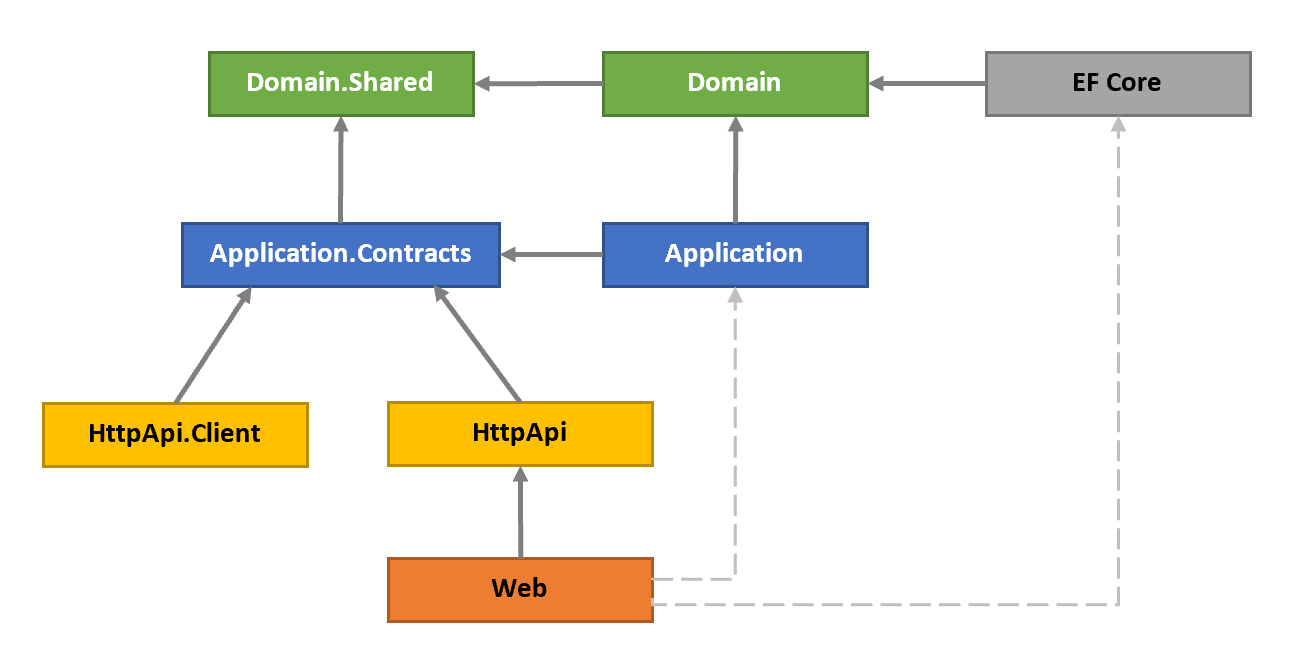
**HRM Architecture - DDD**



Domain-Driven Design (DDD) is an approach to software development that focuses on understanding and modeling the business domain. It provides principles and patterns for designing complex software systems, and it can be applied to various architectural styles, including web APIs. Here's a high-level overview of how DDD principles can be applied to the architecture of a Web API:

* Define the Business Domain:

Identify the core business domain and its relevant concepts.

Clearly define entities, value objects, aggregates, and services that represent the business logic.

* Bounded Contexts:

Divide the system into bounded contexts, each with a distinct and well-defined responsibility.

Establish clear boundaries between different parts of the system to avoid ambiguity and conflicts.

* Entities and Value Objects:

Identify and model entities (objects with a distinct identity) and value objects (objects without identity) in the domain.

Ensure that entities and value objects capture the essential aspects of the business rules.

* Aggregates:

Group related entities and value objects into aggregates.

Define consistency boundaries within aggregates to ensure data integrity.

* Repositories:

Use repositories to manage the lifecycle of aggregates and to abstract data access.

Repositories provide a way to persist and retrieve aggregates while encapsulating the underlying data storage details.

* Domain Services:

Implement domain services to encapsulate business logic that doesn't naturally fit within the boundaries of entities or value objects.

Domain services are stateless and operate on multiple aggregates or entities.

* Application Layer:

Define an application layer that orchestrates the interactions between different parts of the domain.

Implement application services that provide a high-level interface for external clients.

* Infrastructure Layer:

Implement infrastructure components such as data access, external service integration, and Web API controllers.

Keep infrastructure concerns separate from the domain logic.

* Web API Controllers:

Design controllers that act as the entry point for the Web API.

Controllers should delegate the processing of requests to application services and return appropriate responses.

* DTOs (Data Transfer Objects):

Use DTOs to transfer data between the Web API and the application layer.

DTOs help in decoupling the internal domain model from the external representation.

\* Validation:

Implement validation at various levels, including input validation in controllers, domain validation in entities, and validation in application services.

\* Asynchronous Communication:

Consider asynchronous communication patterns for handling long-running processes or improving system scalability.

How can see the project -

