ASSIGNMENT-2

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.

SOLUTION:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Smart City Route Planner</title>
    <link rel="stylesheet" href="styles.css">
    <script src="script.js" defer></script>
</head>
<body>
    <header>
        <h1>Smart City Route Planner</h1>
    </header>
    <main>
        <section id="routeSelection">
            <h2>Plan Your Route</h2>
            <form id="routeForm">
                <label for="startLocation">Starting Point:</label>
                <input type="text" id="startLocation" name="startLocation" required>
```

```
            <label for="endLocation">Destination:</label>
            <input type="text" id="endLocation" name="endLocation" required>

            <label for="transportMode">Transport Mode:</label>
            <select id="transportMode" name="transportMode">
                <option value="car">Car</option>
                <option value="bike">Bike</option>
                <option value="publicTransport">Public Transport</option>
            </select>

            <button type="submit">Plan Route</button>
        </form>
    </section>

    <section id="routeOptions">
        <h2>Route Options</h2>
        <ul id="routeList">
            <!-- Dynamic content will be added here -->
        </ul>
    </section>
</main>
<footer>
    <p>&copy; 2024 Smart City Transportation Management System</p>
</footer>
```

```
</body>
</html>
```

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

SOLUTION:

```
/* Reset styles and basic setup */
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

body {
    font-family: Arial, sans-serif;
    line-height: 1.6;
    background-color: #f0f0f0;
}

header, footer {
    background-color: #333;
    color: #fff;
    padding: 10px;
    text-align: center;
}

main {
```

```
        max-width: 800px;
        margin: 20px auto;
        padding: 20px;
        background-color: #fff;
        border: 1px solid #ccc;
        border-radius: 5px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

form {
        display: grid;
        grid-template-columns: 1fr 2fr;
        gap: 10px;
        margin-bottom: 20px;
}

label {
        font-weight: bold;
}

input[type="text"],
select,
button {
        padding: 8px;
        font-size: 1rem;
```

```
        border: 1px solid #ccc;
        border-radius: 3px;
        width: 100%;
}

button {
        background-color: #333;
        color: #fff;
        border: none;
        cursor: pointer;
        transition: background-color 0.3s ease;
}

button:hover {
        background-color: #555;
}

#routeOptions {
        margin-top: 20px;
}

#routeList {
        list-style-type: none;
        padding: 0;
}
```

```
#routeList li {
    padding: 10px;
    border: 1px solid #ccc;
    margin-bottom: 10px;
    background-color: #f9f9f9;
}
```

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

SOLUTION:

```
document.addEventListener('DOMContentLoaded', function() {
    const routeForm = document.getElementById('routeForm');
    const routeList = document.getElementById('routeList');

    routeForm.addEventListener('submit', function(event) {
        event.preventDefault();

        const startLocation = document.getElementById('startLocation').value;
        const endLocation = document.getElementById('endLocation').value;
        const transportMode = document.getElementById('transportMode').value;

        // Simulate fetching route options from a server (example)
        const routes = [
            { id: 1, description: 'Route A: Fastest route', time: '15 mins' },
            { id: 2, description: 'Route B: Scenic route', time: '20 mins' },
            { id: 3, description: 'Route C: Avoids tolls', time: '18 mins' }
```

```
        ];

        // Clear previous options
        routeList.innerHTML = '';

        // Display new route options
        routes.forEach(route => {
            const li = document.createElement('li');
            li.textContent = `${route.description} - ${route.time}`;
            routeList.appendChild(li);
        });
    });
});
```

Day 2: JavaScript/Bootstrap - Interactive Transit Maps

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

SOLUTION:

```
<!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>
    <!-- Bootstrap CSS -->
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <!-- Custom CSS -->
```

```
    <link href="styles.css" rel="stylesheet">
    <!-- 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@2.5.2/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
    <!-- Custom JavaScript -->
    <script src="script.js" defer></script>
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-lg navbar-light bg-light">
            <a class="navbar-brand" href="#">Transit Maps</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarNav"
                    aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle
navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav ml-auto">
                    <li class="nav-item">
                        <a class="nav-link" href="#">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">About</a>
                    </li>
```

```
                    <li class="nav-item">
                        <a class="nav-link" href="#">Contact</a>
                    </li>
                </ul>
            </div>
        </nav>
    </header>

    <main class="container mt-4">
        <div class="row">
            <div class="col-lg-8">
                <div id="map" class="map-container">
                    <!-- Interactive map goes here -->
                </div>
            </div>
            <div class="col-lg-4">
                <div id="transitInfo">
                    <!-- Transit information panel -->
                </div>
            </div>
        </div>
    </main>

    <footer class="footer mt-auto py-3 bg-light">
        <div class="container text-center">
```

```
                <span class="text-muted">&copy; 2024 Transit Maps. All rights reserved.</span>
            </div>
        </footer>
</body>
</html>
```

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

SOLUTION:

```
<!-- Example of using Bootstrap Modals and Tooltips -->
<button type="button" class="btn btn-primary" data-toggle="modal" data-target="#transitModal">
    View Transit Details
</button>

<!-- Modal -->
<div class="modal fade" id="transitModal" tabindex="-1" role="dialog"
aria-labelledby="transitModalLabel"
    aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="transitModalLabel">Transit Details</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                <p>Real-time information about transit.</p>
```

```
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary"
data-dismiss="modal">Close</button>
                    <button type="button" class="btn btn-primary">Save changes</button>
                </div>
            </div>
        </div>
    </div>

<!-- Tooltip example -->
<button type="button" class="btn btn-secondary" data-toggle="tooltip" data-placement="top"
title="Tooltip on top">
    Tooltip on top
</button>
```

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

SOLUTION:

```
// Example of using JavaScript for map interactions and live updates
document.addEventListener('DOMContentLoaded', function() {
    // Fetch real-time transit data
    function fetchTransitData() {
        // Example fetch call to fetch real-time data
        // Replace with actual API call
        fetch('https://api.transit.com/realtime')
            .then(response => response.json())
            .then(data => {
```

```
                // Update UI with real-time data
                displayTransitInfo(data);
            })
            .catch(error => console.error('Error fetching data:', error));
    }

    // Display transit information in the UI
    function displayTransitInfo(data) {
        const transitInfoElement = document.getElementById('transitInfo');
        // Example: Update transit information panel
        transitInfoElement.innerHTML = `<h3>Transit Information</h3>
                                        <p>Next bus: ${data.nextBus}</p>
                                        <p>Next train: ${data.nextTrain}</p>`;
    }

    // Initialize tooltips (if any)
    $('[data-toggle="tooltip"]').tooltip();

    // Example: Initialize a map (using a library like Leaflet.js)
    // Replace with actual map initialization code
    const map = L.map('map').setView([51.505, -0.09], 13);
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: 'Map data &copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
    }).addTo(map);
```

```
    // Example: Handle map interactions
    map.on('click', function(e) {
        alert(`Clicked on coordinates: ${e.latlng}`);
    });

    // Example: Fetch real-time transit data periodically
    setInterval(fetchTransitData, 60000); // Fetch data every minute
});
```

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

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

SOLUTION:

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

public class TrafficDataServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Process GET request for traffic data
        String location = request.getParameter("location");
```

```
        // Example logic to fetch real-time traffic data based on location
        String trafficInfo = TrafficService.getTrafficInfo(location);

        // Set response content type
        response.setContentType("text/html");

        // Write response content
        response.getWriter().println("<html><body>");
        response.getWriter().println("<h1>Real-time Traffic Information</h1>");
        response.getWriter().println("<p>Location: " + location + "</p>");
        response.getWriter().println("<p>Traffic Info: " + trafficInfo + "</p>");
        response.getWriter().println("</body></html>");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Process POST request for user queries
        String query = request.getParameter("query");

        // Example logic to handle user queries
        String result = QueryService.processQuery(query);

        // Set response content type
        response.setContentType("text/html");
```

```
        // Write response content
        response.getWriter().println("<html><body>");
        response.getWriter().println("<h1>User Query Response</h1>");
        response.getWriter().println("<p>Query: " + query + "</p>");
        response.getWriter().println("<p>Result: " + result + "</p>");
        response.getWriter().println("</body></html>");
    }
}
```

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

SOLUTION:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Dynamic Traffic Information</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <header>
        <h1>Real-time Traffic Information</h1>
    </header>
    <main>
```

```
            <div id="trafficInfo">
                <h2>Traffic Information</h2>
                <p>Location: <%= request.getParameter("location") %></p>
                <p>Traffic Info: <%= TrafficService.getTrafficInfo(request.getParameter("location"))
%></p>
            </div>
            <div id="alternativeRoutes">
                <h2>Alternative Routes</h2>
                <ul>
                    <li>Route A: Fastest route</li>
                    <li>Route B: Scenic route</li>
                    <li>Route C: Avoids congestion</li>
                </ul>
            </div>
        </main>
        <footer>
            <p>&copy; 2024 Traffic Data Processing</p>
        </footer>
</body>
</html>
```

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

SOLUTION:

```
import java.io.Serializable;

public class TrafficDataBean implements Serializable {
    private String location;
```

```
    private String trafficInfo;

    // Default constructor
    public TrafficDataBean() {
    }

    // Getters and setters
    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public String getTrafficInfo() {
        return trafficInfo;
    }

    public void setTrafficInfo(String trafficInfo) {
        this.trafficInfo = trafficInfo;
    }
}
```

Day 4: Spring Core - System Configuration and User Management

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

SOLUTION:

UserManagementService.java:

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

@Service
public class UserManagementService {

    public void registerUser(User user) {
        // Logic to register user
    }

    public void deleteUser(String userId) {
        // Logic to delete user
    }

    public User getUserById(String userId) {
        // Logic to fetch user by ID
        return new User(); // Replace with actual implementation
    }

    // Other methods for user management
}
```

SessionHandlingService.java:

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

@Service
public class SessionHandlingService {

    public void startSession(User user) {
        // Logic to start session for the user
    }

    public void endSession(User user) {
        // Logic to end session for the user
    }

    public boolean isSessionActive(User user) {
        // Logic to check if session is active for the user
        return false; // Replace with actual implementation
    }

    // Other methods for session handling
}
```

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

SOLUTION:

TrafficDataService.java:

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

```
@Service
public class TrafficDataService {

    public String getTrafficInfo(String location) {
        // Logic to fetch traffic information based on location
        return "Traffic information for " + location; // Replace with actual implementation
    }

    public List<String> getAlternativeRoutes(String location) {
        // Logic to fetch alternative routes based on location
        return Arrays.asList("Route A", "Route B", "Route C"); // Replace with actual implementation
    }

    // Other methods for traffic data service
}
```

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

SOLUTION:

SecurityConfig.java:

```
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity
```

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {


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

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.

SOLUTION:

RouteController.java:

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
```

```
@Controller
public class RouteController {

    @GetMapping("/admin/routes")
    public String manageRoutes(Model model) {
        // Logic to fetch and display routes
        List<Route> routes = routeService.getAllRoutes();
        model.addAttribute("routes", routes);
        return "admin/routes";
    }

    // Other methods for managing routes (add, edit, delete)
}
```

routes.html (Thymeleaf template):

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Admin Portal - Manage Routes</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
    <div class="container mt-5">
        <h2>Manage Routes</h2>
```

```
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>Name</th>
                        <th>Description</th>
                        <th>Actions</th>
                    </tr>
                </thead>
                <tbody>
                    <tr th:each="route : ${routes}">
                        <td th:text="${route.id}"></td>
                        <td th:text="${route.name}"></td>
                        <td th:text="${route.description}"></td>
                        <td>
                            <a th:href="@{/admin/routes/edit/{id}(id=${route.id})}" class="btn
btn-primary">Edit</a>
                            <a th:href="@{/admin/routes/delete/{id}(id=${route.id})}" class="btn
btn-danger">Delete</a>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
</body>
</html>
```

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

SOLUTION:

ScheduleController.java:

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

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;


@Controller

public class ScheduleController {


	@GetMapping("/admin/schedule")

	public String manageSchedule(Model model) {

		// Logic to fetch and display schedule

		List<Schedule> schedules = scheduleService.getAllSchedules();

		model.addAttribute("schedules", schedules);

		return "admin/schedule";

	}


	// Other methods for managing schedule (add, edit, delete)

}
```

schedule.html (Thymeleaf template):

```
<!DOCTYPE html>

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

<head>

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

```
    <title>Admin Portal - Manage Schedule</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
    <div class="container mt-5">
        <h2>Manage Schedule</h2>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Route</th>
                    <th>Departure Time</th>
                    <th>Actions</th>
                </tr>
            </thead>
            <tbody>
                <tr th:each="schedule : ${schedules}">
                    <td th:text="${schedule.id}"></td>
                    <td th:text="${schedule.route.name}"></td>
                    <td th:text="${schedule.departureTime}"></td>
                    <td>
                        <a th:href="@{/admin/schedule/edit/{id}(id=${schedule.id})}" class="btn
btn-primary">Edit</a>
                        <a th:href="@{/admin/schedule/delete/{id}(id=${schedule.id})}"
class="btn btn-danger">Delete</a>
                    </td>
```

```
                    </tr>
                </tbody>
            </table>
        </div>
</body>
</html>
```

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

SOLUTION:

IncidentController.java:

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.ModelAttribute;

@Controller
public class IncidentController {

    @GetMapping("/admin/incident")
    public String showIncidentForm(Model model) {
        model.addAttribute("incident", new Incident());
        return "admin/incidentForm";
    }

    @PostMapping("/admin/incident")
    public String submitIncidentForm(@ModelAttribute("incident") Incident incident) {
```

```
        // Process incident report
        incidentService.processIncident(incident);
        return "redirect:/admin/incident/success";
    }

    @GetMapping("/admin/incident/success")
    public String showSuccessPage() {
        return "admin/incidentSuccess";
    }
}
```

incidentForm.html (Thymeleaf template:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Admin Portal - Report Incident</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
    <div class="container mt-5">
        <h2>Report Incident</h2>
        <form action="#" th:action="@{/admin/incident}" th:object="${incident}" method="post">
            <div class="form-group">
                <label for="title">Title</label>
                <input type="text" class="form-control" id="title" th:field="*{title}">
```

```
            </div>
            <div class="form-group">
                <label for="description">Description</label>
                <textarea class="form-control" id="description"
th:field="*{description}"></textarea>
            </div>
            <button type="submit" class="btn btn-primary">Submit</button>
        </form>
    </div>
</body>
</html>
```

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

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

SOLUTION:

Route.java (Hibernate Entity):

```
import javax.persistence.*;

@Entity
@Table(name = "routes")
public class Route {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;
```

```
    @Column(name = "name")
    private String name;

    @Column(name = "description")
    private String description;

    // Getters and setters, constructors, etc.
}
```

Schedule.java (Hibernate Entity):

```
import javax.persistence.*;

@Entity
@Table(name = "schedules")
public class Schedule {

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

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "route_id")
    private Route route;
```

```
    @Column(name = "departure_time")
    private LocalDateTime departureTime;

    // Getters and setters, constructors, etc.
}
```

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

SOLUTION:

RouteDAO.java:

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class RouteDAO {

    @Autowired
    private SessionFactory sessionFactory;

    public void saveOrUpdate(Route route) {
        Session session = sessionFactory.getCurrentSession();
        session.saveOrUpdate(route);
    }

    public Route findById(Long id) {
```

```
        Session session = sessionFactory.getCurrentSession();
        return session.get(Route.class, id);
    }

    // Other CRUD methods (delete, findAll, etc.)
}
```

ScheduleDAO.java:

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class ScheduleDAO {

    @Autowired
    private SessionFactory sessionFactory;

    public void saveOrUpdate(Schedule schedule) {
        Session session = sessionFactory.getCurrentSession();
        session.saveOrUpdate(schedule);
    }

    public Schedule findById(Long id) {
        Session session = sessionFactory.getCurrentSession();
```

```
        return session.get(Schedule.class, id);
    }

    // Other CRUD methods (delete, findAll, etc.)
}
```

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

SOLUTION:

HQL Query:

```
import org.hibernate.Session;
import org.hibernate.query.Query;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public class RouteDAO {

    @Autowired
    private SessionFactory sessionFactory;

    public List<Route> findRoutesByName(String name) {
        Session session = sessionFactory.getCurrentSession();
        Query<Route> query = session.createQuery("FROM Route WHERE name = :name",
Route.class);
```

```
        query.setParameter("name", name);
        return query.getResultList();
    }

    // Other HQL queries for analytics and reporting
}
```

Criteria API Query:

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.query.Query;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public class ScheduleDAO {

    @Autowired
    private SessionFactory sessionFactory;

    public List<Schedule> findSchedulesByRouteId(Long routeId) {
        Session session = sessionFactory.getCurrentSession();
        CriteriaBuilder builder = session.getCriteriaBuilder();
        CriteriaQuery<Schedule> criteria = builder.createQuery(Schedule.class);
```

```
        Root<Schedule> root = criteria.from(Schedule.class);
        criteria.select(root).where(builder.equal(root.get("route").get("id"), routeId));
        Query<Schedule> query = session.createQuery(criteria);
        return query.getResultList();
    }

    // Other Criteria API queries for analytics and reporting
}
```

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.

SOLUTION:

TrafficMonitoringServiceApplication.java (Main Spring Boot Application):

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class TrafficMonitoringServiceApplication {

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

ZoneTrafficService.java (Example Microservice):

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

@Service
public class ZoneTrafficService {

    public String getTrafficStatus(String zoneName) {
        // Logic to fetch traffic status for a specific zone
        return "Traffic status for " + zoneName; // Replace with actual implementation
    }

    // Other methods for zone traffic monitoring
}
```

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

SOLUTION:

EurekaServerApplication.java (Eureka Server Setup):

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {

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

```
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

TrafficMonitoringServiceApplication.java (Eureka Client Setup):

```
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
@EnableEurekaClient
public class TrafficMonitoringServiceApplication {

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

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

SOLUTION:

ConfigServerApplication.java (Spring Cloud Config Server Setup):

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
```

```
@EnableConfigServer
public class ConfigServerApplication {

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

TrafficMonitoringServiceApplication.java (Spring Cloud Config Client Setup):

```
import org.springframework.cloud.context.config.annotation.RefreshScope;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.beans.factory.annotation.Value;

@RestController
@RefreshScope
public class TrafficMonitoringController {

	@Value("${traffic.status}")
	private String trafficStatus;

	@GetMapping("/traffic/status")
	public String getTrafficStatus() {
		return "Current traffic status: " + trafficStatus;
	}
```

}

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.

SOLUTION:

TrafficAlertService.java:

```
import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import reactor.core.publisher.Flux;

import reactor.core.publisher.Mono;


@Service

public class TrafficAlertService {


    @Autowired

    private TrafficAlertRepository alertRepository;


    public Flux<TrafficAlert> getAllTrafficAlerts() {

        return alertRepository.findAll();

    }


    public Mono<TrafficAlert> getTrafficAlertById(String id) {

        return alertRepository.findById(id);

    }
```

```
    public Mono<TrafficAlert> createTrafficAlert(TrafficAlert alert) {
        return alertRepository.save(alert);
    }

    public Mono<Void> deleteTrafficAlert(String id) {
        return alertRepository.deleteById(id);
    }
}
```

TrafficAlertController.java (Spring WebFlux Controller):

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

@RestController
@RequestMapping("/alerts")
public class TrafficAlertController {

    @Autowired
    private TrafficAlertService alertService;

    @GetMapping
    public Flux<TrafficAlert> getAllTrafficAlerts() {
        return alertService.getAllTrafficAlerts();
```

```
    }

    @GetMapping("/{id}")
    public Mono<TrafficAlert> getTrafficAlertById(@PathVariable String id) {
        return alertService.getTrafficAlertById(id);
    }

    @PostMapping
    public Mono<TrafficAlert> createTrafficAlert(@RequestBody TrafficAlert alert) {
        return alertService.createTrafficAlert(alert);
    }

    @DeleteMapping("/{id}")
    public Mono<Void> deleteTrafficAlert(@PathVariable String id) {
        return alertService.deleteTrafficAlert(id);
    }
}
```

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

SOLUTION:

TrafficAlertRepository.java (R2DBC Repository):

```
import org.springframework.data.r2dbc.repository.Query;
import org.springframework.data.repository.reactive.ReactiveCrudRepository;
import org.springframework.stereotype.Repository;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;
```

```
@Repository
public interface TrafficAlertRepository extends ReactiveCrudRepository<TrafficAlert, String> {

    @Query("SELECT * FROM traffic_alert WHERE status = :status")
    Flux<TrafficAlert> findByStatus(String status);

    Mono<Void> deleteById(String id);
}
```

application.properties (Configure R2DBC):

```
spring.r2dbc.url=r2dbc:h2:mem:///testdb
spring.r2dbc.username=sa
spring.r2dbc.password=
spring.r2dbc.pool.initial-size=1
spring.r2dbc.pool.max-size=5
```

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

SOLUTION:

WebSocketConfig.java (WebSocket Configuration):

```
import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;
import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;
```

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/topic");
        config.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/ws").withSockJS();
    }
}
```

TrafficNotificationController.java (WebSocket Controller):

```
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Controller;

@Controller
public class TrafficNotificationController {

    @MessageMapping("/notify")
```

```
    @SendTo("/topic/traffic")
    public TrafficNotification sendTrafficNotification(TrafficNotification notification) {
        // Process notification logic (e.g., broadcasting to clients)
        return notification;
    }
}
```