Project Structure

This project is organized into two main sections: **backend** and **frontend**. Each section is separated to ensure modularity, scalability, and ease of maintenance. The backend follows a microservice architecture, while the frontend is a standalone application.

Backend

The backend is organized into multiple microservices, each having its own folder with its own codebase, infrastructure, and database (if necessary). Here's the full nested project structure for the backend:

```
/backend
│
├── IdentityService/
│   ├── src/
│   │   ├── IdentityService.Api/
```

```
│ │ ├── IdentityService.Core/
│ │ └── IdentityService.Infrastructure/
│ │ ├── Events/ # Event publishing, subscribing handlers
│ │ └── MessageBus/ # Message bus interface (RabbitMQ)
│ ├── tests/
│ │ └── IdentityService.Tests/
│ ├── Dockerfile
│ ├── appsettings.json
│ └── README.md
│
├── CandleService/
│ ├── src/
│ │ ├── CandleService.Api/
│ │ ├── CandleService.Core/
│ │ └── CandleService.Infrastructure/
│ │ ├── Events/ # Event handlers (e.g., handle new candle or average calculation
event)
│ │ └── MessageBus/ # Messaging infrastructure (RabbitMQ)
│ ├── tests/
│ │ └── CandleService.Tests/
│ ├── Dockerfile
│ ├── appsettings.json
│ └── README.md
│
├── CandleGeneratorService/
│ ├── src/
│ │ ├── CandleGeneratorService.Api/
│ │ ├── CandleGeneratorService.Core/
│ │ └── CandleGeneratorService.Infrastructure/
│ │ ├── Events/ # Event publishing after new candle is generated
│ │ └── MessageBus/ # Publish new candle events
│ ├── tests/
│ │ └── CandleGeneratorService.Tests/
│ ├── Dockerfile
│ ├── appsettings.json
│ └── README.md
│
├── CandleAverageService/
│ ├── src/
│ │ ├── CandleAverageService.Api/
│ │ ├── CandleAverageService.Core/
│ │ └── CandleAverageService.Infrastructure/
│ │ ├── Events/ # Event handlers for processing candle average calculation
│ │ └── MessageBus/ # Subscribe to candle events, publish results
│ ├── tests/
│ │ └── CandleAverageService.Tests/
│ ├── Dockerfile
│ ├── appsettings.json
│ └── README.md
│
├── LoggingService/
│ ├── src/
│ │ └── LoggingService.Infrastructure/
│ │ ├── Events/ # Listens to events from other services to log information
│ │ └── MessageBus/ # Subscribe to logs or errors
│ ├── Dockerfile
│ ├── appsettings.json
```

```
│   └── README.md
│
├── common/
│   ├── Contracts/ # Shared event DTOs, Enums, Models
│   ├── Events/ # Common event handling mechanisms
│   └── MessageBus/ # Message bus implementation shared across microservices
└── README.md
```

Description of Each Folder:

- **IdentityService/**: Handles user registration, authentication, and JWT token management.
  - `src/`: Contains the main application code split into API, Core (business logic), and Infrastructure layers (data access, external services).
  - `tests/`: Contains unit and integration tests for the Identity service.
  - `Dockerfile`: Docker configuration for containerizing the Identity service.
  - `appsettings.json`: Configuration for connection strings, JWT settings, etc.
- **CandleService/**: Manages all operations related to candle data (Open, Close, High, Low).
  - `src/`: Contains the API, Core, and Infrastructure layers for managing candle data.
  - `tests/`: Unit and integration tests for the Candle service.
  - `Dockerfile`: Docker configuration for containerizing the Candle service.
- **CandleGeneratorService/**: A background service that generates random candles every minute.
  - `src/`: Contains logic for generating random candles and inserting them into the CandleService database.
  - `tests/`: Unit tests for the generator logic.
- **CandleAverageService/**: A background service that calculates the average of candle prices every two minutes.
  - `src/`: Contains logic to calculate the average of each candle and store it in the database.
  - `tests/`: Unit tests for the average calculation logic.

- **common/**: Shared code used across different microservices, such as Data Transfer Objects (DTOs), enums, utilities, and middleware.

Here is a detailed list of your project's features, covering both the frontend and backend services, while also highlighting key technologies and architectural decisions like Event-Driven Architecture (EDA), JWT authentication, and more.

Project Features

1. Frontend (Next.js 14 with TypeScript)

- **Next.js 14 (App Directory)**:
  - The project uses the **App Directory** structure introduced in Next.js 14, providing a clear separation of pages and server-side logic.
  - It leverages **Server Components** for improved performance and scalability.
- **TypeScript**:
  - Strongly typed development with **TypeScript** ensures maintainability and helps avoid common errors.
- **TailwindCSS**:
  - The frontend uses **TailwindCSS** for rapid UI development and consistent styling.
- **React (Hooks and Context)**:
  - Utilizes modern **React hooks** (`useState`, `useEffect`, etc.) and **Context API** for managing global state such as authentication and user data.
- **Responsive Design**:
  - The frontend is fully responsive and adjusts seamlessly across mobile, tablet, and desktop screen sizes.
- **Candle Form and Market Data Visualization**:
  - Provides a form to input candle data (Open, Close, High, Low prices).

    - Displays market data in real-time, reflecting data generated by the backend microservices.

2. Backend (Microservices with .NET Core)

The backend follows a **microservices architecture**, with each service handling specific functionality. Here are the key microservices and their features:

IdentityService:

- **JWT Authentication and Authorization**:
    - The **IdentityService** provides user registration, login, and secure authentication using **JWT tokens**.
    - Tokens are issued upon login and are set to expire every 30 minutes, after which they are refreshed.
- **Secure Password Hashing**: Implements secure password hashing for user data.

CandleService:

- **Candle Data Management**:
    - Manages the CRUD operations for candle data (Open, Close, High, Low prices).
- **API for Candle Submission and Retrieval**:
    - Provides APIs to submit candle data from the frontend and retrieve historical candle information for display.

CandleGeneratorService:

- **Random Candle Generation**:
    - Runs as a background service that generates random candle data every 1 minute.
- **Event-Driven**:
    - Publishes `CandleGeneratedEvent` to notify other services about newly generated candle data.

CandleAverageService:

- **Average Candle Price Calculation**:

    - A background service that calculates the average price of each candle every 2 minutes.
- **Event-Driven**:
    - Subscribes to `CandleGeneratedEvent` to calculate the average and stores the result in the database.
    - Publishes `AverageCalculatedEvent` to notify other services, e.g., for logging or monitoring.

LoggingService:

- **Centralized Logging**:
    - Captures logs and event data from multiple services, ensuring easy debugging and performance monitoring.
- **Event-Driven**:
    - Subscribes to events like `UserLoggedInEvent`, `CandleGeneratedEvent`, and `AverageCalculatedEvent` for logging success/failure events and performance metrics.

GatewayService:

- **API Gateway**:
    - Acts as a reverse proxy, routing frontend requests to the appropriate microservices (e.g., IdentityService, CandleService).
- **Centralized Authentication**:
    - Verifies JWT tokens and manages user authentication before forwarding requests to other services.

3. Event-Driven Architecture (EDA)

- **Kafka or RabbitMQ as Event Bus**:
    - The backend is built on **Event-Driven Architecture (EDA)** using **Kafka** or **RabbitMQ** as the messaging backbone.
    - Microservices communicate asynchronously by publishing and subscribing to events, ensuring loose coupling and scalability.
- **Event Flow**:
    - **CandleGeneratorService** emits `CandleGeneratedEvent` every 1 minute.

    - **CandleService** listens for these events to store the candle data.
    - **CandleAverageService** listens to `CandleGeneratedEvent` and calculates the average price, publishing `AverageCalculatedEvent`.
    - **LoggingService** listens to multiple events for logging purposes.

4. Database

- **SQL Server**:
    - Each microservice manages its own database (or schema) using **SQL Server**, ensuring data is stored independently and in a scalable manner.
    - Services such as **IdentityService** handle user data, while **CandleService** manages candle and market-related data.

5. Docker & Containerization

- **Dockerized Microservices**:
    - Each microservice (IdentityService, CandleService, etc.) is containerized using **Docker**.
    - The project includes a `docker-compose.yml` file at the root of the repository, allowing easy orchestration of all services (including the frontend and databases).

6. CI/CD Pipeline (Optional)

- **GitHub Actions or Azure Pipelines** (Optional):
    - CI/CD pipeline configurations can be included for automated testing, building, and deployment of microservices.
    - These pipelines ensure the backend and frontend are built, tested, and deployed continuously to production or staging environments.

7. Security

- **JWT Token Security**:

    - The project uses **JWT tokens** for secure API requests. Token expiration and refresh mechanisms ensure ongoing secure communication.

- **HTTPS Enforcement**:
    - All communications between the frontend, backend, and databases are secured using **HTTPS**.

This project structure and feature set ensure a **scalable**, **modular**, and **maintainable** system that leverages the best practices of **microservices architecture**, **event-driven communication**, and modern frontend development using **Next.js**. It is designed to be **easy to scale** as the application grows, with a focus on **performance**, **security**, and **real-time data handling**.

Running the Backend Services

To run the backend services, use the `docker-compose.yml` file located in the root of the repository. This file defines the services, networks, and volumes necessary to bring up the backend microservices and any dependencies (e.g., databases).

Steps to Run the Backend:

1. **Clone the repository**:

```
git clone https://github.com/ArsacidTechnologies/kasbo-tech-project.git
cd kasbo-tech-project
```

2. **Build and run the services** :

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

2. **Access the services** :

- IdentityService: `http://localhost:5001`
- CandleService: `http://localhost:5002`
- CandleGeneratorService: Runs in the background.
- CandleAverageService: Runs in the background.

2. **Stop the services** :

```
docker-compose down
```

For more detailed information about each service, refer to the `README.md` file in each service's folder.