QuitQueue

Problem statement:

In this case study, we will design and develop a fullstack E-Commerce application called "QuitQueue". System is a comprehensive application designed to facilitate e-commerce transactions between buyers and sellers. This system encompasses the entire shopping process, from product browsing and selection to payment processing and order fulfillment. It aims to provide a seamless and secure shopping experience for customers while enabling efficient management for sellers.

Scope

1. **User Registration and Authentication:** Allow users to register, log in, and securely manage their accounts.
2. **Product Catalog:** Display a wide range of products with detailed information, including images, descriptions, and prices.
3. **Shopping Cart:** Enable users to add, remove, and manage items in their shopping carts before checkout.
4. **Order Processing:** Handle the purchase process, including order placement, payment processing, and order confirmation.
5. **Inventory Management:** Help sellers manage product listings, track inventory, and update product availability.
6. **Seller Dashboard:** Provide sellers with tools to manage their online stores, track orders, and update product listings.

Technologies:

- Frontend: React.js / Angular Js.
- Backend: Java, Spring Boot/C#, .Net / Python Djnago for API development.
- Database: MySql / Sql Server.
- Authentication: JSON Web Tokens (JWT) for secure user authentication.

Use case Diagram:

Use Cases:

Actor: User

- Use Case: User Registration and login.
- Use Case: Product Browsing.
- Use Case: Cart Management.
- Use Case: Checkout and Payment.
- Use Case: View orders history.

Actor: Seller

- Use Case: Log In as an seller
- Use Case: Seller Product Management
- Use Case: Order Processing
- Use Case: Sales Reports

Actor: Administrator

- Use Case: Log In
- Use Case: User Management
- Use Case: Product Catalog Management
- Use Case: Generate Reports(As simple list view)

System: Security and Authentication

- Use Case: Authenticate User

System: Database Management

- Use Case: Store Product Information
- Use Case: Store Order Information
- Use Case: Store User and Seller Data

Development Process:

1. **User and seller Registration:**
    - User and seller can create accounts, providing personal details (name, gender, contact number, address, etc.)
    - The system validates the information and creates user profiles.
    - Users and sellers log in using their credentials (username/email and password).
2. **User's Dashboard:**
    - Users can browse products, view detailed descriptions, images, and prices.
    - Users navigate through product categories (such as Electronics, Fashion, Mobile, Home & Furniture) or search for specific items.
        1. Include filters for brands, and price ranges.
        2. Implement a search feature with auto-suggestions and predictive text.
        3. Clicking on product from listing product to view more information about a product. Product details, images, pricing, and seller name are displayed.
    - Users can add products to their shopping carts for purchase.
        1. Users click an "Add to Cart" button on the product page.
        2. The selected product is added to their shopping cart.

3. **User's and Cart Management:**

    - Users can view and manage items in their shopping carts, including adding, removing, or updating quantities.

        1. Calculate and display the total order cost.
        2. Enable customers to review and edit their cart before placing an order.
        3. They can remove items, update quantities, or proceed to checkout.
        4. Users click the "Checkout" button from their shopping cart.
        5. They provide shipping details and select a payment method.
        6. Payment information is securely processed.

4. **Seller Dashboard:**

    - Seller can create new product listings, edit existing ones, and mark products as out of stock.
    - Seller can add product with following details (category name, product name, price, stock number)
    - Sellers access their order management dashboard.
    - They can view the status of orders, including processing and shipping updates.
    - The seller can view history of orders of their product.

5. **Administrators Dashboard**

    - Administrators can manage(delete) user accounts and seller accounts.
    - Admins can add, modify, or remove product categories.

6. **Security and Compliance:**

    - User authentication and authorization are enforced to ensure data privacy.

    1. JWT Authentication:

    JWT authentication involves generating a token upon successful user login and sending it to the client. The client includes this token in subsequent requests to authenticate the user.

    - User Login: Upon successful login (using valid credentials), generate a JWT token on the server.
    - Token Payload: The token typically contains user-related information (e.g., user ID, roles, expiration time).
    - Token Signing: Sign the token using a secret key known only to the server. This ensures that the token hasn't been tampered with.

- Token Transmission: Send the signed token back to the client as a response to the login request.
- Client Storage: Store the token securely on the client side (e.g., in browser storage or cookies).

2. JWT Authorization:

JWT authorization involves checking the token on protected routes to ensure that the user has the required permissions.

- Protected Routes: Define routes that require authentication and authorization.
- Token Verification:
    1. Extract the token from the request header.
    2. Verify the token's signature using the server's secret key.
- Payload Verification:
    1. Decode the token and extract user information.
    2. Check user roles or permissions to determine access rights.
- Access Control: Grant or deny access based on the user's roles and permissions.

Logout:

- Logging out involves invalidating the JWT token on both the client and the server to prevent further unauthorized requests.

Project Development Guidelines

The project to be developed based on the below design considerations.

1	**Backend Development**	• Use Rest APIs (Springboot/ASP.Net Core WebAPI to develop the services • Use Java/C# latest features • Use ORM with database • perform backend data validation • Use Swagger to invoke APIs • Implement API Versioning • Implement security to allow/disallow CRUD operations • Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant. • Any error message or exception should be logged and should be user-readable (not technical) • Database connections and web service URLs should be configurable

		• Implement Unit Test Project for testing the API • Implement JWT for Security • Implement Logging • Follow Coding Standards with proper project structure.

Frontend Constraints

1.	**Layout and Structure**	Create a clean and organized layout for your registration and login pages. You can use a responsive grid system (e.g., Bootstrap or Flexbox) to ensure your design looks good on various screen sizes.
2	**Visual Elements**	**Logo:** Place your application's logo at the top of the page to establish brand identity.
		Form Fields: Include input fields for email/username and password for both registration and login. For registration, include additional fields like name and possibly a password confirmation field.
		Buttons: Design attractive and easily distinguishable buttons for "Register," "Login," and "Forgot Password" (if applicable).
		Error Messages: Provide clear error messages for incorrect login attempts or registration errors.
		Background Image: Consider using a relevant background image to add visual appeal.
		Hover Effects: Change the appearance of buttons and links when users hover over them.
		Focus Styles: Apply focus styles to form fields when they are selected
3.	**Color Scheme and Typography**	Choose a color scheme that reflects your brand and creates a visually pleasing experience. Ensure good contrast between text and background colors for readability. Select a legible and consistent typography for headings and body text.
4.	**Registration Page, add product page by seller, add shipping address page by user**	**Form Fields:** Include fields for users to enter their name, email, password, and any other relevant information. Use placeholders and labels to guide users.
		Validation: Implement real-time validation for fields (e.g., check email format) and provide immediate feedback for any errors. **Form Validation:** Implement client-side form validation to ensure required fields are filled out correctly before submission.
	Registration Page	**Password Strength:** Provide real-time feedback on password strength using indicators or text. **Password Requirements**: Clearly indicate password requirements (e.g., minimum length, special characters) to help users create strong passwords.
		Registration Success: Upon successful registration, redirect users to the login page.
5.	**Login Page**	**Form Fields:** Provide fields for users to enter their email and password.
		Password Recovery: Include a "Forgot Password?" link that allows users to reset their password.

6.	**Common to React/Angular**	• Use Angular/React to develop the UI. • Implement Forms, databinding, validations, error message in required pages. • Implement Routing and navigations. • Use JavaScript to enhance functionalities. • Implement External and Custom JavaScript files. • Implement Typescript for Functions Operators. • Any error message or exception should be logged and should be user-readable (and not technical). • Follow coding standards. • Follow Standard project structure. • Design your pages to be responsive so they adapt well to different screen sizes, including mobile devices and tablets.

Good to have implementation features

- Generate a SonarQube report and fix the required vulnerability.
- Use the Moq framework as applicable.
- Create a Docker image for the frontend and backend of the application .
- Implement OAuth Security.
- Implement design patterns.
- Deploy the docker image in AWS EC2 or Azure VM.
- Build the application using the AWS/Azure CI/CD pipeline. Trigger a CI/CD pipeline when code is checked-in to GIT. The check-in process should trigger unit tests with mocked dependencies.
- Use AWS RDS or Azure SQL DB to store the data.