Project Documentation

E-commerce Application

Project Overview

This e-commerce platform is a containerized web application that enables users to authenticate, manage their profiles, and perform CRUD operations on products with images. The application demonstrates the implementation of Docker containerization for a multi-tier application with a database, backend API, and frontend UI.

Purpose

The application serves as a practical demonstration of:

1. Modern web application development using .NET Core and Flutter
2. Clean Architecture principles in a real-world application
3. Docker containerization for multi-component applications
4. Service orchestration using Docker Compose

Core Features

1. **User Authentication and Management**
   - Registration with email validation
   - JWT-based authentication
   - User profile viewing and updating
2. **Product Management**
   - Adding products with image uploads
   - Viewing product listings
   - Updating product details
   - Deleting products
3. **Responsive UI**
   - Web and mobile support through Flutter
   - State management using BLoC pattern
   - Image handling across platforms

Tech Stack

Backend

- **Framework**: ASP.NET Core 8.0
- **ORM**: Entity Framework Core
- **Database**: SQL Server
- **Authentication**: JWT (JSON Web Tokens)

Frontend

- **Framework**: Flutter
- **State Management**: BLoC Pattern
- **HTTP Client**: Dio
- **Image Handling**: Multi-platform support (Web, Android, iOS)

DevOps

- **Containerization**: Docker
- **Orchestration**: Docker Compose
- **Database**: SQL Server in Docker

Architecture

The application implements Clean Architecture principles with clear separation of concerns:

Backend

- **Controllers**: REST API endpoints for authentication and product operations
- **Models**: Domain entities and data transfer objects
- **Data Access**: Repository pattern with Entity Framework Core

Frontend

- **Presentation Layer**: UI components and screens
- **Business Logic Layer**: BLoC components for state management
- **Data Layer**: Repositories and remote data sources

Getting Started with Docker Setup

To set up and run the application with Docker:

Prerequisites

1. Docker and Docker Compose installed
2. Git for cloning the repository

Step 1: Clone the Repository

bash

```bash
git clone https://github.com/yourusername/ecommerce-project.git
cd ecommerce-project
```

Step 2: Set Up the Backend Dockerfile

Create a file named `Dockerfile` in the `backend/loginPage` directory with the following content:

dockerfile

```dockerfile
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY ["loginPage.csproj", "./"]
RUN dotnet restore "loginPage.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "loginPage.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "loginPage.csproj" -c Release -o /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "loginPage.dll"]
```

Step 3: Create Docker Compose File

Create a file named `docker-compose.yml` in the root directory:

yaml

```yaml
services:
  sql-server:
    image: mcr.microsoft.com/mssql/server:2019-latest
    environment:
      - ACCEPT_EULA=Y
      - SA_PASSWORD=YourStrongPassword123!
    ports:
      - "1433:1433"
    volumes:
      - sql-data:/var/opt/mssql
    networks:
      - ecommerce-network

  backend:
    build:
      context: ./backend/loginPage
      dockerfile: Dockerfile
    ports:
      - "5163:80"
    depends_on:
      - sql-server
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ConnectionStrings__conStr=Server=sql-server;Database=DotNetCore-ECommerce.Identity;User
    networks:
      - ecommerce-network

networks:
  ecommerce-network:
    driver: bridge

volumes:
  sql-data:
```

Step 4: Build and Run the Application

bash

```bash
docker-compose up --build
```

This command will:

1. Build the backend Docker image
2. Pull the SQL Server image
3. Start all services

Step 5: Push to Docker Hub

After successfully building the application:

```bash
# Login to Docker Hub
docker login

# Tag the image
docker tag ecommerce-project_backend yourusername/ecommerce-backend:latest

# Push to Docker Hub
docker push yourusername/ecommerce-backend:latest
```

Step 6: Accessing the Application

- Backend API will be available at: http://localhost:5163
- API endpoints:
  - Authentication: `/api/Account/Login`, `/api/Account/Register`
  - Products: `/api/Item/GetItems`, `/api/Item/AddItem`

Team

- Developer: [Your Name]

Future Enhancements

1. Implement product categories and search functionality
2. Add payment processing capabilities
3. Enhance security with HTTPS and additional authentication options
4. Implement caching layer for improved performance