**CloudCart Documentation**

**Overview**

The **CloudCart** API is an e-commerce backend system designed to manage products, users, orders, and payments. Built using **.NET 8** and **C# 12**, the project adheres to **Clean Architecture** principles, utilizes the **Result Pattern** for operation handling, and implements the **CQRS (Command Query Responsibility Segregation)** pattern for separating read and write operations. Additionally, **Hangfire** is used for background job processing, such as fetching products daily.

**Entities**

**1. Product**

Represents items available for purchase.

* **Properties**:
  + Id: Guid - Unique identifier for the product.
  + Name: string - Name of the product.
  + Description: string - Description of the product.
  + Price: decimal - Price of the product.
  + StockQuantity: int - Number of items in stock.
  + CategoryId: Guid - Identifier for the product category.

**2. User**

Represents customers or administrators within the system.

* **Properties**:
  + Id: Guid - Unique identifier for the user.
  + Username: string - User's username.
  + Email: string - User's email address.
  + PasswordHash: string - Hashed password for security.
  + Role: enum (Admin, Customer) - User's role in the system.
  + CreatedAt: DateTime - Timestamp when the user was created.

**3. Order**

Represents a purchase transaction by a user.

* **Properties**:
  + Id: Guid - Unique identifier for the order.
  + UserId: Guid - Identifier of the user who placed the order.
  + OrderDate: DateTime - Date and time when the order was placed.
  + TotalAmount: decimal - Total cost of the order.
  + OrderStatus: enum (Pending, Completed, Cancelled) - Current status of the order.
  + Items: List<OrderProduct> - List of products included in the order.

**4. OrderProduct**

Represents the relationship between orders and products, detailing which products are included in an order.

* **Properties**:
  + OrderId: Guid - Identifier of the related order.
  + ProductId: Guid - Identifier of the product.
  + Quantity: int - Number of units of the product in the order.
  + UnitPrice: decimal - Price per unit of the product.

**5. Payment**

Represents the payment details for an order.

* **Properties**:
  + Id: Guid - Unique identifier for the payment.
  + OrderId: Guid - Identifier of the related order.
  + PaymentDate: DateTime - Date and time of the payment.
  + Amount: decimal - Amount paid.
  + PaymentStatus: enum (Pending, Completed, Failed) - Current status of the payment.
  + PaymentMethod: enum (CreditCard, PayPal, BankTransfer) - Method used for payment.

**Architecture**

The CloudCart project follows **Clean Architecture**, which divides the system into multiple layers, each with distinct responsibilities, promoting loose coupling and high cohesion. This structure helps in maintaining the codebase, scaling the system, and adapting to changes over time.

**Key Layers in Clean Architecture:**

1. **Domain Layer**: Contains the core business logic, entities, and domain services. It is the most stable part of the application and is independent of external frameworks or infrastructure. This layer only depends on the abstractions, not the implementations.
2. **Application Layer**: Includes the use cases that orchestrate business logic according to the application requirements. It contains CQRS handlers (commands and queries), services, and the use case logic. This layer depends on the domain layer and defines the application's behavior.
3. **Infrastructure Layer**: Handles all the concerns outside the core business logic, such as data access, external services, and message brokers. This layer provides concrete implementations of the interfaces defined in the domain and application layers, ensuring that the business logic is not coupled with external systems.
4. **API Layer**: Serves as the entry point of the application, exposing RESTful endpoints. It orchestrates the flow of data by interacting with the application layer to fulfill HTTP requests. This layer should be as thin as possible, containing minimal logic and delegating business decisions to the application layer.

**Architectural Benefits:**

* **Decoupling**: Ensures that the business rules are isolated from implementation details, such as data access or UI. This makes it easier to modify or replace one part of the system without affecting others.
* **Testability**: By isolating logic into different layers, it becomes easier to write unit and integration tests, improving code quality and reliability.
* **Maintainability and Scalability**: Clean separation of concerns and adherence to SOLID principles make the system easier to maintain and scale over time.

**Key Features**

1. **Separation of Concerns**: By using Clean Architecture, CloudCart ensures that business logic, application logic, infrastructure, and UI concerns are well-separated, enhancing maintainability.
2. **Robust Error Handling**: The Result Pattern provides a unified way to handle both successes and errors, reducing the need for exception handling and simplifying code.
3. **Optimized Read/Write Operations**: CQRS allows for the separation of read and write operations, which can lead to more optimized and scalable applications.
4. **Automated Background Processing**: Hangfire is used for tasks like daily fetching of products, ensuring that operations that do not require immediate response time are handled efficiently in the background.