

Vendor Management System

Objective:

You will be building Vendor Management System using React (Frontend), Spring Boot Microservices (Backend), and MySQL (database)

Project Overview:

A vendor management system (VMS) is a software solution that helps organizations manage their relationships and interactions with vendors or suppliers. It provides a centralized platform to streamline various aspects of vendor management, including vendor selection, onboarding, performance tracking, contract management, and payments.

Key components and features typically found in a Vendor Management System:

Vendor Database: The VMS maintains a comprehensive database of vendors or suppliers. It includes vendor profiles with details such as company name, contact information, products or services offered, certifications, and performance history.

Vendor Selection and Onboarding: The VMS provides tools and workflows to support the vendor selection and onboarding process. It includes features for requesting and evaluating vendor proposals, conducting due diligence, verifying credentials, and managing vendor contracts. The system helps streamline the process of identifying and engaging with new vendors.

Contract Management: The VMS facilitates contract management activities with vendors. It allows users to create, store, and manage vendor contracts electronically. The system may include features for contract negotiation, version control, automated contract renewals, and tracking key contract terms and milestones. This ensures compliance, reduces manual paperwork, and enhances contract lifecycle management.

Compliance and Risk Management: The VMS helps ensure compliance with regulatory requirements and mitigates vendor-related risks. It may include features for tracking vendor certifications, managing compliance documentation, and conducting risk assessments.

Vendor Performance Reviews: The VMS supports the process of conducting vendor performance reviews or evaluations. It enables users to schedule and document performance reviews, capture feedback from relevant stakeholders, and generate performance reports.

Payments and Invoicing: The VMS may integrate with financial systems or provide invoicing capabilities to manage vendor payments. It enables users to

generate purchase orders, track invoices, and streamline the payment process. Integration with payment gateways or accounting software ensures accurate and timely vendor payments.

Security: Implementation of encryption, secure connections (HTTPS), and protection against fraud and unauthorized access.

DAY 1: Creating the login and registration forms using React

Phase 1: Create a UML use case diagram to represent the different actors and their interactions with the system. Use arrows to show the relationships and dependencies between the actors and use cases for the complete case study mentioned above.

- Set up the environment according to the application needs.
- Set up a new React project using Create React App or your preferred method.
- Create basic (UI) forms for registering and logging in a user using react. Give specific endpoints for the two forms.
- After registration, make the login page functional, i.e. it should redirect the user to the home page (blank as of now). In case of error in any of the credentials, the appropriate error must be displayed.
- In the process, make sure that the user is added to the database and the form reloads on submission and clear errors upon successful registration/login.

DAY 2: Getting user data into Global state and creating app components

- Use Redux to make user data available on global state. This will pass the props from parent to grandchild components without having to pass the props through all child components.
- Design page header component and side panel
- Use React Router for navigation.
- Create navigation section of the header page which includes icons and log-out functionality

DAY 3: Create and design homepage component

- The homepage of any website serves as the default page of that website. That is reason enough for the homepage to be really expressive as well as creative.

- Include a footer section with links to important pages, such as terms and conditions, privacy policy, FAQ, and contact information.
- Include social media icons or links to encourage users to follow your website on various social platforms.

DAY 4: Design other components

- Create and design all other necessary components.
- To attain reusability, pass parameters (referred to as props in React.js) to your functional component.

DAY 5: Adding more functionality to the application

- You can also use React Context API to share data values between components without having to pass a prop through every level of the app tree.
- Add more functionality to the application.

DAY 6: Design the database schema

- Determine the entities and relationships required for the application.
- Create the necessary tables and define the relationships between them.

DAY 7: Set up the backend with Spring Boot

- Create a new Spring Boot projects using your preferred IDE.
- Set up the project dependencies, including Spring Web, Spring Data JPA, and any additional dependencies required for your database (e.g., MySQL).
- Configure the database connection in the application.properties file.
- Create entity classes representing your database tables, with appropriate annotations for mapping to the database.

DAY 8: Spring Boot CRUD operation implementation

- Implement repository interfaces using Spring Data JPA for database operations for all the applications.
- Create RESTful APIs using Spring Web to handle CRUD operations for different entities.
- Create APIs for all other functionality which is required for the application.

DAY 9: Implement Spring security

- Add authentication and authorization mechanisms to secure your APIs,
- Make use of JSON Web Tokens (JWT) or OAuth 2.0.

DAY 10: Implement security measures:

- Implement user registration, login, and session management functionalities.
- Make sure you build a secure application.

DAY 11: Build microservices architecture

- Identify different microservices within your application.
- Create separate modules or projects for each microservice.
- Implement business logic and functionality within each microservice.
- Set up database connections for each microservice using Spring Data JPA or your preferred method.
- Implement RESTful APIs for each microservice, focusing on specific functionality

DAY 12: Setup Service Discovery and Communication

- Choose a service discovery mechanism like Netflix Eureka
- Configure service registration and discovery within each microservice.
- Implement communication between microservices using RESTful APIs or messaging queues.

DAY 13: Implement API gateway with Spring Cloud Gateway

- Create a new Spring Boot project for your API Gateway.
- Include the Spring Cloud Gateway dependency.
- Configure the API routes, load balancing, and routing rules in the API Gateway project.

DAY 14: Set up messaging with RabbitMQ

- Configure RabbitMQ connection details in each microservice project.
- Define message queues, exchanges, and bindings for the communication between microservices.
- Implement event producers and consumers within microservices to publish and consume messages.

Day 15: Integrate frontend with microservices via API Gateway

- Connect the frontend components with the API Gateway.
- Implement API calls to interact with the API Gateway using libraries like Axios.
- Make HTTP requests to the API Gateway routes, which will route the requests to the appropriate microservices.
- Handle responses and update the UI accordingly.

DAY 16: Implement additional features

- Design and implement features like search functionality, filters, sorting, and pagination for listings.
- Implement features like user reviews, ratings.
- Integrate Stripe payment in your React application .
- Make application more creative and attractive by adding more features.
- Give a new name to your application and make a new logo for it

DAY 17: Write test cases

- Identify the critical paths and functionalities in your application.
- Write unit tests for each microservice to verify the behaviour of individual components.
- Use testing frameworks like JUnit to write the test cases.
- Mock external dependencies using tools like Mockito to isolate the components under test.

DAY 18: Test and debug

- Consider writing end-to-end tests to simulate user interactions and verify the overall system behaviour.
- Execute the written test cases and verify that they pass successfully.
- Debug any failures or unexpected behaviours and fix the issues in the code.
- Conduct integration testing to verify the communication between microservices and the API Gateway.
- Debug and fix any issues that arise during testing.

Day 19: Dockerizing Spring Boot (Microservices) App and deployment

- Build Docker images for each of your microservices and the API Gateway.
- Create a Dockerfile in each microservice and API Gateway project to define the image build process.

- Include all necessary dependencies and configurations in the Docker images.
- Push the Docker images for your microservices and API Gateway to the container registry (Docker hub).
- Log in to the AWS Management Console and navigate to Amazon ECS.
- Create a new ECS cluster that will host your Docker containers.

Day 20: Deploy the Microservices, API Gateway, and RabbitMQ

- Create an ECS task definition and ECS service for each microservice and the API Gateway.
- Setup load balancing by creating an Application Load Balancer (ALB) in AWS
- Configure the ALB to distribute incoming traffic to your microservices and API Gateway.
- Set up listeners and target groups to route requests to the appropriate ECS services.
- Create security groups to control inbound and outbound traffic for your ECS instances.
- Use the AWS Management Console or the AWS Command Line Interface (CLI) to deploy your ECS services.
- Create ECS service tasks based on the task definitions you created earlier.
- Monitor the deployment process and ensure that the tasks are running successfully.

Congrats!!! You have successfully built and deployed Vendor Management System