

[prajwalhonashetti143@gmail.com](mailto:prajwalhonashetti143@gmail.com)

## Development Scenario: Smart City Transportation Management System

### Day 1: HTML, CSS, and JavaScript - User Interface for Route Planning

Task 1: Build the HTML structure for the city's transportation route planner interface.

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>City Transportation Route Planner</title>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

  <header>

    <h1>City Transportation Route Planner</h1>

  </header>

  <main>

    <section class="route-selection">

      <form id="routeForm">

        <label for="startPoint">Start Point:</label>

        <select id="startPoint" name="startPoint">

          <option value="">Select Start Point</option>

          <option value="A">Point A</option>

          <option value="B">Point B</option>

          <option value="C">Point C</option>

        </select>

        <label for="endPoint">End Point:</label>
```

```

        <select id="endPoint" name="endPoint">
            <option value="">Select End Point</option>
            <option value="X">Point X</option>
            <option value="Y">Point Y</option>
            <option value="Z">Point Z</option>
        </select>

        <button type="button" onclick="getRoutes()">Get Routes</button>
    </form>
</section>
<section class="route-options" id="routeOptions">
    <!-- Dynamic route options will be displayed here -->
</section>
</main>
<script src="scripts.js"></script>
</body>
</html>

```

**Task 2: Style the planner interface with CSS for a user-friendly experience across multiple devices.**

```
/* styles.css */
```

```

body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

```

```
header {
```

```
background-color: #0044cc;
color: white;
text-align: center;
padding: 1rem 0;
}
```

```
main {
  padding: 1rem;
}
```

```
.route-selection {
  margin-bottom: 1rem;
}
```

```
form {
  display: flex;
  flex-direction: column;
  gap: 1rem;
  max-width: 400px;
  margin: auto;
}
```

```
label {
  font-weight: bold;
}
```

```
select, button {
  padding: 0.5rem;
  font-size: 1rem;
```

```
}
```

```
button {  
    background-color: #0044cc;  
    color: white;  
    border: none;  
    cursor: pointer;  
}
```

```
button:hover {  
    background-color: #003399;  
}
```

```
.route-options {  
    max-width: 600px;  
    margin: auto;  
}
```

```
.route-option {  
    background-color: #f9f9f9;  
    border: 1px solid #ddd;  
    padding: 1rem;  
    margin-bottom: 0.5rem;  
    border-radius: 5px;  
}
```

**Task 3: Implement JavaScript to dynamically update route options based on user selections.**

```
// scripts.js
```

```
function getRoutes() {
```

```
const startPoint = document.getElementById('startPoint').value;
const endPoint = document.getElementById('endPoint').value;
const routeOptions = document.getElementById('routeOptions');
```

```
routeOptions.innerHTML = '';
```

```
if (startPoint && endPoint) {
```

```
  const routes = [
```

```
    { start: 'A', end: 'X', route: 'Route 1: A -> B -> C -> X' },
```

```
    { start: 'A', end: 'Y', route: 'Route 2: A -> C -> Y' },
```

```
    { start: 'B', end: 'X', route: 'Route 3: B -> C -> X' },
```

```
    { start: 'B', end: 'Z', route: 'Route 4: B -> A -> Z' },
```

```
    { start: 'C', end: 'Y', route: 'Route 5: C -> B -> Y' }
```

```
  ];
```

```
  const filteredRoutes = routes.filter(route => route.start === startPoint && route.end ===
endPoint);
```

```
  if (filteredRoutes.length > 0) {
```

```
    filteredRoutes.forEach(route => {
```

```
      const routeOption = document.createElement('div');
```

```
      routeOption.className = 'route-option';
```

```
      routeOption.textContent = route.route;
```

```
      routeOptions.appendChild(routeOption);
```

```
    });
```

```
  } else {
```

```
    routeOptions.innerHTML = '<p>No routes available for the selected points.</p>';
```

```
  }
```

```
  } else {
```

```
    routeOptions.innerHTML = '<p>Please select both start and end points.</p>';
```

```
}  
}
```

- **HTML Structure:** We created the basic structure with a header, a form for selecting routes, and a section to display the route options.
- **CSS Styling:** We added styles to make the interface user-friendly and responsive.
- **JavaScript Functionality:** We implemented a function to dynamically update route options based on user selections.

## Day 2: JavaScript/Bootstrap - Interactive Transit Maps

Task 1: Integrate Bootstrap to develop a responsive layout for interactive transit maps.

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>Interactive Transit Maps</title>
```

```
  <link                                     rel="stylesheet"  
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
```

```
  <link rel="stylesheet" href="styles.css">
```

```
</head>
```

```
<body>
```

```
  <div class="container">
```

```
    <header class="text-center my-4">
```

```
      <h1>Interactive Transit Maps</h1>
```

```
    </header>
```

```
    <main>
```

```
      <div class="row">
```

```
        <div class="col-md-8">
```

```
          <div id="transitMap" class="border rounded" style="height: 500px;">
```

```
            <!-- Map will be embedded here -->
```

```
          </div>
```

```

</div>

<div class="col-md-4">

  <h3>Transit Information</h3>

  <ul class="list-group" id="transitInfo">

    <!-- Real-time transit information will be listed here -->

  </ul>

</div>

</div>

</main>

</div>

<!-- Bootstrap JS and dependencies -->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.3/dist/umd/popper.min.js"></scr
ipt>

<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

<script src="scripts.js"></script>

</body>

</html>

```

**Task 2: Use Bootstrap components to display real-time transit data in modals and tooltips.**

```

<!-- Add to the body section of the HTML -->

<!-- Transit Data Modal -->

<div class="modal fade" id="transitDataModal" tabindex="-1" aria-
labelledby="transitDataModalLabel" aria-hidden="true">

  <div class="modal-dialog modal-lg">

    <div class="modal-content">

      <div class="modal-header">

        <h5 class="modal-title" id="transitDataModallLabel">Transit Data</h5>

```

```

        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="modal-body">
        <!-- Real-time transit data will be displayed here -->
        <div id="transitDataContent"></div>
    </div>
    <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-
dismiss="modal">Close</button>
    </div>
</div>
</div>
</div>

```

**Task 3: Write JavaScript to handle live updates of transit statuses and to interact with the map.**

**// scripts.js**

```

document.addEventListener("DOMContentLoaded", function () {

    function fetchTransitData() {
        const transitData = [
            { id: 1, name: "Bus 101", status: "On Time", location: "Main St & 5th Ave" },
            { id: 2, name: "Tram A", status: "Delayed", location: "Central Station" },
            { id: 3, name: "Bus 202", status: "On Time", location: "Broadway & 7th Ave" }
        ];

        const transitInfo = document.getElementById("transitInfo");
    }

```



```

transitInfo.innerHTML = "";

transitData.forEach(data => {
    const listItem = document.createElement("li");
    listItem.className = "list-group-item d-flex justify-content-between align-items-center";
    listItem.textContent = `${data.name} - ${data.status}`;
    listItem.setAttribute("data-toggle", "tooltip");
    listItem.setAttribute("data-placement", "right");
    listItem.setAttribute("title", `Location: ${data.location}`);

    listItem.addEventListener("click", () => {
        document.getElementById("transitDataContent").textContent = `Transit:
${data.name}\nStatus: ${data.status}\nLocation: ${data.location}`;
        $('#transitDataModal').modal('show');
    });

    transitInfo.appendChild(listItem);
});

 $('[data-toggle="tooltip"]').tooltip();
}

function initMap() {
    const map = L.map('transitMap').setView([51.505, -0.09], 13);
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        maxZoom: 19
    }).addTo(map);
    const marker = L.marker([51.505, -0.09]).addTo(map)
        .bindPopup('A pretty CSS3 popup.<br> Easily customizable.')
        .openPopup();
}

```

```

    }

    fetchTransitData();

    initMap();

    setInterval(fetchTransitData, 60000);

});

```

**Responsive Layout with Bootstrap:** We integrated Bootstrap to create a responsive layout for the interactive transit maps.

**Bootstrap Components:** We used Bootstrap modals and tooltips to display real-time transit data.

**JavaScript Functionality:** We implemented JavaScript to handle live updates of transit statuses and interact with the map.

### Day 3: Servlet/JSP, Introduction to JSP - Traffic Data Processing

**Task 1: Create Servlets to process real-time traffic data and user queries.**

#### 1.1 TrafficDataServlet

This servlet will simulate the retrieval and processing of real-time traffic data.

```

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/trafficData")

public class TrafficDataServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        List<TrafficData> trafficDataList = getTrafficData();

        request.setAttribute("trafficData", trafficDataList);

```

```

        request.getRequestDispatcher("/trafficData.jsp").forward(request, response);
    }

    private List<TrafficData> getTrafficData() {
        List<TrafficData> trafficDataList = new ArrayList<>();
        trafficDataList.add(new TrafficData("Highway 1", "Heavy", "Accident at Exit 14"));
        trafficDataList.add(new TrafficData("Downtown Blvd", "Moderate", "Roadwork near
5th Ave"));
        trafficDataList.add(new TrafficData("Route 66", "Light", "No incidents"));

        return trafficDataList;
    }
}

```

## 1.2 UserQueryServlet

This servlet will handle user queries for alternative routes based on current traffic conditions.

```

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/userQuery")
public class UserQueryServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        String startPoint = request.getParameter("startPoint");
        String endPoint = request.getParameter("endPoint");
        String alternativeRoute = findAlternativeRoute(startPoint, endPoint);
    }
}

```

```
request.setAttribute("alternativeRoute", alternativeRoute);  
request.getRequestDispatcher("/userQueryResult.jsp").forward(request, response);  
}
```

```
private String findAlternativeRoute(String startPoint, String endPoint) {  
    return "Alternative route from " + startPoint + " to " + endPoint + " via Route B";  
}  
}
```

**Task 2: Use JSP to present dynamic traffic information and alternative routes.**

### **2.1 trafficData.jsp**

This JSP will display the real-time traffic information retrieved by TrafficDataServlet.

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>  
  
<html>  
  
<head>  
    <title>Real-Time Traffic Data</title>  
</head>  
  
<body>  
    <h1>Real-Time Traffic Data</h1>  
    <ul>  
        <c:forEach var="data" items="${trafficData}">  
            <li>${data.roadName}: ${data.trafficCondition} - ${data.incidentDescription}</li>  
        </c:forEach>  
    </ul>  
</body>  
</html>
```

## 2.2 userQueryResult.jsp

This JSP will display the result of the user query for alternative routes.

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Alternative Route</title>
</head>
<body>
    <h1>Alternative Route</h1>
    <p>${alternativeRoute}</p>
</body>
</html>
```

Task 3: Leverage JavaBeans to store and manage traffic data and user preferences.

### 3.1 TrafficData JavaBean

This JavaBean will store traffic data.

```
public class TrafficData {
    private String roadName;
    private String trafficCondition;
    private String incidentDescription;

    public TrafficData(String roadName, String trafficCondition, String incidentDescription) {
        this.roadName = roadName;
        this.trafficCondition = trafficCondition;
        this.incidentDescription = incidentDescription;
    }
}
```

```
public String getRoadName() {  
    return roadName;  
}
```

```
public void setRoadName(String roadName) {  
    this.roadName = roadName;  
}
```

```
public String getTrafficCondition() {  
    return trafficCondition;  
}
```

```
public void setTrafficCondition(String trafficCondition) {  
    this.trafficCondition = trafficCondition;  
}
```

```
public String getIncidentDescription() {  
    return incidentDescription;  
}
```

```
public void setIncidentDescription(String incidentDescription) {  
    this.incidentDescription = incidentDescription;  
}  
}
```

### 3.2 UserPreferences JavaBean

- This JavaBean will store user preferences.

```
public class UserPreferences {  
    private String preferredStartPoint;
```

```

private String preferredEndPoint;

public String getPreferredStartPoint() {
    return preferredStartPoint;
}

public void setPreferredStartPoint(String preferredStartPoint) {
    this.preferredStartPoint = preferredStartPoint;
}

public String getPreferredEndPoint() {
    return preferredEndPoint;
}

public void setPreferredEndPoint(String preferredEndPoint) {
    this.preferredEndPoint = preferredEndPoint;
}
}

```

## Day 4: Spring Core - System Configuration and User Management

Task 1: Configure Spring Beans for user management and session handling.

### 1.1 Define User Management Beans

User.java (Entity)

```

import javax.persistence.*;
import java.util.Set;

```

@Entity

@Table(name = "users")

```

public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "username", nullable = false, unique = true)
    private String username;

    @Column(name = "password", nullable = false)
    private String password;

    @Column(name = "roles", nullable = false)
    private String roles;

    // Getters and Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
    public String getRoles() { return roles; }
    public void setRoles(String roles) { this.roles = roles; }
}

```

- UserRepository.java (Repository)

```

import org.springframework.data.jpa.repository.JpaRepository;

```

```

public interface UserRepository extends JpaRepository<User, Long> {

    User findByUsername(String username);
}

```



```
}
```

- UserService.java (Service)

```
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;
```

```
@Service
```

```
public class UserService {
```

```
    @Autowired
```

```
    private UserRepository userRepository;
```

```
    @Autowired
```

```
    private BCryptPasswordEncoder passwordEncoder;
```

```
    public User registerUser(User user) {
```

```
        user.setPassword(passwordEncoder.encode(user.getPassword()));
```

```
        return userRepository.save(user);
```

```
    }
```

```
    public User findByUsername(String username) {
```

```
        return userRepository.findByUsername(username);
```

```
    }
```

```
}
```

- SecurityConfig.java (Configuration)

```
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import
org.springframework.security.config.annotation.web.builders.AuthenticationManagerBuilder;

import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
```

**@Configuration**

**@EnableWebSecurity**

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {
```

**@Autowired**

```
private UserService userService;
```

**@Bean**

```
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

**@Override**

```
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(username -> {
        User user = userService.findByUsername(username);
        if (user != null) {
            return org.springframework.security.core.userdetails.User
                .withUsername(user.getUsername())
```

```

        .password(user.getPassword())
        .roles(user.getRoles().split(","))
        .build();
    } else {
        throw new UsernameNotFoundException("User not found");
    }
});
}

```

**@Override**

```

protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .antMatchers("/admin/**").hasRole("ADMIN")
            .antMatchers("/user/**").hasRole("USER")
            .anyRequest().permitAll()
        .and()
        .formLogin()
            .loginPage("/login")
            .permitAll()
        .and()
        .logout()
            .permitAll();
    }
}

```

## 1.2 Configure Session Handling

- SessionConfig.java (Configuration)

```

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

```

```
import
org.springframework.session.data.redis.config.annotation.web.http.EnableRedisHttpSessi
on;
```

**@Configuration**

**@EnableRedisHttpSession**

```
public class SessionConfig {
```

**@Bean**

```
    public LettuceConnectionFactory redisConnectionFactory() {
        return new LettuceConnectionFactory("localhost", 6379);
    }
}
```

**Task 2: Set up Spring's Dependency Injection to manage services related to traffic data.**

### **2.1 Define Traffic Data Beans**

- **TrafficService.java (Service)**

```
import org.springframework.stereotype.Service;
```

**@Service**

```
public class TrafficService {

    public String getTrafficData(String route) {
        // Simulate traffic data retrieval
        return "Traffic data for route: " + route;
    }
}
```

- **TrafficController.java (Controller)**

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

```
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
```

```
public class TrafficController {
```

```
    @Autowired
```

```
    private TrafficService trafficService;
```

```
    @GetMapping("/traffic")
```

```
    public String getTraffic(@RequestParam String route) {
```

```
        return trafficService.getTrafficData(route);
```

```
    }
```

```
}
```

## 2.2 Configure Dependency Injection

Spring automatically manages dependency injection via annotations like `@Autowired`, `@Service`, and `@Controller`. The configuration above ensures that dependencies are injected where needed, such as injecting `TrafficService` into `TrafficController`.

**Task 3: Establish a secure Application Context for user data processing.**

### 3.1 Configure Security Context

- `ApplicationContextConfig.java` (Configuration)

```
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.security.core.context.SecurityContext;
```

```
import org.springframework.security.core.context.SecurityContextHolder;
```

```
@Configuration
```

```
public class ApplicationContextConfig {
```

```
    @Bean
```

```

public SecurityContext securityContext() {
    return SecurityContextHolder.getContext();
}
}

```

- UserSecurityService.java (Service for Security Context)

```

import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Service;

```

**@Service**

```

public class UserSecurityService {
    public String getCurrentUser() {
        return SecurityContextHolder.getContext().getAuthentication().getName();
    }
}

```

### 3.2 Secure User Data Processing

- UserController.java (Controller)

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

```

**@RestController**

```

public class UserController {

```

**@Autowired**

```

    private UserSecurityService userSecurityService;

```

**@GetMapping("/user/me")**

```

    public String getCurrentUser() {

```

```
        return "Current user: " + userSecurityService.getCurrentUser();
    }
}
```

## Day 5: Spring MVC - Administration Portal for Transit Management

Task 1: Utilize Spring MVC to create an admin portal for transit officials to manage routes and schedules.

### 1.1 Set Up Spring MVC

- First, set up your Spring Boot project with the necessary dependencies. Add these dependencies to your pom.xml:

pom.xml:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
  </dependency>
</dependencies>
```

## 1.2 Define the Model

TransitRoute.java

```
import javax.persistence.*;
```

```
import java.util.Set;
```

```
@Entity
```

```
@Table(name = "transit_route")
```

```
public class TransitRoute {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    @Column(name = "route_name", nullable = false)
```

```
    private String routeName;
```

```
    @OneToMany(mappedBy = "transitRoute", cascade = CascadeType.ALL, orphanRemoval  
= true)
```

```
    private Set<Schedule> schedules;
```

```
    // Getters and Setters
```

```
}
```

Schedule.java

```
import javax.persistence.*;
```

```
import java.time.LocalDateTime;
```

```
@Entity
```

```
@Table(name = "schedule")
```

```
public class Schedule {
```

```
    @Id
```



```
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
private Long id;
```

```
@Column(name = "departure_time", nullable = false)
```

```
private LocalDateTime departureTime;
```

```
@ManyToOne(fetch = FetchType.LAZY)
```

```
@JoinColumn(name = "route_id", nullable = false)
```

```
private TransitRoute transitRoute;
```

```
// Getters and Setters
```

```
public Long getId() { return id; }
```

```
public void setId(Long id) { this.id = id; }
```

```
public LocalDateTime getDepartureTime() { return departureTime; }
```

```
public void setDepartureTime(LocalDateTime departureTime) { this.departureTime =  
departureTime; }
```

```
public TransitRoute getTransitRoute() { return transitRoute; }
```

```
public void setTransitRoute(TransitRoute transitRoute) { this.transitRoute = transitRoute;  
}
```

```
}
```

### 1.3 Create Repositories

TransitRouteRepository.java

```
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface TransitRouteRepository extends JpaRepository<TransitRoute, Long> {  
}
```

ScheduleRepository.java

java

Copy code

```
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface ScheduleRepository extends JpaRepository<Schedule, Long> {  
}
```

#### 1.4 Create Controllers

- AdminController.java

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.ui.Model;
```

```
import org.springframework.web.bind.annotation.*;
```

```
import java.time.LocalDateTime;
```

```
import java.util.Optional;
```

```
@Controller
```

```
@RequestMapping("/admin")
```

```
public class AdminController {
```

```
    private final TransitRouteRepository routeRepo;
```

```
    private final ScheduleRepository scheduleRepo;
```

```
    public AdminController(TransitRouteRepository routeRepo, ScheduleRepository  
scheduleRepo) {
```

```
        this.routeRepo = routeRepo;
```

```
        this.scheduleRepo = scheduleRepo;
```

```
    }
```

```
@GetMapping("/routes")
```

```
public String listRoutes(Model model) {
```

```
    model.addAttribute("routes", routeRepo.findAll());
```

```
    return "routes";
```

```
}
```

```
@GetMapping("/routes/add")  
  
public String showAddRouteForm(Model model) {  
    model.addAttribute("route", new TransitRoute());  
    return "addRoute";  
}
```

```
@PostMapping("/routes")  
  
public String addRoute(@ModelAttribute TransitRoute route) {  
    routeRepo.save(route);  
    return "redirect:/admin/routes";  
}
```

```
@GetMapping("/schedules")  
  
public String listSchedules(Model model) {  
    model.addAttribute("schedules", scheduleRepo.findAll());  
    return "schedules";  
}
```

```
@GetMapping("/schedules/add")  
  
public String showAddScheduleForm(Model model) {  
    model.addAttribute("schedules", scheduleRepo.findAll());  
    model.addAttribute("routes", routeRepo.findAll());  
    model.addAttribute("schedule", new Schedule());  
    return "addSchedule";  
}
```

```
@PostMapping("/schedules")  
  
public String addSchedule(@ModelAttribute Schedule schedule) {  
    scheduleRepo.save(schedule);  
}
```

```
        return "redirect:/admin/schedules";
    }
}
```

```
@GetMapping("/routes/edit/{id}")
public String showEditRouteForm(@PathVariable("id") Long id, Model model) {
    Optional<TransitRoute> route = routeRepo.findById(id);
    if (route.isPresent()) {
        model.addAttribute("route", route.get());
        return "editRoute";
    }
    return "redirect:/admin/routes";
}
```

```
@PostMapping("/routes/{id}")
public String updateRoute(@PathVariable("id") Long id, @ModelAttribute TransitRoute
route) {
    route.setId(id);
    routeRepo.save(route);
    return "redirect:/admin/routes";
}
}
```

## 1.5 Create Thymeleaf Templates

templates/routes.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Transit Routes</title>
</head>
<body>
<h1>Transit Routes</h1>
```

```
<a href="/admin/routes/add">Add New Route</a>
```

```
<table>
```

```
  <thead>
```

```
    <tr>
```

```
      <th>Route Name</th>
```

```
      <th>Actions</th>
```

```
    </tr>
```

```
  </thead>
```

```
  <tbody>
```

```
    <tr th:each="route : ${routes}">
```

```
      <td th:text="${route.routeName}"></td>
```

```
      <td>
```

```
        <a th:href="@{/admin/routes/edit/{id}(id=${route.id})}">Edit</a>
```

```
      </td>
```

```
    </tr>
```

```
  </tbody>
```

```
</table>
```

```
</body>
```

```
</html>
```

- templates/addRoute.html

```
<!DOCTYPE html>
```

```
<html xmlns:th="http://www.thymeleaf.org">
```

```
<head>
```

```
  <title>Add Route</title>
```

```
</head>
```

```
<body>
```

```
<h1>Add New Route</h1>
```

```
<form action="#" th:action="@{/admin/routes}" th:object="${route}" method="post">
```

```
<label for="routeName">Route Name:</label>
<input type="text" id="routeName" th:field="*{routeName}" />
<button type="submit">Add Route</button>
</form>
</body>
</html>
```

**Task 2: Integrate Thymeleaf with Spring MVC for real-time updates and schedule changes.**

### **2.1 Integrate Thymeleaf**

Ensure Thymeleaf is properly set up in your project for rendering views. Thymeleaf will handle dynamic updates and view rendering.

### **2.2 Real-Time Updates**

To achieve real-time updates, you can use JavaScript and WebSocket. Thymeleaf templates will render initial data, and WebSocket can push updates to the browser.

- **Add WebSocket Configuration**

**WebSocketConfig.java**

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.socket.server.support.WebSocketHandlerAdapter;
import org.springframework.web.socket.server.support.WebSocketHandlerMapping;
```

**@Configuration**

```
public class WebSocketConfig {
```

**@Bean**

```
public WebSocketHandlerAdapter handlerAdapter() {
```

```
    return new WebSocketHandlerAdapter();  
}
```

**@Bean**

```
public WebSocketHandlerMapping webSocketHandlerMapping(WebSocketHandler  
handler) {  
    Map<String, WebSocketHandler> map = new HashMap<>();  
    map.put("/ws/updates", handler);  
  
    WebSocketHandlerMapping handlerMapping = new WebSocketHandlerMapping();  
    handlerMapping.setUrlMap(map);  
    return handlerMapping;  
}  
}
```

- Add WebSocket Handler

**UpdateWebSocketHandler.java**

```
import org.springframework.web.socket.WebSocketHandler;  
import org.springframework.web.socket.WebSocketSession;  
import org.springframework.web.socket.WebSocketMessage;  
import reactor.core.publisher.Flux;  
import reactor.core.publisher.Mono;
```

**@Component**

```
public class UpdateWebSocketHandler implements WebSocketHandler {
```

**@Override**

```
public Mono<Void> handle(WebSocketSession session) {  
    Flux<WebSocketMessage> messageFlux = Flux.interval(Duration.ofSeconds(5))  
        .map(sequence -> session.textMessage("Update message " + sequence));  
}
```

```
        return session.send(messageFlux);
    }
}
```

- Add JavaScript to Thymeleaf Templates

templates/routes.html

```
<script>

    const socket = new WebSocket('ws://localhost:8080/ws/updates');

    socket.onmessage = function(event) {

        console.log('Message from server ', event.data);

        // Handle message and update the DOM as needed

    };
</script>
```

**Task 3: Develop form handling in Spring MVC for incident reporting and user feedback.**

**.1 Define Entities for Feedback and Incident Reporting**

- IncidentReport.java

```
import javax.persistence.*;
import java.time.LocalDateTime;
```

**@Entity**

**@Table(name = "incident\_report")**

**public class IncidentReport {**

**@Id**

**@GeneratedValue(strategy = GenerationType.IDENTITY)**

**private Long id;**



```
@Column(name = "description", nullable = false)
```

```
private String description;
```

```
@Column(name = "reported_at", nullable = false)
```

```
private LocalDateTime reportedAt;
```

```
// Getters and Setters
```

```
public Long getId() { return id; }
```

```
public void setId(Long id) { this.id = id; }
```

```
public String getDescription() { return description; }
```

```
public void setDescription(String description) { this.description = description; }
```

```
public LocalDateTime getReportedAt() { return reportedAt; }
```

```
public void setReportedAt(LocalDateTime reportedAt) { this.reportedAt = reportedAt; }
```

```
}
```

- IncidentReportRepository.java

```
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface IncidentReportRepository extends JpaRepository<IncidentReport, Long> {
```

```
}
```

### 3.2 Create Form Handling in Controller

```
IncidentController.java
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.ui.Model;
```

```
import org.springframework.web.bind.annotation.*;
```

```
import javax.validation.Valid;
```

```
import java.time.LocalDateTime;
```

```
@Controller
```

```

@RequestMapping("/incidents")
public class IncidentController {

    private final IncidentReportRepository incidentRepo;

    public IncidentController(IncidentReportRepository incidentRepo) {
        this.incidentRepo = incidentRepo;
    }

    @GetMapping("/report")
    public String showReportForm(Model model) {
        model.addAttribute("incident", new IncidentReport());
        return "reportIncident";
    }

    @PostMapping("/report")
    public String reportIncident(@Valid @ModelAttribute IncidentReport incident) {
        incident.setReportedAt(LocalDateTime.now());
        incidentRepo.save(incident);
        return "redirect:/incidents/report";
    }
}

```

### 3.3 Create Thymeleaf Templates for Incident Reporting

templates/reportIncident.html

```

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

    <title>Report Incident</title>

</head>

<body>

```

```
<h1>Report an Incident</h1>
```

```
<form      action="#"      th:action="@{/incidents/report}"      th:object="${incident}"
method="post">
```

```
    <label for="description">Description:</label>
```

```
    <textarea id="description" th:field="*{description}" rows="4" cols="50"></textarea>
```

```
    <button type="submit">Submit Report</button>
```

```
</form>
```

```
</body>
```

```
</html>
```

## Day 6: Object Relational Mapping and Hibernate - Transit Data Modeling

Task 1: Define Hibernate mappings for transit routes, schedules, and vehicle data.

### 1.1 Define Entity Classes

TransitRoute.java

```
import javax.persistence.*;
```

```
import java.util.Set;
```

```
@Entity
```

```
@Table(name = "transit_route")
```

```
public class TransitRoute {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    @Column(name = "route_name", nullable = false)
```

```
    private String routeName;
```

```
    @OneToMany(mappedBy = "transitRoute", cascade = CascadeType.ALL, orphanRemoval
= true)
```

```

private Set<Schedule> schedules;

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getRouteName() { return routeName; }

public void setRouteName(String routeName) { this.routeName = routeName; }

public Set<Schedule> getSchedules() { return schedules; }

public void setSchedules(Set<Schedule> schedules) { this.schedules = schedules; }
}

```

- Schedule.java

```

import javax.persistence.*;
import java.time.LocalDateTime;

```

```
@Entity
```

```
@Table(name = "schedule")
```

```
public class Schedule {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    @Column(name = "departure_time", nullable = false)
```

```
    private LocalDateTime departureTime;
```

```
    @ManyToOne(fetch = FetchType.LAZY)
```

```
    @JoinColumn(name = "route_id", nullable = false)
```

```
    private TransitRoute transitRoute;
```

```

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "vehicle_id", nullable = false)
private Vehicle vehicle;

// Getters and Setters
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }
public LocalDateTime getDepartureTime() { return departureTime; }
public void setDepartureTime(LocalDateTime departureTime) { this.departureTime =
departureTime; }
public TransitRoute getTransitRoute() { return transitRoute; }
public void setTransitRoute(TransitRoute transitRoute) { this.transitRoute = transitRoute;
}
public Vehicle getVehicle() { return vehicle; }
public void setVehicle(Vehicle vehicle) { this.vehicle = vehicle; }
}

```

- Vehicle.java

```
import javax.persistence.*;
```

```

@Entity
@Table(name = "vehicle")
public class Vehicle {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "vehicle_number", nullable = false)
    private String vehicleNumber;
}

```

```

@Column(name = "capacity", nullable = false)
private int capacity;

// Getters and Setters
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }
public String getVehicleNumber() { return vehicleNumber; }
public void setVehicleNumber(String vehicleNumber) { this.vehicleNumber =
vehicleNumber; }
public int getCapacity() { return capacity; }
public void setCapacity(int capacity) { this.capacity = capacity; }
}

```

## 1.2 Hibernate Configuration

Make sure you have the Hibernate configuration set up in hibernate.cfg.xml or through your application.properties if using Spring Boot.

```

<!-- hibernate.cfg.xml -->
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/your_database</property
>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">password</property>

        <!-- Specify dialect -->
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>

```

```

<!-- Enable Hibernate's automatic session context management -->
<property name="hibernate.current_session_context_class">thread</property>

<!-- Echo all executed SQL to stdout -->
<property name="hibernate.show_sql">true</property>

<!-- Drop and re-create the database schema on startup -->
<property name="hibernate.hbm2ddl.auto">update</property>

<!-- Mention annotated class -->
<mapping class="com.example.TransitRoute"/>
<mapping class="com.example.Schedule"/>
<mapping class="com.example.Vehicle"/>
</session-factory>
</hibernate-configuration>

```

**Task 2: Create DAOs using Hibernate for persisting and querying transit operational data.**

- 2.1 TransitRouteDAO.java

```

import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.query.Query;
import java.util.List;

public class TransitRouteDAO {

    public void save(TransitRoute route) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {

```

```

        transaction = session.beginTransaction();
        session.save(route);
        transaction.commit();
    } catch (Exception e) {
        if (transaction != null) transaction.rollback();
        e.printStackTrace();
    }
}

```

```

public TransitRoute getByld(Long id) {
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        return session.get(TransitRoute.class, id);
    }
}

```

```

public List<TransitRoute> getAll() {
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        Query<TransitRoute> query = session.createQuery("from TransitRoute",
TransitRoute.class);
        return query.list();
    }
}
}

```

- 2.2 ScheduleDAO.java

```

import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.query.Query;
import java.util.List;

```

```

public class ScheduleDAO {

```



```

public void save(Schedule schedule) {
    Transaction transaction = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        transaction = session.beginTransaction();
        session.save(schedule);
        transaction.commit();
    } catch (Exception e) {
        if (transaction != null) transaction.rollback();
        e.printStackTrace();
    }
}

```

```

public Schedule getById(Long id) {
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        return session.get(Schedule.class, id);
    }
}

```

```

public List<Schedule> getByRoute(Long routeId) {
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        Query<Schedule> query = session.createQuery("from Schedule where transitRoute.id = :routeId", Schedule.class);
        query.setParameter("routeId", routeId);
        return query.list();
    }
}

```

- 2.3 VehicleDAO.java

```

import org.hibernate.Session;

```

```
import org.hibernate.Transaction;
```

```
import org.hibernate.query.Query;
```

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

```
public class VehicleDAO {
```

```
    public void save(Vehicle vehicle) {
```

```
        Transaction transaction = null;
```

```
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
```

```
            transaction = session.beginTransaction();
```

```
            session.save(vehicle);
```

```
            transaction.commit();
```

```
        } catch (Exception e) {
```

```
            if (transaction != null) transaction.rollback();
```

```
            e.printStackTrace();
```

```
        }
```

```
    }
```

```
    public Vehicle getByid(Long id) {
```

```
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
```

```
            return session.get(Vehicle.class, id);
```

```
        }
```

```
    }
```

```
    public List<Vehicle> getAll() {
```

```
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
```

```
            Query<Vehicle> query = session.createQuery("from Vehicle", Vehicle.class);
```

```
            return query.list();
```

```
        }
```

```
}  
}
```

**Task 3: Formulate complex HQL and Criteria API queries for analytics and reporting.**

### **.1 HQL Query Examples**

#### **Get All Routes with Their Schedules**

```
import org.hibernate.Session;
```

```
import org.hibernate.query.Query;
```

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

```
public class TransitRouteDAO {
```

```
    public List<Object[]> getRoutesWithSchedules() {
```

```
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
```

```
            String hql = "select r.routeName, s.departureTime from TransitRoute r join  
r.schedules s";
```

```
            Query<Object[]> query = session.createQuery(hql, Object[].class);
```

```
            return query.list();
```

```
        }
```

```
    }
```

```
}
```

#### **Get Vehicles with Capacity Greater Than a Certain Value**

```
import org.hibernate.Session;
```

```
import org.hibernate.query.Query;
```

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

```
public class VehicleDAO {
```

```
    public List<Vehicle> getVehiclesWithMinCapacity(int minCapacity) {
```

```
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
```

```
            String hql = "from Vehicle where capacity > :minCapacity";
```

```

        Query<Vehicle> query = session.createQuery(hql, Vehicle.class);
        query.setParameter("minCapacity", minCapacity);
        return query.list();
    }
}

```

### 3.2 Criteria API Query Examples

#### Get Schedules for a Specific Route

```

import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.query.Query;
import java.util.List;
import org.hibernate.Criteria;
import org.hibernate.criterion.Restrictions;

public class ScheduleDAO {

    public List<Schedule> getSchedulesForRoute(Long routeId) {
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            Criteria criteria = session.createCriteria(Schedule.class);
            criteria.createAlias("transitRoute", "route");
            criteria.add(Restrictions.eq("route.id", routeId));
            return criteria.list();
        }
    }
}

```

- Get Vehicles of a Specific Capacity Range

```

import org.hibernate.Session;
import org.hibernate.Transaction;

```

```

import org.hibernate.query.Query;
import java.util.List;
import org.hibernate.Criteria;
import org.hibernate.criterion.Restrictions;

public class VehicleDAO {

    public List<Vehicle> getVehiclesWithinCapacityRange(int minCapacity, int maxCapacity) {
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            Criteria criteria = session.createCriteria(Vehicle.class);
            criteria.add(Restrictions.between("capacity", minCapacity, maxCapacity));
            return criteria.list();
        }
    }
}

```

- **Hibernate Mappings:** Defined mappings for TransitRoute, Schedule, and Vehicle entities.
- **DAO Implementation:** Created DAOs for persisting and querying transit operational data.
- **Complex Queries:** Formulated complex HQ

## **Day 7: Spring Boot and Microservices - Scalable Traffic Monitoring**

**Task 1: Migrate to Spring Boot for a streamlined setup of microservices for different city zones.**

- **Setup Spring Boot Project:**
- Create a new Spring Boot project for each microservice representing different city zones.
- Use Spring Initializr to generate the projects with dependencies like Web, Actuator, and Spring Boot DevTools.
- **Define Application Properties:**

- In `src/main/resources/application.properties` (or `application.yml`), define the basic properties like server port, application name, etc.

`server.port=8081`

`spring.application.name=traffic-monitoring-zone1`

Create Controllers and Services:

Create RESTful endpoints to monitor traffic data.

`@RestController`

`@RequestMapping("/traffic")`

`public class TrafficController {`

`@GetMapping("/status")`

`public ResponseEntity<String> getTrafficStatus() {`

`return ResponseEntity.ok("Traffic status for Zone 1");`

`}`

`}`

Build and Run:

- Use Maven or Gradle to build the project.
- Run the microservices and test the endpoints.

Convert Existing Configuration:

Move configuration files (`application.properties` or `application.yml`) to the Spring Boot application.

Refactor existing code to fit Spring Boot's conventions. For example, replace XML-based configuration with Java-based configuration if needed.

Implement REST Controllers:

Define REST endpoints using `@RestController` and `@RequestMapping` or `@GetMapping`, `@PostMapping` annotations in your Spring Boot application.

Database Integration:

Set up data source configurations using Spring Data JPA or Spring Data MongoDB depending on your database.

Testing:

Test your microservices using Spring Boot's built-in testing support.

Example: Basic Spring Boot Application

@SpringBootApplication

```
public class CityZoneAServiceApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(CityZoneAServiceApplication.class, args);  
    }  
}
```

@RestController

@RequestMapping("/api/traffic")

```
public class TrafficController {  
  
    @GetMapping("/status")  
    public ResponseEntity<String> getTrafficStatus() {  
        return ResponseEntity.ok("Traffic is smooth");  
    }  
}
```

Task 2: Implement Eureka for service discovery among traffic monitoring microservices.

Add Eureka Server:

- Create a new Spring Boot application for Eureka Server.
- Add the spring-cloud-starter-netflix-eureka-server dependency in pom.xml.

Configure Eureka Server:

- Add @EnableEurekaServer annotation to the main application class.
- Configure application.yml or application.properties for Eureka server settings.

**server:**

**port: 8761**

**eureka:**

**client:**

**registerWithEureka: false**

**fetchRegistry: false**

**server:**

**enableSelfPreservation: false**

**Register Microservices:**

**In each microservice, add spring-cloud-starter-netflix-eureka-client dependency.**

**Configure application.yml to register with Eureka Server.**

**spring:**

**application:**

**name: city-zone-a-service**

**cloud:**

**discovery:**

**client:**

**serviceUrl:**

**defaultZone: <http://localhost:8761/eureka/>**

**Testing Service Discovery:**

**Start the Eureka Server and then start your microservices. Verify that they are registered with Eureka via the Eureka Dashboard.**

**Example: Eureka Server Application**

**@SpringBootApplication**

**@EnableEurekaServer**

**public class EurekaServerApplication {**



```

public static void main(String[] args) {
    SpringApplication.run(EurekaServerApplication.class, args);
}
}

```

**Task 3: Configure Spring Cloud Config for managing microservice settings during peak and off-peak hours.**

**Set Up Config Server:**

- Create a new Spring Boot application for the Config Server.
- Add the spring-cloud-config-server dependency.

**Configure Config Server:**

- Add `@EnableConfigServer` annotation to the main application class.
- Configure `application.yml` to point to the location of your configuration files (e.g., Git repository).

**server:**

**port: 8888**

**spring:**

**cloud:**

**config:**

**server:**

**git:**

**uri: <https://github.com/your-config-repo>**

**clone-on-start: true**

**Configure Microservices to Use Config Server:**

- Add spring-cloud-starter-config dependency to your microservices.
- Configure `bootstrap.yml` or `bootstrap.properties` in each microservice to point to the Config Server.

**spring:**

**application:**

**name:** city-zone-a-service

**cloud:**

**config:**

**uri:** <http://localhost:8888>

### Manage Configuration for Different Scenarios:

- Organize configurations in your Git repository for different profiles (e.g., application-peak.yml, application-off-peak.yml).
- Use Spring Cloud Config to dynamically load configurations based on the profile or environment.
- 

### Example: Config Server Application

**@SpringBootApplication**

**@EnableConfigServer**

**public class ConfigServerApplication {**

**public static void main(String[] args) {**

**SpringApplication.run(ConfigServerApplication.class, args);**

**}**

**}**

### Day 8: Reactive Spring - Real-Time Alerts and Notifications

**Task 1: Apply Spring WebFlux to develop a non-blocking, reactive system for sending real-time traffic alerts.**

#### **.1 Set Up Spring WebFlux**

**First, we need to set up a Spring Boot project with Spring WebFlux. Add the necessary dependencies in your pom.xml or build.gradle.**

**pom.xml:**

**<dependencies>**

**<dependency>**

```

    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
<dependency>
    <groupId>io.r2dbc</groupId>
    <artifactId>r2dbc-postgresql</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-r2dbc</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>
</dependencies>

```

## 1.2 Define the Reactive Model and Repository

TrafficAlert.java

```

import org.springframework.data.annotation.Id;
import org.springframework.data.relational.core.mapping.Table;
import java.time.LocalDateTime;

```

```

@Table("traffic_alerts")

```

```

public class TrafficAlert {

```

```

    @Id

```

```

    private Long id;

```

```

    private String message;

```

```

    private LocalDateTime timestamp;

```

```

// Getters and Setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getMessage() { return message; }

public void setMessage(String message) { this.message = message; }

public LocalDateTime getTimestamp() { return timestamp; }

public void setTimestamp(LocalDateTime timestamp) { this.timestamp = timestamp; }
}

```

**TrafficAlertRepository.java**

```

import org.springframework.data.repository.reactive.ReactiveCrudRepository;
import reactor.core.publisher.Flux;

public interface TrafficAlertRepository extends ReactiveCrudRepository<TrafficAlert, Long>
{
    Flux<TrafficAlert> findByTimestampAfter(LocalDateTime timestamp);
}

```

### 1.3 Create the Reactive Service

**TrafficAlertService.java**

```

import org.springframework.stereotype.Service;
import reactor.core.publisher.Flux;
import java.time.LocalDateTime;

```

**@Service**

```

public class TrafficAlertService {

    private final TrafficAlertRepository repository;

    public TrafficAlertService(TrafficAlertRepository repository) {

        this.repository = repository;
    }
}

```

```
public Flux<TrafficAlert> getRecentAlerts(LocalDateTime fromTimestamp) {  
    return repository.findByTimestampAfter(fromTimestamp);  
}
```

```
public Flux<TrafficAlert> getAllAlerts() {  
    return repository.findAll();  
}  
}
```

#### 1.4 Create the Reactive Controller

TrafficAlertController.java

```
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
import reactor.core.publisher.Flux;  
import java.time.LocalDateTime;
```

@RestController

@RequestMapping("/alerts")

```
public class TrafficAlertController {  
    private final TrafficAlertService service;
```

```
    public TrafficAlertController(TrafficAlertService service) {  
        this.service = service;  
    }
```

@GetMapping("/recent")

```
public Flux<TrafficAlert> getRecentAlerts() {  
    return service.getRecentAlerts(LocalDateTime.now().minusMinutes(10));
```

```

    }

    @GetMapping("/all")
    public Flux<TrafficAlert> getAllAlerts() {
        return service.getAllAlerts();
    }
}

```

**Task 2: Use R2DBC for integrating reactive data updates to the traffic management system.**

### **2.1 R2DBC Configuration**

Ensure that your `application.properties` is configured for R2DBC with your PostgreSQL database.

`application.properties`:

```

spring.r2dbc.url=r2dbc:postgresql://localhost:5432/your_database
spring.r2dbc.username=your_username
spring.r2dbc.password=your_password

spring.sql.init.platform=postgres
spring.sql.init.schema-locations=classpath:schema.sql
spring.sql.init.data-locations=classpath:data.sql

```

### **2.2 Schema and Data Initialization**

Create `schema.sql` and `data.sql` files for initializing the database.

`schema.sql`:

```

CREATE TABLE traffic_alerts (
    id SERIAL PRIMARY KEY,
    message VARCHAR(255) NOT NULL,
    timestamp TIMESTAMP NOT NULL
);

```

**INSERT INTO traffic\_alerts (message, timestamp) VALUES**

**('Accident on Highway 1', '2024-07-10T12:00:00'),**

**('Roadwork on Main St', '2024-07-10T12:05:00');**

**Task 3: Set up WebSocket channels for broadcasting city-wide transportation notifications and updates.**

### **3.1 WebSocket Configuration**

- **Create a WebSocket configuration class to register WebSocket endpoints.**

**WebSocketConfig.java**

**import org.springframework.context.annotation.Configuration;**

**import org.springframework.web.reactive.config.EnableWebFlux;**

**import org.springframework.web.reactive.config.WebFluxConfigurer;**

**import**

**org.springframework.web.reactive.socket.server.support.WebSocketHandlerAdapter;**

**import**

**org.springframework.web.reactive.socket.server.support.WebSocketHandlerMapping;**

**@Configuration**

**@EnableWebFlux**

**public class WebSocketConfig implements WebFluxConfigurer {**

**@Bean**

**public WebSocketHandlerAdapter handlerAdapter() {**

**return new WebSocketHandlerAdapter();**

**}**

**@Bean**

**public    WebSocketHandlerMapping    webSocketHandlerMapping(WebSocketHandler  
handler) {**

```

    Map<String, WebSocketHandler> map = new HashMap<>();
    map.put("/ws/alerts", handler);

    WebSocketHandlerMapping handlerMapping = new WebSocketHandlerMapping();
    handlerMapping.setUrlMap(map);
    handlerMapping.setOrder(10); // order less than annotated controllers

    return handlerMapping;
}
}

```

### 3.2 WebSocket Handler

- Create a WebSocket handler to manage WebSocket connections and send real-time notifications.

AlertWebSocketHandler.java

```

import org.springframework.stereotype.Component;
import org.springframework.web.reactive.socket.WebSocketHandler;
import org.springframework.web.reactive.socket.WebSocketSession;
import org.springframework.web.reactive.socket.WebSocketMessage;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;
import java.time.Duration;

```

@Component

```

public class AlertWebSocketHandler implements WebSocketHandler {
    private final TrafficAlertService alertService;

    public AlertWebSocketHandler(TrafficAlertService alertService) {
        this.alertService = alertService;
    }
}

```



**@Override**

```
public Mono<Void> handle(WebSocketSession session) {  
    Flux<WebSocketMessage> messageFlux =  
    alertService.getRecentAlerts(LocalDateTime.now().minusMinutes(10))  
        .repeatWhen(longFlux -> longFlux.delayElements(Duration.ofSeconds(10)))  
        .map(alert -> session.textMessage("New Alert: " + alert.getMessage()));  
  
    return session.send(messageFlux);  
}  
}
```