Flight Search App: Reflections

Ivan Novasak

Southern New Hampshire University

IT 634: Distributed Application Development

Ace Gayhart

2024 September 29

# Flight Search App: Reflections

The purpose of this paper is to describe the challenges the author experienced in developing a Java-based web flight search application, how he overcame them, how development was done, what methods were used, and what data schema were used, as well as why.

**Challenges**

The primary challenges the author faced with developing the app involved learning how to use Spring Boot as well as the security aspects. Regarding Spring Boot, the main difficulties involved the differences between the Model/View/Controller (MVC) setup Spring Boot uses vs ‘standard’ Java with objects and methods. The documentation, most notably the *Spring Boot Cookbook* book by Alex Antonov from 2015 (Antonov, 2015) and some of the videos by SNHU Media, were a bit out of date, so it was a challenge finding and using the correct versions of the software. Another challenge the author faced concerned the SQL database. Initially, it was not working on the author’s desktop computer but Professor Gayhart confirmed by email that the database linking worked fine on their system. The biggest overall issue the author faced concerned the security setup; the author followed the advice on the referenced web page from Module 8 (Broadcom, n.d.) and the video (SNHU Media, 2021b). The final challenge was how best to arrange the web page, given there are an infinite number of ways to arrange a web page depending on the style and audience. In this area, one challenge the author faced was finding a suitable background that both looked like it was appealing to someone looking to fly and did not take up too much hard drive space (under 250 KB).

So, to summarise, the author’s challenges were:

* Understanding Model/View/Controller in Spring Boot
* Getting and using up-to-date versions of the software to follow along with the videos and web pages with steps
* How to incorporate the SQL database into the app
* Security handling
* Web page layout and background size.

**Addressed Challenges**

For understanding MVC, the videos in (SNHU Media, 2021a/b) certainly helped the author lay out the setup of the program and where each Java class file will go. The author understands that the model is where aspects of a flight are stored, the view is for the user-facing parts, which are the web pages, and the controller is more of a backend area where security and the database handling take place.

To acquire the current versions of the software, the author looked on the SNHU Brightspace discussion board for posts about the software. It turned out that <<https://start.spring.io/>> is still the correct place to get both Spring Boot and the dependencies needed to make the web app.

To incorporate the SQL database, the author spoke to Professor via email about how to best incorporate it, and was given the advice to all the following lines into the **application.properties** file which is stored in the **/src/main/resources** directory of the app:

spring.jpa.hibernate.ddl-auto=update

spring.datasource.url=jdbc:mysql://localhost:3306/your\_database\_name?createDatabaseIfNotExist=true

A description of creating the SQL database is in another section of this document.

Laying out the web page should be under a responsive design such as mentioned in (Frain, 2015, pp 5–11) and Bootstrap (SNHU Media, 2021c) will be important so that users on all types of devices - not just computers - can access the flights web app. The author was inspired by the recommended layout of separating the HTML5 code file from the JavaScript and CSS3 files mentioned by Paul J. Williams (Williams, 2020) and in this course’s materials. The author preferred a simple layout and got inspiration from his peers in the critique modules to make a flight search design that has as little that could confuse the average user as possible. The author chose a background created by Tananuru Kummaru that gives off a neutral but also ‘fly the friendly skies’ feeling as referenced in (Kummaru, n.d.). One issue with the background that Professor Gayhart informed the author about and instructed how to fix was that the image was too large (around 18 MB). Professor Gayhart sent a link to a smaller file size version on the same page (Kummaru, n.d.) to use in the app that was way smaller but the same quality to the eye: 61 KB.

Another challenge discovered concerned imports, because some of the guides using older versions said to use **javax.\*** as opposed to **jakarta.\*** in the import statements. The author had Professor Gayhart review the code to confirm the imports needed to be updated due to changes in how the imported features were supported in current versions of Java.

For security, the author modelled the approach after what Broadcom did in (Broadcom, n.d.), where extra web pages are made for the user to log in and be greeted into the system. This portion of the app requires 2 Spring Boot dependencies: Security and Thymeleaf. Security is for handling the login/password matching system and Thymeleaf is for displaying the login web page and accepting the user’s credential entries via a login button. The system is similar to an earlier login system developed by the author for an earlier task given by Professor Gayhart which required the user to have the ability to both create an account and login to get a greeting.

**Development Explanation**

According to the course, the sequence covered for developing this app was:

1. Make a web page using HTML5, JavaScript, and CSS3 with a form for the user to enter the desired date of travel, origin, destination, date, and number of passengers and submit the data using a JavaScript button. Save each file as separate files. Find a suitable graphic that evokes nice skies / flying that is under 100 KB and save this into the same directory as the HTML, CSS, and JavaScript files. In the CSS file, the choice of the SimSun and Serif fonts are from the author’s personal preference, and may be changed to another highly visible font if desired.
2. Use the <<https://start.spring.io/>> website and leave the settings at defaults (Maven, whatever versions it is already set to), enter the desired app name into the Artifact and Name boxes, and add the dependencies Web (for general web access), Thymeleaf (for the login web page), JPA (for database support), and Security. Click the Generate button and save the ZIP file to the working directory of where the program is being written (currently, wherever the web page and associated files are being stored).
3. Unzip the file that was downloaded from Start.Spring.io. Move the 3 web page files and saved background picture into the /src/main/resources/static/ subdirectory that is inside the unzipped file from Spring.
4. Submit a draft of the work done so far to the designated peer discussion area for first draft critique. A few days later, critique peers’ work and draw on each others’ experience.
5. Create the SQL database modelled after the one shown in the video by (SNHU Media, 2021d). Elements to be tracked are the airline, flight number, origin, destination, departure date, departure time, arrival date, arrival time, number of travellers, and the price. Name the SQL file **flights.sql** and save it in the **/src/main/resources/** subdirectory.
6. Create the following directories in the app’s root directory:

* controller
* model
* repository
* service

1. Inside the **controller** subdirectory, create a Java class file called **FlightController.java** that will have 2 functions: get a mapping of all flights from the list as well as get a mapping of flights based on a search of original and destination. A **UserController.java** file is to be created which will handle user registration and authentication. This file is responsible for displaying the messages to the user indicating registration success/failure, or invalid credentials. A **FlightBookingController.java** file will also reside here which is for calling methods to book flights and getting a user’s existing flight bookings.
2. Inside the **model** subdirectory, create a Java class file called **Flight.java** that stores characteristics about an individual flight. These are typically the same as the attributes/columns in the SQL database. Create private fields named for each attribute and any other relevant ones specified in the database or requirements. Next, create accessor and mutator functions for each private field so the app can access the fields via encapsulated functions. Remember to use the Spring Boot @Entity and @Table entries for the private fields as these will interact with the database. Similar Java classes other other relevant objects will be created in the same manner: **FlightBooking.java**, which is for storing attributes an individual booking needs (username, date, origin, destination); **User.java**, which stores information about a user’s attributes; and **XMLUtils.java**, which contains the utilities the web portion needs for the XML file that stores the users and their credentials. There is another class called **UsersWrapper.java**, which is for getting and setting the users in a list. It is for operations that involve all users, such as displaying all the users registered, etc.
3. Inside the **repository** subdirectory, create a Java class file called **FlightRepository** that is an interface that extends the JPA Repository. This class will store the flights from the database in a repository list based on the origin and destination to search for later on. A similar second class called **FlightBookingRepository.java** will also be created here for storing flight bookings.
4. In the **service** subdirectory, create a Java class called **FlightService** that will have functions that use the recently created repository for searching for and getting all flights in the repository. Another class, called **FlightBookingService.java**, is for creating new flight bookings for users, storing them, and getting a specific user’s existing bookings from a list. The User Service, defined in a class called **UserService.java**, is for authenticating users and registering new ones. This is similar to the earlier **UserController.java**, except that controller class was more for displaying the messages involving whether the username and password was valid.
5. The Main method for the program should stored in a Java file called **FlightsearchApplication.java** and contain the following code:

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class FlightsearchApplication {

public static void main(String[] args) {

SpringApplication.run(FlightsearchApplication.class, args);

}

}

**Method Defense**

The author chose an SQL database to use for the flight booking app because the entities are mostly text-based, require strong data consistency, relationships between the entities and adherence to ‘ACID’ (Atomicity, Consistency, Isolation, and Durability) principles. Jahid Momin of LinkedIn, states that other types of databases, such as NoSQL and graph databases, are better orientated to multimedia applications and social networks, as well as ‘large amounts of unstructured data’, all of which are outside the scope of the flight booking app (Momin, 2023). Note that even though a portion of the app has some multimedia-like data, such as the background image and CSS for font colours/styles, these files are not required to be inside the database because they are unchanging single files and the database is only being used to track the text-based flight data. The flight data is also not unstructured - all flights must have a specific origin, destination, departure date/time, arrival date/time, and number of passengers onboard - a very structured layout. Hence the choice of an SQL implementation.

**Data Defense**

The database schema the author decided to choose was the following (in Table 1):

**Table 1**

*SQL database schema for the flight search and booking app*

| **Attribute** | **Description** | **Data Type** | **Domain** | **Primary Key?** | **Foreign Key?** | **Not Null?** |
| --- | --- | --- | --- | --- | --- | --- |
| flightNr | The flight number assigned by the airline | VARCHAR | 16 Characters | YES | NO | YES |
| airline | The airline name | VARCHAR | 100 Characters | NO | NO | YES |
| origin | The origin city/town of the flight | VARCHAR | 100 Characters | NO | NO | YES |
| destination | The destination city/town of the flight | VARCHAR | 100 Characters | NO | NO | YES |
| departureDate | The departure date in yyyy-mm-dd format | DATE | >1903-01-01 & <9999-12-31 | NO | NO | YES |
| departureTime | The departure time in 24-hour HH:mm format | CHAR | 5 Characters | NO | NO | YES |
| arrivalDate | The arrival date in yyyy-mm-dd format | DATE | >1903-01-01 & <9999-12-31 | NO | NO | YES |
| arrivalTime | The arrival time in 24-hour HH:mm format | CHAR | 5 Characters | NO | NO | YES |
| nrTravellers | Number of travellers on the flight | INT | >0 | NO | NO | YES |
| price | The price of the flight, in US dollars | DOUBLE | >= 0.00 | NO | NO | YES |

The VARCHAR, CHAR, and DATE types are intended to be used as strings for use in the Java app. Domains were chosen out of what is reasonable, for instance prices and number of travellers can never be negative, but $0.00 should be an allowed price to account for free flights. The year 1903 was chosen as the earliest date for flights in the database because of the plane being invented by the Wright brothers in 1903 (Crouch, 2024). The maximum date of 9999-12-31 was chosen to ensure no-one enters a year with more than 4 digits.

# References

Antonov, A. (2015). *Spring Boot Cookbook*. Packt Publishing.

Broadcom. (n.d.). *Getting started | Securing a web application*. Broadcom Inc. <https://spring.io/guides/gs/securing-web>

Crouch, T. D. (2024, September 13). ‘Wright brothers’. *Encyclopedia Britannica*. <https://www.britannica.com/biography/Wright-brothers>

Frain, B. (2015). *Responsive Web Design with HTML5 and CSS3 : Learn the HTML5 and CSS3 You Need to Help You Design Responsive and Future-proof Websites That Meet the Demands of Modern Web Users*, Second edition. Packt Publishing.

Kummaru, T. (n.d.). *The sky at sunset*. Vecteezy. <https://www.vecteezy.com/photo/2192615-dramatic-panorama-sky-with-cloud-on-sunrise-and-sunset-time-panoramic-image>

Momin, J. (2023, February 27). *Choosing the Right Database for Your Spring Boot Application*. LinkedIn Corporation. <https://www.linkedin.com/pulse/choosing-right-database-your-spring-boot-application-jahid-momin/>

SNHU Media. (2021a, March 24). *Introduction to SpringBoot MVC* [Video]. YouTube. <https://www.youtube.com/watch?v=Q5raGLCJ_Ac>

SNHU Media. (2021b, March 24). *Spring Boot Basic Security* [Video]. YouTube. <https://www.youtube.com/watch?v=czgwdIWsBCA>

SNHU Media. (2021c, March 24). *Add navigation bar using Bootstrap* [Video]. YouTube. <https://www.youtube.com/watch?v=Vsw0bAlxb-0>

SNHU Media. (2021d, March 24). *CRUD Operations* [Video]. YouTube. <https://www.youtube.com/watch?v=8qKNujjhMIU>

Williams, P. J. (2020, September 23). *Link to CSS and JavaScript in an HTML file*. DEV Community. <https://dev.to/pauljwil/linking-to-css-and-javascript-in-an-html-file-306m>