Core Layer Overview (Clean Architecture)

Core Layer Overview

1. **Core Layer Overview**

The Core Layer is the foundation of the application, containing all the business logic and domain entities. It is divided into several sub-layers, each responsible for specific aspects of the business logic.

2. **Core.Domain**

Overview: The `Core.Domain` layer represents the core business logic and domain entities of the application. This layer is crucial as it defines the business rules, domain events, and the entities themselves that interact with the other layers in the architecture. The `Core.Domain` layer is independent of any other layer, meaning it does not have dependencies on external libraries or infrastructure, ensuring a clean separation of concerns.

Structure and Files:

- **Core.Domain.csproj**:
  - **Purpose**: Project file that defines dependencies, compilation options, and other configurations specific to the domain layer.
  - **Pros**: Centralizes project-level configurations and dependencies.
  - **Cons**: Any misconfiguration can lead to build issues or incorrect dependency management.

Core Layer Overview (Clean Architecture)

- **GlobalUsings.cs**:
  - **Purpose**: Contains global using directives that are made available across the project.
  - **Pros**: Reduces redundancy in the codebase.
  - **Cons**: Overuse can lead to namespace conflicts or unclear code.

- **Entities**:
  - **Overview**: Contains domain entities representing business objects.
  - **Files**:
    - **UiSchema.cs**: Defines the schema for UI elements.
    - **UiSchemaDesign.cs**: Manages the design aspects of the UI schema.
    - **ButtonControl.cs**: Represents a button control element.
    - **ComboBoxControl.cs**: Represents a combo box control element.
    - **ControlBase.cs**: Serves as the base class for all controls.
    - **DatePickerControl.cs**: Represents a date picker control element.
    - **FormControl.cs**: Manages form-related controls.
    - **RadioButtonControl.cs**: Represents a radio button control element.
    - **TextBoxControl.cs**: Represents a text box control element.
  - **Pros**: Centralizes business logic and ensures a single source of truth for domain entities.
  - **Cons**: Complex business rules can make the domain layer challenging to manage.

3. **Core.Application**

Overview: The `Core.Application` layer handles the application logic, including the use cases and application services. It orchestrates the interaction between the domain layer and the infrastructure layer. The `Core.Application` layer is where you implement CQRS (Command Query Responsibility Segregation) and Mediator patterns.

Structure and Files:

- **Core.Application.csproj**:
  - **Purpose**: Project file for the application layer, managing dependencies and configurations.
  - **Pros**: Ensures application logic is separate from domain logic.
  - **Cons**: Can become complex with many dependencies.

- **GlobalUsings.cs**:
  - **Purpose**: Global using directives for the application layer.
  - **Pros**: Simplifies code by removing the need for repetitive using statements.
  - **Cons**: Potential for namespace conflicts.

- **Commands**:
  - **Overview**: Contains command classes for CQRS operations.
  - **Files**:
    - **AddCommand.cs**: Handles the logic for adding new entities.
    - **DeleteCommand.cs**: Manages the deletion of entities.
    - **UpdateCommand.cs**: Handles updating existing entities.
  - **Pros**: Implements separation of concerns by isolating commands.
  - **Cons**: Overhead of managing multiple commands.

Core Layer Overview (Clean Architecture)

- **Controllers**:
  - **Overview**: Contains API controllers.
  - **Files**:
    - **BaseApiController.cs**: Base class for all API controllers, ensuring consistent behavior.
  - **Pros**: Centralized place for handling API requests.
  - **Cons**: Complex controllers can lead to code duplication.

- **Helpers**:
  - **Overview**: Utility classes for common tasks.
  - **Files**:
    - **JsonHelper.cs**: Provides methods for JSON serialization and deserialization.
  - **Pros**: Reusable utility methods.
  - **Cons**: Overuse can lead to helper classes doing too much.

- **Options**:
  - **Overview**: Manages configuration options.
  - **Files**:
    - **OptionsExtensions.cs**: Extensions for configuring options.
  - **Pros**: Centralizes configuration logic.
  - **Cons**: Mismanagement can lead to configuration errors.

- **Queries**:
  - **Overview**: Contains query classes for CQRS operations.
  - **Files**:

    - **GetAllQuery.cs**: Handles retrieval of all entities.
  - **Pros**: Implements separation of concerns by isolating queries.
  - **Cons**: Similar to commands, can add overhead.

Design Patterns and Architecture:

- **CQRS Pattern**: The Command Query Responsibility Segregation (CQRS) pattern is used to separate read and write operations, ensuring that each is handled by a different model.
- **Mediator Pattern**: This pattern is used in conjunction with CQRS to manage requests and their handlers, promoting loose coupling and single responsibility.

Integration with Other Layers:

- **Core.Domain**: Provides the foundational business logic that the Core.Application layer operates on.
- **Core.Infrastructure**: Implements the abstractions defined in the Core.Application.Abstractions layer, enabling interaction with external systems.

Q: 1. How does the Core.Domain layer ensure business logic is isolated?

A: By defining domain entities and business rules within a dedicated layer, separated from infrastructure and application concerns.

Q: 2. What patterns are implemented in the Core.Application layer?

A: The Core.Application layer implements CQRS and Mediator patterns, which separate read and write operations and manage request handling.

Core Layer Overview (Clean Architecture)

Q: 3. Why are DTOs used in the Core.Application.Dto layer?

A: DTOs decouple data representation from business logic, facilitating data transfer between layers.

Q: 4. How does the Core.Infrastructure layer interact with external systems?

A: The Core.Infrastructure layer implements abstractions defined in Core.Application.Abstractions, enabling communication with databases, file systems, and other external resources.

Core Layer Overview (Clean Architecture)

High-Level Architecture Diagram

```
High-Level Architecture Diagram


         +-------------------------------------------+
         |             Application Layer             |
         |  +-----------+       +----------------+ |
         |  |   API     | <-->  | Core.Application| |
         |  +-----------+       +----------------+ |
         |                                           |
         +-------------------------------------------+
                    |                  |
                    V                  V
         +--------------------+  +--------------------+
         | Core.Domain Layer   |  | Core.Infrastructure |
         | (Business Logic)    |  |  (External Systems) |
         +--------------------+  +--------------------+
```