

Ecommerce application

Microservices and Containerization based assignment



April 10, 2025

COUSE: SCALABLE SERVICES

Submitted By: VISHAL SHARMA, ID: 2023TM93550

#### **Github URL:** <https://github.com/vissharm/ecommerce-app/tree/feature/submodules> **Demo:** <https://drive.google.com/file/d/1nrV_Jwk2QDcnzVpjbdjJCRxSF97TYh4R/view?usp=drive_link>

*Thanks.  
THE ASSIGNMENT HAS BEEN POSSIBLE DUE TO THE effort you put in covering modern technologies and the use cases for applicatons of these technologies.*

Contents

[**Assignment objective** 1](#_Toc195451191)

[Key highlights: 1](#_Toc195451192)

[**Technologies Stack** 1](#_Toc195451193)

[Backend 1](#_Toc195451194)

[Frontend 2](#_Toc195451195)

[DevOps & Infrastructure 2](#_Toc195451196)

[**Development phases** 2](#_Toc195451197)

[**Phase 1: Local Development (Monolithic Repository)** 2](#_Toc195451198)

[Key achievements: 2](#_Toc195451199)

[o Directory structure 2](#_Toc195451200)

[**Phase 2: Microservices Architecture (Multi-Repository)** 3](#_Toc195451201)

[Key changes: 3](#_Toc195451202)

[Submodules setup 3](#_Toc195451203)

[**Phase 3: Docker Containerization** 4](#_Toc195451204)

[Key implementations: 5](#_Toc195451205)

[**Phase 4: Minikube cluster deployment** 5](#_Toc195451206)

[Key implementations: 5](#_Toc195451207)

[**Service Details** 5](#_Toc195451208)

[Required Software Versions 6](#_Toc195451209)

[Framework Versions 6](#_Toc195451210)

[**System Architecture** 7](#_Toc195451211)

[**Data Flow diagram** 7](#_Toc195451212)

[**Architecture Documentation** 7](#_Toc195451213)

[**System Components** 7](#_Toc195451214)

[**Event Flow Patterns** 8](#_Toc195451215)

[Order Processing Flow 8](#_Toc195451216)

[**Security Architecture** 8](#_Toc195451217)

[**Repository Setup** 8](#_Toc195451218)

[**Deployment Options** 9](#_Toc195451219)

[1. Local Development (Without Containers) 9](#_Toc195451220)

[NPM Commands added in root package.json for local development 9](#_Toc195451221)

[Actual scripts created and called by above commands are mentioned below. 10](#_Toc195451222)

[Key points for start-services.bat: 10](#_Toc195451223)

[2. Docker Compose Deployment 10](#_Toc195451224)

[NPM Commands added in root package.json for local development 10](#_Toc195451225)

[Additional docker commands 11](#_Toc195451226)

[3. **Minikube(Kubernetes) Deployment** 11](#_Toc195451227)

[NPM Commands added in root package.json for local development 11](#_Toc195451228)

[Deploy services: 11](#_Toc195451229)

[Check deployment status: 11](#_Toc195451230)

[Get application URL: 12](#_Toc195451231)

[Pod management: 12](#_Toc195451232)

[Debugging: 12](#_Toc195451233)

[**Service URLs** 12](#_Toc195451234)

[**API Documentation** 12](#_Toc195451235)

[Authentication Endpoints 12](#_Toc195451236)

[User Service (Port: 3001) 13](#_Toc195451237)

[Product Service (Port: 3004) 13](#_Toc195451238)

[Order Service (Port: 3002) 13](#_Toc195451239)

[Notification Service (Port: 3003) 13](#_Toc195451240)

[**Directory structure** 14](#_Toc195451241)

[Deployment Script (k8s-deploy.ps1) 15](#_Toc195451242)

[Purpose 15](#_Toc195451243)

[Setup Script (setup.ps1) 16](#_Toc195451244)

[Purpose 16](#_Toc195451245)

[Key Operations 16](#_Toc195451246)

[Database Initialization Script (initialize\_database.js) 17](#_Toc195451247)

[Purpose 17](#_Toc195451248)

[Data Initialization 17](#_Toc195451249)

[Features 17](#_Toc195451250)

[Usage 17](#_Toc195451251)

[Service Starter Script (start-services.bat) 17](#_Toc195451252)

[Purpose 17](#_Toc195451253)

[Service Launch Order 17](#_Toc195451254)

[Features 17](#_Toc195451255)

[Usage 18](#_Toc195451256)

[**MongoDB compass screenshot** 19](#_Toc195451257)

[**Application screenshots** 20](#_Toc195451258)

[Landing page / homepage 20](#_Toc195451259)

[Responsive UX & dashboard/home page 21](#_Toc195451260)

[Login screen with validations 21](#_Toc195451261)

[Login screen 22](#_Toc195451262)

[Products page / Inventory management 23](#_Toc195451263)

[Add Product 23](#_Toc195451264)

[Notifications support 24](#_Toc195451265)

[Profile page with editing support 24](#_Toc195451266)

[Top navigation bar 24](#_Toc195451267)

[Order management – Create order 25](#_Toc195451268)

[Order list 25](#_Toc195451269)

[Order creation and Kafka-websocket real time notification integration & flow 27](#_Toc195451270)

# **Assignment objective**

**Microservices-Based Application Development**

1. Develop a **microservices-based application** with at least **four microservices**. Each microservice should be maintained in a separate **code repository** to enable independent development, deployment, and testing.
2. Choose an appropriate **database** and apply a suitable **database design pattern** for these services.
3. Implement an efficient **communication mechanism** between the microservices, ensuring **low coupling** to maintain flexibility and scalability.

**Exploring Deployment Tools**

1. Deploy each microservice in an **individual Docker container**.
2. Set up a **Minikube cluster** on your local machine, explore its various functionalities, and attempt to deploy your application on it.

#### **Github URL:** <https://github.com/vissharm/ecommerce-app/tree/feature/submodules> **Demo:** <https://drive.google.com/file/d/1nrV_Jwk2QDcnzVpjbdjJCRxSF97TYh4R/view?usp=drive_link>

###### Application domain selected for assignment

A modern **basic e-commerce platform** built using microservices architecture. The application demonstrates enterprise-level patterns with Node.js-based services, React frontend, and event-driven communication. It features real-time order processing, inventory management, and user notifications through WebSocket integration.

## Key highlights:

* Microservices architecture with API Gateway pattern
* Event-driven design using Kafka for service communication
* Real-time updates via Socket.IO
* Containerized deployment with Docker and Kubernetes support
* Shared library approach for code reusability
* Comprehensive development and deployment options

The platform supports essential e-commerce operations including user management, product catalog, shopping cart, order processing, and real-time notifications. Built with scalability in mind, it utilizes MongoDB for data persistence, Redis for caching, and includes monitoring capabilities.

## **Technologies Stack**

### Backend

* **Node.js** - Runtime environment
* **Express.js** - Web framework
* **MongoDB** - Primary database
* **Kafka** - Message broker for event-driven architecture
* **JWT** - Authentication and authorization
* **Socket.IO** - Real-time notifications

### Frontend

* **React.js** - UI library
* **Redux** - State management
* **Material-UI** - Component library
* **Axios** - HTTP client

### DevOps & Infrastructure

* **Docker** - Containerization
* **Kubernetes** / Minikube- Container orchestration

# **Development phases**

I have done the development in four phases or steps.

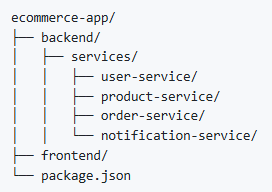
## **Phase 1: Local Development (Monolithic Repository)**

* Created master branch with all services in a single repository
* Basic project structure setup
* Local development with Express server

### Key achievements:

* + Basic service functionality
  + Local inter-service communication
  + Frontend-backend integration
  + Basic authentication flow
  + Database connectivity

### Directory structure



## **Phase 2: Microservices Architecture (Multi-Repository)**

* Split services into separate repositories
* Added repositories as submodules to main project
* Updated local development workflow

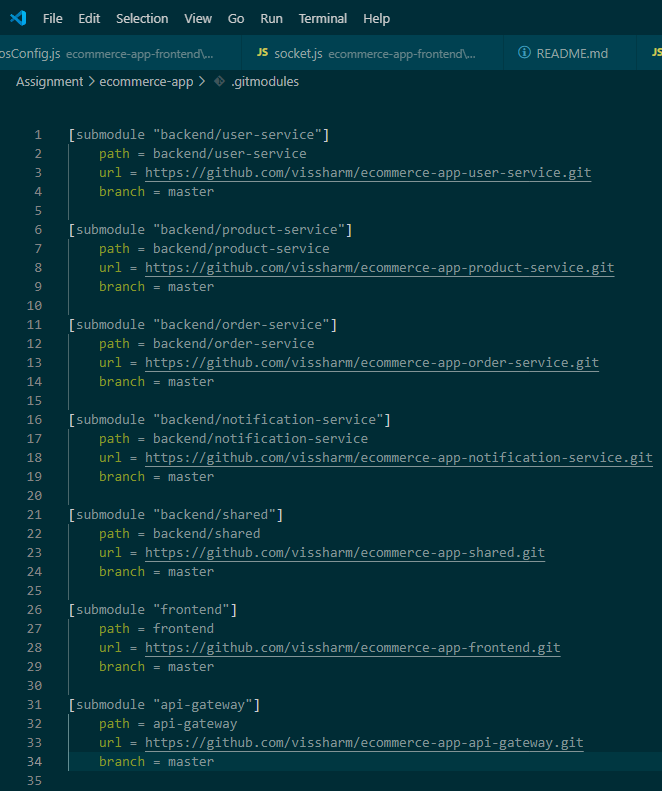
### Key changes:

* + Created separate repositories for each service
  + Implemented shared library
  + Updated service communication
  + Modified development scripts
  + Each service accesses its own database
  + Teams can do parallel development on submodules and test the api’s via postman locally

### Submodules setup

1. "**backend/user-service**" = "https://github.com/vissharm/ecommerce-app-user-service.git"
2. "**backend/product-service**" = "https://github.com/vissharm/ecommerce-app-product-service.git"
3. "**backend/order-service**" = "https://github.com/vissharm/ecommerce-app-order-service.git"
4. "**backend/notification-service**" = "https://github.com/vissharm/ecommerce-app-notification-service.git"
5. "**backend/shared**" = "https://github.com/vissharm/ecommerce-app-shared.git"
6. "**frontend**" = "https://github.com/vissharm/ecommerce-app-frontend.git"
7. "**api-gateway**" = <https://github.com/vissharm/ecommerce-app-api-gateway.git>
8. All these git sub-modules are referred to in **ecommerce-app** main root repository. **Repository**: [ecommerce-app/submodules](https://github.com/vissharm/ecommerce-app/tree/feature/submodules)

**Note:** .gitmodules file added in the root repository which cotains all the linked repositories information and allow user to setup and work on whole application also via ecommerce-app repository.



**# Initialize and update ecommerce-app with all git modules**

git submodule init

git submodule update --recursive –remote

## **Phase 3: Docker Containerization**

I used docker-compose to begin with easy containerization and deployment.

* Implemented Docker deployment
* Created docker-compose configuration
* Resolved container networking issues

### Key implementations:

* + Service Dockerfiles
  + Multi-stage builds
  + Shared library container

## **Phase 4: Minikube cluster deployment**

* Implemented Kubernetes manifests
* Created deployment automation scripts
* Configured service discovery

### Key implementations:

* + Kubernetes manifests for all services
  + Health check endpoints
  + Resource limits and requests
  + Automated deployment scripts

# **Service Details**

**API Gateway**

* Route management
* Request validation
* Authentication & Authorization
* Metrics/Logging

**User Service**

* User management
* Authentication
* Profile management

**Product Service**

* Product catalog management
* Inventory management

**Order Service**

* Order processing
* Order status management

**Notification Service**

* Push notifications
* Real-time updates

**Frontend Service**

* React-based user interface
* State management
* Material-UI components
* Responsive design
* Client-side routing
* Form validation
* Real-time updates via WebSocket

**Shared Library**

* Common utilities and models
* Versioned package (1.0.0)
* Used by all services
* Built as npm package
* Mounted in Kubernetes deployments

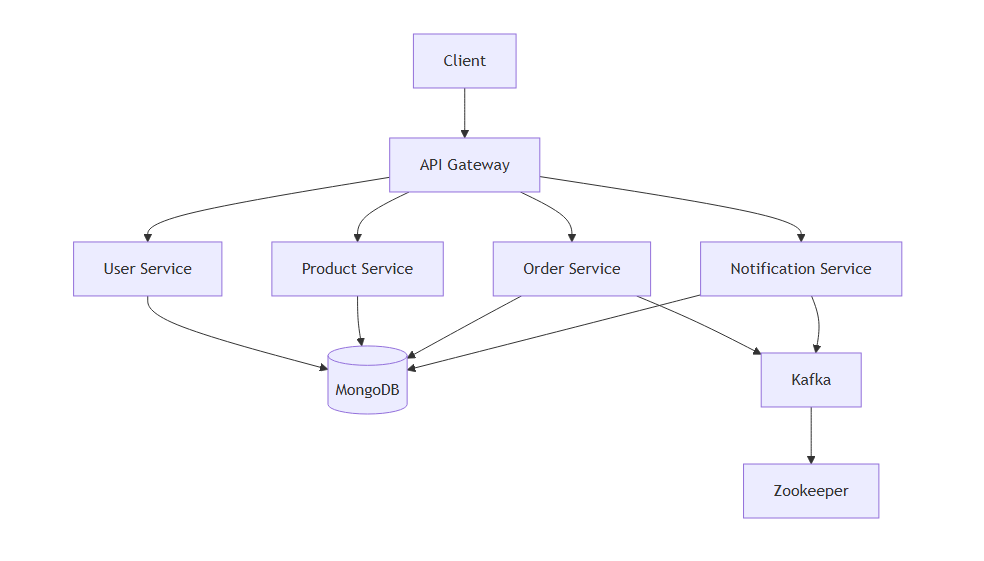
## Required Software Versions

* Node.js >= 14.x
* npm >= 6.x
* Docker >= 20.10.x
* Kubernetes >= 1.19.x
* Minikube >= 1.25.x
* Git >= 2.x
* MongoDB >= 4.4.x
* Apache Kafka >= 3.6.x
* Apache Zookeeper >= 3.7.x

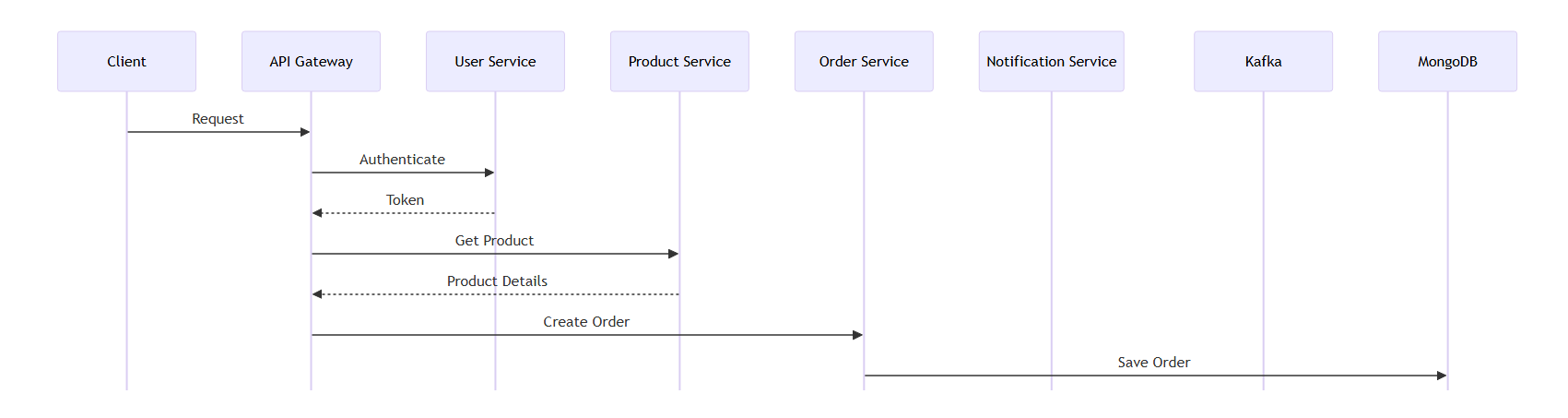
## Framework Versions

* Express.js >= 4.17.x
* React.js >= 17.0.x
* Material-UI >= 4.11.x
* Socket.IO >= 4.8.x
* JWT >= 8.5.x

# **System Architecture**



# **Data Flow diagram**



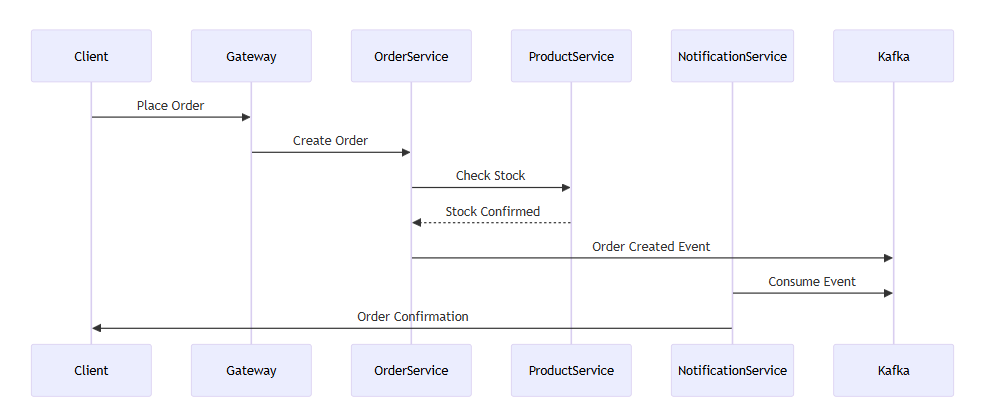
# **Architecture Documentation**

## **System Components**

1. **API Gateway**
   * Rate limiting: 100 requests/minute
   * JWT validation
   * Request routing
   * Load balancing
   * Response caching
   * Error handling
2. **Service Communication**
   * Synchronous: REST APIs
   * Asynchronous: Kafka events
   * WebSocket: Real-time updates
3. **Data Storage**
   * MongoDB: Primary database
   * Redis: Caching layer
   * Kafka: Event store

## **Event Flow Patterns**

### Order Processing Flow

  
  
Not explaining other flow like inventory/ stock update as I will covereing lot of things as part of demo video too. It will make the document lengthier.

## **Security Architecture**

1. **Authentication Flow**
   * JWT-based authentication
   * Token refresh mechanism
   * API key validation for service-to-service communication
2. **Data Security**
   * Password encryption
   * Audit logging (api-gateway and other services)

# **Repository Setup**

1. Clone the main repository:

git clone https://github.com/vissharm/ecommerce-app.git

1. Run the setup script to initialize and update all submodules:

# For Windows PowerShell

.\scripts\setup.ps1

**Note** - It internally calls .\scripts\**initialized\_database.js,** which configures the mongodb database connection and creates all the required database tables.

**Or using npm from the root ecommerce-app folder. This command added in the package.json scripts section and calls setup.ps1**

npm run start

This setup script will:

* Initialize git repository if not already initialized
* Set up all submodules with correct branches
* Update submodules to latest versions
* Install root level dependencies specified in package.json files
* Build and link shared library
* Install dependencies for all services as per respective package.json files
* Set up MongoDB

1. Manual submodule management (if needed):

# Initialize submodules

git submodule init

# Update submodules

git submodule update --init --recursive

# Pull latest changes for all submodules

git submodule update --remote –merge

#### Commands added in root package.json for ease.



# **Deployment Options**

I started with local development using console, then moved to docker-compose container deployment and then deployment with minikube cluster or Kubernetes as per phases I mentioned earlier.

## 1. Local Development (Without Containers)

* Services run directly on host
* MongoDB runs in Docker
* Hot reloading enabled
* Shared library linked locally

### NPM Commands added in root package.json for local development

# Initialize all submodules and build the complete monorepo

npm run initialize:monorepo\_dev\_with\_db\_initialization

# Initialize only the database

npm run initialize:db

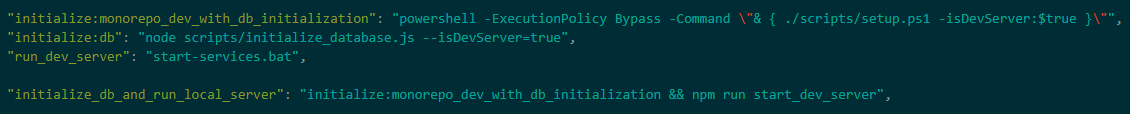
# Start the local development server using command prompt

npm run run\_dev\_server

# Single command to initialize everything and run local development server

npm run initialize\_db\_and\_run\_local\_server

#### Commands added in root package.json for ease.



### Actual scripts created and called by above commands are mentioned below.

1. Install dependencies and setup:

# Run the setup script

.\scripts\setup.ps1

# Or use npm command

npm run setup

1. Start all services:

# Using start-services.bat – runs development server for all services

.\start-services.bat

### Key points for start-services.bat:

* Ensure Zookeeper and Kafka are running first
* Opens separate terminal windows for each service
* Links shared library automatically
* Builds and starts frontend application

## 2. Docker Compose Deployment

* All services in containers
* Development-specific configurations
* Volume mounts for hot reloading
* Docker Compose override support

### NPM Commands added in root package.json for local development

# Deploy and run application using docker-compose

npm run docker-compose:run

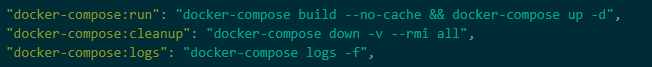
# Cleanup docker-compose created/initialized resources

npm run docker-compose:cleanup

# View logs for troubleshooting

npm run docker-compose:logs

#### Commands added in root package.json for ease.



### Additional docker commands

# View logs for specific service

docker-compose logs -f [service-name]

# Stop all services

docker-compose down

# Stop and remove volumes

docker-compose down -v

# Check service status

docker-compose ps

# Restart specific service

docker-compose restart [service-name]

## 3. **Minikube(Kubernetes) Deployment**

* Local Kubernetes development with minikube cluster
* Service discovery enabled
* ConfigMap for environment variables
* Ingress controller for routing

### NPM Commands added in root package.json for local development

# Deploy and run application using kubernetes/minikube

npm run k8s\_deploy\_with\_cleanup

# Quick deploy without cleaning up resources and rebuilding images

npm run k8s\_deploy:fast\_without\_cleanup

# Run below command to expose the api-gateway port to access the application - https://localhost:<port\_returned\_by\_below\_command>

# This is only needed if above scripts not emitting the port in the end. In windows, sometime it is needed to expose the port and access app.

minikube service api-gateway -n ecommerce –url

#### Commands added in root package.json for ease.



### Deploy services:

# Deploy all services

kubectl apply -f k8s/

# Deploy specific service

kubectl apply -f k8s/[service-name].yaml

### Check deployment status:

# View all pods in ecommerce namespace

kubectl get pods -n ecommerce

# View all services

kubectl get svc -n ecommerce

# View all deployments

kubectl get deployments -n ecommerce

### Get application URL:

# Get API Gateway URL to access the application in browser

minikube service api-gateway -n ecommerce –url

### Pod management:

# Get pod logs

kubectl logs [pod-name] -n ecommerce

# Get pod details

kubectl describe pod [pod-name] -n ecommerce

# Execute command in pod

kubectl exec -it [pod-name] -n ecommerce -- /bin/sh

# Delete pod (will recreate)

kubectl delete pod [pod-name] -n ecommerce

# Rollout restart

kubectl rollout restart deployment [deployment-name] -n ecommerce

### Debugging:

# Check pod events

kubectl get events -n ecommerce --sort-by='.lastTimestamp'

# Check init container logs

kubectl logs [pod-name] -c [init-container-name] -n ecommerce

# Port forward service - if face access issue

kubectl port-forward service/[service-name] [local-port]:[service-port] -n ecommerce

# **Service URLs**

Local/Docker:

* Frontend**: http://localhost:3000**
* API Gateway: **http://localhost:3000**
* User Service: **http://localhost:3001**
* Order Service: **http://localhost:3002**
* Notification Service: **http://localhost:3003**
* Product Service: [**http://localhost:3004**](http://localhost:3004)

# **API Documentation**

**Note:** Very few api’s are not implemented as it was not necessary as per scope and objective of assignment.

### Authentication Endpoints

POST /auth/login

POST /auth/register

POST /auth/refresh-token

GET /auth/verify-email/:token

POST /auth/forgot-password

POST /auth/reset-password

### User Service (Port: 3001)

GET /users # List users (Admin only)

GET /users/:id # Get user details

PUT /users/:id # Update user

DELETE /users/:id # Delete user

GET /users/profile # Get own profile

PUT /users/profile # Update own profile

GET /users/:id/addresses # Get user addresses

POST /users/addresses # Add address

PUT /users/addresses/:id # Update address

DELETE /users/addresses/:id # Delete address

### Product Service (Port: 3004)

GET /products # List products

GET /products/:id # Get product details

POST /products # Create product (Admin)

PUT /products/:id # Update product (Admin)

DELETE /products/:id # Delete product (Admin)

GET /products/categories # List categories

POST /products/categories # Create category (Admin)

PUT /products/categories/:id # Update category (Admin)

DELETE /products/categories/:id # Delete category (Admin)

POST /products/:id/reviews # Add product review

GET /products/:id/reviews # Get product reviews

DELETE /products/reviews/:id # Delete review (Admin/Owner)

### Order Service (Port: 3002)

GET /orders # List orders

GET /orders/:id # Get order details

POST /orders # Create order

PUT /orders/:id/status # Update order status

DELETE /orders/:id # Cancel order

GET /orders/cart # Get cart

POST /orders/cart # Add to cart

PUT /orders/cart/:id # Update cart item

DELETE /orders/cart/:id # Remove from cart

POST /orders/checkout # Checkout process

GET /orders/history # Order history

### Notification Service (Port: 3003)

WebSocket: /socket.io # Real-time notifications

GET /notifications # List notifications

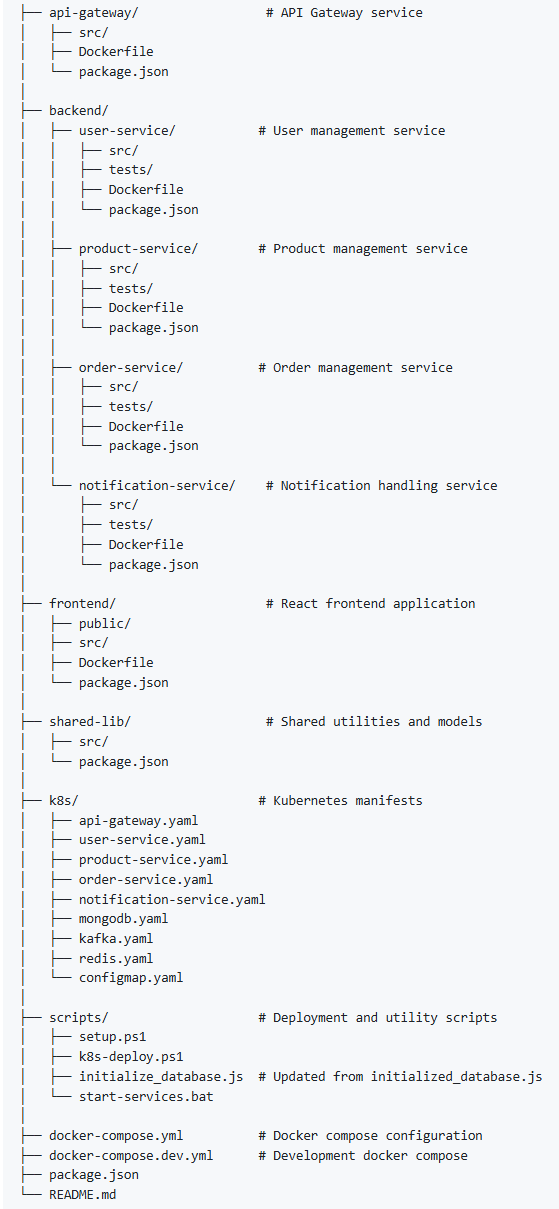
PUT /notifications/:id # Mark as read

DELETE /notifications/:id # Delete notification

POST /notifications/subscribe # Subscribe to notifications

POST /notifications/unsubscribe # Unsubscribe from notifications

# **Directory structure**



# Deployment Script (k8s-deploy.ps1)

### Purpose

The script automates the deployment of the e-commerce application to a Minikube cluster. It handles environment setup, service deployment, and health checks.

#### Key Functions

##### 1. Environment Setup

Test-KubernetesConnection

* Verifies Minikube status
* Starts Minikube if not running
* Ensures correct Kubernetes context
* Validates cluster connectivity

##### 2. Deployment Process

Main deployment flow

* Cleans up existing deployments (optional)
* Configures Docker to use Minikube daemon
* Creates namespace and applies ConfigMaps
* Builds and deploys services in order:
  + Shared library
  + Core services (user, product, order, notification)
  + API Gateway

##### 3. Infrastructure Components

* MongoDB (with initialization)
* Kafka
* Zookeeper

###### Service deployments

* user-service
* product-service
* order-service
* notification-service
* api-gateway

###### Usage Options

**Full deployment with cleanup:** ./scripts/k8s-deploy.ps1

**Deploy without cleanup (faster):** ./scripts/k8s-deploy.ps1 -SkipCleanup

#### Key Features

* Automated environment setup
* Health checks and readiness probes
* Error handling and recovery
* Pod status monitoring
* Database initialization
* Service dependency management

#### Monitoring and Debugging

* Pod status checks
* Service connectivity tests
* Detailed error logging
* Debug commands for troubleshooting

# Setup Script (setup.ps1)

### Purpose

Initial project setup script that prepares the development environment by cloning and initializing submodules in the main ecommerce-app repository. It takes care of installing dependencies, making builds and initializing database with table creation and dummy data.

### Key Operations

#### 1. Dependencies Installation

* Root level npm packages
* Shared library setup and linking
* Service-specific dependencies
* Frontend dependencies

#### 2. API Gateway Setup

* Install dependencies
* Link shared library
* Configure routes

#### 3. Database Initialization

* Calls initialize\_database.js
* Sets up initial data
* Configures development environment

#### Usage

**Full setup:** .\scripts\setup.ps1

**Through npm:** npm run setup

# Database Initialization Script (initialize\_database.js)

### Purpose

Sets up the MongoDB databases with initial data for development and testing.

### Data Initialization

1. User Service Data

2. Product Service Data

### Features

- Safe collection dropping

- Schema validation

- Connection management

- Error handling

### Usage

**Direct execution:** node scripts/initialize\_database.js

**With development server flag:** node scripts/initialize\_database.js --isDevServer=true

# Service Starter Script (start-services.bat)

### Purpose

Windows batch script that launches all required services in separate terminal windows for local development.

### Service Launch Order

#### 1. Infrastructure Services

* Zookeeper (Port: 2181)
* Kafka (Port: 9092)

#### 2. Microservices

* User Service (Port: 3001)
* Product Service (Port: 3004)
* Order Service (Port: 3002)
* Notification Service (Port: 3003)

#### 3. Frontend & Gateway

* Frontend Application (Port: 3000)
* API Gateway (Port: 8080)

### Features

* Parallel service execution
* Dependency management
* Shared library linking
* Development hot-reloading
* Terminal window organization

### Usage

**Start all services:** .\start-services.bat

##### Services are launched in separate Windows Terminal tabs

# Each service shows its own logs

# Terminal Window Layout

[Services Info] [Zookeeper] [Kafka] [Order Service]

[User Service] [Product Service] [Notification Service]

[Frontend] [API Gateway]

##### Service Dependencies

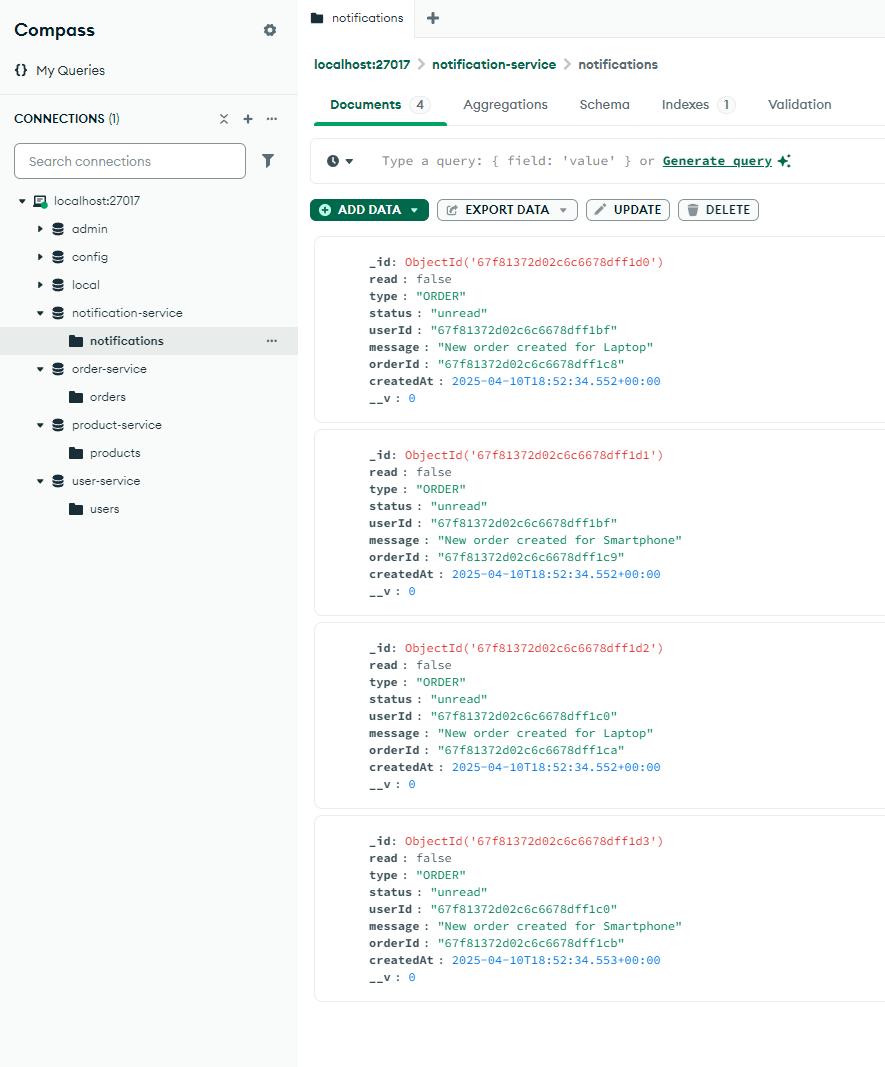
* Zookeeper must start before Kafka
* Kafka must be ready before microservices
* All services must be up before API Gateway
* Frontend builds after services start

##### These scripts work together to provide:

* Consistent development environment setup
* Automated dependency management
* Initial data population
* Service orchestration
* Development workflow automation

The combination ensures developers can quickly set up and start working with the full application stack in a development environment.

# **MongoDB compass screenshot**



# **Application screenshots**

## Landing page / homepage

A screenshot of a computer

AI-generated content may be incorrect.

## Responsive UX & dashboard/home page A screenshot of a computer AI-generated content may be incorrect. A screenshot of a phone AI-generated content may be incorrect.

## Login screen with validations

A screenshot of a computer

AI-generated content may be incorrect.

## Login screen

A screenshot of a login form

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

## Products page / Inventory management A screenshot of a computer AI-generated content may be incorrect.

## Add Product

A screenshot of a computer

AI-generated content may be incorrect.

## Notifications support A screenshot of a computer AI-generated content may be incorrect.

## Profile page with editing support A screenshot of a computer AI-generated content may be incorrect.

## Top navigation bar

A blue and white rectangle

AI-generated content may be incorrect.

A blue sign with white text

AI-generated content may be incorrect.

## Order management – Create order

A screenshot of a computer

AI-generated content may be incorrect.

## Order list

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Order creation and Kafka-websocket real time notification integration & flow

1. Order created -> Order service process order and put in pending state and publish message to kafka for notification service
2. Notification service listens to kafka message for order -> Create and save notification in database and pushes notification to client/browser that order created successfully. -> Updates the order state from pending to completed and pushes another notification to connected client that order completed successfully.
3. Order status also updated dynamically in grid via websocket notificationsA screenshot of a computer

   AI-generated content may be incorrect.
4. Notification also automatically updated to read from unread once notification is displayed on browser. If not due to socket connection issue, user could manually click on status column and it will be updated from unread to read.
5. Delete support also available for notifications

A screenshot of a computer

AI-generated content may be incorrect.