Techademy

CAPSTONE Assessment-ANGULAR

Flight Booking Application

Contents

[1 Problem Statement 2](#_Toc173100328)

[2 Screen Flows 2](#_Toc173100329)

[3 Technical Details 5](#_Toc173100330)

# Problem Statement

**Develop a flight ticket booking application using Angular and SpringBoot and MySQL.**

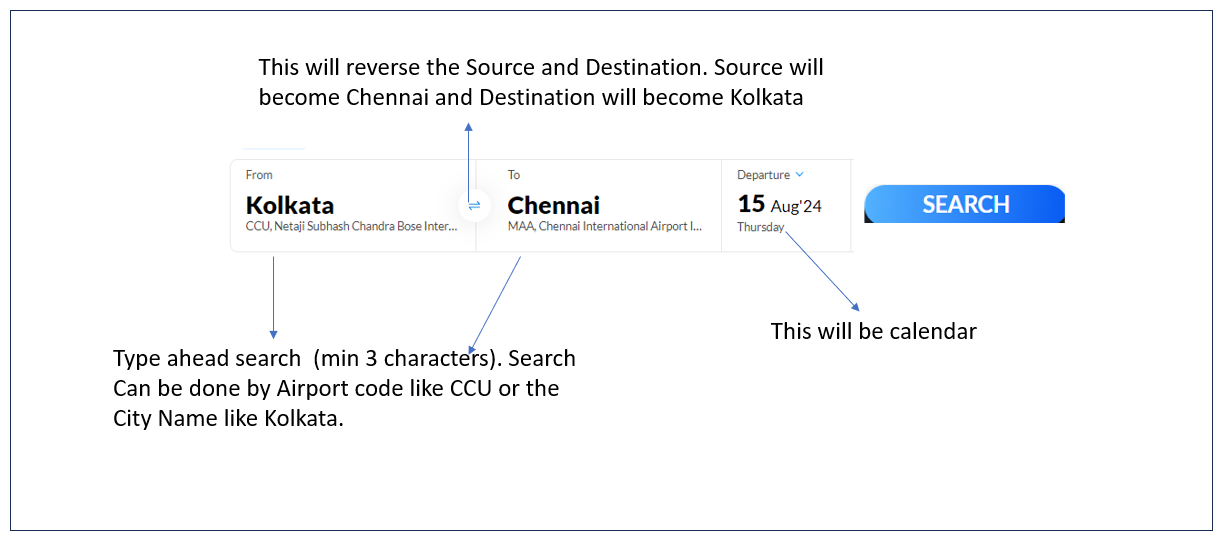
**Scope of the project will be to :**

1. **Analyze the requirement.**
2. **UI Design.**
3. **Database design.**
4. **REST endpoint design (i.e designing of REST urls, methods and input data).**
5. **Development and Testing.**

# Screen Flows

**Note :** All screens are indicative and can be made improvised.

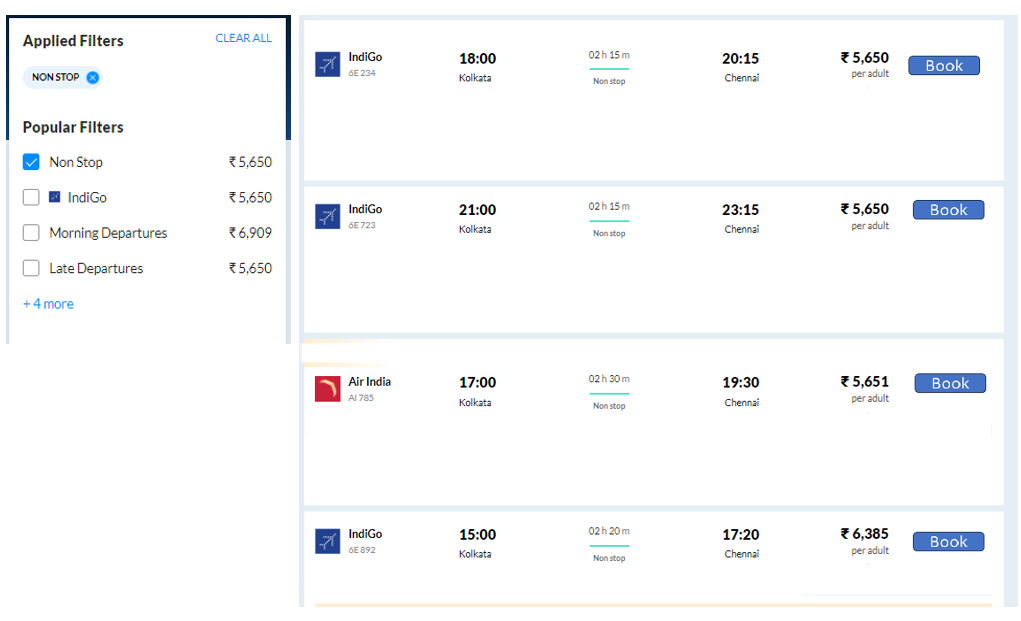
**Home Screen: Screen-01**



**Search Result Screen: Screen-02**

Once the search button is clicked the following screen will appear .

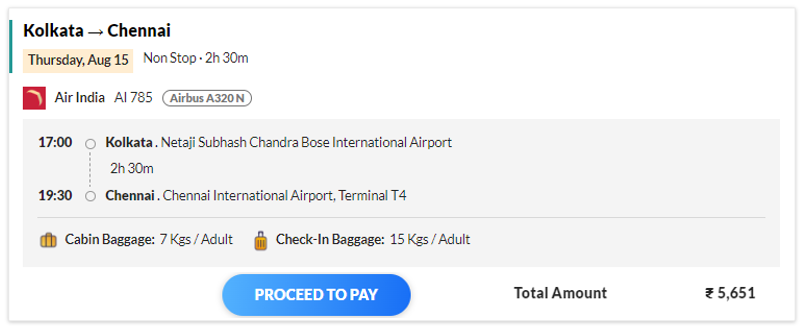
User will be able to apply some filters and based on that the search results will be filtered. User will also be able to clear filters.



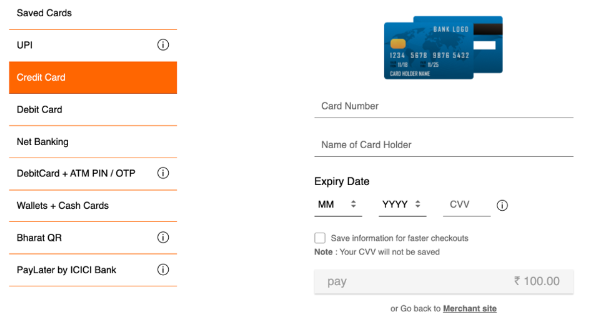
If there is no data available for the specified search criterion then a message will be shown (in the screen – no popups) saying “No Flights available matching your criterion” under the search screen.

**Booking Screen: Screen-03**

Once the search button is clicked

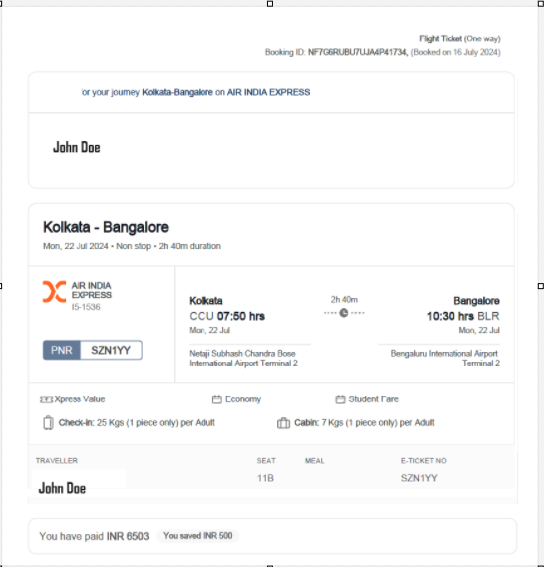


**Payment Screen: Screen-04**



**Successful Payment: Screen-05**

Ticket will be generated – same can be downloaded as PDF.



**Failed Payment :** **Screen-06**

In case of failed payment a message will be displayed saying the payment has failed and user can then start from the beginning.

# Technical Details

1. There will be three Microservices
   1. Flight Data MS – Will contain Flight details like Aircraft name , flight number , Flight route and price for that route for a particular day.
   2. Booking MS : This is the master microservice. All screens will be fed data from this microservice and it will be responsible for search , payment etc by interacting with the other microservices (payment and flight detail ms).
   3. Payment MS – Generating the payment status.
2. Overall logic:
   1. When a search is made with Source, Destination and Travel date (will assume that the booking will be made for one traveller only) – that will invoke an endpoint say /search of the Booking MS and that in turn will call another end point of Flight Data MS (which contains all master data required for flights along with routes and price for a particular date range).
   2. If the search date falls inside the range available in the Flight data then all the information required for populating that screen will be returned by the Flight Data MS.
   3. Each search result returned by Flight Data MS will have an unique id so that if a booking is made for any particular results then that id is stored in the Booking MS.
   4. For additional filter conditions either a different end point of Flight Data MS can be used or some additional parameters are passed as filter conditions to the search end point.
   5. If the book button is clicked then a payment is initiated for that booking. A booking id will be generated with status “initiated”. If the payment is successful then the status will be changed to “successful”. The other possible values are “failed” if payment failed or “cancelled” if cancelled by the user.
   6. No real payment gateway integration is required. You may integrate only one payment methods.
   7. Some basic validations to be implemented like if card payment is implemented then the card number should be 16 digits and expiry date should be later than today’s date.
   8. Once the payment is initiated then a message will be generated with the booking id, amount ,mode of payment etc to a Kafka topic (say T1).
   9. The Payment MS will then read that data from the T1 topic and then will do some processing and return the result against that booking id in the T2 topic with status “Successful”/”Failed” (or some number or code) indicating that the payment is successful or failed along with the booking id. The Booking MS will then update the status against that booking id and will either generate a Eticket (if it is successful) or show a message saying that the booking has failed.

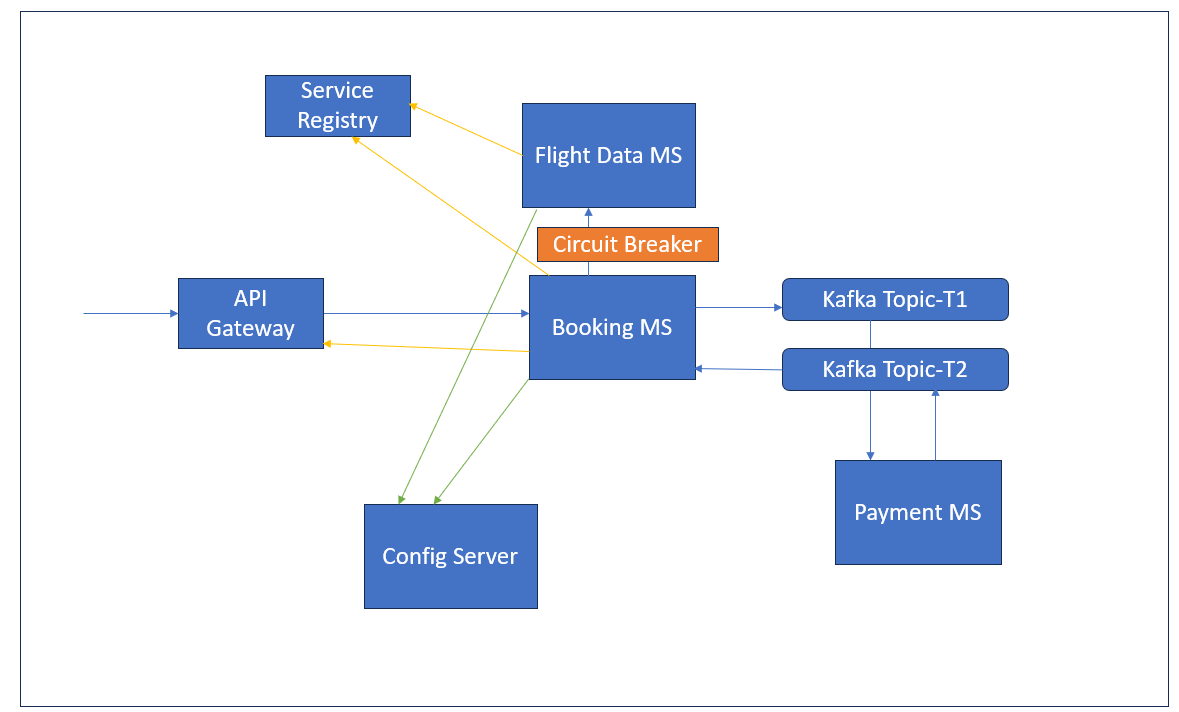


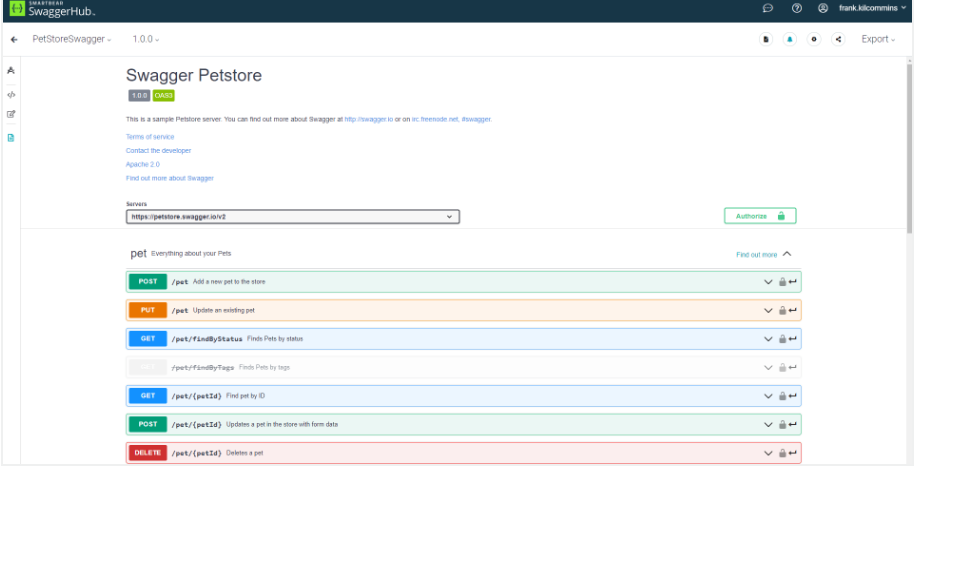
Figure Application Architecture

***Swagger Documentation Guideline:***

* Use @Tag , @Operation, ApiResponses and @ApiResponse annotations for swagger documentation. A sample class is provided below as reference:



The api documentation should be visible through swagger-ui as below (image is for reference only):



***UI/UX Guideline:***

***Core UX Guidelines for UI Development***

* + Consistency: Maintain consistent design elements (colors, fonts, buttons).
  + Clarity: Make UI elements clear and understandable.
  + User control: Give users control over interactions.
  + Feedback: Provide feedback for user actions (e.g., button clicks).
  + Clear messaging: Use clear, concise language for labels, buttons, errors.

***UI Design Best Practices***

* + Simple and intuitive: Minimize complexity in UI.
  + Use familiar patterns: Use common UI patterns users are familiar with.
  + Responsive design: Ensure UI works well across devices/screen size

General Guidelines:

* Use Java version 17/21.
* Use layering in Microservices.
* Use proper package structure like com.wipro.flightms
* Write JUnit test cases for testing.
* The Microservices and the database could be dockized (optional).
* Use Swagger for API documentation (mandatory).
* Angular Routing should be used.
* Usage of Bootstrap is advisable.
* Code should be maintained in a Git repository.

Core Technology:

* HTML/CSS/Angular/Spring Boot

Deliverables:

* Screen shots/Videos showing all operations and data in the database.
* The html , css and .js files as source code in proper folder structure.