**PPTify Architecture Documentation**

**Table of Contents**

1. Overview
2. Solution Structure
3. Layer-by-Layer Breakdown
4. Folder Responsibilities
5. Clean Architecture Principles
6. Benefits

**1. Overview**

The PPTify solution follows the principles of Clean Architecture combined with Domain-Driven Design (DDD). The structure is organized into well-defined layers to promote maintainability, testability, and scalability. Each layer has a clear role and set of responsibilities, minimizing coupling and maximizing cohesion.

**2. Solution Structure**

PPTify/

│

├── src/

│ ├── Api/

│ ├── Application/

│ ├── Domain/

│ ├── Infrastructure/

│ └── Shared/

│

├── tests/

├── docs/

├── scripts/

├── docker-compose.yml

├── .gitignore

└── README.md

**3. Layer-by-Layer Breakdown**

**Domain Layer (PPTify.Domain)**

**Purpose**:  
Encapsulates core business logic and domain entities. This layer is independent and has no external dependencies.

**Contents**:

* Entities/: Core domain models (e.g., User, Presentation)
* ValueObjects/: Immutable value types (e.g., Email, Name)
* Enums/: Domain-specific enumerations
* Exceptions/: Domain-specific exceptions
* Events/: Domain events triggered by changes
* Interfaces/: Domain-level contracts
* Common/: Base classes or abstractions used across domain models

**Application Layer (PPTify.Application)**

**Purpose**:  
Implements use cases of the application. It contains the business logic and coordinates between domain and infrastructure layers.

**Contents**:

* Features/ExampleFeature/: Organized per use case using CQRS
  + Commands/: Write operations
  + Queries/: Read operations
  + Handlers/: Command/Query logic
  + Validators/: FluentValidation rules
* Contracts/: Interfaces for services (e.g., IEmailService)
* Behaviors/: Pipeline behaviors like logging, caching
* Exceptions/: Application-level exceptions
* Common/: Reusable types across features
* Services/: Internal application logic
* DependencyInjection.cs: Registers application services with the DI container

**Infrastructure Layer (PPTify.Infrastructure)**

**Purpose**:  
Handles interactions with external resources such as the database, authentication, and external services.

**Contents**:

* Persistence/: Database access layer
  + DbContexts/: EF Core contexts
  + Repositories/: Repository pattern implementation
  + Configurations/: EF configurations for entities
  + Migrations/: EF Core migrations
* Services/: External service integrations
  + EmailService/: Email functionality
  + StorageService/: File or blob storage
* Authentication/: Auth mechanisms
  + Jwt/: JWT token handling
* DependencyInjection.cs: Registers infrastructure services

**Shared Layer (PPTify.Shared)**

**Purpose**:  
Houses shared utilities and helpers used by multiple layers of the application.

**Contents**:

* Helpers/: Utility classes
* Constants/: Application-wide constants
* Extensions/: Extension methods
* Utilities/: Miscellaneous utility classes

**API Layer (PPTify.Api)**

**Purpose**:  
Serves as the entry point to the application through a Web API.

**Contents**:

* Controllers/: ASP.NET Core API controllers
* Extensions/: Extension methods for API-specific concerns
* Middleware/: Custom middlewares (e.g., exception handling)
* Filters/: Action filters for validation, logging
* Properties/: Standard .NET project files
* Program.cs: Application bootstrap and startup logic
* appsettings.json: Configuration settings
* Dockerfile: Containerization support

**4. Additional Folders**

**Tests (/tests)**

Contains all automated test projects organized by test type.

* PPTify.Tests.Unit: Unit tests for domain and application logic
* PPTify.Tests.Integration: Integration tests for infrastructure components
* PPTify.Tests.Functional: End-to-end tests of application features

**Documentation (/docs)**

Project documentation for internal and external use.

* architecture/: System design documents and diagrams
* api-documentation/: API usage, Swagger setup
* setup-guides/: Environment setup, onboarding docs
* usage-guides/: How-to guides for extending or using the system

**Scripts (/scripts)**

Automation scripts for build, deployment, migration, and maintenance.

* PowerShell, Bash, and CMD scripts
* Used in CI/CD and DevOps pipelines

**5. Clean Architecture Principles**

* Separation of concerns: Each layer is responsible for a single area of functionality.
* Independence: Inner layers (Domain, Application) are unaware of outer layers (Infrastructure, API).
* Testability: Business rules can be tested without involving infrastructure.
* Flexibility: Easy to swap out infrastructure components like databases or services.
* Maintainability: Clear folder and responsibility separation make onboarding and scaling easier.

**6. Benefits**

* Scalable: Well-suited for growing applications and teams
* Modular: Supports separation into microservices or modular monolith
* Test-Driven: Encourages unit, integration, and functional testing
* DevOps-Ready: Easily integrated into CI/CD pipelines with containerization support
* Extensible: New features can be added in isolation under the Features/ folder

**Summary**

The PPTify architecture is designed to offer a clean, maintainable, and scalable foundation for building production-grade .NET applications. By following Clean Architecture and DDD practices, the system encourages well-structured code, separation of concerns, and high testability while remaining flexible for future enhancements.