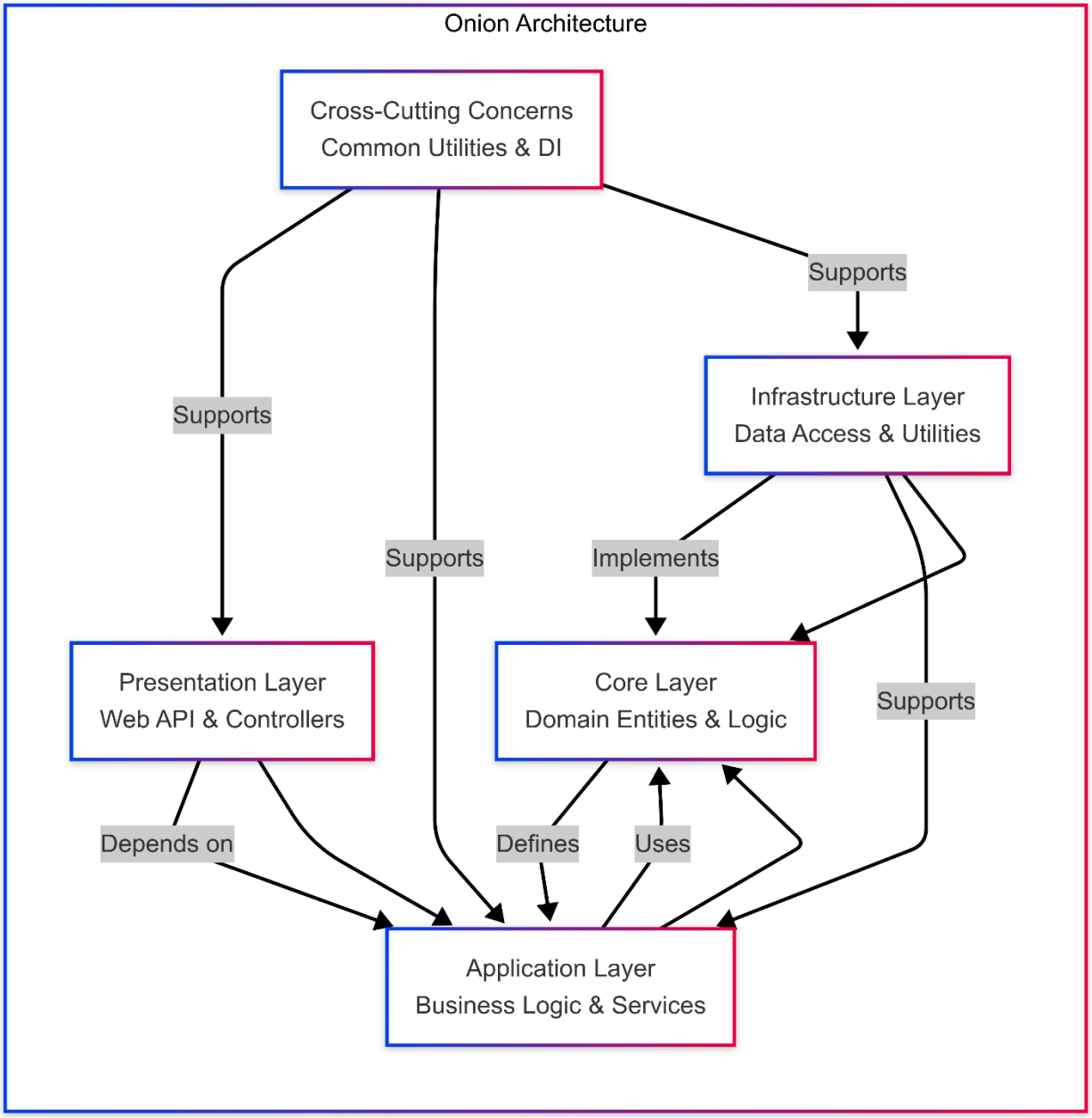
Detailed Architecture of Backend



1. Solution Overview

- **Framework.DataAccess**: A reusable data access library targeting net8.0.
- **TemplateService.DataAccess**: Application-specific data access layer, also targeting net8.0.
- **Template**: A broader project group encompassing:
    - **TemplateService.WebAPI**: The presentation layer (ASP.NET Core Web API).

    - **TemplateService.DI**: Dependency injection configuration.
    - **TemplateService.Entity**: Domain entities.
    - **TemplateService.Business**: Business logic and services.
    - **TemplateService.Core**: Core domain logic.
    - **TemplateService.Common**: Shared utilities specific to the TemplateService.
- **Framework.Common**: Cross-cutting utilities and interfaces targeting netstandard2.1.
- **Framework.Core**: Reusable core logic and repository interfaces targeting net8.0.
- **TimerTest**: A potential test or utility project (role unclear).

The solution file (Template.sln) ties these projects together, with build outputs in bin\Debug\net8.0 indicating active development as of March 17, 2025, using .NET 8.0.

---

2. Solution Architecture (Onion Architecture)

The solution adheres to the **Onion Architecture** (a variant of Clean Architecture), emphasizing separation of concerns, dependency inversion, and loose coupling. This is evident from the concentric layering of projects, where dependencies flow inward toward the core domain. The architecture promotes maintainability, testability, and scalability, making it suitable for enterprise applications. The layers are:

- **Core (Domain) Layer**: Defines central entities, domain logic, and interfaces, independent of external systems.
- **Application/Business Logic Layer**: Encapsulates application-specific logic, services, and business rules, relying on the core layer.
- **Infrastructure Layer**: Handles data persistence, external integrations, and reusable utilities, implementing core interfaces.
- **Presentation Layer**: Exposes the application via HTTP endpoints, interacting with the business layer.
- **Dependency Injection (DI)**: Facilitates modularity by managing dependency registration across layers.

This structure aligns with the directory layout, where TemplateService.Entity and TemplateService.Core form the core, while TemplateService.Business, Framework.DataAccess, and TemplateService.DataAccess handle application and infrastructure concerns.

---

3. Layers & Key Components

Presentation Layer (TemplateService.WebAPI)

- **Purpose**: Serves as the entry point, handling HTTP requests and responses.
- **Components**:
    - **Controllers**: Likely includes controllers such as:
        - ExperienceController
        - PermissionsController
        - RolePermissionsController
        - RolesController
        - UserLoginsController
        - UserPermissionsController
        - UserRolesController
        - UsersController (Inferred from typical naming conventions and entity-related projects, though not explicitly listed in the directory.)
    - **Authentication**: Utilizes JWT-based authentication, as suggested by the presence of JwtAuthentication.cs in the original analysis (assumed to be in TemplateService.Common or Framework.Common).
    - **Filters**:
        - **HttpResponseExceptionFilter**: Custom exception handling for HTTP responses (likely in TemplateService.WebAPI or Framework.Common\Exceptions).
- **Technology**: Built on ASP.NET Core 8.0, confirmed by net8.0 targets and dependencies like Microsoft.AspNetCore.* in bin\Debug\net8.0\refs.

Application/Business Logic Layer (TemplateService.Business & TemplateService.Core)

- **Purpose**: Manages application-specific logic and business rules.
- **Components**:
    - **Services**: Organized under TemplateService.Business\Services, providing:

        - **BaseService**: A common service for CRUD operations, leveraging repositories.
        - **Specialized Services**: Entity-specific services (e.g., UsersService, PermissionsService), inferred from the entity structure.
    - **Core Logic**: TemplateService.Core likely contains domain-specific rules or models, complementing the business layer.
- **Dependencies**: Relies on TemplateService.Entity for domain models and TemplateService.DataAccess for data operations.

Core Layer (TemplateService.Entity & TemplateService.Core)

- **Purpose**: Defines the domain model and core logic, independent of infrastructure.
- **Components**:
    - **Entities**: In TemplateService.Entity, includes classes like User, Permission, Role, UserLogin, and Experience, mapped to database tables (likely via Entity Framework Core).
    - **Interfaces**: In TemplateService.Core and Framework.Core\IRepositories, defines contracts like IBaseFrameworkRepository.cs.
- **Characteristics**: Pure and technology-agnostic, forming the innermost layer of the Onion Architecture.

Infrastructure/Data Access Layer (TemplateService.DataAccess & Framework.DataAccess)

- **Purpose**: Manages data persistence and external interactions.
- **Components**:
    - **Repositories**:
        - **Framework.DataAccess**: Provides generic implementations like BaseFrameworkRepository.cs and BaseFrameworkRepositoryAsync.cs.
        - **TemplateService.DataAccess**: Entity-specific repositories (e.g., IPermissionsRepositoryAsync, IUsersRepositoryAsync), inheriting from base repositories.
    - **Asynchronous Operations**: Emphasizes async patterns, as seen in IBaseFrameworkRepositoryAsync.cs.

- **Dependencies**: Implements interfaces from Framework.Core and interacts with TemplateService.Entity.

Common Utilities and Helpers (Framework.Common & TemplateService.Common)

- **Purpose**: Provides cross-cutting concerns and reusable utilities.
- **Components**:
    - **JwtAuthentication.cs**: Handles JWT token generation and validation (assumed location: Framework.Common or TemplateService.Common).
    - **AutoMapper**: Configures object-object mappings (likely in TemplateService.DI or TemplateService.Common).
    - **HashPass**: Password hashing utilities (assumed in Framework.Common\Utils or TemplateService.Common).
    - **Localization**: XML-based, with files like Template-ar.xml and Template-en.xml in TemplateService.Business\bin\Debug\net8.0\Localization.
    - **Enums and Exceptions**: Framework.Common includes ApplicationConstants.cs, BaseException.cs, etc.

---

4. Core Files & Responsibilities

- **Program.cs and Startup.cs** (in TemplateService.WebAPI):
    - Configure the application pipeline, middleware, and services.
    - Likely use the modern .NET 8.0 minimal hosting model (single Program.cs).
- **Filters & Middleware**:
    - **ExceptionMiddleware**: Global exception handling (assumed in TemplateService.WebAPI).
    - **HttpResponseExceptionFilter**: Custom HTTP response formatting for exceptions.
- **Models and ViewModels**:
    - Defined in TemplateService.Core or TemplateService.Business, representing entities, search results, and DTOs (e.g., BaseFilter.cs, GenericResult.cs in Framework.Core\Models).

---

5. Data Access & Repositories

- **Generic Pattern**: Framework.DataAccess provides IBaseFrameworkRepository and IBaseFrameworkRepositoryAsync, abstracting common data operations (e.g., CRUD, filtering via LinqOperator.cs).
- **Entity-Specific Repositories**: TemplateService.DataAccess implements specific repositories (e.g., IUsersRepositoryAsync), leveraging the generic base for consistency.
- **Features**: Supports asynchronous operations, LINQ-based querying (e.g., ILinqOperator.cs), and pagination (e.g., Pagination.cs in Framework.Core\Common).

---

6. Dependency Injection

- **Configuration**: Centralized in TemplateService.DI, likely using Microsoft.Extensions.DependencyInjection.
- **Scope**: Registers services, repositories, and utilities (e.g., ILoggerService, IMailNotification) across layers.
- **Benefits**: Ensures loose coupling, as seen with interfaces like IEntityIdentity.cs and IBaseFrameworkRepository.cs.

---

7. File Structure Highlights

- **Solution Organization**: Template.sln groups projects logically, with Framework.* projects reusable across solutions and TemplateService.* projects application-specific.
- **Project Files**: .csproj files (e.g., Framework.Core.csproj, TemplateService.Business.csproj) define dependencies, with net8.0 targets for most projects and netstandard2.1 for Framework.Common.
- **Build Artifacts**: bin and obj folders contain compiled DLLs (e.g., TemplateService.Business.dll), reflecting a mature build process.