IT Lab Report – Assignment 3

TITLE

Name – Biswarup Ray

Roll – 001810501082

Class – BCSE 3rd year

Group – A3

Assignment Number – 3

Implement a web application for "Travel Thru Air" using servlets to support the following two use cases:

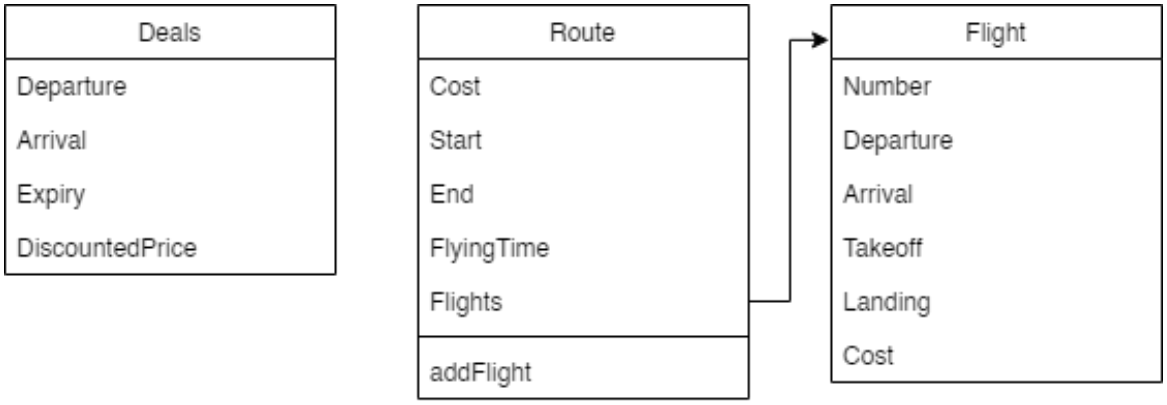
1. A list of current special deals must appear on the home page. Each special deal must display the departure city, the arrival city, and the cost. These special deals are set up by the marketing department and change during the day, so it can't be static. Special deals are only good for a limited amount of time.
2. A user may search for flights, given a departure city, time and an arrival city. The results must display the departure city, the arrival city, the total cost, and how many legs the flight will have.

State and explain why and where you have used design patterns. If possible, please write the front end using React JS.

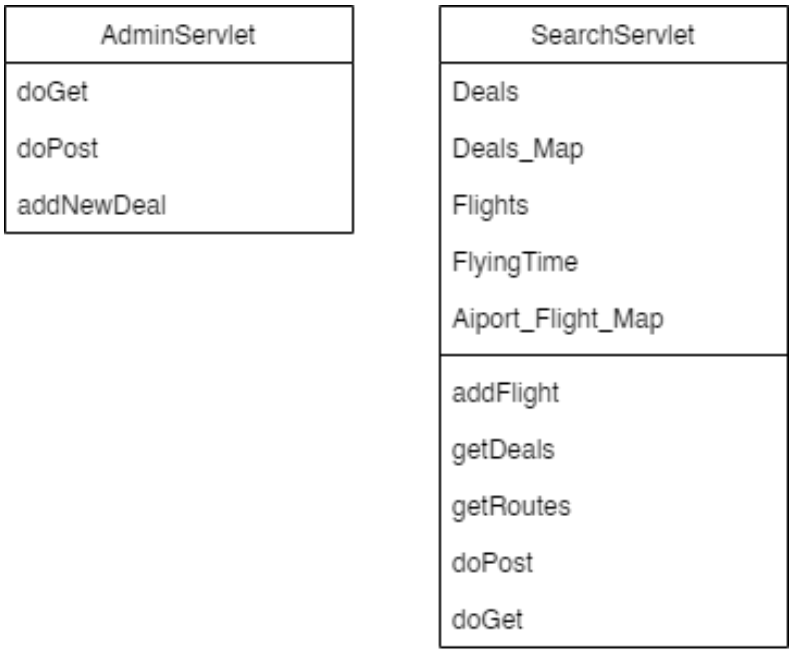
Solution Approach

The class diagram and the servlet format for the solution approach used has been shown below.

Class Diagram



Servlet

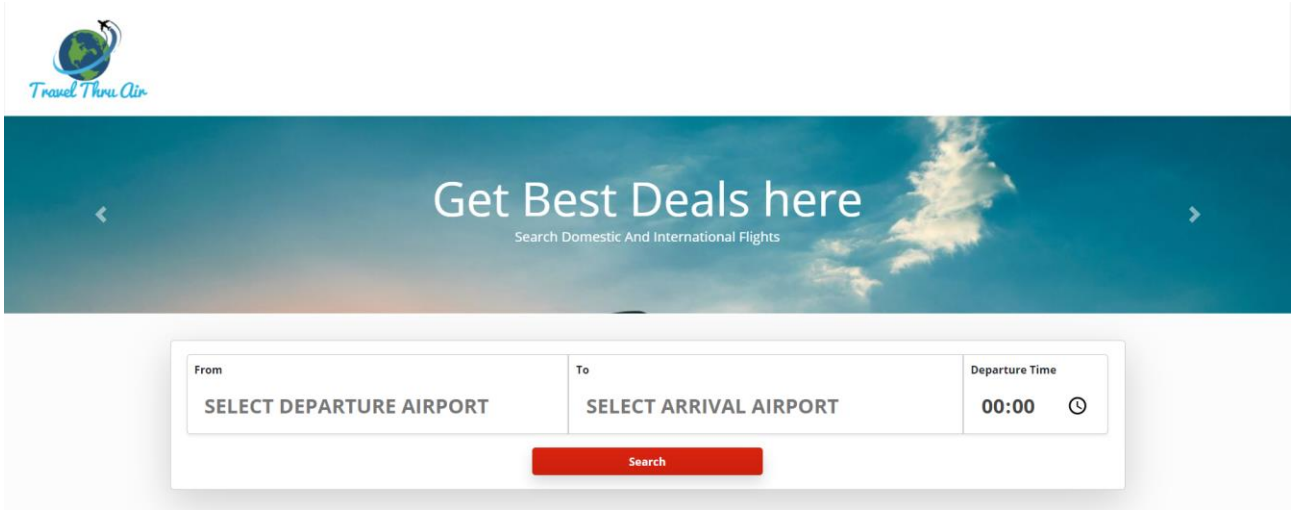


Frontend and backend modules.

Frontend

The frontend has been done using jsp files and have been coded in html.

User Interface :



Backend modules

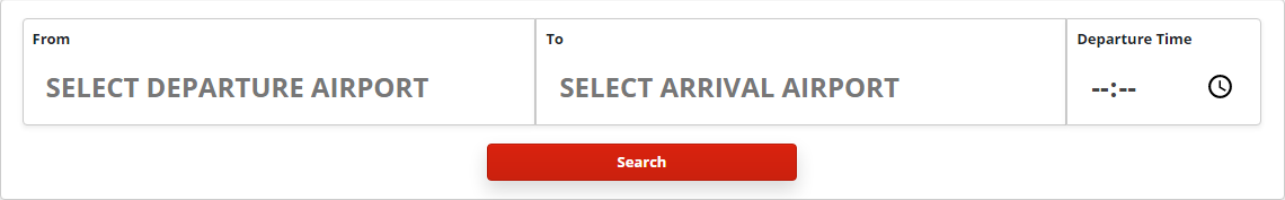
Java servlet class has been used to implement the backend of the website. The data are stored and read from csv files.

Technology and Features

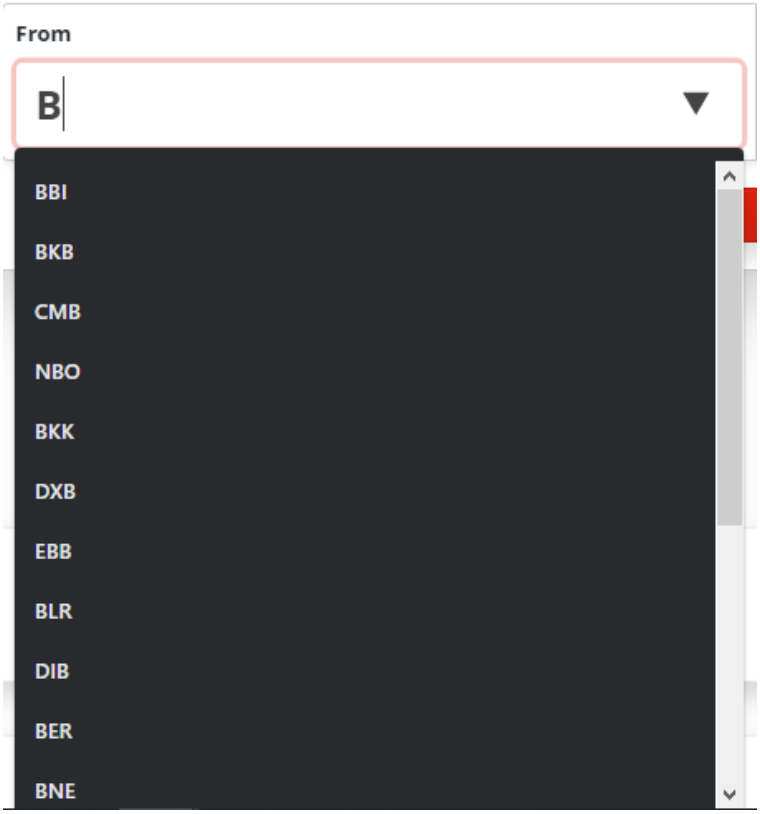
Deals

Carousel like display, continuously scrolling horizontally displaying active deals.

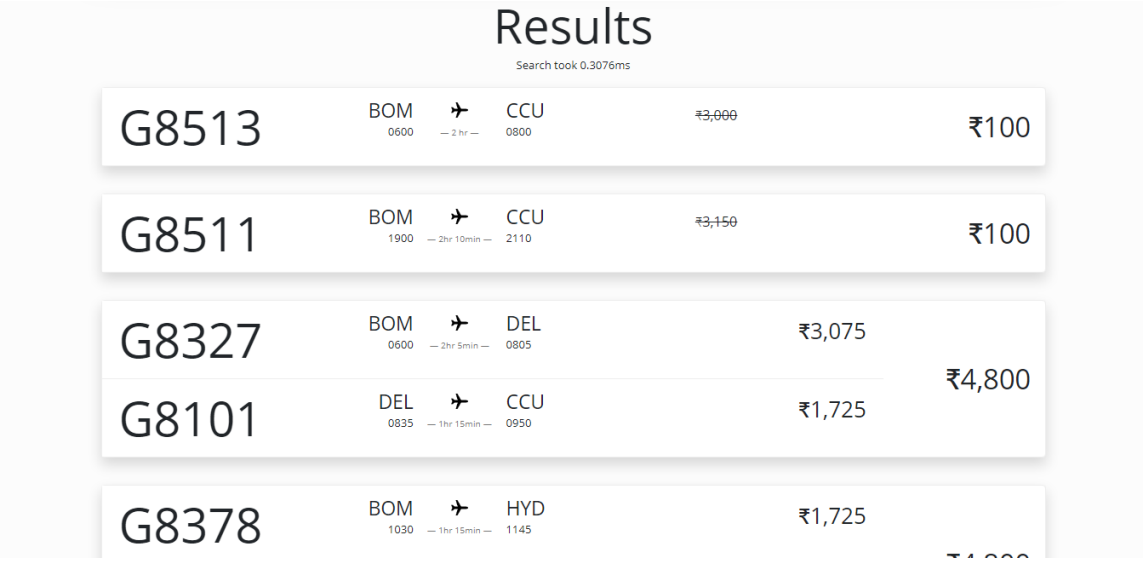
Search box



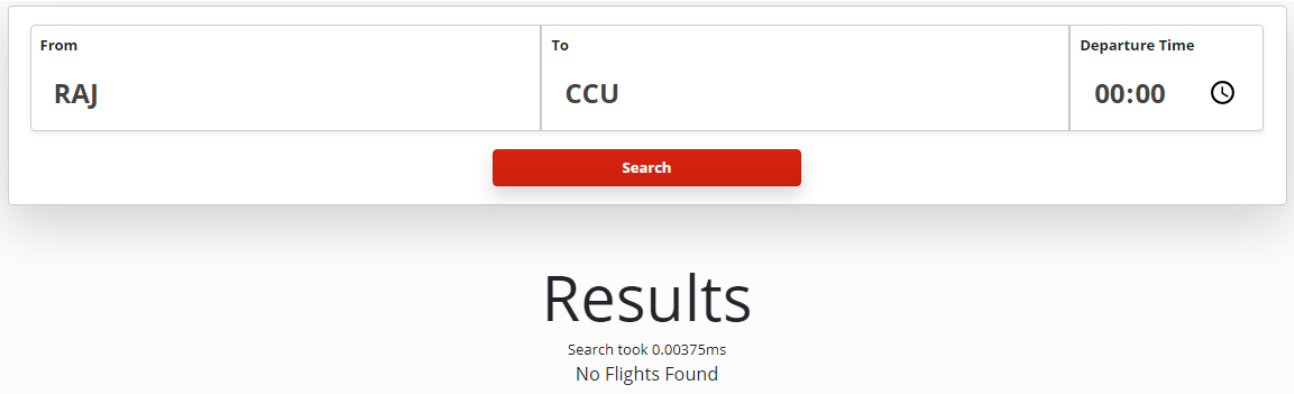
Airport Code Suggestor



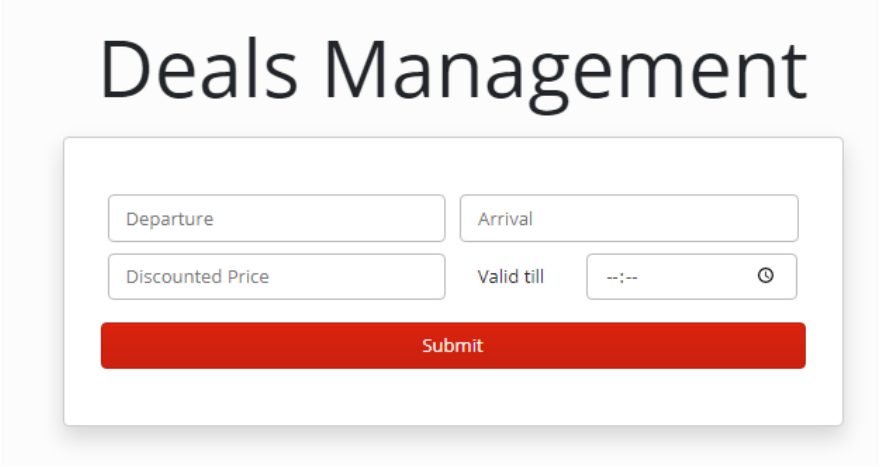
Multi Leg Flights



Proper Error Message



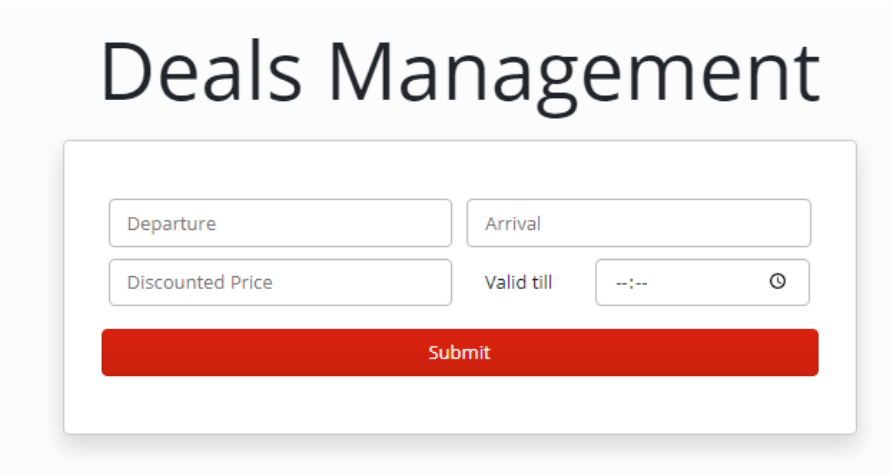
Admin Deals Management



Backend Implementation

Backend has been coded using java servlet classes. Instead of using DBMS a simplistic approach of storing data in csv has been used.

The admin page has been also created from where the admin can update the deals present in csv.



Code for updating the deals:

```
   private static void addNewDeal(String departure, String arrival, int exp, int
discount_price)
   // Function to store a new deal if not present or already present deal into a hash map
   {
      deals.add(new Deals(departure, arrival, exp, discount_price));
      String sector = departure + arrival;
      HashMap<Integer, Integer> temp; // A HashMap to store the discount price of a deal
with its expire time as a key

      if (deals_map.containsKey(sector))
         temp = deals_map.get(sector);
      else
         temp = new HashMap<Integer, Integer>();

      temp.put(exp, discount_price);
      deals_map.put(sector, temp);
   }

   public static void addDeal(String departure, String arrival, int exp, int
discount_price, String path)
   // Function to add a new deal to the deals.csv and add deal to the deals_map using the
addNewDeal declared above
   {
      try {
         FileWriter csvWriter = new FileWriter(path, true);
         csvWriter.write(
               '\n' + departure + "," + arrival + "," + Integer.toString(exp) + "," +
Integer.toString(discount_price));
         csvWriter.close();
         addNewDeal(departure, arrival, exp, discount_price);
      } catch (Exception e) {
         e.printStackTrace();
      }}
```

The flights have been stored by random web scraping into a csv named ***"flights.csv"*** in the column format:

Flight Number	Departure airport code	Departure time	Arrival airport code	Arrival time	Cost of the flight

Some deals have been stored initially by default in a csv named ***"deals.csv"*** the deals could later be updated using the admin deals management page. The data is present in the column format:

Departure airport code	Arrival airport code	Arrival time	Expiry time of the deal

Code to get the deals:

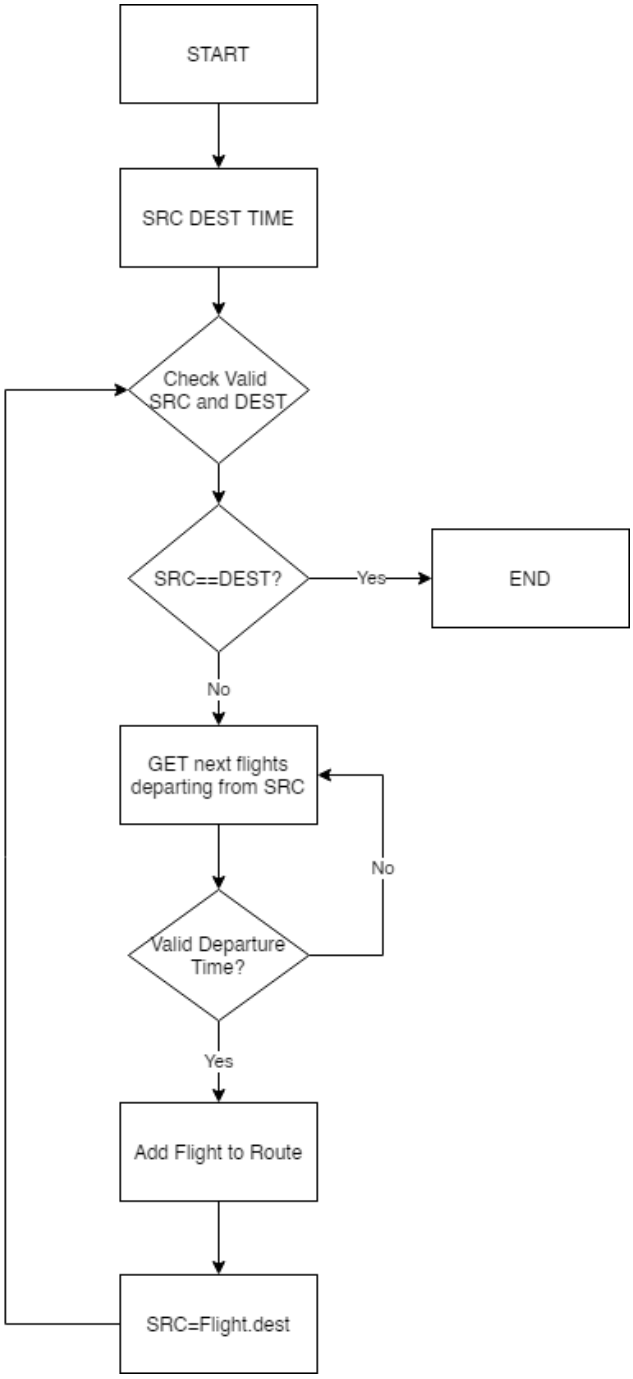
```
   private void getDeals(String path)
   // Function to get the values of the deals from the deals.csv file and add add deal to
the deals_map using the addNewDeal declared above
   {
      try {
         Scanner sc = new Scanner(new File(path));
         while (sc.hasNext()) {
            String[] row = sc.next().split(",");
            addNewDeal(row[0], row[1], Integer.parseInt(row[2]), Integer.parseInt(row[3]));
         }
         sc.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
                                                }

   private ArrayList<Deals> getActiveDeals()
   // Function that stores the active deals (after current time) in a sorted order in an
ArrayList and returns it
   {
      Calendar now = Calendar.getInstance();
      int time_now = 100 * now.get(Calendar.HOUR_OF_DAY) + now.get(Calendar.MINUTE); // Get
the current time
      ArrayList<Deals> activeDeals = new ArrayList<Deals>(deals.stream()
                                    .filter(deal -> deal.getExpiry() > time_now)
                                    .sorted(Comparator.comparing(Deals::getExpiry))
                                    .collect(Collectors.toList())); // Use the stream
function of java taught in PPL
      return activeDeals;
                                                }
```

Search Implementation

The search algorithm has been implemented using Depth First Search Algorithm (**DFS).**

A flowchart representing the algorithm used:



Code for the search algorithm:

```
   private LinkedList<Route> getRoutes(String src, String dest, int time)
   // Function that stores all the possible routes in a LinkedList and returns it
   {
      src = src.toUpperCase();
      dest = dest.toUpperCase();
      LinkedList<Route> routes = new LinkedList<>();
      LinkedList<Flight> flights = new LinkedList<>();
      this.getRoutes(src, dest, time, 0, "", flights, routes); // Calling the
overloaded getRoutes present below
      routes.sort((r1, r2) -> {
         int res = r1.cost - r2.cost;
         if (res == 0) {
            res = r1.flights.size() - r2.flights.size();
            if (res == 0)
               return r1.flyingTime - r2.flyingTime;
            return res;
         }
         return res;
      });
      return routes;
   }

   private void getRoutes(String src, String dest, int time, int cost, String
path, LinkedList<Flight> flights,
         LinkedList<Route> routes)
   // Overloaded Function that uses DFS to search possible the routes
   {
      if (!airport_flights.containsKey(dest) || !airport_flights.containsKey(src))
         return;
      if (src.equals(dest)) {
         if (flights.size() > 0)
            routes.add(new Route(flights));
         return;
      }
      for (Flight flight : airport_flights.get(src)) {
         if (flight.getTakeOff() < time)
            continue;
         String next_airport = flight.getArr();
         if (!path.contains(next_airport)) {
            LinkedList<Flight> flights_copy = new LinkedList<Flight>(flights);
            flights_copy.add(flight);
            getRoutes(next_airport, dest, flight.getLanding() + 30, cost +
flight.getCost(), path + src + "~",
                  flights_copy, routes);
         }
      }
                                                }
```