ASSIGNMENT-1

Development Scenario: Insurance Claim Processing System

Day 1: HTML, CSS, and JavaScript - User Authentication and Profile Setup

Task 1: Design and code the HTML forms for user registration and login, ensuring accessibility standards are met.

SOLUTION:

User Registration Form :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User Registration - Insurance Claim Processing System</title>
    <link rel="stylesheet" href="styles.css">
    <script src="script.js" defer></script> <!-- Include JavaScript for form validation -->
</head>
<body>
    <div class="container">
        <h2>User Registration</h2>
        <form id="registrationForm" action="register.php" method="POST">
            <div class="form-group">
                <label for="username">Username:</label>
                <input type="text" id="username" name="username" aria-label="Username"
aria-required="true" required>
            </div>
            <div class="form-group">
```

```
                    <label for="email">Email:</label>
                    <input type="email" id="email" name="email" aria-label="Email"
aria-required="true" required>
                </div>
                <div class="form-group">
                    <label for="password">Password:</label>
                    <input type="password" id="password" name="password" aria-label="Password"
aria-required="true" required>
                </div>
                <button type="submit">Register</button>
            </form>
        </div>
</body>
</html>
```

User Login Form:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User Login - Insurance Claim Processing System</title>
    <link rel="stylesheet" href="styles.css">
    <script src="script.js" defer></script> <!-- Include JavaScript for form validation -->
</head>
<body>
    <div class="container">
```

```
        <h2>User Login</h2>
        <form id="loginForm" action="login.php" method="POST">
            <div class="form-group">
                <label for="username">Username:</label>
                <input type="text" id="username" name="username" aria-label="Username"
aria-required="true" required>
            </div>
            <div class="form-group">
                <label for="password">Password:</label>
                <input type="password" id="password" name="password" aria-label="Password"
aria-required="true" required>
            </div>
            <button type="submit">Login</button>
        </form>
    </div>
</body>
</html>
```

Task 2: Apply CSS to style the forms for a consistent look and feel that aligns with the company's branding.

SOLUTION:

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

.container {
    max-width: 400px;
```

```
        margin: 50px auto;
        background-color: #fff;
        padding: 20px;
        border-radius: 5px;
        box-shadow: 0 0 10px rgba(0,0,0,0.1);
}

h2 {
        text-align: center;
        color: #333;
}

.form-group {
        margin-bottom: 15px;
}

label {
        display: block;
        font-weight: bold;
        margin-bottom: 5px;
}

input[type="text"],
input[type="email"],
input[type="password"] {
```

```
    width: 100%;
    padding: 8px;
    font-size: 16px;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
}

button {
    background-color: #4CAF50;
    color: white;
    padding: 10px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
    width: 100%;
}

button:hover {
    background-color: #45a049;
}
```

Task 3: Implement JavaScript form validations to provide immediate feedback on user input errors before submission.

SOLUTION:

```
// JavaScript for form validation
```

```
document.addEventListener("DOMContentLoaded", function() {
    const registrationForm = document.getElementById("registrationForm");
    const loginForm = document.getElementById("loginForm");

    if (registrationForm) {
        registrationForm.addEventListener("submit", function(event) {
            event.preventDefault(); // Prevent form submission

            // Validate username
            const username = registrationForm.username.value.trim();
            if (username.length < 5) {
                alert("Username must be at least 5 characters");
                return false;
            }

            // Validate email
            const email = registrationForm.email.value.trim();
            if (!isValidEmail(email)) {
                alert("Please enter a valid email address");
                return false;
            }

            // Validate password
            const password = registrationForm.password.value;
```

```
            if (password.length < 6) {
                alert("Password must be at least 6 characters");
                return false;
            }

            // If all validations pass, submit the form
            registrationForm.submit();
        });
}

if (loginForm) {
        loginForm.addEventListener("submit", function(event) {
            event.preventDefault(); // Prevent form submission

            // Validate username
            const username = loginForm.username.value.trim();
            if (username.length < 5) {
                alert("Username must be at least 5 characters");
                return false;
            }

            // Validate password
            const password = loginForm.password.value;
            if (password.length < 6) {
                alert("Password must be at least 6 characters");
```

```
                return false;
            }

            // If all validations pass, submit the form
            loginForm.submit();
        });
    }
});

// Function to validate email format
function isValidEmail(email) {
    const re = /\S+@\S+\.\S+/;
    return re.test(email);
}
```

Day 2: JavaScript/Bootstrap - Responsive Dashboard for Policy Management

Task 1: Create a dashboard layout with Bootstrap ensuring responsiveness across devices.

SOLUTION:

Create a dashboard layout with Bootstrap:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Policy Management Dashboard</title>
```

```
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <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>
    <link rel="stylesheet" href="styles.css"> <!-- Optional custom styles -->
</head>
<body>
    <div class="container-fluid">
        <div class="row">
            <div class="col-lg-3">
                <!-- Sidebar -->
                <div class="bg-light border-right" id="sidebar">
                    <div class="sidebar-heading">Policy Management</div>
                    <div class="list-group list-group-flush">
                        <a href="#" class="list-group-item
list-group-item-action">Dashboard</a>
                        <a href="#" class="list-group-item list-group-item-action">Policies</a>
                        <a href="#" class="list-group-item list-group-item-action">Claims</a>
                        <a href="#" class="list-group-item list-group-item-action">Settings</a>
                    </div>
                </div>
            </div>
            <div class="col-lg-9">
                <!-- Content Area -->
                <div id="content">
```

```
                    <nav class="navbar navbar-expand-lg navbar-light bg-light">
                        <div class="container-fluid">
                            <button type="button" id="sidebarCollapse" class="btn btn-info">
                                <i class="fas fa-align-left"></i>
                                <span>Toggle Sidebar</span>
                            </button>
                        </div>
                    </nav>
                    <div class="container">
                        <h2>Welcome to Policy Management Dashboard</h2>
                        <p>This is the main content area where policies and claims will be displayed.</p>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <script src="script.js"></script> <!-- Optional JavaScript for interactivity -->
</body>
</html>
```

Task 2: Utilize Bootstrap's JavaScript components like tabs and modals to enrich the policy management interface.

SOLUTION:

Utilize Bootstrap's JavaScript components:

```
<!-- Inside the main content area -->
<div class="container">
```

```
    <ul class="nav nav-tabs" id="myTab" role="tablist">
        <li class="nav-item">
            <a class="nav-link active" id="policies-tab" data-toggle="tab" href="#policies" role="tab" aria-controls="policies" aria-selected="true">Policies</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" id="claims-tab" data-toggle="tab" href="#claims" role="tab" aria-controls="claims" aria-selected="false">Claims</a>
        </li>
    </ul>
    <div class="tab-content" id="myTabContent">
        <div class="tab-pane fade show active" id="policies" role="tabpanel" aria-labelledby="policies-tab">
            <!-- Content for Policies tab -->
            <h4>List of Policies</h4>
            <p>Here you can view and manage policies.</p>
        </div>
        <div class="tab-pane fade" id="claims" role="tabpanel" aria-labelledby="claims-tab">
            <!-- Content for Claims tab -->
            <h4>List of Claims</h4>
            <p>Here you can view and manage claims.</p>
        </div>
    </div>
</div>
```

Task 3: Enhance dashboard interactivity with JavaScript for policy sorting and detailed views.

SOLUTION:

```
// JavaScript for sorting policies
```

```
document.addEventListener("DOMContentLoaded", function() {
    const policies = [
        { name: "Policy A", premium: 200 },
        { name: "Policy C", premium: 150 },
        { name: "Policy B", premium: 180 }
    ];

    // Display initial list of policies
    displayPolicies(policies);

    // Example: Sort policies by name
    document.getElementById("sortByNameBtn").addEventListener("click", function() {
        policies.sort((a, b) => a.name.localeCompare(b.name));
        displayPolicies(policies);
    });

    // Function to display policies
    function displayPolicies(policies) {
        const policyList = document.getElementById("policyList");
        policyList.innerHTML = ''; // Clear existing list

        policies.forEach(policy => {
            const listItem = document.createElement("li");
            listItem.textContent = `${policy.name} - Premium: ${policy.premium}`;
            policyList.appendChild(listItem);
```

```
            });
        }
});
```

Day 3: Servlet/JSP, Introduction to JSP - Claims Submission Process

Task 1: Develop Servlets to manage the workflow of submitting insurance claims.

SOLUTION:

Develop Servlets to manage the workflow of submitting insurance claims:

```
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("/SubmitClaimServlet")
public class SubmitClaimServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Retrieve parameters from the form
        String policyNumber = request.getParameter("policyNumber");
        String claimType = request.getParameter("claimType");
        String claimDetails = request.getParameter("claimDetails");

        // Perform backend processing (e.g., save to database, send notifications)
        // Example: Save claim data to a database
```

```
            // Redirect to a confirmation page
            response.sendRedirect("claimConfirmation.jsp");
        }
}
```

Task 2: Construct JSP pages for entering claim information and confirmations.

SOLUTION:

Example JSP for Claim Submission Form:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Submit Insurance Claim</title>
</head>
<body>
    <h2>Submit Insurance Claim</h2>
    <form action="SubmitClaimServlet" method="post">
        <label for="policyNumber">Policy Number:</label>
        <input type="text" id="policyNumber" name="policyNumber" required><br><br>

        <label for="claimType">Claim Type:</label>
        <select id="claimType" name="claimType" required>
            <option value="Car">Car</option>
            <option value="Home">Home</option>
```

```
                <option value="Health">Health</option>
            </select><br><br>

            <label for="claimDetails">Claim Details:</label><br>
            <textarea id="claimDetails" name="claimDetails" rows="4" cols="50"
required></textarea><br><br>

            <button type="submit">Submit Claim</button>
        </form>
</body>
</html>
```

Example JSP for Claim Confirmation:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Claim Submission Confirmation</title>
</head>
<body>
    <h2>Claim Submission Confirmation</h2>
    <p>Your claim has been successfully submitted.</p>
    <!-- Add more details or options as needed -->
</body>
</html>
```

Task 3: Employ JavaBeans to manage the transition of data in the claim submission process.

SOLUTION:

```
public class ClaimBean {
	private String policyNumber;
	private String claimType;
	private String claimDetails;

	// Getters and Setters
	public String getPolicyNumber() {
		return policyNumber;
	}

	public void setPolicyNumber(String policyNumber) {
		this.policyNumber = policyNumber;
	}

	public String getClaimType() {
		return claimType;
	}

	public void setClaimType(String claimType) {
		this.claimType = claimType;
	}

	public String getClaimDetails() {
		return claimDetails;
```

```
    }

    public void setClaimDetails(String claimDetails) {
        this.claimDetails = claimDetails;
    }
}
```

Day 4: Spring Core - Policy Administration Backend

Task 1: Refactor policy-related operations to utilize Spring Beans and Dependency Injection.

SOLUTION:

Refactor policy-related operations to utilize Spring Beans and Dependency Injection:

1.PolicyService Interface:

```
public interface PolicyService {
    void createPolicy(Policy policy);
    Policy getPolicyById(int policyId);
    List<Policy> getAllPolicies();
    void updatePolicy(Policy policy);
    void deletePolicy(int policyId);
}
```

2.PolicyServiceImpl Implementation

```
@Service
public class PolicyServiceImpl implements PolicyService {

    private final PolicyRepository policyRepository; // Assume PolicyRepository is already defined
```

```
@Autowired
public PolicyServiceImpl(PolicyRepository policyRepository) {
    this.policyRepository = policyRepository;
}

@Override
public void createPolicy(Policy policy) {
    policyRepository.save(policy);
}

@Override
public Policy getPolicyById(int policyId) {
    return policyRepository.findById(policyId).orElse(null);
}

@Override
public List<Policy> getAllPolicies() {
    return policyRepository.findAll();
}

@Override
public void updatePolicy(Policy policy) {
    policyRepository.save(policy); // Assuming save method handles update as well
}
```

```
    @Override
    public void deletePolicy(int policyId) {
        policyRepository.deleteById(policyId);
    }
}
```

3.Policy Entity and Repository

```
@Entity
public class Policy {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String policyNumber;
    private String policyType;
    private double premium;

    // Getters and setters, constructors
}

@Repository
public interface PolicyRepository extends JpaRepository<Policy, Integer> {
    // Custom queries if needed
}
```

Task 2: Implement Spring validation on the server side to ensure policy data integrity.

SOLUTION:

Implement Spring validation on the server side to ensure policy data integrity:

1.Policy Entity with Validation Annotations

```
@Entity
public class Policy {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @NotBlank(message = "Policy number is required")
    private String policyNumber;

    @NotBlank(message = "Policy type is required")
    private String policyType;

    @Min(value = 0, message = "Premium must be positive")
    private double premium;

    // Getters and setters, constructors
}
```

2.Controller Advice for Global Exception Handling

```
@ControllerAdvice
public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {

    @Override
    protected ResponseEntity<Object>
handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers,
```

```
HttpStatus status, WebRequest request) {
        Map<String, Object> body = new LinkedHashMap<>();
        body.put("timestamp", LocalDateTime.now());
        body.put("status", status.value());

        // Get all errors
        List<String> errors = ex.getBindingResult()
                                .getFieldErrors()
                                .stream()
                                .map(x -> x.getDefaultMessage())
                                .collect(Collectors.toList());

        body.put("errors", errors);

        return new ResponseEntity<>(body, headers, status);
    }
}
```

Task 3: Set up Application Context and Bean Factory for a scalable backend structure.

SOLUTION:

1.Spring Configuration Class:

```
@Configuration
@ComponentScan(basePackages = "com.yourpackage")
@EnableJpaRepositories(basePackages = "com.yourpackage.repository")
@EntityScan(basePackages = "com.yourpackage.entity")
public class AppConfig {
```

```
    @Bean
    public DataSource dataSource() {
        // Configure and return DataSource bean
        return DataSourceBuilder.create()
                                    .driverClassName("com.mysql.jdbc.Driver")
                                    .url("jdbc:mysql://localhost:3306/your_database")
                                    .username("username")
                                    .password("password")
                                    .build();
    }

    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
        LocalContainerEntityManagerFactoryBean em = new
LocalContainerEntityManagerFactoryBean();
        em.setDataSource(dataSource());
        em.setPackagesToScan("com.yourpackage.entity");

        JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
        em.setJpaVendorAdapter(vendorAdapter);

        Properties properties = new Properties();
        properties.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQLDialect");
        em.setJpaProperties(properties);
```

```
        return em;
    }

    @Bean
    public PlatformTransactionManager transactionManager(EntityManagerFactory emf) {
        JpaTransactionManager transactionManager = new JpaTransactionManager();
        transactionManager.setEntityManagerFactory(emf);

        return transactionManager;
    }
}
```

Day 5: Spring MVC - User Claim Interaction Workflow

Task 1: Migrate front-end form handling to Spring MVC controllers.

SOLUTION:

Migrate front-end form handling to Spring MVC controllers:

```
@Controller
@RequestMapping("/claims")
public class ClaimController {

    @Autowired
    private ClaimService claimService; // Assume ClaimService is already defined

    @GetMapping("/submit")
    public String showClaimForm(Model model) {
```

```
        model.addAttribute("claim", new Claim());
        return "claimForm"; // Return Thymeleaf view name
    }

    @PostMapping("/submit")
    public String submitClaim(@Valid @ModelAttribute("claim") Claim claim, BindingResult bindingResult, Model model) {
        if (bindingResult.hasErrors()) {
            return "claimForm"; // Return to form with validation errors
        }

        claimService.submitClaim(claim); // Process claim submission

        model.addAttribute("claim", claim);
        return "claimConfirmation"; // Return Thymeleaf view name for confirmation
    }
}
```

Task 2: Configure Thymeleaf as the view layer for dynamic content rendering in Spring MVC.

SOLUTION:

Maven Dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Thymeleaf Template:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Submit Claim</title>
</head>
<body>
    <h2>Submit Claim</h2>
    <form action="#" th:action="@{/claims/submit}" th:object="${claim}" method="post">
        <label for="policyNumber">Policy Number:</label>
        <input type="text" id="policyNumber" th:field="*{policyNumber}" required><br><br>

        <label for="claimType">Claim Type:</label>
        <select id="claimType" th:field="*{claimType}" required>
            <option value="">Select</option>
            <option value="Car">Car</option>
            <option value="Home">Home</option>
            <option value="Health">Health</option>
        </select><br><br>

        <label for="claimDetails">Claim Details:</label><br>
        <textarea id="claimDetails" th:field="*{claimDetails}" required></textarea><br><br>

        <button type="submit">Submit Claim</button>
    </form>
```

```
</body>

</html>
```

Task 3: Implement data binding and server-side validation within the Spring MVC framework.

SOLUTION:

Model Class (Claim.java)

```
public class Claim {


    @NotEmpty(message = "Policy number is required")

    private String policyNumber;


    @NotEmpty(message = "Claim type is required")

    private String claimType;


    @NotEmpty(message = "Claim details are required")

    private String claimDetails;


    // Getters and setters

}
```

Controller Method (ClaimController.java):

```
@PostMapping("/submit")

public String submitClaim(@Valid @ModelAttribute("claim") Claim claim, BindingResult bindingResult,
Model model) {

    if (bindingResult.hasErrors()) {

        return "claimForm"; // Return to form with validation errors

    }
```

```
        claimService.submitClaim(claim); // Process claim submission


        model.addAttribute("claim", claim);
        return "claimConfirmation"; // Return Thymeleaf view name for confirmation
}
```

Global Exception Handling (ControllerAdvice):

```
@ControllerAdvice
public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {


        @ExceptionHandler(MethodArgumentNotValidException.class)
        protected ResponseEntity<Object>
handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers,
HttpStatus status, WebRequest request) {
                Map<String, Object> body = new LinkedHashMap<>();
                body.put("timestamp", LocalDateTime.now());
                body.put("status", status.value());


                // Get all errors
                List<String> errors = ex.getBindingResult()
                                                .getFieldErrors()
                                                .stream()
                                                .map(x -> x.getDefaultMessage())
                                                .collect(Collectors.toList());


                body.put("errors", errors);
```

```
        return new ResponseEntity<>(body, headers, status);
    }
}
```

Day 6: Object Relational Mapping and Hibernate - Database Integration for Claims and Policies

Task 1: Define Hibernate entity mappings for claim and policy data models.

SOLUTION:

Define Hibernate entity mappings for claim and policy data models:

1.Claim Entity:

```
@Entity
@Table(name = "claims")
public class Claim {

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

    @Column(name = "claim_number")
    private String claimNumber;

    @Column(name = "claim_type")
    private String claimType;

    @Column(name = "claim_details")
```

```
    private String claimDetails;

    // Getters and setters
}
```

2.Policy Entity:

```
@Entity
@Table(name = "policies")
public class Policy {

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

    @Column(name = "policy_number")
    private String policyNumber;

    @Column(name = "policy_type")
    private String policyType;

    @Column(name = "premium")
    private double premium;

    // Getters and setters
}
```

Task 2: Develop Hibernate DAOs to handle CRUD operations for claims and policies.

SOLUTION:

Develop Hibernate DAOs to handle CRUD operations for claims and policies:

1.Claim DAO:

```
@Repository
public class ClaimDAO {

    @Autowired
    private EntityManager entityManager;

    public void save(Claim claim) {
        entityManager.persist(claim);
    }

    public Claim findById(int id) {
        return entityManager.find(Claim.class, id);
    }

    public List<Claim> findAll() {
        CriteriaBuilder cb = entityManager.getCriteriaBuilder();
        CriteriaQuery<Claim> cq = cb.createQuery(Claim.class);
        Root<Claim> rootEntry = cq.from(Claim.class);
        CriteriaQuery<Claim> all = cq.select(rootEntry);
```

```
        TypedQuery<Claim> allQuery = entityManager.createQuery(all);
        return allQuery.getResultList();
    }

    public void update(Claim claim) {
        entityManager.merge(claim);
    }

    public void delete(int id) {
        Claim claim = findById(id);
        entityManager.remove(claim);
    }
}
```

2.Policy DAO

```
@Repository
public class PolicyDAO {

    @Autowired
    private EntityManager entityManager;

    public void save(Policy policy) {
        entityManager.persist(policy);
    }

    public Policy findById(int id) {
```

```
        return entityManager.find(Policy.class, id);
    }

    public List<Policy> findAll() {
        CriteriaBuilder cb = entityManager.getCriteriaBuilder();
        CriteriaQuery<Policy> cq = cb.createQuery(Policy.class);
        Root<Policy> rootEntry = cq.from(Policy.class);
        CriteriaQuery<Policy> all = cq.select(rootEntry);

        TypedQuery<Policy> allQuery = entityManager.createQuery(all);
        return allQuery.getResultList();
    }

    public void update(Policy policy) {
        entityManager.merge(policy);
    }

    public void delete(int id) {
        Policy policy = findById(id);
        entityManager.remove(policy);
    }
}
```

Task 3: Write and test HQL and Criteria queries for advanced data retrieval and reporting.

SOLUTION:

1.HQL Query:

```
@Repository
public class ClaimDAO {

    // Other methods...

    public List<Claim> findClaimsByType(String claimType) {
        String hql = "FROM Claim WHERE claimType = :claimType";
        return entityManager.createQuery(hql, Claim.class)
                                    .setParameter("claimType", claimType)
                                    .getResultList();
    }
}
```

2.Criteria Query

```
@Repository
public class PolicyDAO {

    // Other methods...

    public List<Policy> findPoliciesWithPremiumAbove(double premiumThreshold) {
        CriteriaBuilder cb = entityManager.getCriteriaBuilder();
        CriteriaQuery<Policy> cq = cb.createQuery(Policy.class);
        Root<Policy> root = cq.from(Policy.class);

        cq.select(root)
          .where(cb.gt(root.get("premium"), premiumThreshold));
```

```
        TypedQuery<Policy> query = entityManager.createQuery(cq);
        return query.getResultList();
    }
}
```

Day 7: Spring Boot and Microservices - Microservices for Claim Processing

Task 1: Transition the monolithic application structure to a microservices architecture using Spring Boot.

SOLUTION:

Service 1: Claim Service:

```
@SpringBootApplication
@EnableDiscoveryClient
public class ClaimServiceApplication {

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

Service 2: Policy Service:

```
@SpringBootApplication
@EnableDiscoveryClient
public class PolicyServiceApplication {

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

```
    }
}
```

Task 2: Implement service discovery with Eureka and develop Feign clients for inter-service communication.

SOLUTION:

Eureka Server:

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {

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

Feign Client for Inter-Service Communication:

```
@FeignClient(name = "policy-service")
public interface PolicyFeignClient {

    @GetMapping("/policies/{policyId}")
    Policy getPolicyById(@PathVariable("policyId") int policyId);
}
```

Task 3: Set up and configure Spring Cloud Config for centralized configuration management of microservices.

SOLUTION:

Config Server:

```
@SpringBootApplication
```

```
@EnableConfigServer
public class ConfigServerApplication {

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

Config Client:

```
@SpringBootApplication
@EnableDiscoveryClient
public class PolicyServiceApplication {

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

Day 8: Reactive Spring - Real-time Claim Status Updates

Task 1: Introduce Spring WebFlux for handling real-time claim status updates using reactive streams.

SOLUTION:

WebFlux Controller:

```
@RestController
@RequestMapping("/claims")
public class ClaimController {
```

```
    private final ClaimService claimService;

    public ClaimController(ClaimService claimService) {
        this.claimService = claimService;
    }

    @GetMapping("/{claimId}/status")
    public Mono<ClaimStatus> getClaimStatus(@PathVariable("claimId") String claimId) {
        return claimService.getClaimStatus(claimId);
    }

    @PutMapping("/{claimId}/status")
    public Mono<ClaimStatus> updateClaimStatus(@PathVariable("claimId") String claimId,
@RequestBody ClaimStatus newStatus) {
        return claimService.updateClaimStatus(claimId, newStatus);
    }
}
```

Task 2: Configure R2DBC for reactive database connectivity to update claim status dynamically.

SOLUTION:

R2DBC Dependencies:

```
<dependency>
    <groupId>io.r2dbc</groupId>
    <artifactId>r2dbc-postgresql</artifactId>
    <version>VERSION</version>
</dependency>
```

R2DBC Configuration:

```
@Configuration
public class R2DBCConfig extends AbstractR2dbcConfiguration {


    @Override
    @Bean
    public ConnectionFactory connectionFactory() {
        return new PostgresqlConnectionFactory(
                PostgresqlConnectionConfiguration.builder()
                        .host("localhost")
                        .port(5432)
                        .database("your_database")
                        .username("your_username")
                        .password("your_password")
                        .build());
    }
}
```

Task 3: Implement WebSocket communication for real-time interaction between the client and the server.

SOLUTION:

WebSocket Handler:

```
@Component
public class ClaimStatusWebSocketHandler extends TextWebSocketHandler {


    private final ObjectMapper objectMapper;
    private final ClaimService claimService;
```

```
    public ClaimStatusWebSocketHandler(ObjectMapper objectMapper, ClaimService claimService) {
        this.objectMapper = objectMapper;
        this.claimService = claimService;
    }

    @Override
    protected void handleTextMessage(WebSocketSession session, TextMessage message) throws Exception {
        // Handle incoming messages
        String claimId = message.getPayload();
        Mono<ClaimStatus> claimStatusMono = claimService.getClaimStatus(claimId);

        claimStatusMono.subscribe(claimStatus -> {
            try {
                session.sendMessage(new TextMessage(objectMapper.writeValueAsString(claimStatus)));
            } catch (JsonProcessingException e) {
                e.printStackTrace();
            }
        });
    }
}
```

WebSocket Configuration:

```
@Configuration
```

```
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

    private final ClaimStatusWebSocketHandler claimStatusWebSocketHandler;

    public WebSocketConfig(ClaimStatusWebSocketHandler claimStatusWebSocketHandler) {
        this.claimStatusWebSocketHandler = claimStatusWebSocketHandler;
    }

    @Override
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
        registry.addHandler(claimStatusWebSocketHandler, "/claim-status").setAllowedOrigins("*");
    }
}
```