The Application is made up of four components:

1.Card

2.FlightReservation

3.Flights

4.GoogleUserAPI

**Card:**

This provides a rest api for validating the card details provided as an input and post Validation the input amount is deducted from the bank table connected via one to one relation to the card table.

Syntax: GET <http://localhost:3456/cards/>cardNumber/CVV/expiryMonth/expiryYear/CardHolderName/AmountToBeDeducted

The Business Logic Validates the cardNumber fisrt if there are any entries found and then checks for the cvv,expiryMonth,expiryYear,CardHolderName and throws Exceptions respectively if any of the details are not matching.

After Validation it fecthes the AccountEntity which is connected to it and compares the balance to the amount to be deducted.If it is greater than or equal to it will deduct or else would throw an exception “No Balance Found”.

**GoogleUserAPI:**

This provides a rest api where in the user can punch in his/her email id and password of their google accounts which are present in the local googleuser database and the details would be registered to the Application’s Database.

Syntax:GET

<http://localhost:3466/google/details/emailid/Password>

Here, the Business Logic checks for the entries using the email id as it is the primary key and if records found returns a GoogleUser Instance or else would be throwing an Exception “No Data Found”

**Flights:**

This Spring Boot Project provides four APIs for performing the operations on the flight and the seats table which are connected via OneToMany relationship.

* Fetching the flights based on Source, Destination, Date and Number of Passenger:GET.

<http://localhost:3465/flights/getAvailableFlights/Source/Destination/TravelDate/numberofpassengers>.

The Business Logic will run a custom query based on the data input and provides the flight’s data and the seats connected to it.

It will throw an exception”Date is before Today” if entered date is before today and “No Flights Found” if the search did not return any results.

* Fetching the Seats of a flight based on a flightNumber:GET.

http://localhost:3465/flights/flightNumber

The Business logic will be fetching the available seats for a flight or will throw an Exception if there are no Available Seats on a plane.

* Getting the Total Price to be paid based on flightNumber and the total number of passengers:GET.

<http://localhost:3465/flights/flightno/numberofpassengers>

The business logic will fetch the total amount to be paid and will throw exceptions if the flightno is empty or the seats are empty respectively.

* Allocating the Seats and returning the reservationid:PUT.

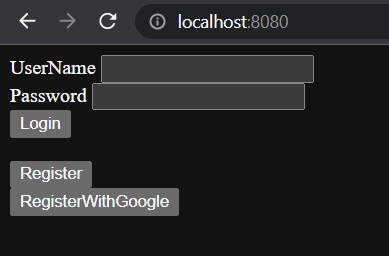
<http://localhost:3465/flights/bookSeats/flightno/seatNumbers(seperated> by a comma)

This API will fetch the list of seats available by flightNumber and set the status of seats passed as pathvariable to Booked and will be allocating the reservation id. This will also reduce the number of seats available in the flights table.

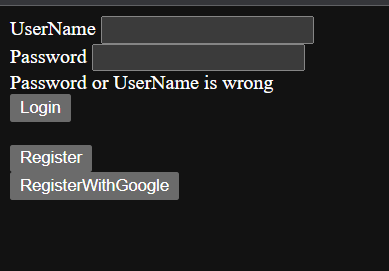
**FlightReservation:**

This is the main project of the Application which provides an interface based on JSPs and Spring MVC Architecture.

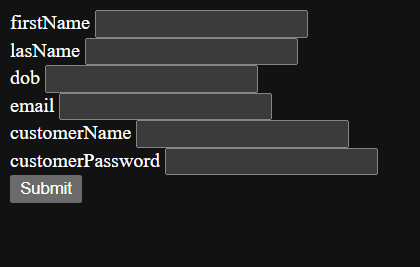
The index.html under static directory present in resources would be the visible as below when you access: <http://localhost:8080>



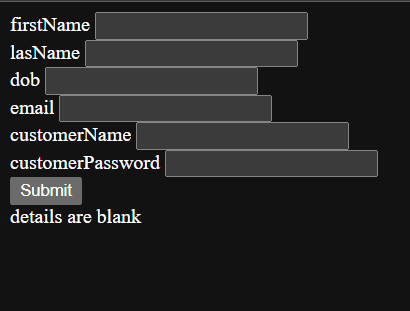
If the entered details for login are not correct erro.jsp is called:



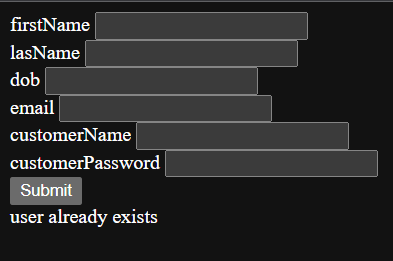
You can click on the Register Button which will call the register.jsp as follows:



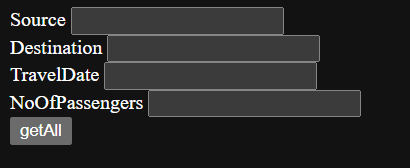
If the details are blank blankregister.jsp is called:



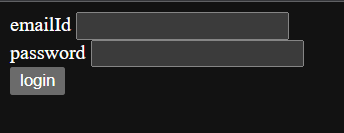
If the details entered are already present register\_error.jsp is called as below:



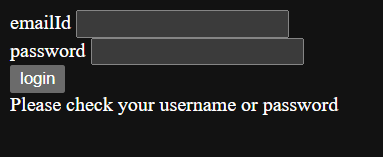
If the User registration is successful they will be logged into the Welcome.jsp page:



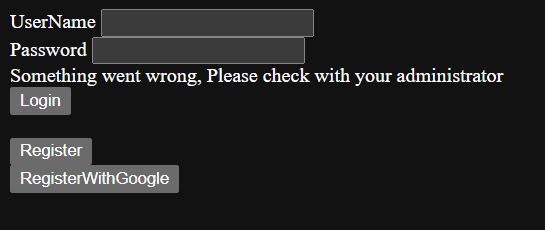
Upon clicking the RegisterWithGoogle registerG.jsp would be called:



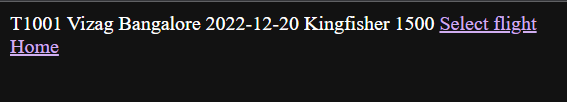
If the username or password is wrong registerGerror.jsp would be called:



If there is any exceptions caught while injecting data GoogleInjectError.jsp is called:



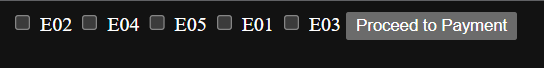
Once you are in the welcome.jsp page after successfully entering the details as per the database flightsview.jsp is called:



Here, the Business Logic will be first fetching an array through RestTemplate and ResponseEntity and that array is added into the flightsview.jsp where it is iterated.

Upon clicking on Home you will be re-directed to welcome.jsp page

Upon clicking on Select Flight seatsview.jsp would be called:



Here the seats list is iterated in the jsp.

If the number of seats selected in the welcome.jsp and the number of seats selected here doesn’t match it will call the InvalidSelection.jsp page:



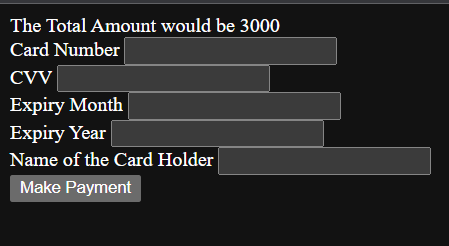
Upon clicking you will be re-directed to the seatsview.jsp page

If nothing is selected and Proceed to Payment is clicked emptySlection.jsp is called:

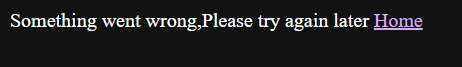


Upon clicking you will be re-directed to the seatsview.jsp page

After selecting the seats successfully payment.jsp page would be called:

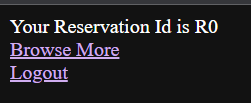


If there is any error in the payment ErrorInPayment.jsp is called:



Upon clicking on Home you will be re-directed to welcome.jsp page

If the payment is successful final.jsp would be called:



Browse More will redirect to welcome.jsp and Logout will take you to the index.html.

The .sql files present under resources needs to be executed first and the the url for database must be setup in application.properties accordingly.

Download the repository into your local and run the Spring Boot Application class.