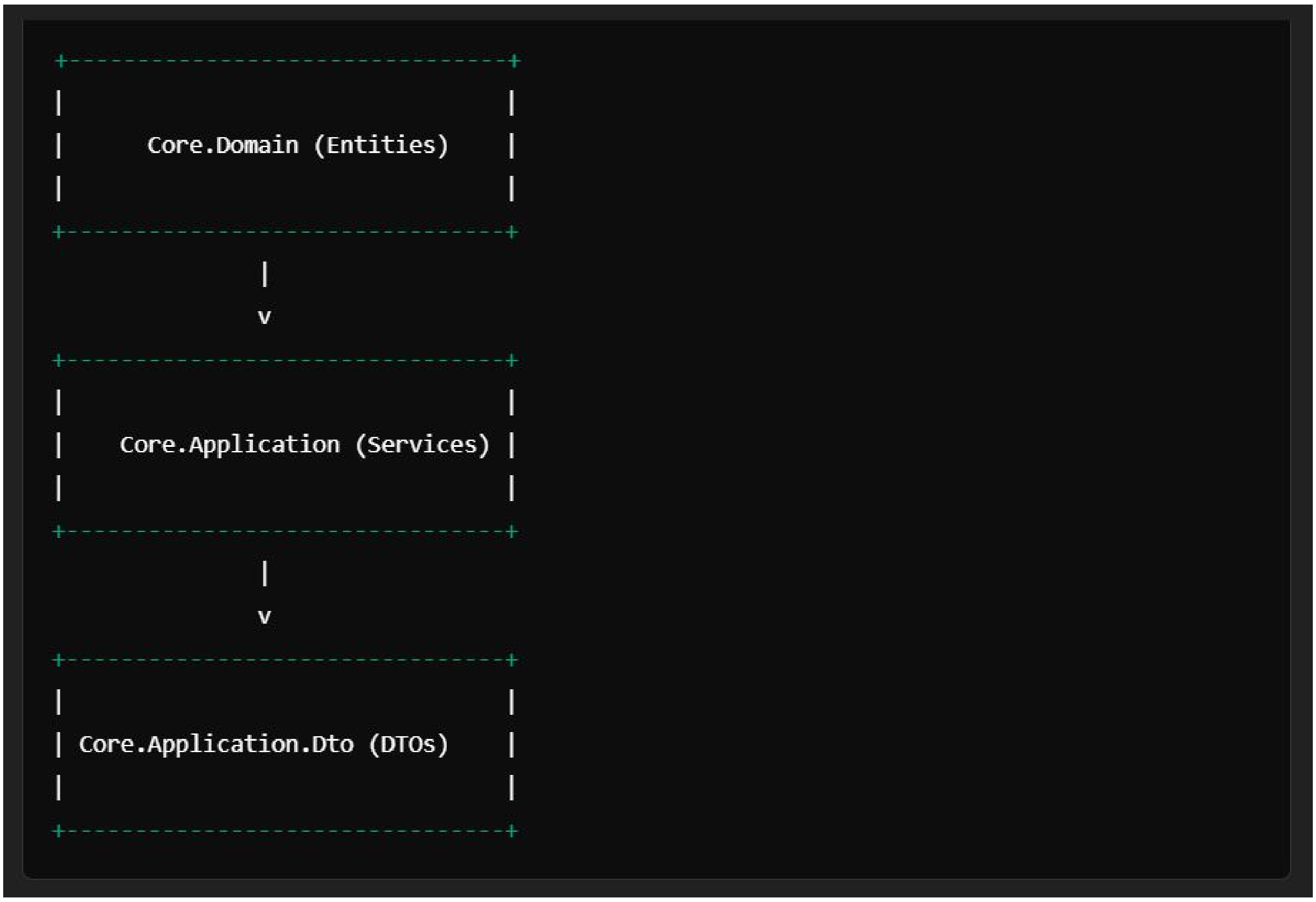
Core Layer Overview

High-Level Architecture Diagram



```
+-------------------------------+
|                               |
|      Core.Domain (Entities)   |
|                               |
+-------------------------------+
              |
              v
+-------------------------------+
|                               |
|    Core.Application (Services) |
|                               |
+-------------------------------+
              |
              v
+-------------------------------+
|                               |
| Core.Application.Dto (DTOs)   |
|                               |
+-------------------------------+
```

Core Layer Overview

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

- **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.

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 to manage interactions, ensuring separation of concerns.

Core Layer Overview

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

A: DTOs are used to transfer data between layers, ensuring that only necessary information is exposed while keeping the business logic separate from data representation.