K DEEPAK **ASSIGNMENT-J2EE**

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, ensuringaccessibility standards are met.

1. 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</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <main>
    <h1>User Registration</h1>
    <form id="registration-form" aria-labelledby="registration-heading">
      <fieldset>
        <legend id="registration-heading">Create Your Account</legend>

        <label for="name">Full Name:</label>
        <input type="text" id="name" name="name" required aria-required="true" placeholder="">

        <label for="email">Email Address:</label>
        <input type="email" id="email" name="email" required aria-required="true" placeholder="
```

```
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" required aria-required="true" placeholder="">

            <label for="confirm-password">Confirm Password:</label>
            <input type="password" id="confirm-password" name="confirm-password" required aria-required="true" placeholder="">

            <button type="submit">Register</button>
        </fieldset>
    </form>
  </main>
  <script src="script.js"></script>
</body>
</html>
```

2. 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</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <main>
    <h1>User Login</h1>
    <form id="login-form" aria-labelledby="login-heading">
      <fieldset>
        <legend id="login-heading">Log In to Your Account</legend>

        <label for="login-email">Email Address:</label>
```

```
            <input type="email" id="login-email" name="login-email" required aria-required="true" placeholder=" ">

            <label for="login-password">Password:</label>
            <input type="password" id="login-password" name="login-password" required aria-required="true" placeholder="">

            <button type="submit">Log In</button>
        </fieldset>
    </form>
  </main>
  <script src="script.js"></script>
</body>
</html>
```

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

```
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f4f4f4;
}

main {
  max-width: 600px;
  margin: auto;
  padding: 20px;
  background-color: #fff;
  border-radius: 8px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h1 {
  text-align: center;
  color: #333;
}

fieldset {
  border: none;
  padding: 0;
}
```

```
legend {
  font-size: 1.2em;
  margin-bottom: 10px;
  color: #333;
}

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

input {
  width: 100%;
  padding: 8px;
  margin-bottom: 15px;
  border: 1px solid #ddd;
  border-radius: 4px;
}

button {
  display: block;
  width: 100%;
  padding: 10px;
  background-color: #007bff;
  color: #fff;
  border: none;
  border-radius: 4px;
  font-size: 16px;
  cursor: pointer;
}

button:hover {
  background-color: #0056b3;
}
```

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

1. User Registration Form with Validation
JavaScript Code (script.js):

```
document.addEventListener('DOMContentLoaded', () => {
  const form = document.getElementById('registration-form');

  const nameInput = document.getElementById('name');
  const emailInput = document.getElementById('email');
  const passwordInput = document.getElementById('password');
  const confirmPasswordInput = document.getElementById('confirm-password');

  const nameError = document.getElementById('name-error');
  const emailError = document.getElementById('email-error');
  const passwordError = document.getElementById('password-error');
  const confirmPasswordError = document.getElementById('confirm-password-
error');

  const validateName = () => {
    if (nameInput.value.trim() === '') {
      nameError.textContent = 'Full Name is required.';
      return false;
    } else {
      nameError.textContent = '';
      return true;
    }
  };

  const validateEmail = () => {
    const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailPattern.test(emailInput.value.trim())) {
      emailError.textContent = 'Enter a valid email address.';
      return false;
    } else {
      emailError.textContent = '';
      return true;
    }
  };

  const validatePassword = () => {
    if (passwordInput.value.length < 6) {
      passwordError.textContent = 'Password must be at least 6 characters long.';
```

```
        return false;
      } else {
        passwordError.textContent = '';
        return true;
      }
    };

    const validateConfirmPassword = () => {
      if (passwordInput.value !== confirmPasswordInput.value) {
        confirmPasswordError.textContent = 'Passwords do not match.';
        return false;
      } else {
        confirmPasswordError.textContent = '';
        return true;
      }
    };

    form.addEventListener('input', () => {
      validateName();
      validateEmail();
      validatePassword();
      validateConfirmPassword();
    });

    form.addEventListener('submit', (event) => {
      const isNameValid = validateName();
      const isEmailValid = validateEmail();
      const isPasswordValid = validatePassword();
      const isConfirmPasswordValid = validateConfirmPassword();

      if (!isNameValid || !isEmailValid || !isPasswordValid ||
!isConfirmPasswordValid) {
        event.preventDefault(); // Prevent form submission if there are validation
errors
      }
    });
});
```

User Login Form with Validation:

```
document.addEventListener('DOMContentLoaded', () => {
    const loginForm = document.getElementById('login-form');

    const loginEmailInput = document.getElementById('login-email');
```

```
  const loginPasswordInput = document.getElementById('login-password');

  const loginEmailError = document.getElementById('login-email-error');
  const loginPasswordError = document.getElementById('login-password-error');

  const validateLoginEmail = () => {
    const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailPattern.test(loginEmailInput.value.trim())) {
      loginEmailError.textContent = 'Enter a valid email address.';
      return false;
    } else {
      loginEmailError.textContent = '';
      return true;
    }
  };

  const validateLoginPassword = () => {
    if (loginPasswordInput.value.trim() === '') {
      loginPasswordError.textContent = 'Password is required.';
      return false;
    } else {
      loginPasswordError.textContent = '';
      return true;
    }
  };

  loginForm.addEventListener('input', () => {
    validateLoginEmail();
    validateLoginPassword();
  });

  loginForm.addEventListener('submit', (event) => {
    const isLoginEmailValid = validateLoginEmail();
    const isLoginPasswordValid = validateLoginPassword();

    if (!isLoginEmailValid || !isLoginPasswordValid) {
      event.preventDefault(); // Prevent form submission if there are validation
errors
    }
  });
});
```

Day 2: JavaScript/Bootstrap - Responsive Dashboard for Policy Management
Task 1: Create a dashboard layout with Bootstrap ensuring responsiveness across devices.

HTML :

```
<!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/5.3.0/css/bootstrap.min.css"
>
  <link rel="stylesheet" href="styles.css"> <!-- Optional: Custom CSS -->
</head>
<body>
  <div class="container-fluid">
    <!-- Sidebar -->
    <div class="row">
      <nav id="sidebar" class="col-md-3 col-lg-2 d-md-block bg-light sidebar">
        <div class="position-sticky">
          <h4 class="sidebar-heading">Policy Management</h4>
          <ul class="nav flex-column">
            <li class="nav-item">
              <a class="nav-link active" href="#">
                <i class="bi bi-house-door"></i> Dashboard
              </a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">
                <i class="bi bi-file-text"></i> Policies
              </a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">
                <i class="bi bi-calendar"></i> Claims
              </a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">
                <i class="bi bi-person"></i> Users
              </a>
```

```
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">
                <i class="bi bi-gear"></i> Settings
              </a>
            </li>
          </ul>
        </div>
      </nav>
      <!-- Main Content -->
      <main class="col-md-9 ms-sm-auto col-lg-10 px-4">
        <div class="d-flex justify-content-between flex-wrap flex-md-nowrap align-
items-center pt-3 pb-2 mb-3 border-bottom">
          <h1 class="h2">Dashboard</h1>
        </div>
        <!-- Dashboard Content -->
        <div class="row">
          <div class="col-md-6 col-lg-3 mb-4">
            <div class="card text-white bg-primary">
              <div class="card-body">
                <h5 class="card-title">Total Policies</h5>
                <p class="card-text">1,234</p>
              </div>
            </div>
          </div>
          <div class="col-md-6 col-lg-3 mb-4">
            <div class="card text-white bg-success">
              <div class="card-body">
                <h5 class="card-title">Active Claims</h5>
                <p class="card-text">456</p>
              </div>
            </div>
          </div>
          <div class="col-md-6 col-lg-3 mb-4">
            <div class="card text-white bg-warning">
              <div class="card-body">
                <h5 class="card-title">Pending Approvals</h5>
                <p class="card-text">789</p>
              </div>
            </div>
          </div>
          <div class="col-md-6 col-lg-3 mb-4">
            <div class="card text-white bg-danger">
              <div class="card-body">
```

```
                    <h5 class="card-title">Expired Policies</h5>
                    <p class="card-text">123</p>
                  </div>
                </div>
              </div>
            </div>
          </main>
        </div>
    </div>

    <script
src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/js/bootstrap.bundle.
min.js"></script>
    <script src="script.js"></script> <!-- Optional: Custom JavaScript -->
</body>
</html>
```

Custom CSS:

```
#sidebar {
    height:  100vh;
    position: fixed;
    top: 0;
    left: 0;
    z-index:          100;
    padding-top: 20px;
}

.sidebar-heading {
    padding: 0 1rem;
    font-size: 1.25rem;
    color: #007bff;
}

.nav-link {
    font-size: 1rem;
}

.nav-link.active {
    background-color: #e9ecef;
}

/* Responsive adjustments */
@media (max-width: 768px) {
```

```
  #sidebar {
    position: static;
    height: auto;
  }
}

JAVASCRIPT:
document.addEventListener('DOMContentLoaded', function() {
  // Toggle Sidebar
  const sidebarToggle = document.querySelector('#sidebarToggle');
  const sidebar = document.querySelector('#sidebar');

  sidebarToggle.addEventListener('click', function() {
    sidebar.classList.toggle('d-md-block');
    sidebar.classList.toggle('bg-light');
  });

  // Example Data for Charts
  const policyData = {
    labels: ['January', 'February', 'March', 'April', 'May', 'June', 'July'],
    datasets: [{
      label: 'Policies Created',
      data: [65, 59, 80, 81, 56, 55, 40],
      backgroundColor: 'rgba(75, 192, 192, 0.2)',
      borderColor: 'rgba(75, 192, 192, 1)',
      borderWidth: 1
    }]
  };

  const claimsData = {
    labels: ['January', 'February', 'March', 'April', 'May', 'June', 'July'],
    datasets: [{
      label: 'Claims Processed',
      data: [28, 48, 40, 19, 86, 27, 90],
      backgroundColor: 'rgba(255, 206, 86, 0.2)',
      borderColor: 'rgba(255, 206, 86, 1)',
      borderWidth: 1
    }]
  };

  // Chart.js Initialization
  const ctxPolicy = document.getElementById('policyChart').getContext('2d');
  const policyChart = new Chart(ctxPolicy, {
    type: 'line',
```

```
        data: policyData,
        options: {
            scales: {
                y: {
                    beginAtZero: true
                }
            }
        }
    });

    const ctxClaims = document.getElementById('claimsChart').getContext('2d');
    const claimsChart = new Chart(ctxClaims, {
        type: 'bar',
        data: claimsData,
        options: {
            scales: {
                y: {
                    beginAtZero: true
                }
            }
        }
    });
});
```

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

Utilizing Bootstrap's JavaScript Components

```
<!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/5.3.0/css/bootstrap.min.css"
>
    <link rel="stylesheet" href="styles.css"> <!-- Optional: Custom CSS -->
</head>
<body>
```

```
<div class="container-fluid">
  <!-- Sidebar -->
  <div class="row">
    <nav id="sidebar" class="col-md-3 col-lg-2 d-md-block bg-light sidebar">
      <div class="position-sticky">
        <h4 class="sidebar-heading">Policy Management</h4>
        <ul class="nav flex-column">
          <li class="nav-item">
            <a class="nav-link active" href="#">
              <i class="bi bi-house-door"></i> Dashboard
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#">
              <i class="bi bi-file-text"></i> Policies
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#">
              <i class="bi bi-calendar"></i> Claims
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#">
              <i class="bi bi-person"></i> Users
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#">
              <i class="bi bi-gear"></i> Settings
            </a>
          </li>
        </ul>
      </div>
    </nav>
    <!-- Main Content -->
    <main class="col-md-9 ms-sm-auto col-lg-10 px-4">
      <div class="d-flex justify-content-between flex-wrap flex-md-nowrap align-
items-center pt-3 pb-2 mb-3 border-bottom">
        <h1 class="h2">Dashboard</h1>
      </div>

      <!-- Tabs for different views -->
      <ul class="nav nav-tabs" id="myTab" role="tablist">
```

```
            <li class="nav-item" role="presentation">
              <button class="nav-link active" id="home-tab" data-bs-toggle="tab"
data-bs-target="#home" type="button" role="tab" aria-controls="home" aria-
selected="true">Overview</button>
            </li>
            <li class="nav-item" role="presentation">
              <button class="nav-link" id="policies-tab" data-bs-toggle="tab" data-
bs-target="#policies" type="button" role="tab" aria-controls="policies" aria-
selected="false">Policies</button>
            </li>
            <li class="nav-item" role="presentation">
              <button class="nav-link" id="claims-tab" data-bs-toggle="tab" data-
bs-target="#claims" type="button" role="tab" aria-controls="claims" aria-
selected="false">Claims</button>
            </li>
          </ul>
          <div class="tab-content" id="myTabContent">
            <div class="tab-pane fade show active" id="home" role="tabpanel" aria-
labelledby="home-tab">
              <!-- Dashboard Content -->
              <div class="row mt-3">
                <div class="col-md-6 col-lg-3 mb-4">
                  <div class="card text-white bg-primary">
                    <div class="card-body">
                      <h5 class="card-title">Total Policies</h5>
                      <p class="card-text">1,234</p>
                    </div>
                  </div>
                </div>
                <div class="col-md-6 col-lg-3 mb-4">
                  <div class="card text-white bg-success">
                    <div class="card-body">
                      <h5 class="card-title">Active Claims</h5>
                      <p class="card-text">456</p>
                    </div>
                  </div>
                </div>
                <div class="col-md-6 col-lg-3 mb-4">
                  <div class="card text-white bg-warning">
                    <div class="card-body">
                      <h5 class="card-title">Pending Approvals</h5>
                      <p class="card-text">789</p>
                    </div>
                  </div>
```

```
                </div>
                <div class="col-md-6 col-lg-3 mb-4">
                  <div class="card text-white bg-danger">
                    <div class="card-body">
                      <h5 class="card-title">Expired Policies</h5>
                      <p class="card-text">123</p>
                    </div>
                  </div>
                </div>
              </div>
            </div>
            <div class="tab-pane fade" id="policies" role="tabpanel" aria-
labelledby="policies-tab">
              <!-- Policies Content -->
              <div class="row mt-3">
                <div class="col-12">
                  <div class="d-flex justify-content-between align-items-center mb-
3">
                    <h2>Policies</h2>
                    <button type="button" class="btn btn-primary" data-bs-
toggle="modal" data-bs-target="#policyModal">
                      Add Policy
                    </button>
                  </div>
                  <table class="table table-striped">
                    <thead>
                      <tr>
                        <th scope="col">#</th>
                        <th scope="col">Policy Name</th>
                        <th scope="col">Policy Holder</th>
                        <th scope="col">Status</th>
                        <th scope="col">Actions</th>
                      </tr>
                    </thead>
                    <tbody id="policyTableBody">
                      <!-- Policy rows will be added dynamically here -->
                    </tbody>
                  </table>
                </div>
              </div>
            </div>
            <div class="tab-pane fade" id="claims" role="tabpanel" aria-
labelledby="claims-tab">
              <!-- Claims Content -->
```

```
                <div class="row mt-3">
                  <div class="col-12">
                    <h2>Claims</h2>
                    <p>Manage claims here...</p>
                  </div>
                </div>
              </div>
            </div>
          </main>
        </div>
      </div>

      <!-- Policy Modal -->
      <div class="modal fade" id="policyModal" tabindex="-1" aria-
labelledby="policyModalLabel" aria-hidden="true">
        <div class="modal-dialog">
          <div class="modal-content">
            <div class="modal-header">
              <h5 class="modal-title" id="policyModalLabel">Add New Policy</h5>
              <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
            </div>
            <div class="modal-body">
              <form id="policyForm">
                <div class="mb-3">
                  <label for="policyName" class="form-label">Policy Name</label>
                  <input type="text" class="form-control" id="policyName" required>
                </div>
                <div class="mb-3">
                  <label for="policyHolder" class="form-label">Policy Holder</label>
                  <input type="text" class="form-control" id="policyHolder"
required>
                </div>
                <div class="mb-3">
                  <label for="policyStatus" class="form-label">Status</label>
                  <select class="form-select" id="policyStatus" required>
                    <option value="Active">Active</option>
                    <option value="Pending">Pending</option>
                    <option value="Expired">Expired</option>
                  </select>
                </div>
                <button type="submit" class="btn btn-primary">Add Policy</button>
              </form>
            </div>
```

```
        </div>
      </div>
    </div>

    <script
src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/js/bootstrap.bundle.
min.js"></script>
    <script src="script.js"></script> <!-- Custom JavaScript -->
</body>
</html>
```

Explanation:

- Tabs: Used to switch between different views (Overview, Policies, Claims) using Bootstrap's tab component.
- Modals: Used to add a new policy with a form inside a modal.

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

```
document.addEventListener('DOMContentLoaded', function() {
    // Handle Policy Form Submission
    const policyForm = document.getElementById('policyForm');
    const policyTableBody = document.getElementById('policyTableBody');

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

        const policyName = document.getElementById('policyName').value;
        const policyHolder = document.getElementById('policyHolder').value;
        const policyStatus = document.getElementById('policyStatus').value;

        const newRow = document.createElement('tr');

        newRow.innerHTML = `
            <td></td>
            <td>${policyName}</td>
            <td>${policyHolder}</td>
            <td>${policyStatus}</td>
            <td>
                <button class="btn btn-sm btn-info"
onclick="viewPolicyDetails(this)">View</button>
                <button class="btn btn-sm btn-danger"
onclick="deletePolicy(this)">Delete</button>
            </td>
        `;
```

```
    policyTableBody.appendChild(newRow);
    updatePolicyIndices();

    // Reset form and close modal
    policyForm.reset();
    document.querySelector('.btn-close').click();
  });

  // Update Policy Indices
  function updatePolicyIndices() {
    const rows = policyTableBody.querySelectorAll('tr');
    rows.forEach((row, index) => {
      row.children[0].textContent = index + 1;
    });
  }

  // View Policy Details
  window.viewPolicyDetails = function(button) {
    const row = button.closest('tr');
    const policyName = row.children[1].textContent;
    const policyHolder = row.children[2].textContent;
    const policyStatus = row.children[3].textContent;

    alert(`Policy Details:\n\nName: ${policyName}\nHolder:
${policyHolder}\nStatus: ${policyStatus}`);
  };

  // Delete Policy
  window.deletePolicy = function(button) {
    if (confirm('Are you sure you want to delete this policy?')) {
      const row = button.closest('tr');
      row.remove();
      updatePolicyIndices();
    }
  };
});
```

Explanation:

- Policy Form Submission: Adds new policy rows to the table dynamically and updates indices.
- View Policy Details: Shows an alert with policy details.
- Delete Policy: Deletes a policy row from the table and updates indices.

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

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

Create a ClaimServlet to handle the claim submission workflow.

```
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;
import javax.servlet.http.HttpSession;

@WebServlet("/ClaimServlet")
public class ClaimServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

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

      String policyNumber = request.getParameter("policyNumber");
      String claimType = request.getParameter("claimType");
      String description = request.getParameter("description");
      double amount = Double.parseDouble(request.getParameter("amount"));

      ClaimBean claim = new ClaimBean();
      claim.setPolicyNumber(policyNumber);
      claim.setClaimType(claimType);
      claim.setDescription(description);
      claim.setAmount(amount);

      HttpSession session = request.getSession();
      session.setAttribute("claim", claim);

      response.sendRedirect("confirmClaim.jsp");
   }
}
```

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

claimForm.jsp

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

    <label for="claimType">Claim Type:</label>
    <input type="text" id="claimType" name="claimType" required><br><br>

    <label for="description">Description:</label>
    <textarea id="description" name="description" required></textarea><br><br>

    <label for="amount">Amount:</label>
    <input type="number" id="amount" name="amount" step="0.01"
required><br><br>

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

confirmClaim.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ page import="javax.servlet.http.HttpSession" %>
<%@ page import="ClaimBean" %>
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
```

```
    <title>Confirm Claim</title>
</head>
<body>
    <h2>Confirm Insurance Claim</h2>
    <%
        HttpSession session = request.getSession();
        ClaimBean claim = (ClaimBean) session.getAttribute("claim");

        if (claim != null) {
    %>
        <p><strong>Policy Number:</strong> <%= claim.getPolicyNumber() %></p>
        <p><strong>Claim Type:</strong> <%= claim.getClaimType() %></p>
        <p><strong>Description:</strong> <%= claim.getDescription() %></p>
        <p><strong>Amount:</strong> $<%= claim.getAmount() %></p>

        <form action="processClaim.jsp" method="post">
            <input type="submit" value="Confirm">
        </form>
        <form action="claimForm.jsp" method="get">
            <input type="submit" value="Edit">
        </form>
    <%
        } else {
            out.println("<p>No claim found to confirm.</p>");
        }
    %>
</body>
</html>
```

processClaim.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ page import="javax.servlet.http.HttpSession" %>
<%@ page import="ClaimBean" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Process Claim</title>
</head>
<body>
    <h2>Claim Processed</h2>
    <%
```

```
        HttpSession session = request.getSession();
        ClaimBean claim = (ClaimBean) session.getAttribute("claim");

        if (claim != null) {
            // Here you can add the logic to process the claim, such as saving it to a
database
            session.removeAttribute("claim");
            out.println("<p>Claim has been successfully submitted and
processed.</p>");
        } else {
            out.println("<p>No claim found to process.</p>");
        }
    %>
</body>
</html>
```

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

ClaimBean.java

```
import java.io.Serializable;

public class ClaimBean implements Serializable {
    private static final long serialVersionUID = 1L;

    private String policyNumber;
    private String claimType;
    private String description;
    private double amount;

    // 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 getDescription() {
        return description;
    }

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

    public double getAmount() {
        return amount;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }
}
```

- Servlets: Handle the workflow of submitting insurance claims.
- JSP Pages: Provide a user interface for entering claim information and confirming submissions.
- JavaBeans: Manage data transitions in the claim submission process.

Day 4: Spring Core - Policy Administration Backend

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

Set Up Spring Configuration

Create a SpringConfig class to define the beans.

SpringConfig.java

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

@Configuration
public class SpringConfig {

    @Bean
    public PolicyService policyService() {
```

```
        return new PolicyServiceImpl();
    }

    @Bean
    public PolicyRepository policyRepository() {
        return new InMemoryPolicyRepository();
    }
}
```

- PolicyService.java

```
import java.util.List;

public interface PolicyService {
    void addPolicy(Policy policy);
    Policy getPolicyById(String id);
    List<Policy> getAllPolicies();
}
```

PolicyRepository.java:

```
import java.util.List;

public interface PolicyRepository {
    void save(Policy policy);
    Policy findById(String id);
    List<Policy> findAll();
}
```

PolicyServiceImpl.java

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

import java.util.List;

@Service
public class PolicyServiceImpl implements PolicyService {

    private final PolicyRepository policyRepository;

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

```
    }

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

    @Override
    public Policy getPolicyById(String id) {
        return policyRepository.findById(id);
    }

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

InMemoryPolicyRepository.java

```
import org.springframework.stereotype.Repository;

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

@Repository
public class InMemoryPolicyRepository implements PolicyRepository {

    private final Map<String, Policy> policyMap = new HashMap<>();

    @Override
    public void save(Policy policy) {
        policyMap.put(policy.getId(), policy);
    }

    @Override
    public Policy findById(String id) {
        return policyMap.get(id);
    }

    @Override
    public List<Policy> findAll() {
        return new ArrayList<>(policyMap.values());
```

```
    }
}
```

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

Add Spring Validation Dependencies
Add the following dependencies to your pom.xml:

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

- **Annotate the Policy Class for Validation**

Policy.java

```
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;

public class Policy {

    @NotBlank(message = "Policy ID cannot be blank")
    private String id;

    @NotBlank(message = "Policy name cannot be blank")
    private String name;

    @NotBlank(message = "Policy holder cannot be blank")
    private String holder;

    @NotNull(message = "Amount cannot be null")
    private Double amount;

    // Getters and Setters
    public String getId() {
        return id;
    }

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

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getHolder() {
        return holder;
    }

    public void setHolder(String holder) {
        this.holder = holder;
    }

    public Double getAmount() {
        return amount;
    }

    public void setAmount(Double amount) {
        this.amount = amount;
    }
}
```

Validate Policy in Service Layer
PolicyServiceImpl.java

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

import javax.validation.Valid;
import java.util.List;

@Service
public class PolicyServiceImpl implements PolicyService {

    private final PolicyRepository policyRepository;

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

```
    @Override
    public void addPolicy(@Valid Policy policy) {
        policyRepository.save(policy);
    }

    @Override
    public Policy getPolicyById(String id) {
        return policyRepository.findById(id);
    }

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

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

1 Create Application Context

Application.java

```
Import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Application {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(SpringConfig.class);

        PolicyService policyService = context.getBean(PolicyService.class);

        Policy policy = new Policy();
        policy.setId("1");
        policy.setName("Health Insurance");
        policy.setHolder("John Doe");
        policy.setAmount(1000.00);

        policyService.addPolicy(policy);

        System.out.println("All Policies: " + policyService.getAllPolicies());

        context.close();
    }
```

}

Day 5: Spring MVC - User Claim Interaction Workflow

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

1 Set Up Spring MVC Configuration

Add the necessary dependencies to your pom.xml:

```
<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-validation</artifactId>
</dependency>
```

2 Create the MVC Controller

ClaimController.java

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import javax.validation.Valid;

@Controller
public class ClaimController {

   @Autowired
   private ClaimService claimService;
```

```
    @GetMapping("/submitClaim")
    public String showClaimForm(Model model) {
        model.addAttribute("claim", new Claim());
        return "claimForm";
    }

    @PostMapping("/submitClaim")
    public String submitClaim(@Valid @ModelAttribute("claim") Claim claim,
BindingResult result, Model model) {
        if (result.hasErrors()) {
            return "claimForm";
        }
        claimService.submitClaim(claim);
        model.addAttribute("message", "Claim submitted successfully");
        return "confirmation";
    }
}
```

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

1 Create Thymeleaf Templates

claimForm.html

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

        <label for="claimType">Claim Type:</label>
        <input type="text" id="claimType" th:field="*{claimType}" required>
```

```
        <div th:if="${#fields.hasErrors('claimType')}" th:errors="*{claimType}">Claim
Type Error</div>
        <br><br>

        <label for="description">Description:</label>
        <textarea id="description" th:field="*{description}" required></textarea>
        <div th:if="${#fields.hasErrors('description')}"
th:errors="*{description}">Description Error</div>
        <br><br>

        <label for="amount">Amount:</label>
        <input type="number" id="amount" th:field="*{amount}" step="0.01"
required>
        <div th:if="${#fields.hasErrors('amount')}" th:errors="*{amount}">Amount
Error</div>
        <br><br>

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

2 confirmation.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Confirmation</title>
</head>
<body>
    <h2>Claim Confirmation</h2>
    <p th:text="${message}"></p>
</body>
</html>
```

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

1 Create the Claim Model with Validation Annotations
Claim.java

```
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;
```

```
public class Claim {

    @NotBlank(message = "Policy Number cannot be blank")
    private String policyNumber;

    @NotBlank(message = "Claim Type cannot be blank")
    private String claimType;

    @NotBlank(message = "Description cannot be blank")
    private String description;

    @NotNull(message = "Amount cannot be null")
    private Double amount;

    // 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 getDescription() {
        return description;
    }

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

    public Double getAmount() {
        return amount;
    }
```

```
    public void setAmount(Double amount) {
        this.amount = amount;
    }
}
```

2 Create the Service Layer
ClaimService.java

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

@Service
public class ClaimService {

    private final ClaimRepository claimRepository;

    @Autowired
    public ClaimService(ClaimRepository claimRepository) {
        this.claimRepository = claimRepository;
    }

    public void submitClaim(Claim claim) {
        claimRepository.save(claim);
    }
}
```

3 Updated Controller for Integration
Ensure your controller is correctly integrated with the service:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import javax.validation.Valid;

@Controller
public class ClaimController {

    private final ClaimService claimService;
```

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

    @GetMapping("/submitClaim")
    public String showClaimForm(Model model) {
        model.addAttribute("claim", new Claim());
        return "claimForm";
    }

    @PostMapping("/submitClaim")
    public String submitClaim(@Valid @ModelAttribute("claim") Claim claim,
BindingResult result, Model model) {
        if (result.hasErrors()) {
            return "claimForm";
        }
        claimService.submitClaim(claim);
        model.addAttribute("message", "Claim submitted successfully");
        return "confirmation";
    }
}
```

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.

Claim Entity

Claim.java

java

Copy code

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;

@Entity
public class Claim {

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

```
@NotBlank(message = "Policy Number cannot be blank")
private String policyNumber;

@NotBlank(message = "Claim Type cannot be blank")
private String claimType;

@NotBlank(message = "Description cannot be blank")
private String description;

@NotNull(message = "Amount cannot be null")
private Double amount;

// Getters and Setters
public Long getId() {
   return id;
}

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

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 getDescription() {
   return description;
}

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

```
    }

    public Double getAmount() {
        return amount;
    }

    public void setAmount(Double amount) {
        this.amount = amount;
    }
}
```

2 Policy Entity

Policy.java

```
import javax.persistence.Entity;
import  javax.persistence.GeneratedValue;
import  javax.persistence.GenerationType;
import javax.persistence.Id;
import  javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;

@Entity
public class Policy {

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

    @NotBlank(message = "Policy Name cannot be blank")
    private String name;

    @NotBlank(message = "Policy Holder cannot be blank")
    private String holder;

    @NotNull(message = "Amount cannot be null")
    private Double amount;

    // Getters and Setters
    public Long getId() {
        return id;
    }
```

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getHolder() {
        return holder;
    }

    public void setHolder(String holder) {
        this.holder = holder;
    }

    public Double getAmount() {
        return amount;
    }

    public void setAmount(Double amount) {
        this.amount = amount;
    }
}
```

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

1 ClaimDAO Interface

ClaimDAO.java

```
import java.util.List;

public interface ClaimDAO {
    void save(Claim claim);
    Claim findById(Long id);
    List<Claim> findAll();
    void update(Claim claim);
    void delete(Claim claim);
}
```

ClaimDAOImpl Implementation

ClaimDAOImpl.java
java
Copy code

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

import java.util.List;

@Repository
public class ClaimDAOImpl implements ClaimDAO {

    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public void save(Claim claim) {
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        session.save(claim);
        transaction.commit();
        session.close();
    }

    @Override
    public Claim findById(Long id) {
        Session session = sessionFactory.openSession();
        Claim claim = session.get(Claim.class, id);
        session.close();
        return claim;
    }

    @Override
    public List<Claim> findAll() {
        Session session = sessionFactory.openSession();
        List<Claim> claims = session.createQuery("from Claim", Claim.class).list();
        session.close();
        return claims;
    }

    @Override
    public void update(Claim claim) {
```

```
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        session.update(claim);
        transaction.commit();
        session.close();
    }

    @Override
    public void delete(Claim claim) {
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        session.delete(claim);
        transaction.commit();
        session.close();
    }
}
```

2 PolicyDAO Interface

PolicyDAO.java

```
import java.util.List;

public interface PolicyDAO {
    void save(Policy policy);
    Policy findById(Long id);
    List<Policy> findAll();
    void update(Policy policy);
    void delete(Policy policy);
}
```

PolicyDAOImpl Implementation

PolicyDAOImpl.java

java

Copy code

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

import java.util.List;

@Repository
public class PolicyDAOImpl implements PolicyDAO {
```

```
@Autowired
private SessionFactory sessionFactory;

@Override
public void save(Policy policy) {
   Session session = sessionFactory.openSession();
   Transaction transaction = session.beginTransaction();
   session.save(policy);
   transaction.commit();
   session.close();
}

@Override
public Policy findById(Long id) {
   Session session = sessionFactory.openSession();
   Policy policy = session.get(Policy.class, id);
   session.close();
   return policy;
}

@Override
public List<Policy> findAll() {
   Session session = sessionFactory.openSession();
   List<Policy> policies = session.createQuery("from Policy", Policy.class).list();
   session.close();
   return policies;
}

@Override
public void update(Policy policy) {
   Session session = sessionFactory.openSession();
   Transaction transaction = session.beginTransaction();
   session.update(policy);
   transaction.commit();
   session.close();
}

@Override
public void delete(Policy policy) {
   Session session = sessionFactory.openSession();
   Transaction transaction = session.beginTransaction();
   session.delete(policy);
   transaction.commit();
```

```
        session.close();
    }
}
```

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

HQL Queries

```
List<Claim> claims = session.createQuery("from Claim", Claim.class).list();
```

- **Retrieve claims by policy number:**

```
List<Claim> claims = session.createQuery("from Claim where policyNumber = :policyNumber", Claim.class)
                    .setParameter("policyNumber", "POL12345")
                    .list();
```

- **Criteria Queries**

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Policy> query = builder.createQuery(Policy.class);
Root<Policy> root = query.from(Policy.class);
query.select(root);
List<Policy> policies = session.createQuery(query).getResultList();
```

- **Retrieve policies with amount greater than a specified value:**

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Policy> query = builder.createQuery(Policy.class);
Root<Policy> root = query.from(Policy.class);
query.select(root).where(builder.gt(root.get("amount"), 1000));
List<Policy> policies = session.createQuery(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.

1 Split the Monolithic Application into Microservices

We will create three microservices: Claim Service, Policy Service, and User Service.

2 Set Up Spring Boot Applications

For each microservice, create a new Spring Boot application.

Claim Service

```
<!-- 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-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-h2</artifactId>
  </dependency>
</dependencies>
```

Policy Service

xml

Copy code

```
<!-- 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-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-h2</artifactId>
  </dependency>
</dependencies>
```

- **User Service**

```
<!-- 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-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-h2</artifactId>
  </dependency>
</dependencies>
```

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

- **Set Up Eureka Server**

Create a new Spring Boot application for Eureka Server**.**

Eureka Server

```
<!-- pom.xml -->
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
  </dependency>
</dependencies>
```

- **Application Configuration**

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

properties

```
# application.properties
spring.application.name=eureka-server
server.port=8761

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

- **Configure Eureka Clients**

Add the following dependencies to each microservice's pom.xml:

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

- **Configure each microservice to register with Eureka Server:**

```
# application.properties
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
```

- **Develop Feign Clients**

Add Feign dependencies to pom.xml:

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

- **Enable Feign Clients in the application:**

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

Create Feign clients for inter-service communication:

- **Claim Service Feign Client**

```
@FeignClient(name = "policy-service")
public interface PolicyClient {
   @GetMapping("/policies/{id}")
   Policy getPolicyById(@PathVariable("id") Long id);
}
```

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

- **Set Up Spring Cloud Config Server**

Create a new Spring Boot application for Config Server.

Config Server

```
<!-- pom.xml -->
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-server</artifactId>
  </dependency>
</dependencies>
```

- **Application Configuration**

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

- **properties**

```
# application.properties
spring.application.name=config-server
server.port=8888

spring.cloud.config.server.git.uri=https://github.com/your-repo/config-repo
spring.cloud.config.server.git.clone-on-start=true
```

- **Configure Microservices to Use Config Server**

Add the following dependency to each microservice's pom.xml:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

Configure each microservice to use the Config Server:

- properties

```
# bootstrap.properties
spring.cloud.config.uri=http://localhost:8888
```

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

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

- Add Dependencies

Add the following dependencies to your pom.xml:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-r2dbc</artifactId>
  </dependency>
  <dependency>
    <groupId>io.r2dbc</groupId>
    <artifactId>r2dbc-h2</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-websocket</artifactId>
  </dependency>
</dependencies>
```

Create Reactive Claim Repository

ReactiveClaimRepository.java

java

Copy code

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

public interface ReactiveClaimRepository extends ReactiveCrudRepository<Claim,
Long> {
  Flux<Claim> findByPolicyNumber(String policyNumber);
}
```

- **Create Reactive Service**

ReactiveClaimService.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 ReactiveClaimService {

   @Autowired
   private ReactiveClaimRepository claimRepository;

   public Mono<Claim> createClaim(Claim claim) {
      return claimRepository.save(claim);
   }

   public Mono<Claim> getClaimById(Long id) {
      return claimRepository.findById(id);
   }

   public Flux<Claim> getClaimsByPolicyNumber(String policyNumber) {
      return claimRepository.findByPolicyNumber(policyNumber);
   }

   public Mono<Claim> updateClaim(Claim claim) {
      return claimRepository.save(claim);
   }

   public Mono<Void> deleteClaim(Long id) {
      return claimRepository.deleteById(id);
   }
}
```

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

- **Configure R2DBC in application.properties**

```
spring.r2dbc.url=r2dbc:h2:mem:///testdb
spring.r2dbc.username=sa
```

spring.r2dbc.password=password
spring.h2.console.enabled=true

- Initialize Database

R2DBCConfiguration.java

```
import io.r2dbc.spi.ConnectionFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;
import org.springframework.r2dbc.connection.init.ConnectionFactoryInitializer;
import org.springframework.r2dbc.connection.init.ResourceDatabasePopulator;

@Configuration
public class R2DBCConfiguration {

   @Bean
   public ConnectionFactoryInitializer initializer(ConnectionFactory
connectionFactory) {
      ConnectionFactoryInitializer initializer = new ConnectionFactoryInitializer();
      initializer.setConnectionFactory(connectionFactory);
      initializer.setDatabasePopulator(new ResourceDatabasePopulator(new
ClassPathResource("schema.sql")));
      return initializer;
   }
}
```

- schema.sql

```
CREATE TABLE claim (
   id IDENTITY PRIMARY KEY,
   policy_number VARCHAR(255) NOT NULL,
   claim_type VARCHAR(255) NOT NULL,
   description VARCHAR(255) NOT NULL,
   amount DOUBLE NOT NULL,
   status VARCHAR(255) DEFAULT 'Pending'
);
```

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

- WebSocket Configuration

WebSocketConfig.java

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

```
import org.springframework.web.reactive.config.EnableWebFlux;
import org.springframework.web.socket.config.annotation.EnableWebSocket;
import org.springframework.web.socket.config.annotation.WebSocketConfigurer;
import
org.springframework.web.socket.config.annotation.WebSocketHandlerRegistry;

@Configuration
@EnableWebSocket
@EnableWebFlux
public class WebSocketConfig implements WebSocketConfigurer {

   private final ClaimWebSocketHandler claimWebSocketHandler;

   public WebSocketConfig(ClaimWebSocketHandler claimWebSocketHandler) {
      this.claimWebSocketHandler = claimWebSocketHandler;
   }

   @Override
   public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
      registry.addHandler(claimWebSocketHandler,
"/ws/claims").setAllowedOrigins("*");
   }
}
```

- WebSocket Handler

ClaimWebSocketHandler.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.server.support.WebSocketHandlerAdapt
er;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

import java.time.Duration;

@Component
public class ClaimWebSocketHandler implements WebSocketHandler {

   private final ReactiveClaimService claimService;

   public ClaimWebSocketHandler(ReactiveClaimService claimService) {
```

```
        this.claimService = claimService;
    }

    @Override
    public Mono<Void> handle(WebSocketSession session) {
        Flux<String> claimFlux = claimService.getClaimsByPolicyNumber("POL12345")
                .map(claim -> "Claim ID: " + claim.getId() + ", Status: " + claim.getStatus())
                .delayElements(Duration.ofSeconds(5));

        return session.send(claimFlux.map(session::textMessage));
    }
}
```

- **WebSocket Client**

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Claim Status Updates</title>
    <script>
        let socket = new WebSocket("ws://localhost:8080/ws/claims");

        socket.onmessage = function(event) {
            let claimStatus = event.data;
            let claimStatusDiv = document.getElementById("claimStatus");
            let p = document.createElement("p");
            p.textContent = claimStatus;
            claimStatusDiv.appendChild(p);
        };
    </script>
</head>
<body>
    <h1>Real-time Claim Status Updates</h1>
    <div id="claimStatus"></div>
</body>
</html>
```