Local Business Digital Transformation System

1.Functional Requirements

The functional requirements define what the system should do. They include:

- **User Management:** The system must allow users to register, log in, log out, change password, and manage their profiles.
- **Vendor Management:** Vendors should be able to register, log in, manage their profiles, and view products associated with them.
- **Product Management:** The system must support adding, updating, deleting, and retrieving product information, along with managing reviews.
- **Order Management:** The system must allow users to place orders, view order history, and track order status.
- **Payment Processing:** The system must handle payment transactions securely and update order statuses accordingly.

2.Non-Functional Requirements

Non-functional requirements define how the system performs its functions:

- **Performance:** The system should handle at least 1000 concurrent users with response times under 200ms.
- **Scalability:** The architecture should allow horizontal scaling to support increased load.
- **Security:** Data must be encrypted in transit and at rest. Use JWT for authentication and authorization.
- **Availability:** The system should have an uptime of 99.9%, with failover mechanisms in place.
- **Maintainability:** Code should follow SOLID principles, with high modularity and low coupling.

3.Architecture Overview

The architecture is based on a microservices pattern, where each service is responsible for a specific business capability. Each microservice runs independently, communicating with others via HTTP REST APIs. The key components include:

- **API Gateway:** Acts as a single-entry point for all client requests, routing them to the appropriate microservice.
- **Authentication Service:** Manages user and vendor authentication using JWT.
- **User Service**: Handles user-related operations such as registration, login, change password, and profile management.
- **Vendor Service**: Manages vendor-related operations, including registration, login, and profile management.
- **Product Service:** Manages the product catalog, including CRUD operations, category management, and review functionality.
- **Order Service:** Processes orders, manages order items, and tracks order status.
- **Payment Service:** Processes payments and updates order statuses.
- **Notification Service:** Sends email and SMS notifications to users regarding their orders.

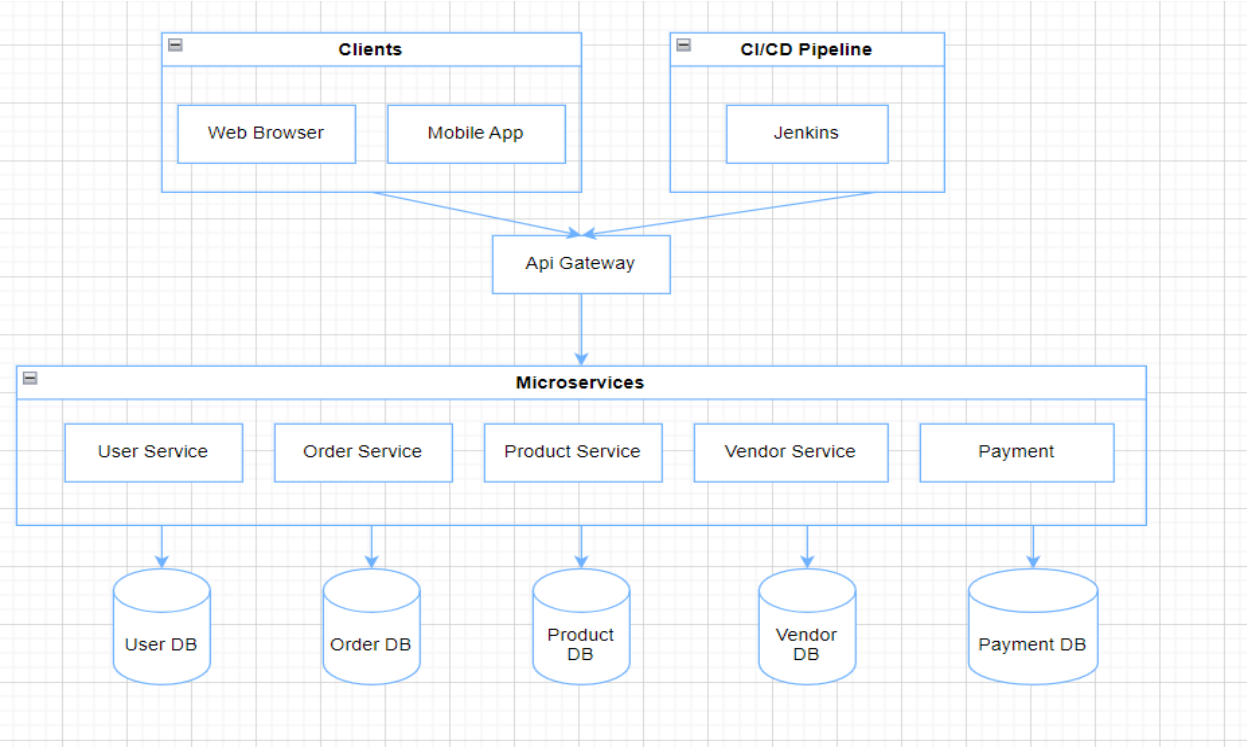

Fig-1: Architecture Diagram

4.Service End Points.

Each service exposes a set of RESTful APIs that can be consumed by other services or the frontend. The endpoints are as follows:

User Service API:

1. ‘POST /api/users/register’: Registers a new user.
2. ‘POST /api/users/login‘: Authenticates a user and returns a JWT token.
3. ‘PUT /api/users/email‘: Updates a user’s details by email.
4. ‘DELETE /api/users/email‘: Deletes a user by email.
5. ‘GET /api/users/email‘: Retrieves a user by email.
6. ‘GET /api/users‘: Lists all users.
7. ‘GET /api/users/userId/orders‘: Retrieves all orders associated with a specific user.

Vendor Service API:

1. ‘POST /api/vendors/register‘: Registers a new vendor.
2. ‘POST /api/vendors/login‘: Authenticates a vendor and returns a JWT token.
3. ‘PUT /api/vendors/id‘: Updates a vendor’s details by ID.
4. ‘DELETE /api/vendors/id‘: Deletes a vendor by ID.
5. ‘GET /api/vendors/id‘: Retrieves a vendor by ID.
6. ‘GET /api/vendors/contact/contactMail‘: Retrieves a vendor by contact email.
7. ‘DELETE /api/vendors/contact/contactMail‘: Deletes a vendor by contact email.
8. ‘GET /api/vendors‘: Lists all vendors.
9. ‘GET /api/vendors/id/products‘: Retrieves all products associated with a specific vendor.

Product Service API:

1. ‘POST /api/products‘: Adds a new product.
2. ‘GET /api/products‘: Retrieves a list of all products.
3. ‘GET /api/products/id‘: Retrieves details of a specific product, including reviews.

4. ‘GET /api/products/vendor/vendorId‘: Retrieves all products associated with a specific vendor.
5. ‘GET /api/products/category/categoryId‘: Retrieves all products under a specific category.
6. ‘PUT /api/products/id‘: Updates an existing product.
7. ‘DELETE /api/products/id‘: Deletes a product.
8. ‘GET /api/products/id/reviews‘: Retrieves all reviews associated with a specific product.

Order Service API:

1. ‘POST /api/orders‘: Creates a new order.
2. ‘GET /api/orders‘: Lists all orders.
3. ‘GET /api/orders/id‘: Retrieves an order by ID.
4. ‘PUT /api/orders/id‘: Updates an existing order by ID.
5. ‘DELETE /api/orders/id‘: Deletes an order by ID.
6. ‘GET /api/orders/user/userId‘: Retrieves all orders associated with a specific user.
7. ‘GET /api/orders/order-items/orderId‘: Retrieves all order items associated with a specific order.

Notification API

1. POST /api/notifications/send`: Send a notification to customers.
2. PUT /api/notifications/preferences`: Update customer notification preferences.
3. GET /api/notifications`: Retrieve notification history for a customer.

Payment API

1. POST /api/payments: Processes a new payment.
2. GET /api/payments: Retrieves a list of all payments.
3. GET /api/payments/id: Retrieves a payment by its ID.
4. GET /api/payments/order/orderId: Retrieves payments associated with a specific order.

5. GET /api/payments/user/userId: Retrieves all payments associated with a specific user.
6. PUT /api/payments/id: Updates an existing payment.
7. DELETE /api/payments/id: Deletes a payment by ID.

5.Activity Flow Diagram

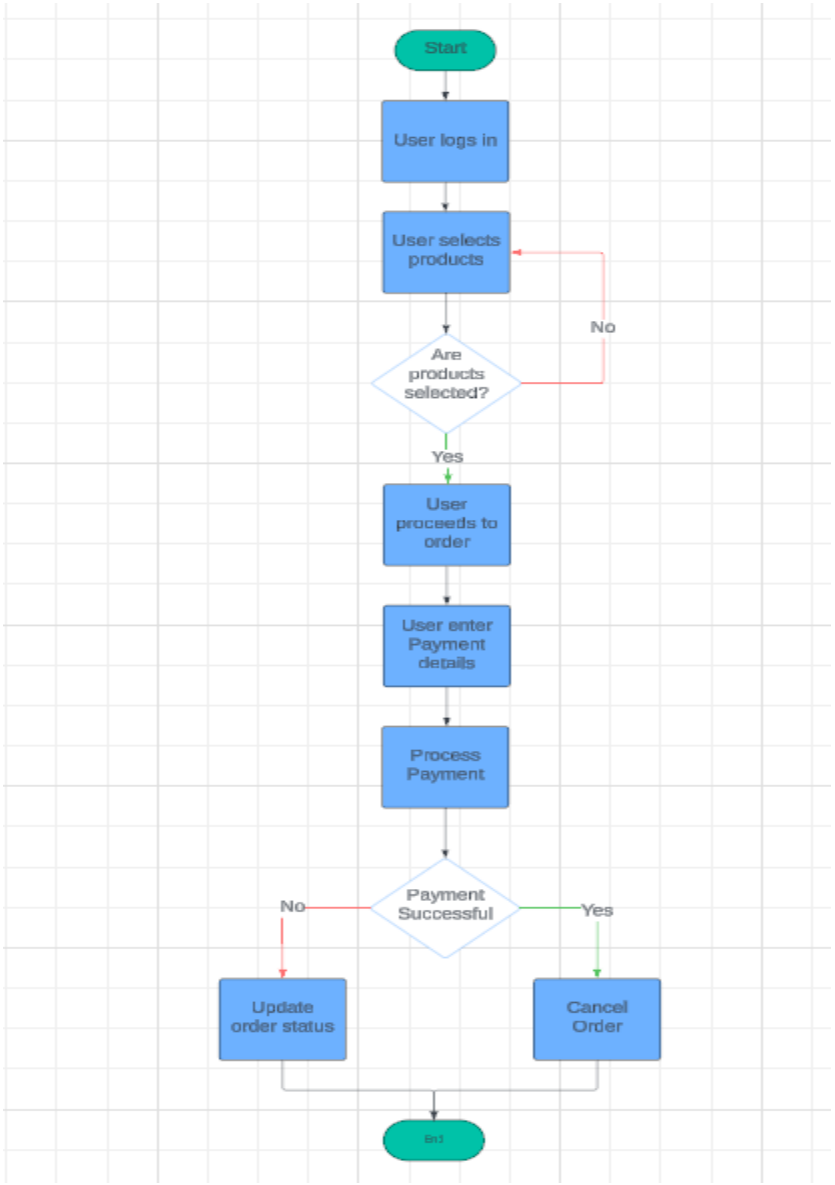

Fig 2: Activity Flow Diagram

6.Technologies Used

The project employs the following technologies:

- Programming Language: Java 21 for backend services.
- Frameworks:
    1. Spring Boot for building microservices.
    2. Spring Security for authentication and authorization.
    3. Spring Cloud Netflix Eureka for service discovery.
    4. Spring Cloud Gateway for API routing.
- Database: MongoDB for storing persistent data.
- API Communication: RESTful APIs using Spring Web.
- Frontend: Angular for building the user interface.
- Containerization: Docker for containerizing microservices

7.Database Schema

The following diagram illustrates the database schema for the E-Commerce platform:

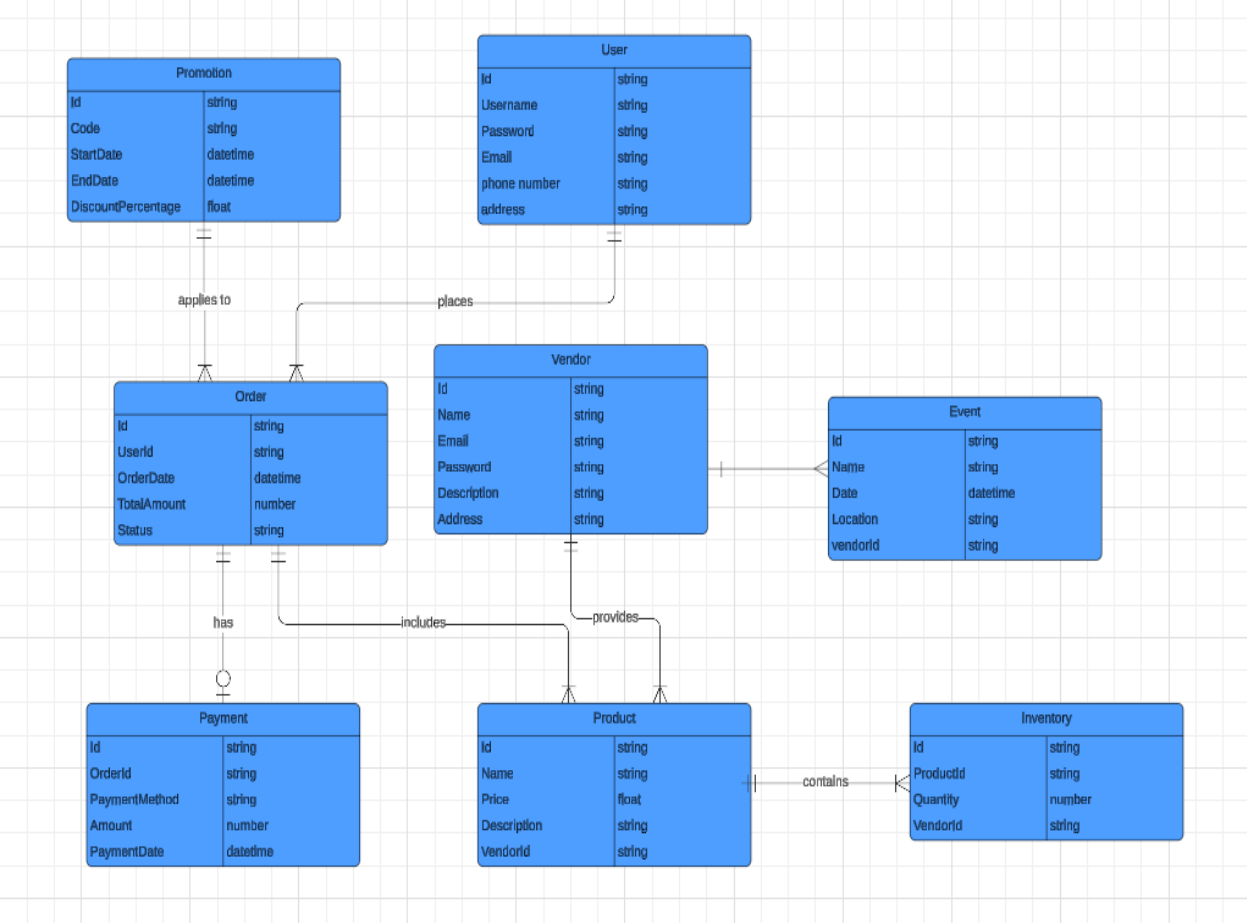

Fig 3: Database Schema

8.Wire Frame Diagrams:

This section describes how different components interact with each other. The interactions are illustrated using a wireframe diagram that shows the flow between different pages and components of the system.

For example:

- The homepage allows users to log in or browse products by category.
- After logging in, users can access their profile, view order history, and manage their cart.
- Vendors can view and manage products they have listed on the platform.
- The checkout process involves interaction between the Order, Payment, and Notification services.

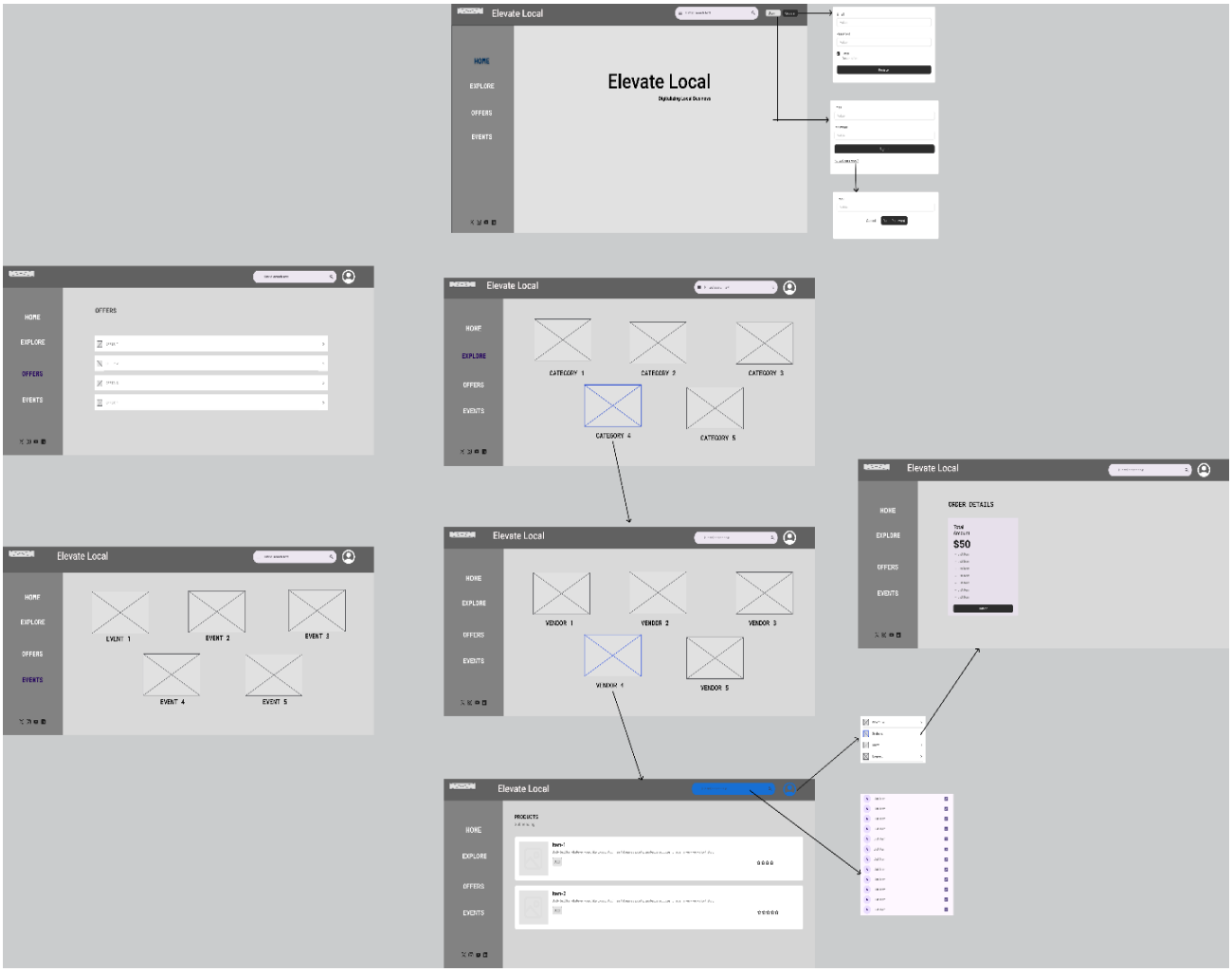

Fig 4 : Wireframe Diagram