Nombre de la solución

* APL.Solution
* APL.Suite (si lo quieres más comercial).

**Proyectos dentro de la solución**

1. **Frontend (aplicación web)**
   * **APL.Web**
   * **APL.WebApp**
   * **APL.Portal** (si será el portal principal para usuarios).
2. **Backend (API REST)**
   * **APL.API**
   * **APL.Service**
   * **APL.RestApi** (más explícito).
3. **Capa de lógica de negocio (opcional, si separas)**
   * **APL.Core**
   * **APL.Domain**
   * **APL.Application**
4. **Capa de datos (opcional)**
   * **APL.Infrastructure**
   * **APL.Data**

APL.Solution

├── APL.WebApp

├── APL.API

├── APL.Core

├── APL.Infrastructure

De esta forma, siempre que alguien vea el repositorio sabrá de inmediato:

APL.WebApp → aplicación web en .NET Core (MVC/Razor o Blazor, según sea).

APL.API → servicios RESTful.

APL.Core / Infrastructure → capas reutilizables y desacopladas.

**Arquitectura propuesta en .NET Core**

**📂 Nombre de la solución**

* **APL.Solution**  
  *(contiene todos los proyectos y capas)*

**📦 Proyectos principales**

1. **Frontend (Aplicación Web)**
   * **APL.WebApp**
   * Función: Portal principal para usuarios (administradores, comerciales, retail).
   * Tecnologías: ASP.NET Core MVC / Razor Pages / Blazor (dependiendo de tu elección).
2. **Backend (API REST)**
   * **APL.API**
   * Función: Exponer servicios RESTful (acuerdos, promociones, liquidaciones, reportes).
   * Comunicación: JSON, OpenAPI/Swagger.

**⚙️ Capas de la lógica y datos**

1. **Lógica de negocio / Aplicación**
   * **APL.Application**
   * Contiene casos de uso (services, handlers, validaciones de negocio).
   * Aquí se gestionan las reglas sobre acuerdos, promociones y liquidaciones.
2. **Dominio (Entidades y contratos)**
   * **APL.Domain**
   * Define entidades (Agreement, Promotion, Liquidation, User, etc.).
   * Contiene interfaces que describen contratos de repositorios.
3. **Infraestructura / Datos**
   * **APL.Infrastructure**
   * Implementa acceso a datos (EF Core, Dapper, etc.).
   * Manejo de persistencia, repositorios, migraciones, conexiones.
   * También puede alojar integraciones con sistemas externos (ERP, CRM).

**🧪 Otros proyectos opcionales**

1. **Testing**
   * **APL.Tests** (unitarios, integración).
   * Frameworks sugeridos: xUnit, NUnit o MSTest.
2. **Common/Shared**
   * **APL.Shared** (opcional).
   * Utilidades comunes: helpers, DTOs, excepciones personalizadas, constantes.

**📂 Ejemplo de estructura en Visual Studio / Git**

APL.Solution

├── APL.WebApp → Aplicación web frontend

├── APL.API → Servicios RESTful

├── APL.Application → Casos de uso / lógica de aplicación

├── APL.Domain → Entidades y contratos

├── APL.Infrastructure → Persistencia / integraciones

├── APL.Shared → Utilidades comunes (opcional)

└── APL.Tests → Pruebas unitarias e integración

**🔗 Flujo de dependencias (clean architecture style)**

* **APL.WebApp** → consume **APL.API**
* **APL.API** → usa **APL.Application**
* **APL.Application** → depende de **APL.Domain**
* **APL.Infrastructure** → implementa contratos de **APL.Domain**
* **APL.Application** → referencia a **APL.Infrastructure** solo a través de interfaces

✅ Esto te da:

* **Escalabilidad**: puedes crecer por módulos sin romper todo.
* **Mantenibilidad**: cada capa cumple un rol claro.
* **Claridad en nombres**: cualquiera que vea la solución entiende qué hace cada proyecto.