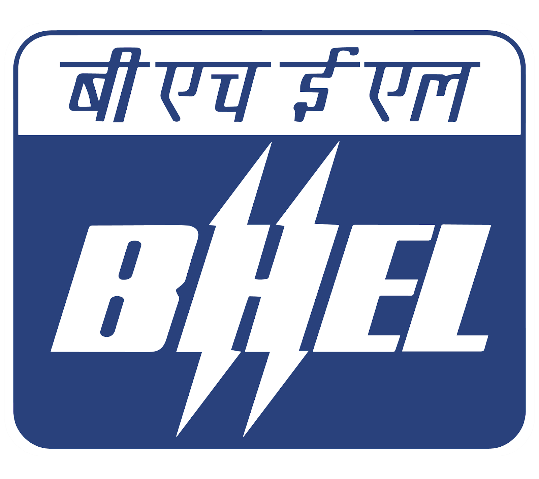
**Bharat Heavy Electricals Limited  
Haridwar**



**Vocational Training (2019)**

**Project Report**

**Submitted By:**

Raghav Kandpal

Reg. No. 201900873

Branch: Computer Science

G.B. Pant University of  
Agriculture & Technology, Pantnagar

**ACKNOWLEDGEMENT**

The internship opportunity I had with Bharat Heavy Electricals Limited, Haridwar was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it.

It is my radiant sentiment to place on record my best regards and deepest sense of gratitude to Mr. Sandeep Kumar for his careful and precious guidance which were extremely valuable for my study both theoretically and practically.

I perceive as this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired career objectives.

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **TITLE** | | **Page No.** |
| 1 | INTRODUCTION | | 3 |
| 2 | Technologies Used | Node.js | 4 |
| MongoDB |
| 3 | Node Dependencies | package.json | 5 |
| Dependencies | 5-8 |
| 4 | APPLICATION | app.js | 9 |
| Index Route | 10 |
| About Route | 10-11 |
| 5 | Application Routes | | 11 |
| 6 | USER Route | User Model | 12 |
| Register Route | 13 |
| Login Route | 14 |
| 7 | BOOKING Route | Flight Model | 15 |
| Book Form | 15-16 |
| Booking Index | 16-18 |
| Book Model | 18 |
| Booking History | 19-20 |
| 8 | ADMIN Route | Admin Login | 21 |
| User Listing | 22 |
| All Bookings | 23 |
| Create Flight | 23-24 |
| Flight Index | 24-25 |
| 9 | Front-End Elements | Bootstrap Navbar | 26 |
| Flash Messages |
| 10 | CONCLUSION | | 27 |

**INTRODUCTION**

The objective of this project is to create an online ticket booking system using JavaScript.

Therefore, using the JavaScript framework called Node.js and the NoSQL database known as MongoDB, the final product is a web application that provides easy and quick services to the user. The application a flight service is called Book-It.

Using an authenticated user system that allows easy login and registration, Book-It lets the user find and book flights instantly. The authentication is handled using Passport.js, a JavaScript framework designed to work with Node.

The code is written entirely on Visual Studio Code, a code editor by Microsoft and has been tested on Google Chrome (version 75.0.3770.100) and Microsoft Edge (version 42.17134.1.0).

Databases are handled using Mongoose, an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.’

The front-end is handled using Handlebars, an HTML templating engine that is both, powerful and easy to use and the code is written using Bootstrap 4 notation.

Other frameworks used are included within the ‘package.json’ file and shall be discussed later.

The application itself is a version 1.0.0, therefore may contain some bugs and issues. Further versions will look towards resolving these issues and adding new functions as well as updating existing ones.

**TECHNOLOGIES USED**

**Node.js**

****

Version 10.15.1

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user’s web browser. Consequently, Node.js represents a ‘JavaScript everywhere’ paradigm, unifying web application development around a single programming language, rather than different languages for server- and client-side scripts.

**MongoDB**



Version 4.0.6

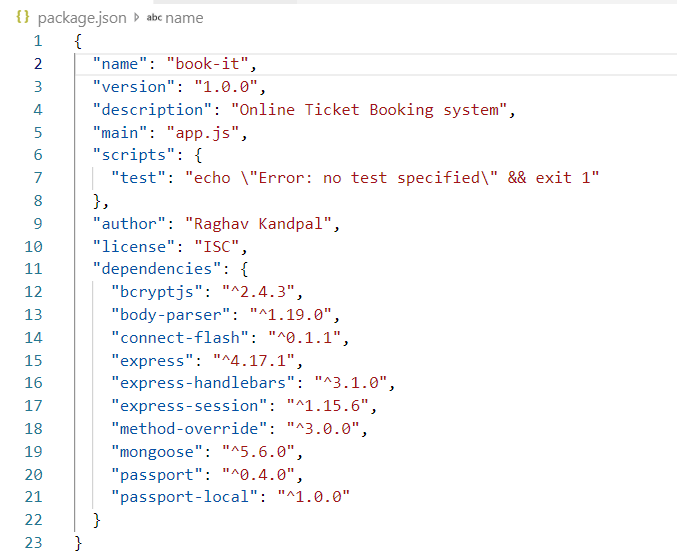
MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema.

A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Motivations for this approach include: simplicity of design, simpler "horizontal" scaling to clusters of machines (which is a problem for relational databases), finer control over availability and limiting the Object-relational impedance mismatch.

**Node dependencies**

**package.json**

The package.json file is the heart of Node.js system. It is the manifest file of any Node.js project and contains the metadata of the project.

****

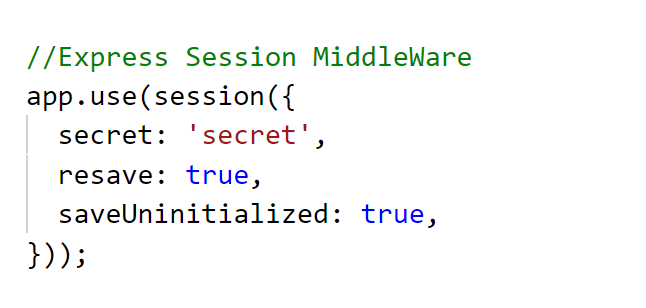
**Dependencies**

The dependencies property of a module's package.json is where dependencies - the other modules that this module uses - are defined. The dependencies property takes an object that has the name and version at which each dependency should be used.

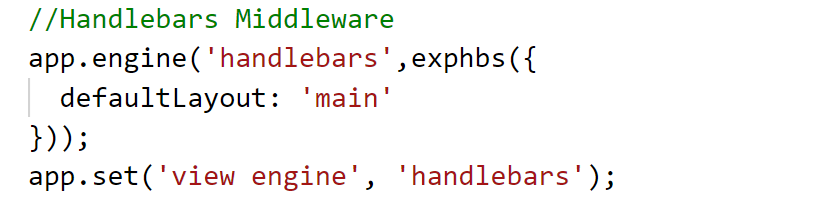
* Express:  
  Express is a minimal, open source and flexible Node.js web app framework designed to make developing websites, web apps, & API's much easier. It helps the user manage everything, from routes, to handling requests and views.

Middleware:  
Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application’s request-response cycle. The next middleware function is commonly denoted by a variable named next.

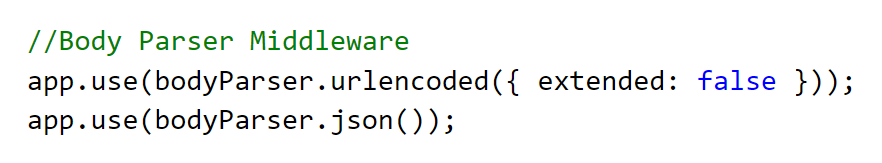
* Express Session:  
  A session is a place to store data that you want access to across requests. Each user that visits your website has a unique session. You can use sessions to store and access user data as they browse your application.



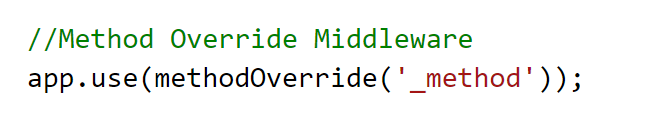
* Handlebars:  
  Handlebars.js is a popular templating engine that is powerful, simple to use and has a large community.  With Handlebars, you can separate the generation of HTMLfrom the rest of your JavaScript and write cleaner code.



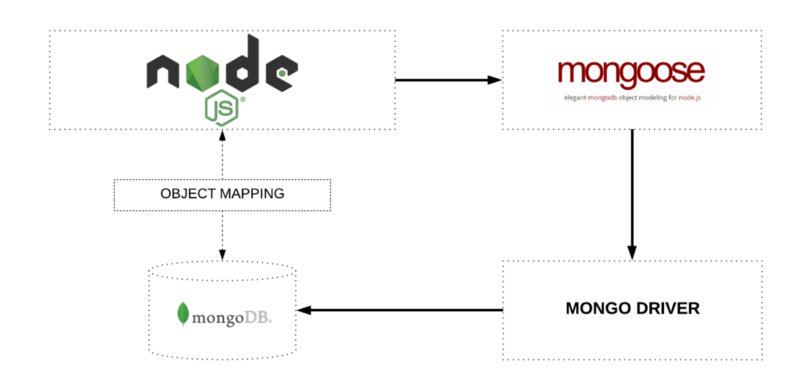
* Body-parser:  
  body-parser extract the entire body portion of an incoming request stream and exposes it on *req.body* . This body-parser module parses the JSON, buffer, string and URL encoded data submitted using HTTP POST request.



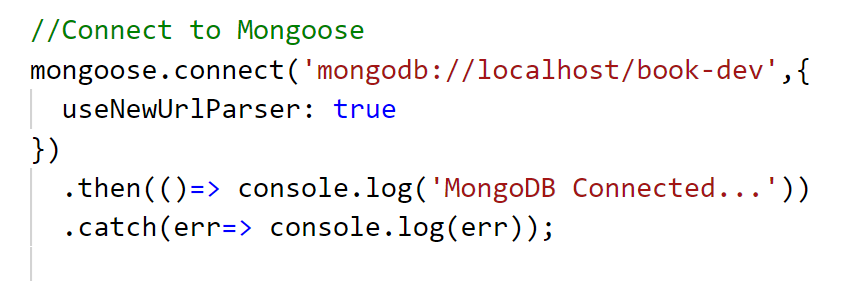
* Method Override:  
  The methodOverride() middleware is for requests from clients that only natively support simple verbs like GET and POST.



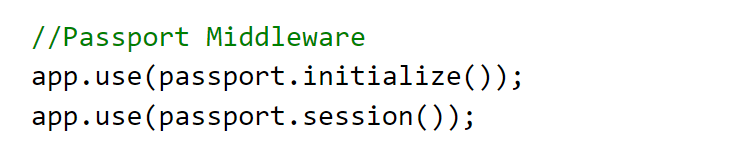
* Mongoose:   
  Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.



Using Mongoose, the user can define the schema for the documents in a collection.



* Passport.js:  
  Passport is authentication middleware for Node.js. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application. A comprehensive set of strategies support authentication using a username and password.



Local Strategy:  
Authentication mechanisms, known as strategies, are packaged as individual modules. Applications can choose which strategies to employ, without creating unnecessary dependencies.

passport-local lets the user authenticate using a username and password in the Node.js applications. By plugging into Passport, local authentication can be easily and unobtrusively integrated into any application or framework that supports Connect-style middleware, including Express.

* Bcryptjs:

The bcrypt library on NPM makes it really easy to hash and compare passwords in Node. If you're coming from a PHP background, these are roughly equivalent to password\_hash() and password\_verify(). Bcrypt is the de facto way to hash and store passwords.

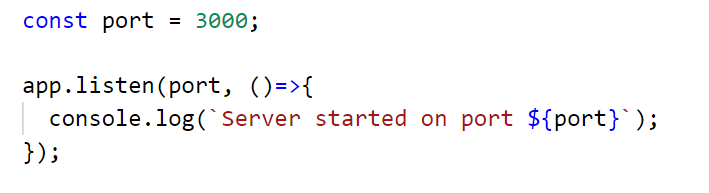
**APPLICATION**

The entry point of the application is **app.js**, which imports the required modules by creating methods.



The **app** object includes methods for routing HTTP requests, configuring middleware, rendering HTML views and registering a template engine.

The **app.listen()** function creates the Node.js web server at the specified host and port.

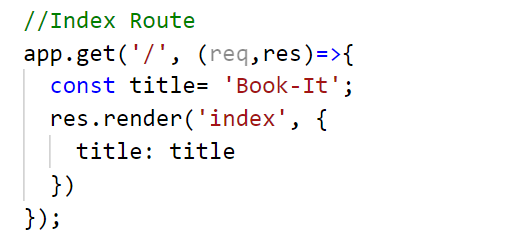


Once the server is started using the **node app.js** command, you can access the application on your browser at **‘http://localhost:3000’**

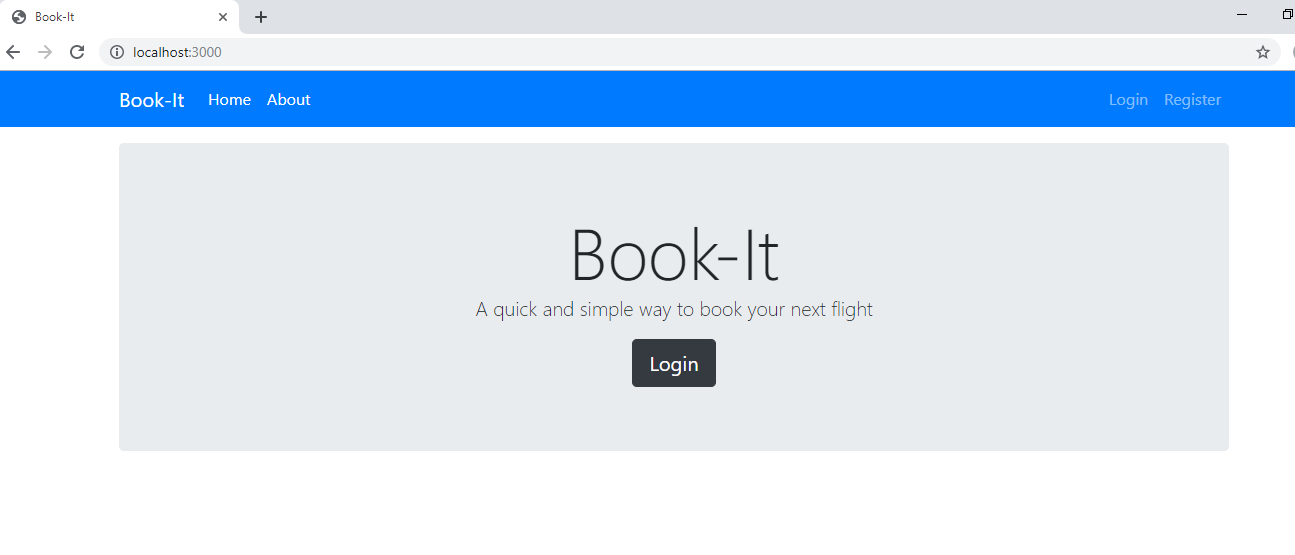
Route:  
A route is a section of Express code that associates an HTTP verb (GET, POST, PUT, DELETE, etc.), a URL path/pattern, and a function that is called to handle that pattern.

**Index Route:**

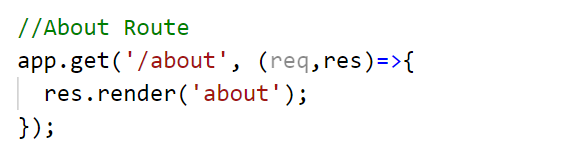
The index route is the main page of the application. It is the page that appears on the browser once you access **http://localhost:3000**.



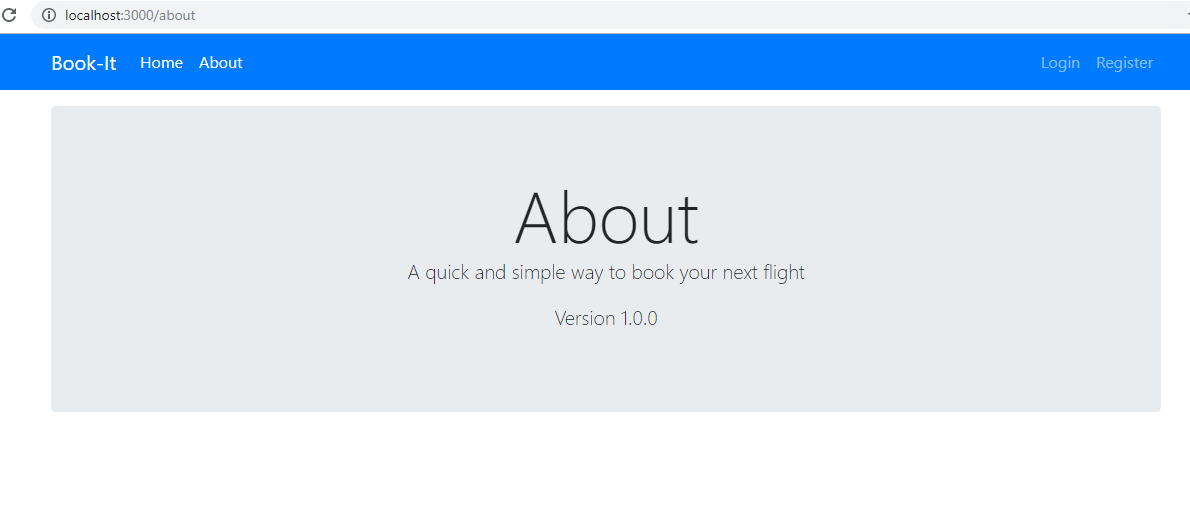
The page that we receive is:



**About Route:**On clicking about, the user is redirected to **http://localhost:3000/about**.



The page that we receive is:

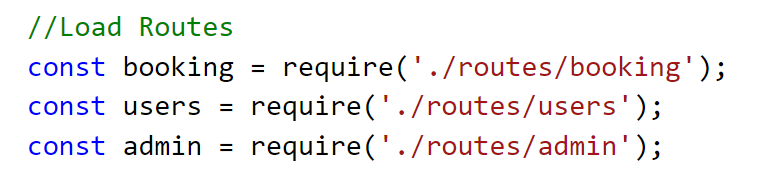


**Application Routes:**

To prevent **app.js** from overcrowding, we created individual application routes for certain functions. These routes are:

* **user** route
* **booking** route
* **admin** route

These are created in a **routes** folder in the main project directory and are then imported into **app.js**.

****

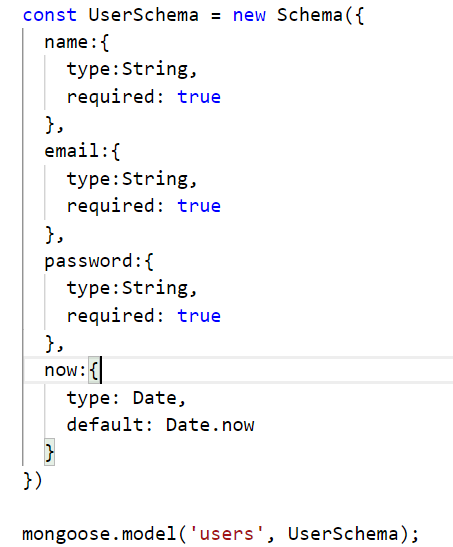
These are saved in separate JavaScript files, but instead of using their methods with the **app** object, in these routes, an object called **Router** is imported and used.

**USER Route**

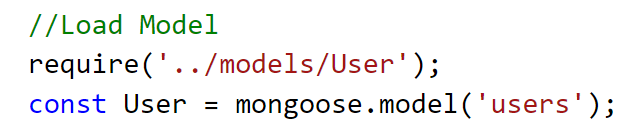
The **user** route contains the front-end and back-end functions of the login and registration part of the application. It is here that we use Passport for authentication of the users that browse the application.

**User Model:**Everything in Mongoose starts with a Schema. Each schema maps to a MongoDB collection and defines the shape of the documents within that collection.

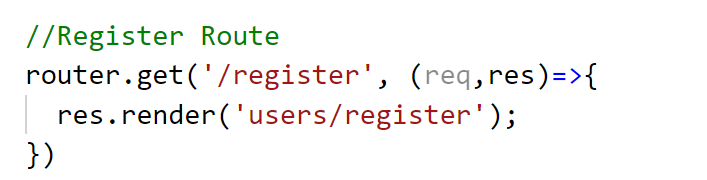
The user model is connected to collection called **users** in MongoDB. The collection is mapped into the application using the schema that we have created.



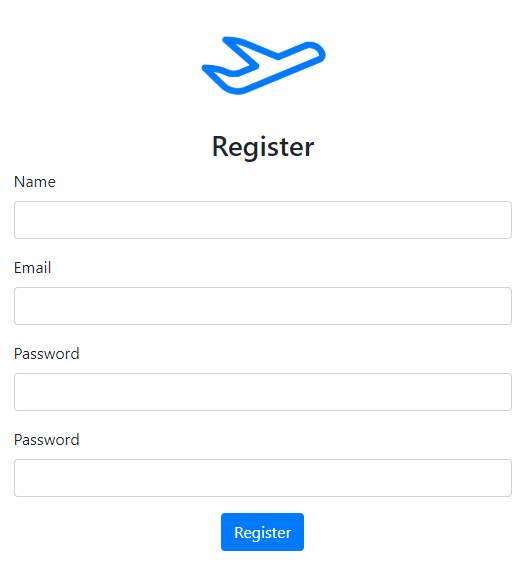
This is imported in **users.js** as follows:



**Register Route:**The register route shows the registration page for the application. It is accessed through **http://localhost:3000/users/register**.



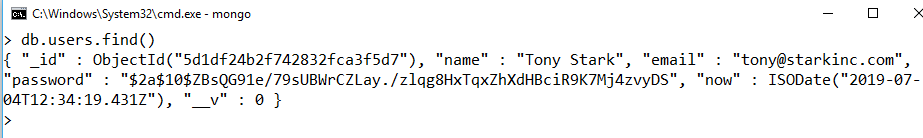
We receive the following form:



Henceforth, the posted data is sent to the **user** collection in MongoDB.

We are using the **bcryptjs** module for securing password, therefore we create a hash as soon as the password is sent to the server and is then saved in the database.

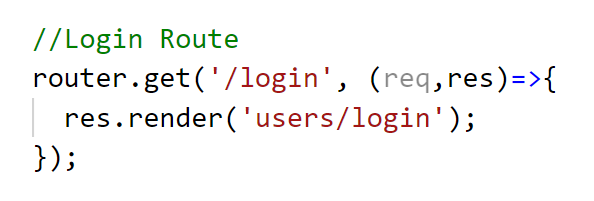
The Mongo shell command **db.users.find()** shows the data stored in the collection **user**:



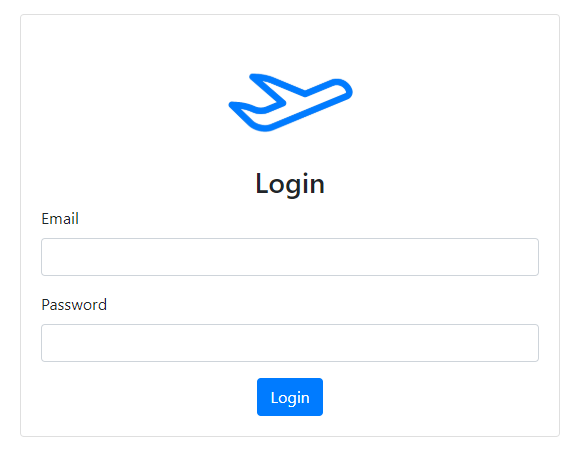
Note that the password is hashed.

Once a user registers, they are redirected to a login page.

**Login Route:**The login page is accessible on **http://localhost:3000/users/login**.



We receive the following form:



The data entered here is sent to be checked through Passport, where the hashed passwords are matched. If the match is found, the user is directed to the booking homepage, on the **booking** route.

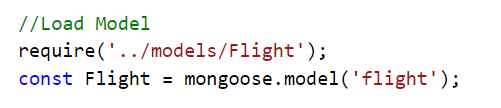
**Ensure Authenticated:**  
All views except for the homepage, about, login and register are accessible only after a user has logged in successfully. Therefore, the ensureAuthenticated function is required to render all other views. This is exported from **auth.js** in the **helpers** folder in the main directory.

**BOOKING Route**

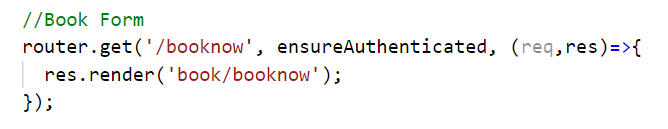
The **booking** route contains the front-end and back-end functions of the actual booking part of the application. Here the user browses the list of sources and destinations to get the flight listing for their provided date. After that, the user can choose the flight they want and book their seat. This generates a ‘Passenger Name Record’ or PNR number, which is unique for every booking.

**Flight Model:**The flight model maps data from the **flights** collection in MongoDB. Each flight object contains a Source, Destination, Time, Date, Class, Capacity and Flight Number.

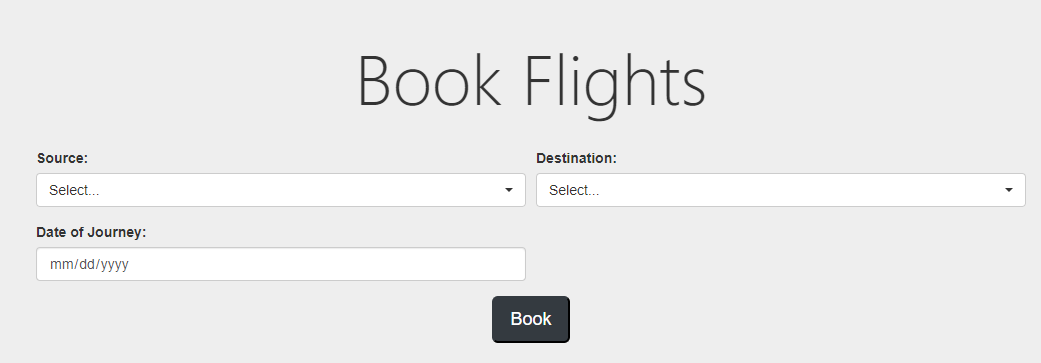
It is imported into **booking.js** as follows:



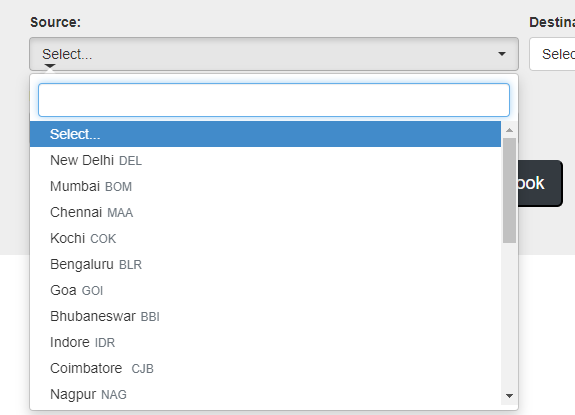
**Book Form:**The book form is accessible on **http://localhost:3000/book/booknow**.



Here the user can search for their required flights.

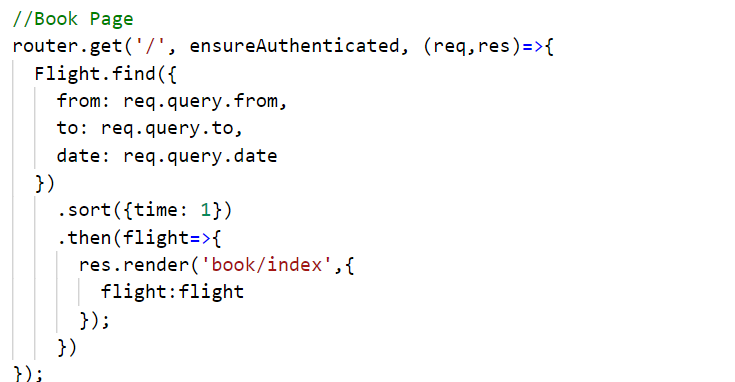


The list of sources and destinations can be browsed as seen below:

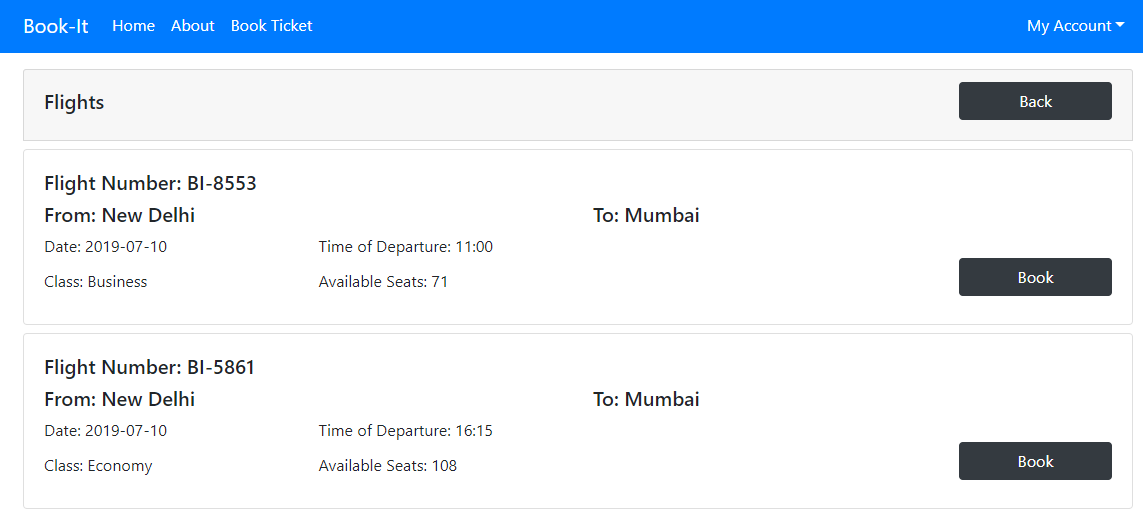


Once the details are entered, the details are sent to the server and the user is redirected to the list of flights.

**Booking Index:**The booking index page is accessed after entering the required details in the book form. Here, the flights are listed as per the users search. On the back-end, the server matched the entered source, destination and date into the **flights** model and returns the ones that match.

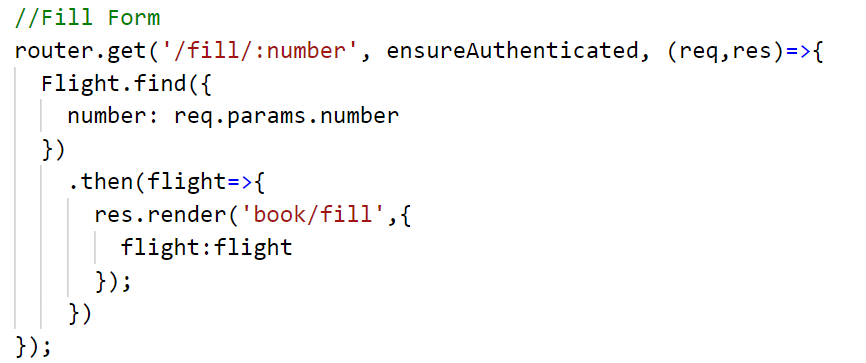


For instance, we search for flights from New Delhi to Mumbai on 10-07-2019. After we press the book button, we are redirected to this list of flights:



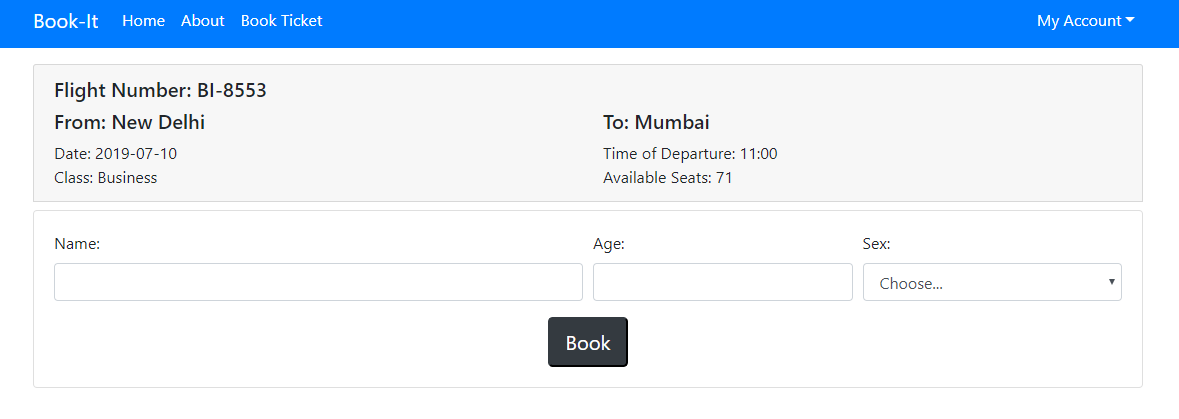
Here, we press book for the flight numbered **BI-8553** and we are taken to the next page for passenger details.

The fill details page is rendered using the following code:



Each page is rendered with reference to the Flight Number of the selected entry.

The fill details page is shown as follows:

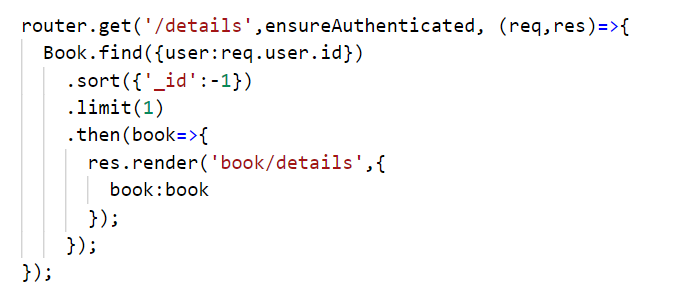


After filling the details, a confirmation page is rendered, showing that the booking has been confirmed.

**Book Model:**The book model maps data from the **books** collection in MongoDB. Each booking object contains a Flight Number, Name, Age, Sex, Source, Destination, Time, Date, Class, Seat Number and PNR number.

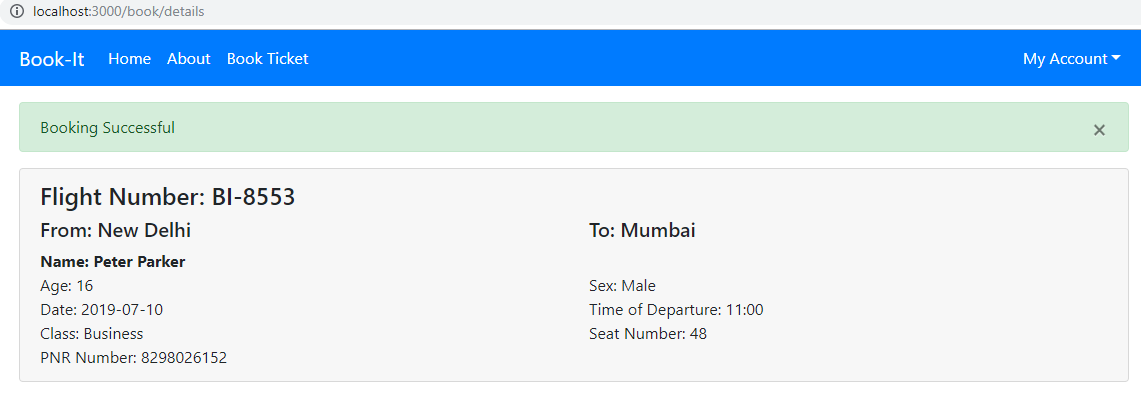
After the details are filled in the previous page, the data so gathered is pushed into the books database.

This data is then retrieved for the confirmation page as such:



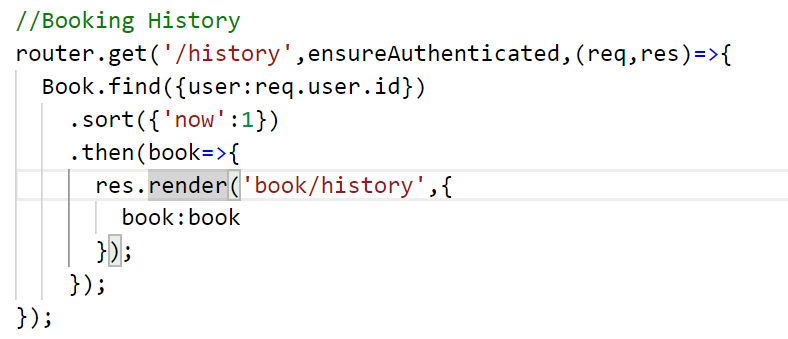
Therefore, the **books** collection is the largest and most important collection in this application.

The confirmation page is shown below:

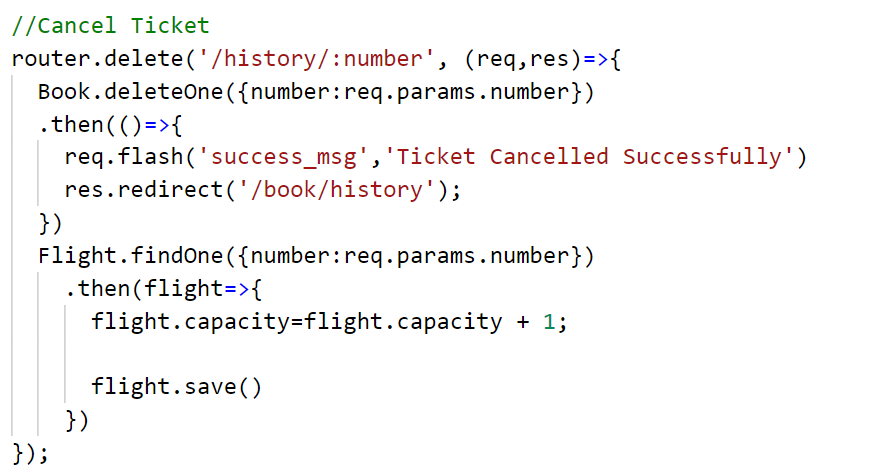


Hence, the booking is completed.

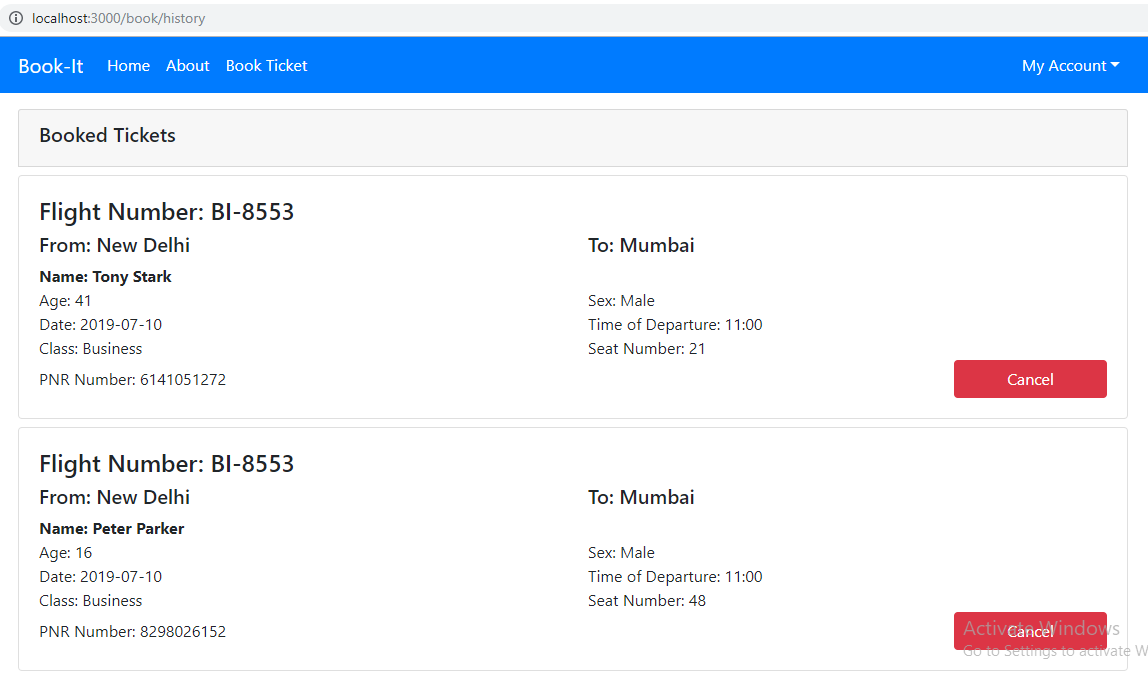
**Booking History:**For every user, there is a ‘My Account’ dropdown on the navigation bar, that gives them access to their ‘Booking History’. It is accessible at **http://localhost:3000/book/history.**



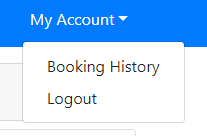
This page also contains the function for cancelling tickets.



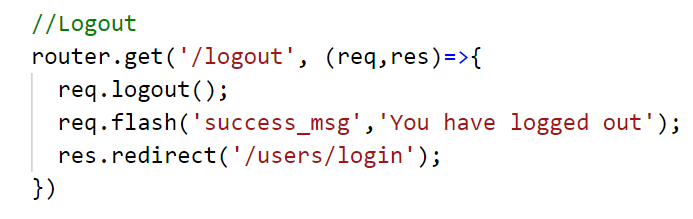
The rendered page is shown below:



The ‘My Account’ dropdown also has the logout feature.



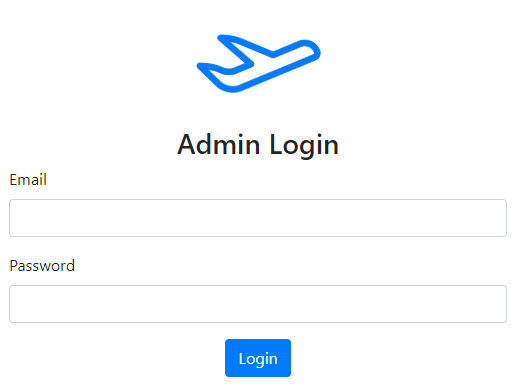
This route is handled by **users.js** and redirects the user to the login page after signing them out.



**ADMIN Route**

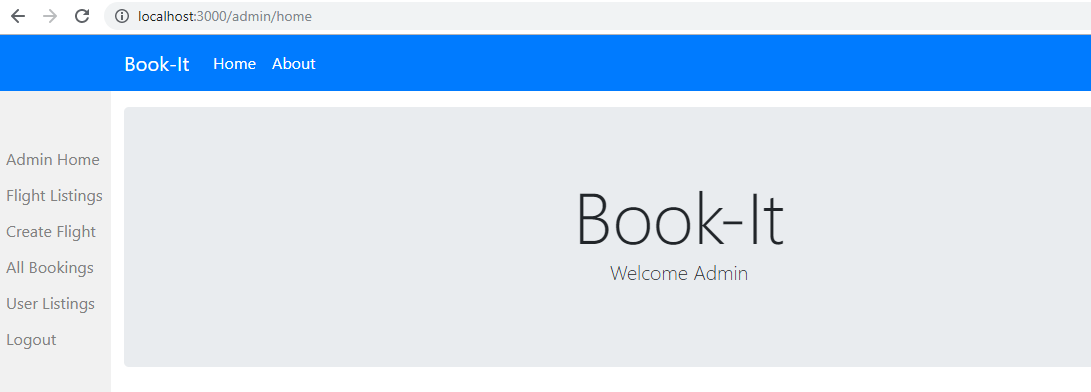
The **admin** route contains the front-end and back-end functions of the administration part of the application. Here, the admin can create flight listings that are accessed by the user. It also provides a list of all booked tickets and all registered users.

**Admin Login:**The admin login can be accessed through **http://localhost:3000/admin/login**.

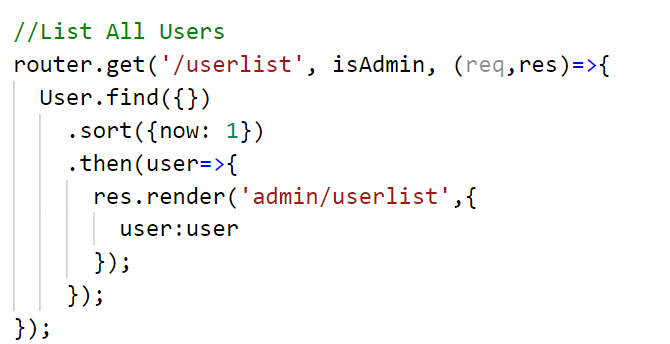


The default admin email is **admin@bookit.com** and the default password is **storm**.

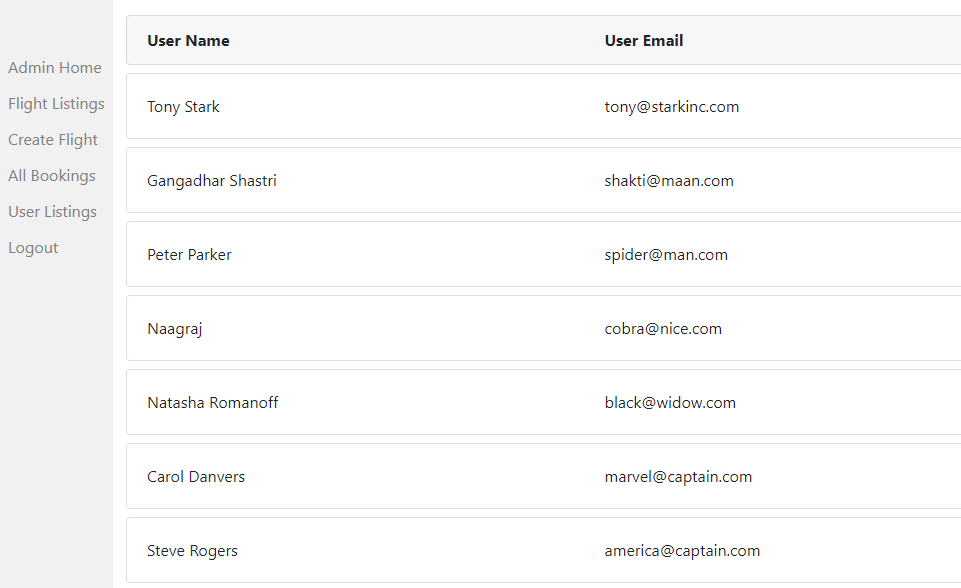
After successful login, the admin homepage is rendered.



**User Listing:**The user listing provides a list of all registered users and their e-mail IDs. It is accessible through the admin sidebar.

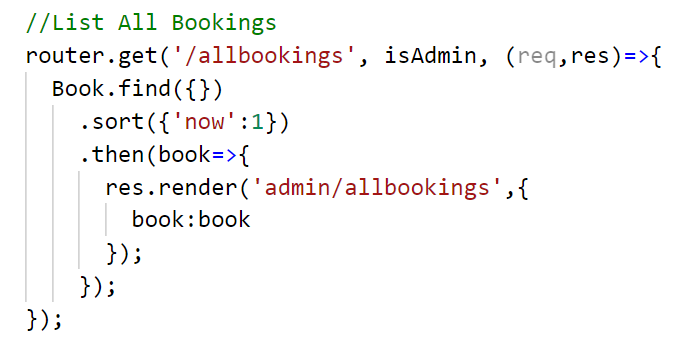


The listing is shown below:

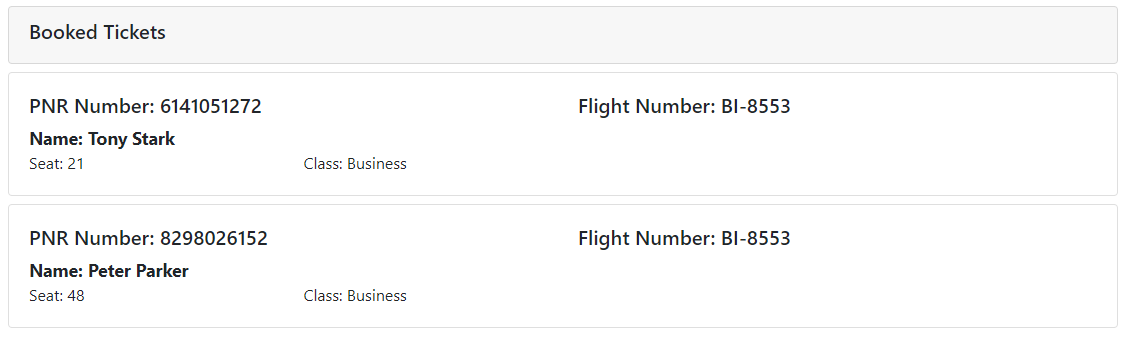


The passwords are hashed and therefore not accessible.

**All Bookings:**This provides a list of all bookings of all flights. It is accessible through the admin sidebar.

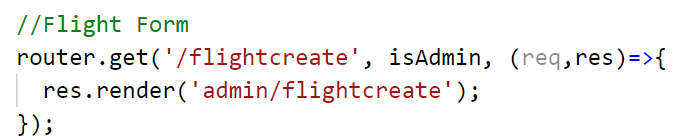


The list is as shown:

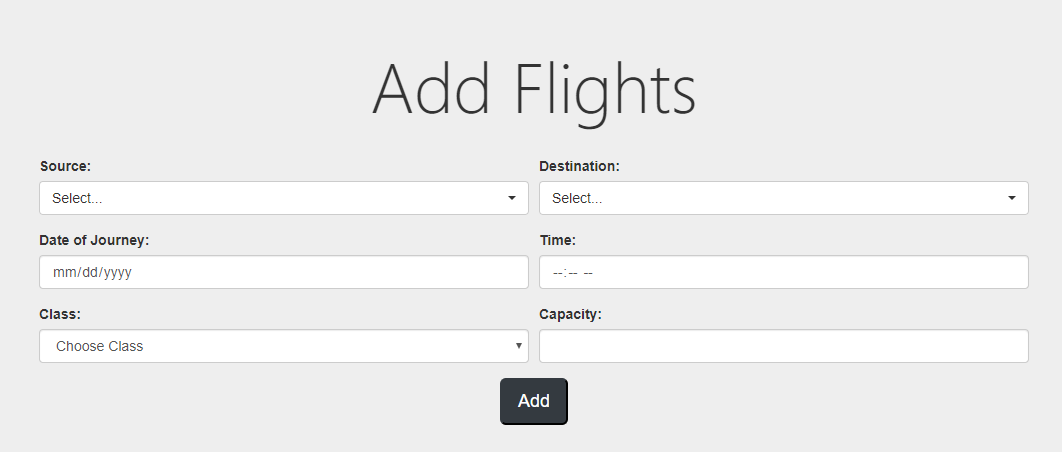


**Create Flight:**Now coming to the most important function of the admin console, creating flight listings. The admin is able to create a flight object that contains a Source, Destination, Time, Date, Class, Capacity and Flight Number. The Flight Number is automatically assigned for every flight.

The other details are filled by the admin on the ‘Add Flight’ page.

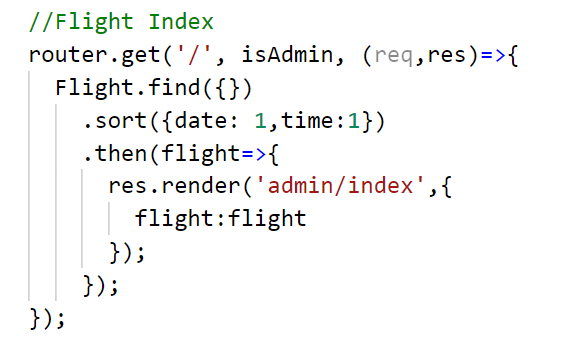


The rendered page is as shown:



After filing the details and pressing the **Add** button, data is sent to the server, where it is stored as an object in the **flights** collection and the user is directed to the flight index page.

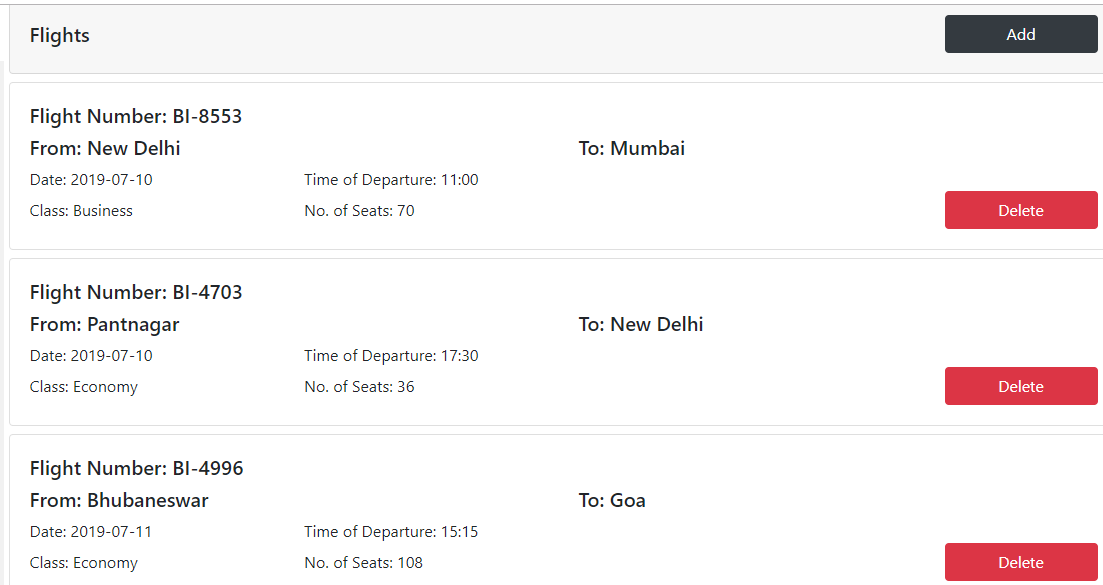
**Flight Index:**This page lists all the flights that have been created by the admin with the added function of deleting desired flights.



The listed flights are sorted according to their dates and time of departure.

Further updates in the application will create a separate tab for the flights of each date.

The rendered page is as shown:



The last item on the admin sidebar is the Logout button, which redirects the user to the admin login page.

**Front-End Elements**

**Bootstrap Navbar:**It is a simple navigation bar template. Using Handlebars, the navbar has been rendered through a partial view, which is then added to all pages.

Using the user global variable, the navbar can be used to show different views when the user is logged in and when user is logged out.

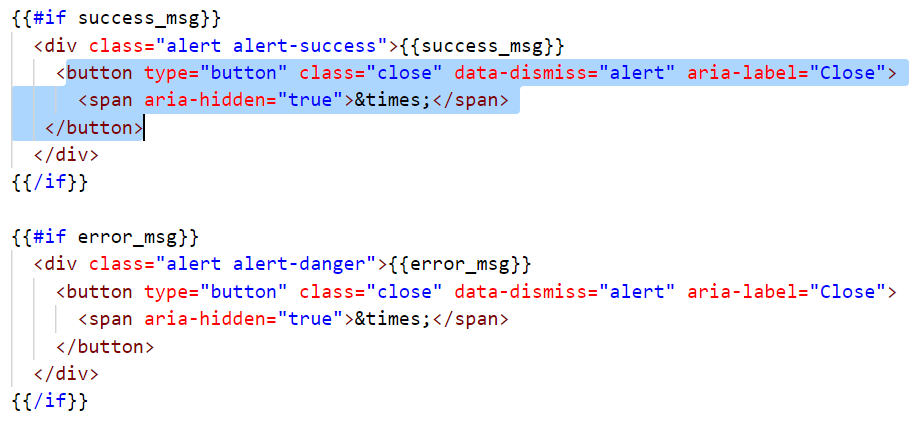
While user is logged out:



While user is logged in:



**Flash Messages:**Flash messages are also put into the layout using Handlebars partials. The file **\_msg.handlebars** is as shown:



And the flash messages:





**CONCLUSION**

Through all these routes and methods, the application is finally complete. There are many interactive front-end elements in the application as well as a lot of simple back-end routes. That being said, the application is still not complete, there are many elements that can be added.

Therefore, I must say that the application is still a work in progress. But still, it is very functional and easy to operate, which was the objective and motivation behind this service.

Further updates and version shall definitely be created and various other functions will be added in time.

This project has given me an opportunity to dive deeper in my knowledge of web technologies and has helped me learn many new skills. Hence, it is a culmination of all the theoretical and practical experience in web development.

Therefore, this training has proven immensely helpful for me.