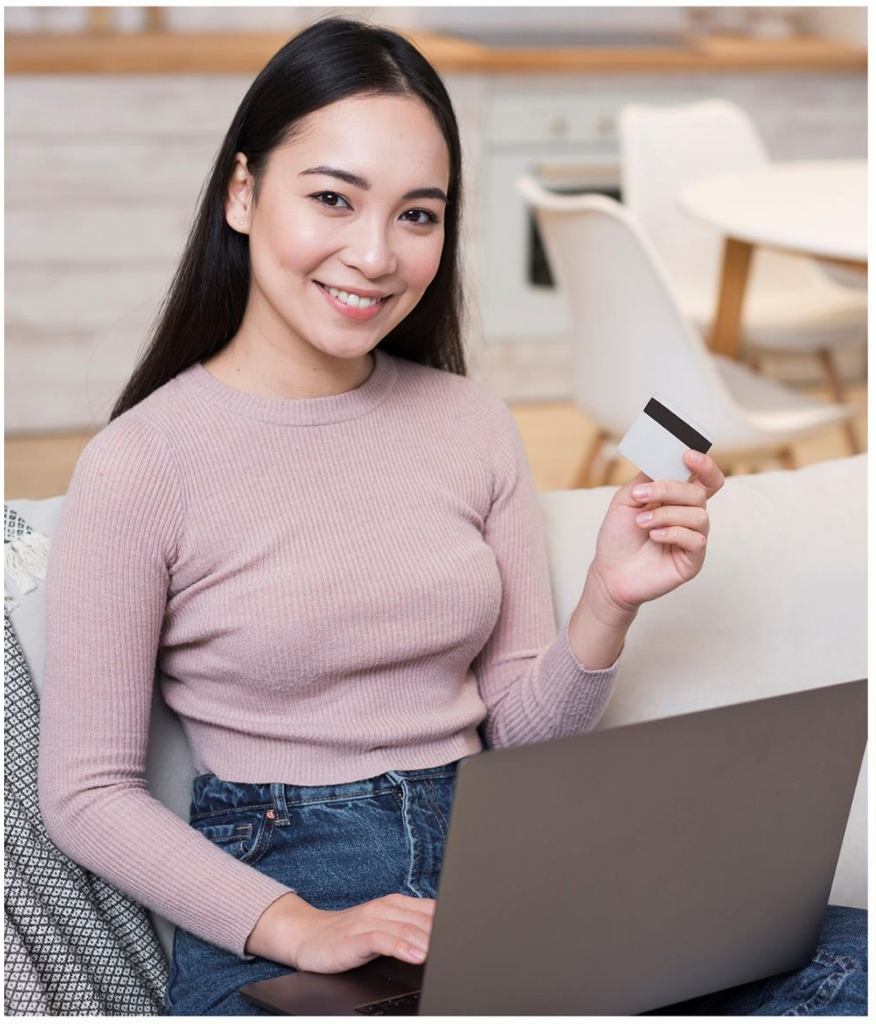
ONLINE

SHOPPING



Team members:

Name	Id	Section
Yasmine Khaled atta	20201701154	Sec 9
Ahmed Esmail Mohamed	20201700024	Sec 1
Haidy Ashraf Mondy	20201700953	Sec 9
Nada Abdallah Mahdy	20201700921	Sec 8

Overview:

- Introduction to the project's goals and objectives.
- Project Structure.
- System Architecture diagram.
- Project components.
- Development environments Set up.
- Deploying the platform.

Introduction

This project aims to create a robust and scalable online shopping platform that leverages modern technologies such as Docker, Docker-Compose, Kubernetes, and Jenkins. By adopting a microservices architecture, the platform will be divided into a collection of services, each responsible for specific business functionalities such as user authentication, product management, order management. These services will be designed to be independent, loosely coupled, and communicate with each other through lightweight protocols such as HTTP, ensuring flexibility and modularity.

project Structure

Each service (user authentication, product management, order management) has its own directory containing its source code, Dockerfile, and any other necessary files. The application directory contains the source code for the application that communicates with the services. The docker-compose.yml file defines the services, their configurations, and their dependencies.

Online-Shopping/

├── Services/

│ ├── User-authentication/

│ │ ├── Dockerfile

│ │ ├── User-Database

│ │ ├── Api

│ │ ├── Models

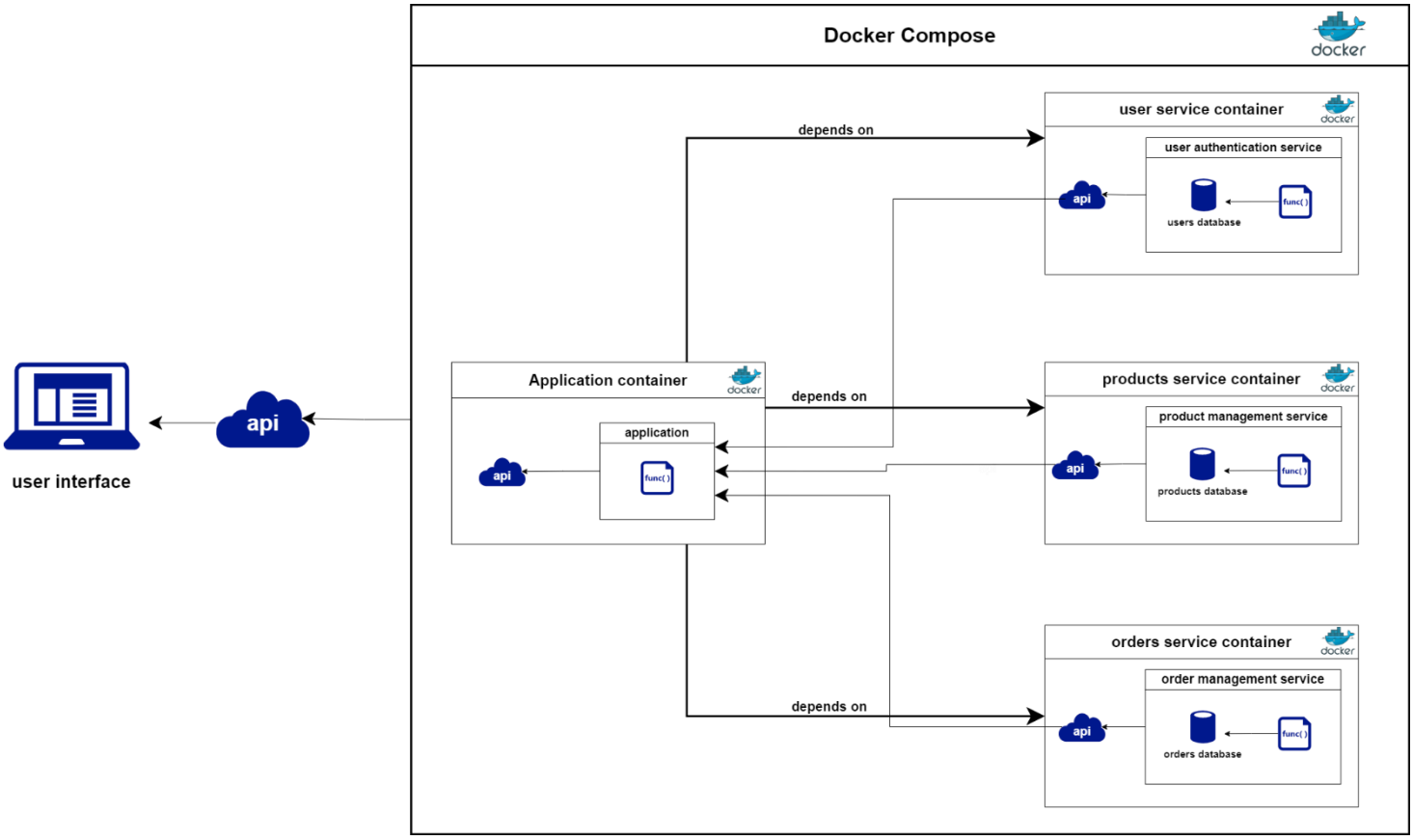
│ │ ├── User-deployment

│ │ ├── User-replacaset

│ │ ├── Database Management

```
│   │   ├── User pod
│   │   ├── Requirements
│   │   └── Unit-tests
│   │
│   ├── Product-Management/
│   │   ├── Dockerfile
│   │   ├── Product-Database
│   │   ├── Product-replacaset
│   │   ├── ……
│   │   └── Api
│   ├── Order-management/
│   │   ├── Dockerfile
│   │   ├── order-Database
│   │   ├── order-replacaset
│   │   ├── ...
│   │   └── Api
├── Application/
│   ├── Dockerfile
│   ├── User-Database
│   ├── Api
│   ├── Models
│   ├── Application deployment
│   ├── Application replacaset
│   ├── Requirements
│   ├── Application pod
│   └── App-tests
│
└── docker-compose.yml
```

System Architecture diagram



Project Component

The project components work together to create a scalable, reliable, and efficient online shopping platform, providing users with a seamless shopping experience from product browsing to order fulfillment.

1. **User Management Service:** Responsible for handling user authentication, registration, and profile management.
2. **Product Management Service:** Manages the catalog of products available for purchase, including details like product name, description, price, and availability.
3. **Order Management Service:** Manages the creation, modification, and fulfillment of orders placed by users.
4. **SQLite Database:** Used as the database system for storing user data, product information, cart contents, orders, and payment details.
5. **Fast API:** A modern, fast (high-performance), web framework for building APIs with Python. Each microservice will be built using FastAPI to provide RESTful APIs for communication between services and clients.
6. **Docker:** Used for containerizing each microservice along with its dependencies, ensuring consistency across different environments and facilitating efficient deployment and scaling.
7. **Docker-Compose:** Helps define and manage multi-container development environments, allowing developers to easily spin up the entire application stack locally for integration testing and development.
8. **Kubernetes**: Utilized for orchestrating and managing the deployment of containerized services, providing features such as scalability, high availability, and efficient resource utilization.
9. **ReplicaSet:** In Kubernetes, a ReplicaSet is a controller that ensures a specified number of pod replicas are running at any given time. It helps maintain the desired number of pods for each microservice, providing scalability, fault tolerance, and high availability.
10. **Pod:** In Kubernetes, a Pod is the smallest deployable unit that represents a single instance of a running process in your cluster. It can contain one or more containers, such as your microservice application container and any accompanying sidecar containers.
11. **Deployment:** In Kubernetes, a Deployment is a higher-level abstraction that manages ReplicaSets and Pods. It enables declarative updates to Pods and ReplicaSets, providing features like rolling updates and rollbacks.

Development environments Set up

Prerequisites:

Before setting up the development environment, ensure that you have the following prerequisites installed:

- Docker Engine
- Minikube
- Kubectl
- Git

Then:

Clone the Repository: Clone the project repository from the version control system (e.g., GitHub)

- **git clone** < https://github.com/AhmedEsmail8/online-shopping >

Deploying the platform

1. **Build Docker Images**: Build Docker images for the application components.
   - **docker build** -t <image-name> <path-to-docker file>
2. **Start Docker Containers**: Start the Docker containers using Docker Compose.
   - **docker-compose up**
3. **Access the Application:** Access the application in your web browser using the provided URL (http://localhost:3030/).