Name – Nityasundar Mondal

Roll – 002110501091

Class – BCSE III

Section – A2

Subject – IT Assignment 4

PROBLEM STATEMENT :-

Implement a web application for “Book My Flight” using Spring framework to implement the following

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.

A customer would select a special deal and should be able to book a flight. (S)He may not register using username

password for booking a flight. A phone no could be used instead.

Technical requirements:

1. State and explain why and where you have used design patterns.

2. If possible, please write the front end using React JS (https://reactjs.org/) or Thymeleaf

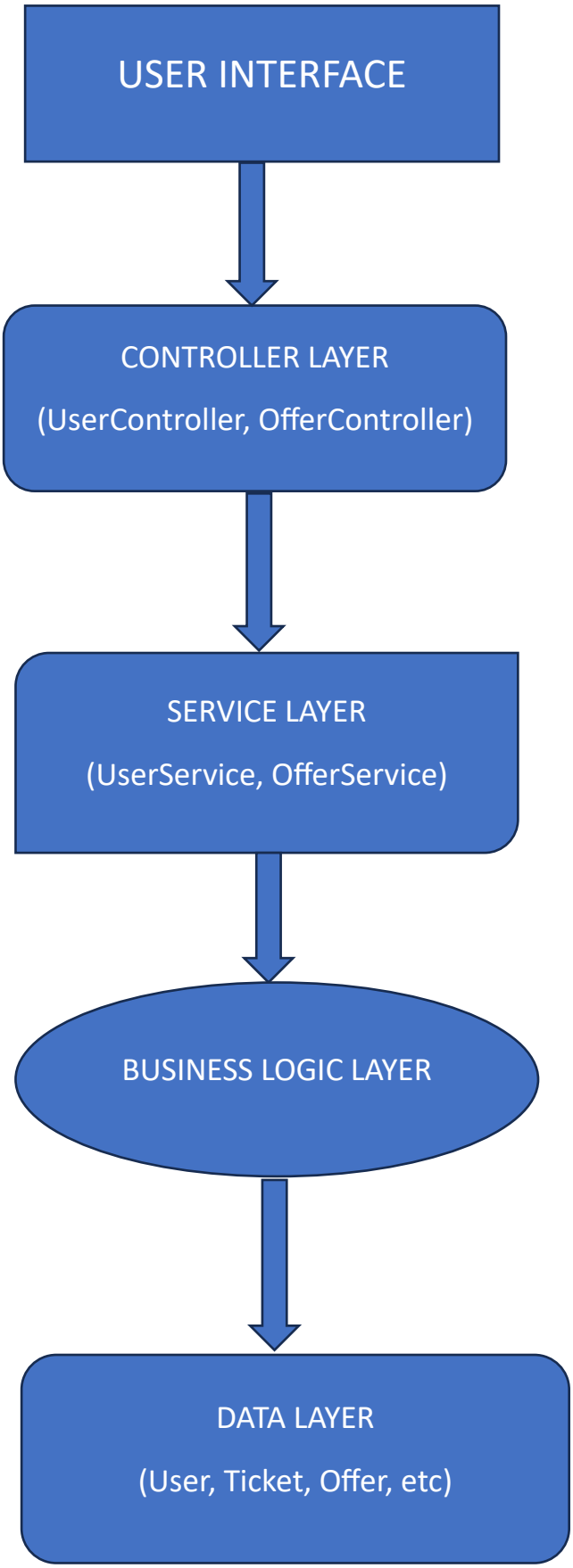
(https://www.thymeleaf.org/). Both will be given same credit.

3. You may or may not use HTTP sessions. It is optional here.

4. CRUD service for the special deals should be given. An admin would typically create or update a deal.

5. Any relational database would be preferred, for example, H2 or MySQL.

Architectural Diagram:-



Main Features:

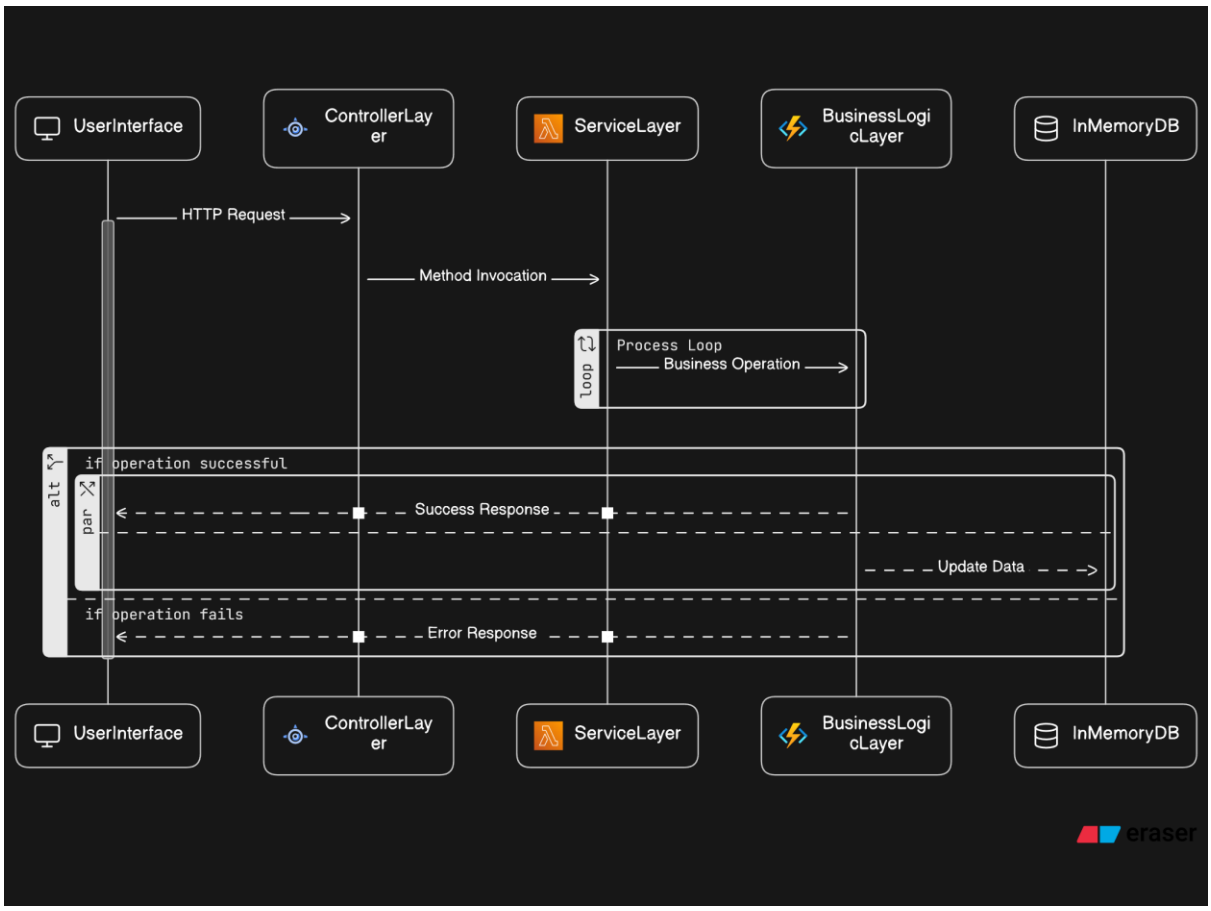
1. **User Management:**
    - Users can sign up and log in to the application using their phone number and password.
    - User accounts are stored in memory, allowing users to access their reservations and book tickets.
2. **Ticket Booking:**
    - Users can browse available offers and book tickets for desired flights.
    - Upon booking, tickets are associated with the user's account and stored in their reservation list.
3. **Offer Management:**
    - Admin users can add new flight offers, specifying details such as origin, destination, and cost.
    - Offers are stored in memory and can be retrieved for users to browse and book.
4. **Exception Handling:**
    - The application includes robust exception handling mechanisms to handle errors and exceptions gracefully.
    - Different types of exceptions, such as invalid login credentials or duplicate ticket bookings, are appropriately handled and communicated to users.

Design Pattern:

Based on the provided code and its structure, the application appears to follow the **MVC (Model-View-Controller)** design pattern.

1. **Model (Entity Classes):**
    - The entity classes (**Offer**, **Ticket**, **User**) represent the data model of the application.
    - They encapsulate data and behavior related to specific entities such as flight offers, tickets, and user accounts.
2. **View (User Interface Layer):**
    - The user interface layer consists of controller classes (**UserController**, **OfferController**) responsible for handling incoming HTTP requests and generating appropriate responses.
    - These controllers interact with the service layer to execute business logic and retrieve data for presentation to users.
3. **Controller (Service Layer):**
    - The service layer contains service classes (**UserService**, **OfferService**) that encapsulate application logic and orchestrate interactions between the controllers and the business logic layer.
    - Service classes handle operations such as user authentication, ticket booking, offer management, and data retrieval.
4. **Business Logic (Business Logic Layer):**
    - The business logic layer contains the core logic and rules of the application.
    - It includes methods for ticket booking, offer management, user authentication, and other business operations.
5. **Exception Handling (Exception Handling Layer):**
    - Although not explicitly defined as a separate layer, the application incorporates exception handling mechanisms to manage errors and exceptions gracefully.

COMPONENT INTERACTION DIAGRAM:-



CODE SNIPPETS:-

Entity Classes –

User.java

```
package com.example.flight.Entity;

import java.util.List;

import com.fasterxml.jackson.annotation.JsonProperty;


public class User {

	String email;

	@JsonProperty("password")

	String password;

	@JsonProperty("phoneNumber")
```

```
String phoneNumber;
@JsonProperty("reservations")
List<Ticket> reservations;
@JsonProperty("isAdmin")
boolean isAdmin;

public String getPhoneNumber() {
        return phoneNumber;
}
public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
}
public List<Ticket> getReservations() {
        return reservations;
}
public void setReservations(List<Ticket> reservations) {
        this.reservations = reservations;
}
public String getPhone() {
        return phoneNumber;
}
public void setPhone(String phone) {
        this.phoneNumber = phone;
}
public String getEmail() {
        return email;
}
public void setEmail(String email) {
        this.email = email;
}
public String getPassword() {
```

```
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
	public boolean isAdmin() {
		return isAdmin;
	}
	public void setAdmin(boolean isAdmin) {
		this.isAdmin = isAdmin;
	}

	public void deleteTicketUsingTicketId(String TicketId) {
		for (Ticket ticket : reservations) {
			if(ticket.ticketId.equals(TicketId)) {
				reservations.remove(ticket);
			}
		}
	}

	@Override
	public String toString() {
		return "User [email=" + email + ", password=" + password + ", phone=" +
phoneNumber + ", isAdmin=" + isAdmin + "]";
	}

}
```

Ticket.java

```
package com.example.flight.Entity;
import java.util.*;
```

```
import javax.persistence.Entity;

import com.fasterxml.jackson.annotation.JsonProperty;
import java.util.UUID;

@Entity
public class Ticket {
	@JsonProperty("ticketId")
	String ticketId;
	@JsonProperty("offer")
	Offer offer;
	@JsonProperty("date")
	String date;
	@JsonProperty("passengers")
	List<Passenger> passengers;

	public Ticket() {
		this.ticketId = UUID.randomUUID().toString();
		this.date = "1-1-2024";
	}

	public Offer getOffer() {
		return offer;
	}
	public void setOffer(Offer offer) {
		this.offer = offer;
	}
	public String getDate() {
		return date;
	}
	public void setDate(String date) {
```

```
		this.date = date;
	}
	public List<Passenger> getPassengers() {
		return passengers;
	}
	public void setPassengers(List<Passenger> passengers) {
		this.passengers = passengers;
	}
	public String getTicketId() {
		return ticketId;
	}
	public void setTicketId(String ticketId) {
		this.ticketId = ticketId;
	}
	@Override
	public String toString() {
		return "Ticket [ticketId=" + ticketId + ", offer=" + offer + ", date=" + date + ",
passengers=" + passengers
				+ "]";
	}
}
```

Offer.java

```
package com.example.flight.Entity;

public class Offer {
	String offerId;
	int cost;
	String origin;
	String destination;

	public String getOfferId() {
```

```
		return offerId;
	}
	public void setOfferId(String id) {
		this.offerId = id;
	}
	public int getCost() {
		return cost;
	}
	public void setCost(int price) {
		this.cost = price;
	}
	public String getOrigin() {
		return origin;
	}
	public void setOrigin(String origin) {
		this.origin = origin;
	}
	public String getDestination() {
		return destination;
	}
	public void setDestination(String destination) {
		this.destination = destination;
	}
}
```

Service Classes –

OfferService.java

```
package com.example.flight.Service;

import java.util.ArrayList;
import java.util.HashMap;
```

```
import java.util.List;
import java.util.Map;

import org.springframework.stereotype.Service;
import org.springframework.util.ObjectUtils;

import com.example.flight.Entity.Offer;
import com.example.flight.Exceptions.TicketAlreadyExistsException;
import com.example.flight.Exceptions.TicketNotFoundException;

@Service
public class OfferService {
	List<Offer> offerList= new ArrayList<Offer>();
	Map<String, Offer> offerMap = new HashMap<String, Offer>();

	public void addOffer(Offer offer) {
		if(offerMap.containsKey(offer.getOfferId())) {
			throw new TicketAlreadyExistsException("Ticket Exists");
		}
		else {
			offerList.add(offer);
			offerMap.put(offer.getOfferId(), offer);
		}
	}

	public void addAllOffer(List<Offer> offerlist) {
		for(Offer offer:offerlist) {
			if(offerMap.containsKey(offer.getOfferId())) {
				throw new TicketAlreadyExistsException("Ticket Exists");
			}
			else {
```

```
                offerList.add(offer);
                offerMap.put(offer.getOfferId(), offer);
            }
        }
    }

    public List<Offer> getAllOffer() {
        return offerList;
    }

    /** This method retrieves a ticket based on its unique id from ticketmap. **/
    public Offer getOfferById(String offerId) {
        if(ObjectUtils.isEmpty(offerMap.get(offerId))) {
            System.out.println("Ticket not found");
        }
        return offerMap.get(offerId);
    }

    public void deleteOffer(String offerId) {
        if(ObjectUtils.isEmpty(offerMap.get(offerId))) {
            throw new TicketNotFoundException("Ticket Doesn't Exists");
        }
        else {
            Offer offer = getOfferById(offerId);
            offerList.remove(offer);
            offerMap.remove(offerId);
        }
    }
    public int getPriceByOfferId(String offerId) {
        if(ObjectUtils.isEmpty(offerMap.get(offerId))) {
            throw new TicketNotFoundException("Ticket Doesn't Exists");
```

```
                }
                else {
                        Offer offer = getOfferById(offerId);
                        return offer.getCost();
                }
        }
}
```

UserService.java

```
package com.example.flight.Service;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.stereotype.Service;
import org.springframework.util.ObjectUtils;

import com.example.flight.Entity.Ticket;
import com.example.flight.Entity.User;
import com.example.flight.Exceptions.TicketAlreadyExistsException;
import com.example.flight.Exceptions.TicketNotFoundException;

@Service
public class UserService {
   private final Map<String, User> userMap = new HashMap<>();

   public void addUser(User user) {
      if (userMap.containsKey(user.getPhone())) {
         throw new TicketAlreadyExistsException("User Exists");
```

```
    }
    userMap.put(user.getPhone(), user);
}

public List<User> getAllUsers() {
    return new ArrayList<>(userMap.values());
}

public User getUserByPhone(String phone) {
    return userMap.get(phone);
}

public void deleteUser(String phone) {
    if (!userMap.containsKey(phone)) {
        throw new TicketNotFoundException("User Doesn't Exist");
    }
    userMap.remove(phone);
}

public boolean checkUserExists(String phone) {
    return userMap.containsKey(phone);
}

// Ticket booking related methods

public void bookTicket(Ticket ticket, String phoneNumber) {
    User user = userMap.get(phoneNumber);
    if (user == null) {
        System.out.println("User not found");
        return;
    }
```

```
        if (user.getReservations() == null) {
            user.setReservations(new ArrayList<>());
        }
        user.getReservations().add(ticket);
    }

    public List<Ticket> getAllTickets(String phoneNumber) {
        User user = userMap.get(phoneNumber);
        return user != null ? user.getReservations() : Collections.emptyList();
    }

    public void deleteTicket(String ticketId, String phoneNumber) {
        User user = userMap.get(phoneNumber);
        if (user != null && user.getReservations() != null) {
            user.getReservations().removeIf(ticket -> ticket.getTicketId().equals(ticketId));
        } else {
            System.out.println("User or ticket not found");
        }
    }
}
```

Controller Classes –

OfferController.java

```
package com.example.flight.Controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.example.flight.Entity.Offer;
import com.example.flight.Service.OfferService;

@RestController
@RequestMapping("/offer")
@CrossOrigin(origins = "*")
public class OfferController {

	@Autowired
	OfferService offerService;

	@PostMapping("/addoffer")
	public void addOffer(@RequestBody Offer offer) {
		offerService.addOffer(offer);
	}

	@PostMapping("/addalloffers")
	public void addOffers(@RequestBody List<Offer> offerList) {
		offerService.addAllOffer(offerList);
	}

	@GetMapping("/alloffer")
	public List<Offer> getAllOffers() {
		return offerService.getAllOffer();
	}
```

```
	@GetMapping("/{offerId}")
	public Offer getOfferById(@PathVariable String offerId) {
		return offerService.getOfferById(offerId);
	}

	@DeleteMapping("/{offerId}")
	public void deleteOfferById(@PathVariable String offerId) {
		offerService.deleteOffer(offerId);
	}
}
```

UserController.java

```
package com.example.flight.Controller;

import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.example.flight.Entity.Ticket;
import com.example.flight.Entity.User;
import com.example.flight.Entity.WrappedTicket;
```

```
import com.example.flight.Entity.WrappedUser;
import com.example.flight.Exceptions.InvalidLoginCredentialException;
import com.example.flight.Exceptions.UserNotFoundException;
import com.example.flight.Service.UserService;

@RestController
@RequestMapping("/user")
@CrossOrigin(origins = "*")
public class UserController {

	@Autowired
	UserService userService;

	@PostMapping("/signup")
	public WrappedUser signUp(@RequestBody User user) {
		userService.addUser(user);
		WrappedUser wUser = new WrappedUser();
		wUser.setAuth(true);
		wUser.setAdmin(user.isAdmin());
		wUser.setEmail(user.getEmail());
		wUser.setPassword(null);
		wUser.setPhone(user.getPhone());
		System.out.println(wUser.toString());
		return wUser;
	}

	@PostMapping("/login")
	public WrappedUser logIn(@RequestBody Map<String, Object> requestBody) {
		String phoneNumber = (String) requestBody.get("phoneNumber");
	String password = (String) requestBody.get("password");
	boolean isAdmin = (boolean) requestBody.get("isAdmin");
```

```
    WrappedUser wUser = new WrappedUser();
if(userService.checkUserExists(phoneNumber)) {
    if(password.equals(userService.getUserByPhone(phoneNumber).getPassword())) {
            //wUser.setEmail(email);
            wUser.setPassword(null);
            wUser.setAdmin(isAdmin);
            wUser.setAuth(true);
            return wUser;
    }
   else {
    throw new InvalidLoginCredentialException("Invalid Login Credentials!");
   }
}
else {
    throw new UserNotFoundException("User Not Found!");
}


    }


    @GetMapping("/allusers")
    public List<User> getAllUsers() {
            return userService.getAllUsers();
    }


    @GetMapping("/{phoneNumber}")
    public User getUserByEmail(@PathVariable String phoneNumber) {
            return userService.getUserByPhone(phoneNumber);
    }


    @DeleteMapping("/{phoneNumber}")
```

```
	public void deleteUserByEmail(@PathVariable String phoneNumber) {
		userService.deleteUser(phoneNumber);
	}

	// related to ticketBooking

	@PostMapping("/book")
	public void bookTicket(@RequestBody WrappedTicket wTicket) {
		wTicket.toString();
		Ticket ticket = new Ticket();
		ticket.setDate(wTicket.getDate());
		ticket.setOffer(wTicket.getOffer());
		ticket.setPassengers(wTicket.getPassengers());
		ticket.setTicketId(wTicket.getTicketId());
		userService.bookTicket(ticket, wTicket.getPhoneNumber());
	}

	@GetMapping("/myreservation")
	public List<Ticket> getAllTickets(@RequestParam("phoneNumber") String phoneNumber) {
		return userService.getAllTickets(phoneNumber);
	}

	@DeleteMapping("/myreservation/{ticketId}")
	public void deleteTicketById(@RequestBody Map<String, Object> requestBody) {
		String ticketId = (String)requestBody.get("ticketId");
		String phoneNumber = (String) requestBody.get("phoneNumber");
		userService.deleteTicket(ticketId, phoneNumber);
	}
}
```

Discussion

System Performance:

The Ticket Booking Application demonstrates satisfactory performance during testing. Response times for common actions such as signing up, logging in, and booking tickets are within acceptable limits. Scalability may be a concern as user traffic increases.

Challenges Faced:

Several challenges were encountered during the development of the Ticket Booking Application. Integrating external APIs for fetching flight data posed initial difficulties due to unfamiliarity with API documentation and authentication mechanisms. Additionally, ensuring data consistency and concurrency in a multi-user environment required careful consideration and implementation of transaction management strategies.

Future Improvements:

Moving forward, several enhancements could be made to the Ticket Booking Application to further enhance its functionality and usability. Integrating additional features such as seat selection , and payment processing would provide users with a more comprehensive booking experience. Improving accessibility and mobile responsiveness would cater to a broader range of users and devices.