operation. It then performs all the required logic sometimes making use of operations provided by other services, communicates with the corresponding repositories for any database-related operations, checks for errors and passes the response back to the controller. The controller then sends the response back in an appropriate HTTP message.

### Authentication

### Users

### Rooms

### Booking and Availability

### Search

### Reviews

### Recommendation System

The recommendation system, contained in the *recommendation* package, is responsible for providing a predefined number of rooms that may be relevant or of interest to the user. As required by the specification, the recommender is based on the [Matrix Factorization](https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems)) algorithm, implemented using [Gradient Descent](https://en.wikipedia.org/wiki/Gradient_descent). Before diving deeper into the implementation details of the recommendation system it is worth examining closely the following problems:

* Providing recommendations to unauthenticated users: to provide recommendations to users that have not signed up or logged in to the app we would need to find a way to track their behavior while using our website. Although this is technically possible using cookies, the privacy concerns and the implementation complexity made us decide against pursuing such an option. Instead, we decided to provide the rooms with the highest rating (tie-break using number of reviews) as recommendations to the unauthenticated user.
* Can the recommendation algorithm run quickly enough to provide suggested rooms with an acceptable amount of delay to the user? The simple answer to this question is no. No matter how much we optimize the code or tinker with the learning parameters, there is no way the gradient descent part of the algorithm will run in an acceptable amount of time (about 2-3 seconds ignoring any other delays

To solve the second of those problems we decided to run the factorization part of the algorithm, which makes up the bulk of the work, periodically as a background task. For demonstration purposes we have set up our Spring Backend to call the matrix factorization routine every 30 minutes and do 500 iterations each time, which should take about 3 minutes to complete.

When the algorithm is done it saves the user and item matrices, along with 2 dictionaries that map user and room IDs to their corresponding row in the matrix, on disk for later use. Whenever the server needs to suggest rooms for a given user, it loads up the latest algorithm results from disk and runs the necessary calculations to return the most relevant rooms. This operation can be performed fast enough for the user to not notice any significant delay, with the small penalty of the recommendations being slightly out of date. However, there are a few edge cases we need to consider:

* The user has signed up to the application after the latest recommendation run.
* The recommendation algorithm, that is also executed on startup, is not yet finished.
* Some I/O operation fails when trying to load the algorithm results from disk.

Our solution to those problems was to simply return the top-rated rooms when something goes wrong. This way the user will never be left without any room to check out in the home page.

### Administrator

### Images

### Messages

### Utilities and Configuration

## Frontend/Website

# API Documentation

# Attributions