فهرست

موضوع کلی پروژه ۲

اجزاء پیش بینی شده برای زیرساخت ۲

آشنایی کلی با معماری اولیه پروژه ۲

ارتقاء پروژه به پلتفرم جدید .NET Core ٤

معرفی اجزاء تشکیل دهنده معماری ٤

1. برنامه اصلی (Single Page Application) ٤
2. کاربر سرویس (Service Client) ٤
3. سرویس وب (Web Service) ٤
4. سرویس های گردش کار (Workflow) ٤
5. دسترسی به اطلاعات – مدل (Model) ٥
6. دسترسی به اطلاعات – نگاشت (Mapping) ٥
7. کلاس های نمایی مدل (View Model) ٥
8. دسترسی به اطلاعات – نگاشت گر دامنه (Domain Mapper) ٥
9. دسترسی به اطلاعات – کلاس های انباره (Repository) ٦
10. دسترسی به اطلاعات – کار با EF Core ٦
11. تزریق وابستگی یا کنترل معکوس (Inversion of Control) ٦
12. تست های واحد (Unit Tests) ۷

آشنایی با ساختار سورس های پروژه ۷

ساختار راه حل (Solution) اصلی پروژه ۷

پوشه BackEnd ۸

پوشه CrossCutting ۱۰

پوشه FrontEnd ۱۲

پوشه Model ۱۲

پوشه Tests ۱٤

موضوع کلی پروژه

تحلیل، طراحی و تولید نسل جدید سیستم مالی و اداری تدبیر با ویژگی های زیر :

۱. مناسب برای استفاده توسط سازمان های بزرگ با شرکت ها و شعب زیر مجموعه متعدد
۲. بر مبنای یک زیرساخت نرم افزاری قدرتمند و انعطاف پذیر
۳. استفاده از رویکرد سرویس گرا

اجزاء پیش بینی شده برای زیرساخت

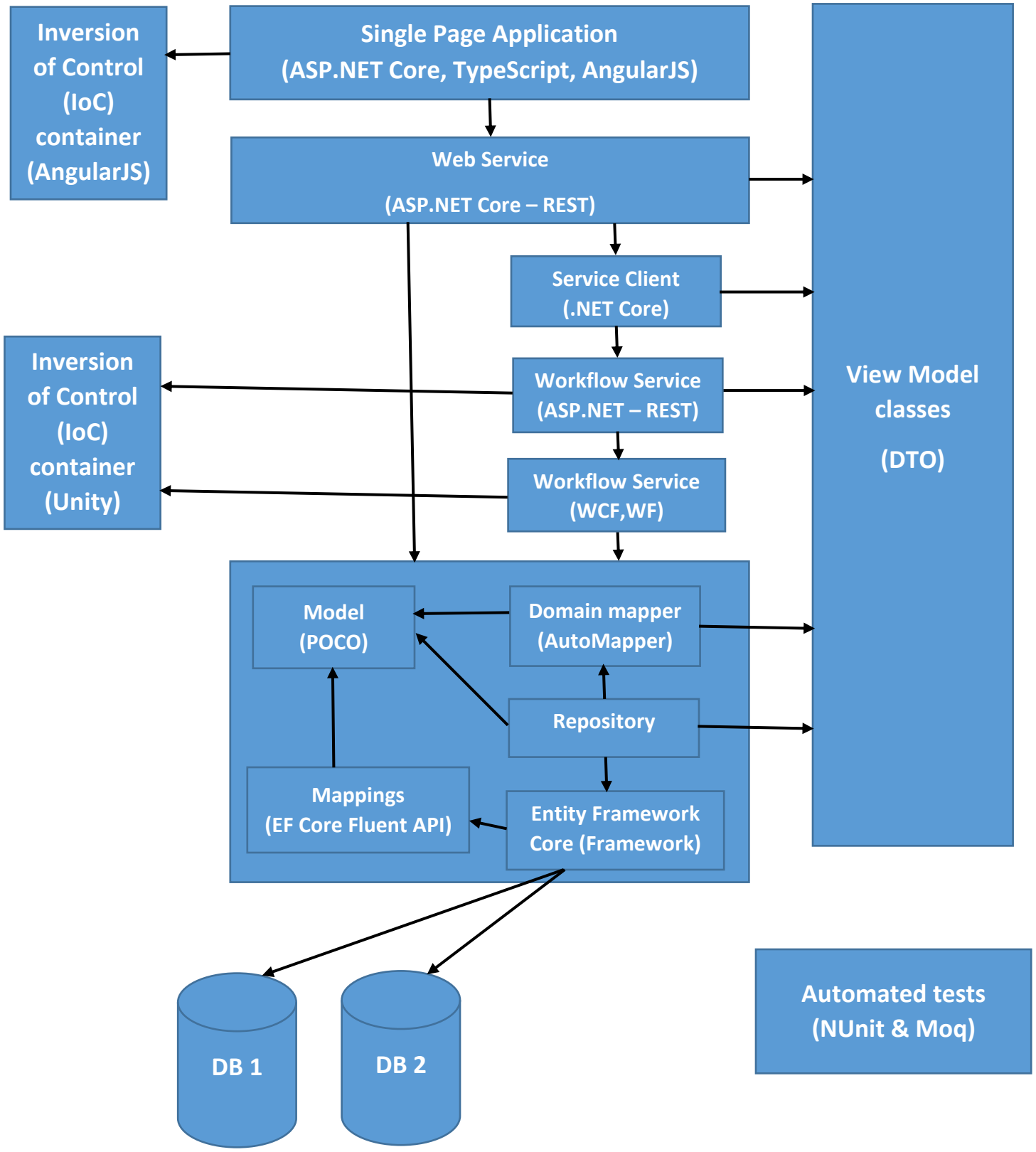
۱. سرویس امنیتی با ویژگی های کلی زیر :
   - بر مبنای نقش (Role-based)
   - پشتیبانی از چندین شرکت و چندین شعبه داخل هر شرکت
   - پشتیبانی از سناریو های پیچیده راهبری برای تامین نیاز های کاربران خاص
۲. مدیریت فرآیند ها و کارتابل سازمانی
۳. واسط کاربری چند زبانه
۴. سرویس های زیرساختی (Logging, Caching, Exception Handling)
۵. امکان توسعه مدل داده ای از طریق اضافه کردن فیلد های کاربری جدید

آشنایی کلی با معماری اولیه پروژه

یکی از نیازمندی های کلان پروژه، استفاده از معماری چندلایه ای (N-tier architecture) است. برای تامین این نیازمندی، یکی از رایج ترین اشکال معماری چندلایه ای یعنی معماری سه لایه ای مورد استفاده قرار گرفته است. لایه های منطقی موجود در این معماری عبارتند از :

۱. لایه واسط کاربری (User Interface layer)
۲. لایه منطق کسب و کار (Business Logic layer)
۳. لایه دسترسی به داده (Data Access layer)

در ادامه این بخش، اجزاء مختلف پیش بینی شده در معماری پروژه را به اختصار معرفی می کنیم. نمای تصویری معماری را در نمودار زیر می توانید مشاهده کنید :


Inversion of Control (IoC) container (AngularJS)
Single Page Application (ASP.NET Core, TypeScript, AngularJS)
Web Service (ASP.NET Core – REST)
Service Client (.NET Core)
Inversion of Control (IoC) container (Unity)
Workflow Service (ASP.NET – REST)
Workflow Service (WCF,WF)
View Model classes (DTO)
Model (POCO)
Domain mapper (AutoMapper)
Repository
Mappings (EF Core Fluent API)
Entity Framework Core (Framework)
DB 1
DB 2
Automated tests (NUnit & Moq)

ارتقاء پروژه به پلتفرم جدید .NET Core

پس از ارائه رسمی ویرایش دوم از پلتفرم .NET Core توسط شرکت مایکروسافت، و با انجام ارزیابی های لازم برای مزایا و معایب استفاده از این پلتفرم در سیستم جدید تدبیر، تصمیم گرفته شد تا تمام کارهای انجام شده به پلتفرم جدید ارتقاء داده شود. در نتیجه این تصمیم، تغییرات متعددی در معماری موجود در برنامه و زیرساخت ایجاد شد. به دلیل عدم پشتیبانی از بعضی فناوری ها و ابزارهای استفاده شده در پلتفرم جدید، به طور عمده بخش گردش کار و NHibernate، بخش های جدیدی در معماری قبلی اضافه شده است.

معرفی اجزاء تشکیل دهنده معماری

۱. **برنامه اصلی (Single Page Application)** : این بخش، واسط کاربری اصلی در برنامه کاربردی کسب و کار (Business Application) است و در معماری سه لایه مورد استفاده، لایه واسط کاربری را شکل می دهد. با توجه به نیازمندی بنیادی سیستم جدید تدبیر (واسط کاربری تحت وب)، و به منظور ایجاد یک واسط کاربری مدرن، این بخش توسط فناوری های ASP.NET Core و AngularJS پیاده سازی می شود.

۲. **کاربر سرویس (Service Client)** : برای تماس با سرویس وب از طریق زبان برنامه نویسی C#، امکانات موجود در پلتفرم دات نت برای تماس با سرویس های REST از طریق پروتکل استاندارد HTTP مورد استفاده قرار می گیرند. این بخش از معماری، وظیفه ارسال درخواست به سرویس ها و دریافت پاسخ از سرویس ها را به عهده دارد. لازم به ذکر است که تماس های برنامه اصلی با سرویس وب به صورت مستقیم و از طریق زبان های TypeScript و JavaScript انجام می شوند، ولی در موارد دیگر (مانند تماس با سرویس گردش کار) می توان از امکانات این بخش استفاده کرد.

۳. **سرویس وب (Web Service)** : این بخش شامل یک یا چند سرویس وب است که قابلیت های اصلی کسب و کار در آنها پیاده سازی می شود. برای پیاده سازی اولین سرویس برنامه، در حال حاضر از معماری REST (REpresentational State Transfer) و فناوری ASP.NET Core استفاده شده است. این تصمیم با هدف پشتیبانی بهتر از واسط های کاربری موبایل صورت گرفته و در صورت لزوم قابل تغییر است.

۴. **سرویس های گردش کار (Workflow)** : پشتیبانی از فرآیندها و گردش های کاری با اتکا به فناوری موجود در چارچوب دات نت (Workflow Foundation یا WF) انجام می شود. چون این فناوری در چارچوب کاری جدید .NET Core. پشتیبانی نمی شود، برای دسترسی به امکانات پیاده سازی شده در معماری قبلی، یک سرویس واسط جدید با معماری استاندارد REST اضافه می شود. تمام قابلیت هایی

که در معماری قبلی برای پیاده سازی گردش های کاری نمونه در سرویس وب اصلی وجود داشت، در معماری جدید در این سرویس واسط قرار می گیرند. مکانیزم دسترسی به این امکانات در سرویس وب اصلی، بخش کاربر سرویس (Service Client) خواهد بود. این بخش از معماری، به همراه بخش کاربر سرویس و سرویس وب، لایه منطق کسب و کار را در معماری سه لایه ای شکل می دهند.

۵. **دسترسی به اطلاعات – مدل (Model)** : برای کار با پایگاه های داده ای داخل برنامه، از Entity Framework Core به عنوان OR Mapper استفاده شده است. این ابزار، مانند بیشتر ابزارهای مشابه، برای کار با اقلام اطلاعاتی نیازمند آن است که مدل رابطه ای (Relational) داخل پایگاه داده به صورت شیئ گرا تعریف شده و به آن ابزار معرفی شود. بخش مدل شامل تمام کلاس هایی است که معادل شیئ گرای جداول اطلاعاتی هستند. کلاس های موجود در این بخش، با رویکرد ساده و به صورت اصطلاحا POCO (Plain Old CLR Object) مدلسازی و پیاده سازی شده اند. در این رویکرد کلاس ها صرفا شامل اطلاعات داده ای بوده و به ندرت عملیات و توابع کاربردی در آنها پیاده سازی می شود.

۶. **دسترسی به اطلاعات – نگاشت (Mapping)** : این بخش نحوه صحیح تبدیل ساختار ارتباطی به شیئ گرا (و بالعکس) را به منظور استفاده توسط ابزار EF Core تعریف می کند. به منظور بالا بردن خوانایی کد برنامه و ایجاد امکان پیاده سازی تست های واحد (Unit Test) برای نگاشت ها، از امکانات موجود در EF Core استفاده شده که توسط آن می توان نگاشت های مورد نیاز را با استفاده از یک API اصطلاحا روان (Fluent) پیاده سازی کرد.

۷. **کلاس های نمایی مدل (View Model)** : برای کاهش وابستگی مستقیم بخش های مختلف معماری به مدل شیئ گرای اطلاعات، از کلاس های نمایی مدل استفاده شده که از الگوی پیاده سازی DTO (Data Transfer Objects) پشتیبانی می کنند. وظیفه اصلی این کلاس ها تعریف شکل خاصی از مدل شیئ گرا است که برای نمایش و مدیریت در واسط های کاربری مناسب تر باشد. استفاده از این کلاس ها به جای کلاس های اصلی بخش مدل، مشابه استفاده از یک نما (View) به جای جدول یا جداول به کار رفته در آن نما است. این کلاس ها به صورت بالقوه مزیت های مختلفی را در اختیار برنامه نویس قرار می دهند. به عنوان مثال، با استفاده از این کلاس ها می توان مقادیر قابل محاسبه را به صورت مستقل از مدل اصلی تعریف و داخل برنامه مورد استفاده قرار داد (مانند جمع مقادیر بدهکار و بستانکار آرتیکل های یک سند).

۸. **دسترسی به اطلاعات – نگاشت گر دامنه (Domain Mapper)** : برای تبدیل اشیا کلاس های مدل به کلاس های نمایی مدل (و بالعکس) می توان کلاس های مبدل جداگانه را به سادگی پیاده سازی کرد. این امر در عین سادگی، بسیار دست و پاگیر و توام با خطا است و حجم کاری و زمانی مورد نیاز برای آن

با رشد ساختار اطلاعاتی برنامه به سرعت افزایش می یابد. برای سادگی این تبدیل ها، از یک ابزار بسیار مفید و محبوب به نام AutoMapper استفاده شده که به دلیل سادگی و کارایی آن، تقریبا به یک ابزار استاندارد تبدیل شده است. در صورت به کارگیری تعدادی قرارداد ساده در نامگذاری ویژگی های (Properties) کلاس های View Model، می توان بخش عمده ای از تبدیل ها را با حداقل کد نویسی ممکن توسط این ابزار انجام داد.

۹. **دسترسی به اطلاعات – کلاس های انباره (Repository)** : کلاس های موجود در این بخش از الگوی طراحی معروف انباره (Repository pattern) پشتیبانی می کنند. مطابق با این الگو، تمامی عملیات دیتابیسی مورد نیاز برای مدیریت اطلاعات یک یا چند جدول مرتبط توسط یک کلاس مجزا پیاده سازی می شوند. همچنین پشتیبانی از تراکنش ها (Transaction) هنگام ارسال دستورات به سرور دیتابیس، از نیازمندی های اصلی در این الگوی طراحی است.

۱۰. **دسترسی به اطلاعات – کار با EF Core** : برای استفاده مستقیم از امکانات فناوری EF Core در کلاس های Repository، از امکانات پیاده سازی شده در زیرساخت جدید تدبیر استفاده شده است. بخش اصلی این زیرساخت که در تمامی کلاس های Repository از آن استفاده شده اینترفیس های IUnitOfWork، IRepository و IAsyncRepository و پیاده سازی موجود برای آنها است. این بخش و سایر بخش های قبل که با پیشوند "دسترسی به اطلاعات" مشخص شده اند، به صورت مشترک لایه دسترسی به اطلاعات را تشکیل می دهند.

۱۱. **تزریق وابستگی یا کنترل معکوس (Inversion of Control)** : برای پشتیبانی از اصل مهم استفاده از وابستگی های نرم (Loose Coupling) در پیاده سازی برنامه، بخش عمده ای از وابستگی های موجود در بخش های مختلف با استفاده از اینترفیس ها مشخص شده است. با توجه به تنوع محیط برنامه نویسی در معماری جدید، برای تزریق وابستگی ها در بخش های مختلف از ابزارهای مختلفی استفاده می شود. در برنامه اصلی که برمبنای فناوری AngularJS و بصورت یک برنامه تک صفحه ای (SPA) پیاده سازی می شود، از امکانات موجود در AngularJS برای تزریق وابستگی ها استفاده می شود. در طراحی جدید چارچوب .NET Core امکانات اولیه ای برای مدیریت وابستگی ها پیش بینی شده که با وجود ساده بودن این امکانات، در حال حاضر برای نیازهای سرویس وب کافی بوده و مورد استفاده قرار می گیرند. لازم به یادآوری است که ابزار Unity مایکروسافت در چارچوب کاری جدید پشتیبانی نشده و به احتمال زیاد هیچوقت به این چارچوب منتقل نخواهد شد. با این حال، مانند معماری قبل برای مدیریت وابستگی ها در سرویس های گردش کاری قابل استفاده است.

۱۲. **تست های واحد (Unit Tests)** : تست های واحد در این مرحله از کار، جزو خروجی های اصلی پروژه نیستند. با این حال این تست های واحد برای تعداد محدودی از کلاس های پروژه، به عنوان نمونه استفاده از ابزارهای انتخابی، پیاده سازی شده اند. برای این کار از ابزارهای متن باز NUnit به عنوان چارچوب تست (Test Framework) و Moq به عنوان چارچوب مقلد سازی (Mocking Framework) استفاده شده است.

آشنایی با ساختار سورس های پروژه

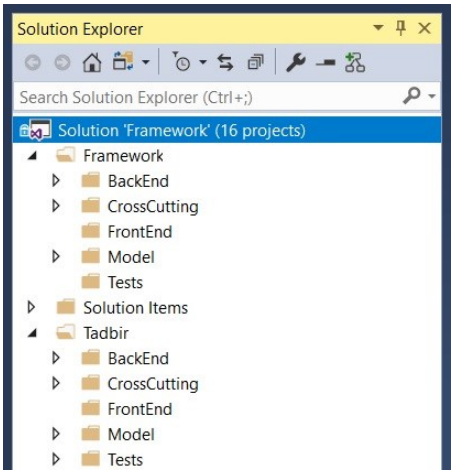
پیاده سازی های موجود تماما داخل برنامه ویژوال استودیو ۲۰۱۷ انجام شده است. سورس کامل پروژه، به همراه سایر فایل های مورد استفاده، روی یک فضای سروری خصوصی در سایت GitHub قرار گرفته است. این فضای خصوصی در حال حاضر از طریق یک حساب کاربری شخصی ایجاد شده و پس از مجوزدهی لازم، توسط کاربران تعریف شده در سایت GitHub قابل دسترسی است. نسخه آفلاین تاریخچه و تغییرات سورس ها روی کامپیوتر شخصی، به صورت منظم و روزانه با نسخه آنلاین روی سرور هماهنگ سازی (Sync) می شود.

ساختار راه حل (Solution) اصلی پروژه

برای سازماندهی بهتر پروژه ها در Solution اصلی، از امکان Solution Folder استفاده شده است. دو پوشه اصلی در این ساختار پیش بینی شده است :

۱. پوشه زیرساخت (Framework) : شامل کلیه پروژه هایی است که ماهیت عمومی و زیرساختی دارند.
۲. پوشه تدبیر (Tadbir) : شامل پروژه هایی است که برای پیاده سازی برنامه کاربردی استفاده می شوند.

این ساختار را در شکل ۱ مشاهده می کنید :

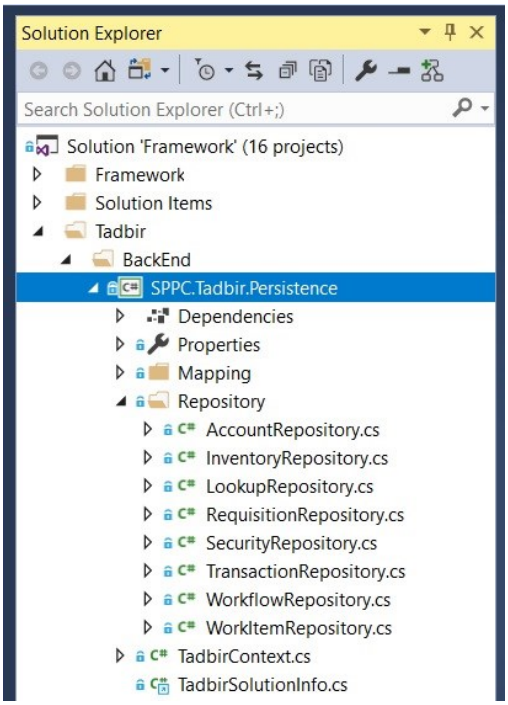


شکل ۱ - ساختار اصلی پوشه ها در زیرساخت و تدبیر

همان طوری که در شکل بالا می بینید، داخل هر یک از پوشه های اصلی ساختار مشابهی پیش بینی شده که برای سازماندهی منطقی اجزاء تشکیل دهنده معماری استفاده شده است. در بخش های بعدی، هر یک از این پوشه ها و پروژه های داخل آنها را برای پوشه اصلی تدبیر به اختصار معرفی می کنیم. ساختار پروژه ها در پوشه زیرساخت (Framework) نیز تا حد زیادی مشابه پوشه تدبیر است.

پوشه BackEnd – این پوشه برای سازماندهی اجزاء سمت سرور پیش بینی شده و در حال حاضر شامل پروژه های زیر است :

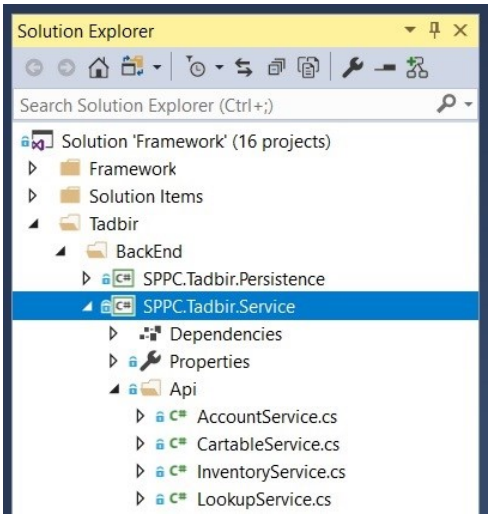
۱. پروژه SPPC.Tadbir.Persistence : کلاس های پیاده سازی شده در بخش های Repository و Mapping درون این پروژه قرار می گیرند. در شکل ۲ نمای ساختار درختی این پروژه نمایش داده شده است :



شکل ۲ - ساختار پروژه Persistence در پوشه تدبیر

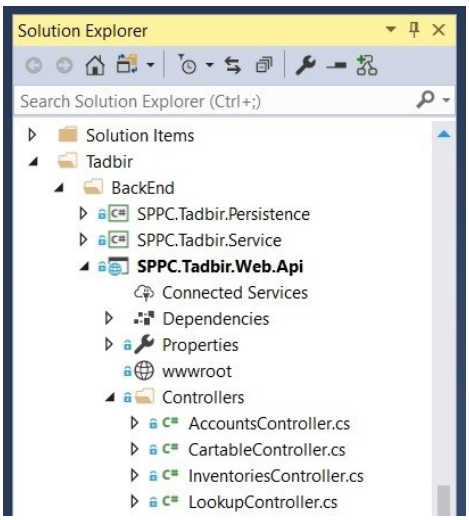
۲. پروژه SPPC.Tadbir.Service : کلاس های پیاده سازی شده در بخش Service Client درون این پروژه قرار می گیرند. با توجه به پیاده سازی سرویس وب اصلی با معماری REST، کلیه تماس ها با این سرویس از طریق ارسال درخواست (Request) و دریافت پاسخ (Response) با ساختار تعریف شده در پروتکل HTTP صورت می گیرد. ابزار اصلی برای مبادله اطلاعات با سرویس وب، کلاس ApiClient در همین پروژه است که امکانات اصلی برای ارسال و دریافت اطلاعات را از کلاس ServiceClient در

پروژه زیرساختی مشابه به ارث برده و قابلیت های تکمیلی برای مجوزدهی امنیتی را به آن اضافه می کند. بیشتر کلاس های این پروژه وابستگی مستقیم به اینترفیس اصلی این کلاس (IApiClient) دارند. ساختار سورس های این پروژه در شکل ۳ نشان داده شده است :



شکل ۳ - ساختار پروژه کاربر سرویس در پوشه تدبیر و زیرساخت

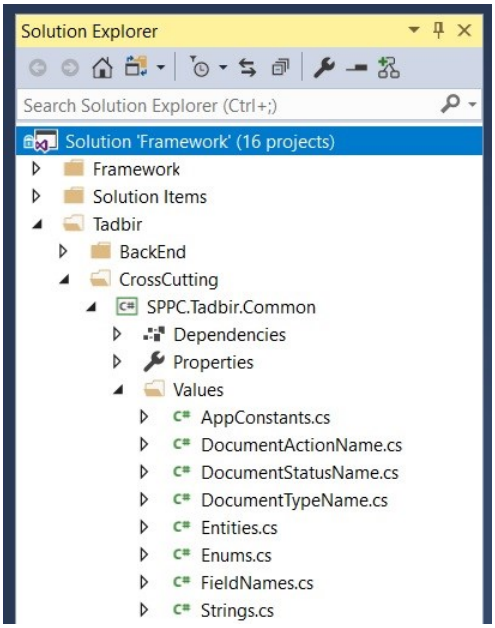
۳. پروژه SPPC.Tadbir.Web.Api : کلاس های پیاده سازی شده در بخش Web Service درون این پروژه قرار می گیرند. پیاده سازی سرویس های وب با استفاده از زیرساخت ASP.NET Core و کلاس‌های Controller انجام می شود. نمای درختی بخش عمده این پروژه در شکل ۴ نمایش داده شده است :



شکل ۴ – ساختار پروژه سرویس وب در پوشه تدبیر

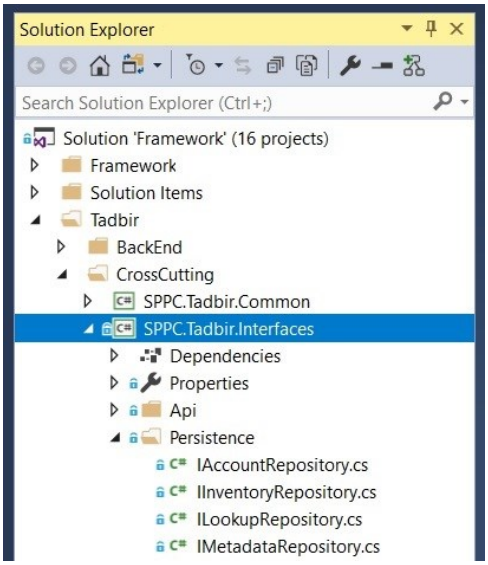
پوشه CrossCutting – این پوشه برای سازماندهی پروژه هایی به کار رفته که به صورت مشترک در دو یا چند پروژه دیگر مورد استفاده قرار گرفته اند. این پوشه در حال حاضر شامل پروژه های زیر است :

۱. پروژه SPPC.Tadbir.Common : این پروژه به هیچ یک از پروژه های دیگر وابستگی ندارد و در پایین ترین سطح وابستگی قرار می گیرد. کلاس های موجود در این پروژه به صورت بالقوه می توانند در تمام پروژه های موجود مورد استفاده قرار بگیرند، بدون اینکه مشکلات فنی از قبیل وابستگی دایره ای (Circular Reference) ایجاد شود. این پروژه در این مرحله از کار شامل کلاس هایی می شود که متن های قابل ترجمه واسط کاربری را به صورت متمرکز تعریف می کنند. با توجه به اینکه قابلیت چند زبانه در مراحل بعدی به زیرساخت و برنامه اضافه می شوند، به عنوان پیش بینی فنی، از قرار دادن متن های ثابت در کلاس های خارج از این پروژه اکیدا خودداری شده است. در پیاده سازی تمام این کلاس ها از الگوی ثابتی استفاده شده است. ساختار سورس های این پروژه را در شکل ۶ می بینید :



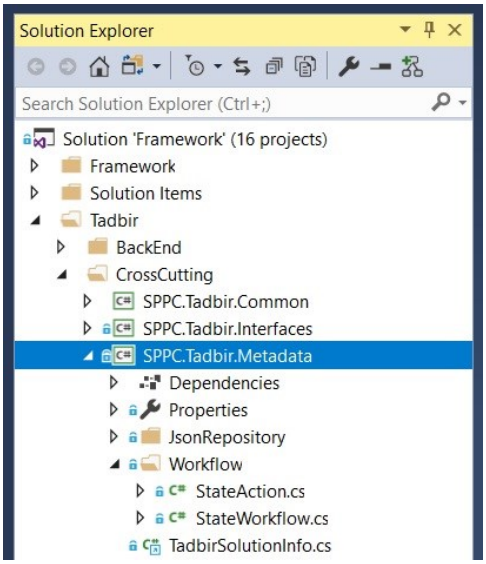
شکل ۶ – ساختار پروژه مشترک (Common) در پوشه تدبیر

۲. پروژه SPPC.Tadbir.Interfaces : این پروژه جنبه زیرساختی داشته و تجلی مستقیم در معماری برنامه ندارد. وظیفه اصلی آن سازماندهی اینترفیس های تعریف شده در سایر اجزاء سیستم در یک پروژه مرکزی است، که این امر ایجاد وابستگی در پروژه های دیگر را کمی ساده تر می کند. با توجه به تعریف اینترفیس ها در بخش های مختلف، برای تفکیک اینترفیس های هر بخش از پوشه های فیزیکی استفاده شده است. در شکل ۷ بخشی از ساختار پروژه را می بینید :



شکل ۷ – ساختار پروژه دربرگیرنده اینترفیس ها در پوشه تدبیر

۳. پروژه SPPC.Tadbir.Metadata : کلاس های درون این پروژه، ساختار اطلاعات فراداده ای برنامه اصلی تدبیر و تسهیلات مورد نیاز برای استفاده از آنها را در لایه های مختلف معماری پیاده سازی می کنند. این پروژه بنا بر ماهیت زیرساختی خود، در بخش های مختلف مورد استفاده قرار می گیرد، بنابراین برای سازماندهی بهتر سورس های آن از پوشه های فیزیکی شده است. در شکل ۸ بخشی از ساختار این پروژه را می بینید :

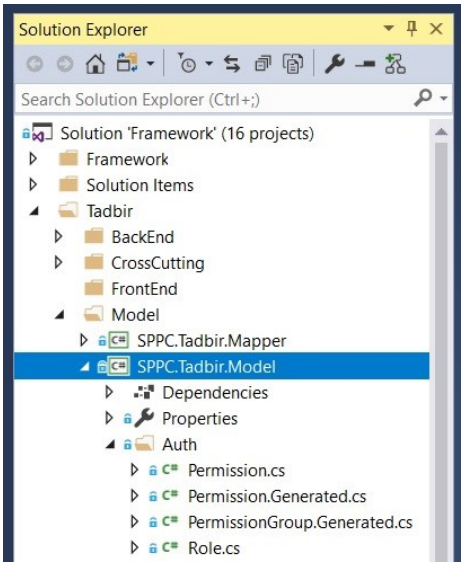


شکل ۸ – ساختار پروژه Metadata در پوشه تدبیر

پوشه FrontEnd – این پوشه برای سازماندهی واسط کاربری برنامه و پروژه های کمکی آن استفاده می شود. در ابتدای انجام پروژه سیستم جدید تدبیر، قابلیت های زیرساخت و سرویس وب اصلی در طول پیشبرد پروژه، در یک برنامه تحت وب نمایشی و با استفاده از فناوری ASP.NET MVC مورد استفاده قرار می گرفتند. پس از شروع به کار تیم جداگانه برای طراحی و پیاده سازی واسط کاربری، این بخش در پروژه از ساختار سورس های اصلی جدا شده و به یک فضای کاری مستقل انتقال پیدا کرد. در نتیجه این تغییر، ممکن است این پوشه از ساختار سورس های solution فعلی حذف شود.

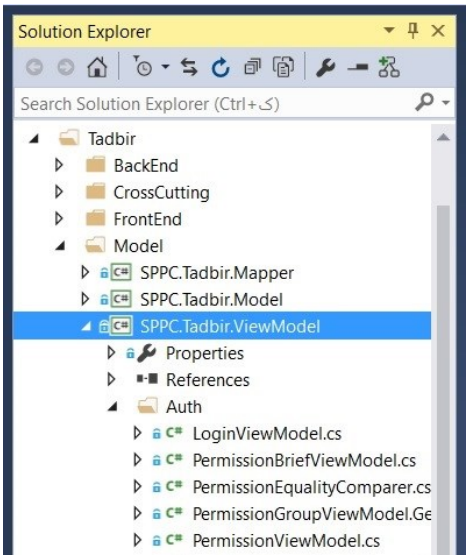
پوشه Model – پروژه های درون این پوشه برای کار با مدل شیئ گرای برنامه استفاده می شوند. با توجه به معماری پروژه که در بخش های قبل بررسی شد، کلاس های مرتبط با Model، View Model و Domain Mapper در پروژه های درون این پوشه قرار می گیرند. این پروژه ها عبارتند از :

۱. پروژه SPPC.Tadbir.Model : کلاس های درون این پروژه قابلیت های بخش Model در معماری اصلی را پیاده سازی می کنند. با توجه به استفاده از Schema ها برای طبقه بندی مفهومی جداول در دیتابیس اصلی برنامه، کلاس های موجودیت نیز درون این پروژه به صورت مشابهی طبقه بندی می شوند. کلاس های پیاده سازی شده در این پروژه با استفاده از یک ابزار شخصی و متادیتای ایجاد شده در این ابزار، تولید (Generate) شده اند و ساختار اطلاعاتی موجود در آنها همواره بر مبنای آخرین تغییرات در ساختار دیتابیس فیزیکی، به روزرسانی می شوند. بخشی از ساختار کلاس های این پروژه را در شکل ۱۰ می بینید :



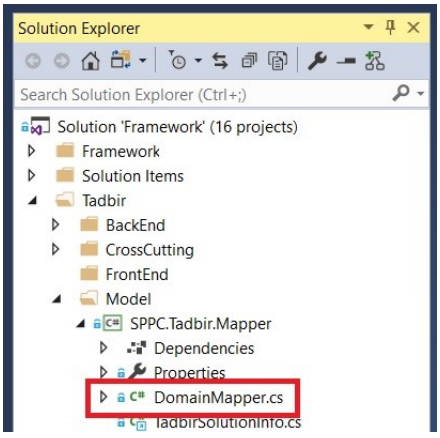
شکل ۱۰ – ساختار پروژه مدل شیئ گرا در پوشه تدبیر

۲. پروژه SPPC.Tadbir.ViewModel : کلاس های درون این پروژه قابلیت های بخش View Model در معماری اصلی را پیاده سازی می کنند. طبقه بندی این کلاس ها نیز بر مبنای Schema های دیتابیس انجام شده است. بخشی از ساختار کلاس های این پروژه را در شکل ۱۱ می بینید :



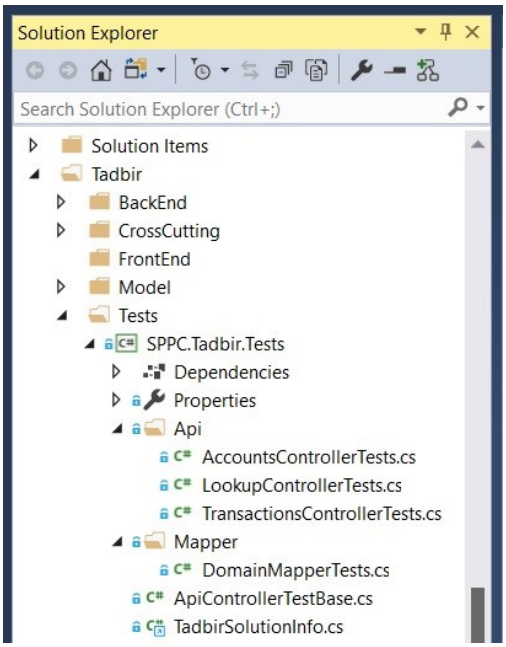
شکل ۱۱ – ساختار کلاس های View Model در پوشه تدبیر

۳. پروژه SPPC.Tadbir.Mapper : کلاس های درون این پروژه قابلیت های بخش Domain Mapper در معماری اصلی را پیاده سازی می کنند. این بخش شامل یک کلاس DomainMapper است که فقط در پروژه Persistence برای تبدیل کلاس های Model به View Model و بالعکس استفاده می شود. تبدیل های پیاده سازی شده در این کلاس با گذشت زمان و پیشرفت پروژه مفصل و طولانی می شوند و ممکن است مطالعه سورس را مشکل کند. بنابراین برای سازماندهی بهتر پیاده سازی کلاس، می توان تبدیل های مربوط به Schema های مختلف را در فایل های جداگانه قرار داد. ساختار کامل این پروژه را در شکل ۱۲ می بینید :



شکل ۱۲ – ساختار کامل پروژه Domain Mapper در پوشه تدبیر

پوشه Tests و پروژه SPPC.Tadbir.Tests : کلاس های این پروژه تست های واحد (Unit Test) در بخش های مختلف را پیاده سازی می کنند. با توجه با اولویت نسبتا پایین برای تهیه این تست ها، در حال حاضر تمام تست های نمونه پیاده سازی شده در یک پروژه قرار داده شده اند. تفکیک این تست ها در پروژه های متعدد (مثلا یک پروژه برای تست های هر بخش معماری) حین پیشرفت پروژه امکان پذیر است. همچنین انواع دیگر تست، مانند تست های یکپارچگی (Integration Test) و تست های پذیرش (Acceptance Test)، را می توان در مراحل بعدی در پروژه های جداگانه پیاده سازی کرد. ساختار کامل کلاس های درون این پروژه را در شکل ۱۳ می بینید :



شکل ۱۳ – ساختار پروژه تست های واحد در پوشه تدبیر

تاریخچه تغییرات

ردیف	نسخه	تهیه کننده	تغییرات
۱	۱,۰	بابک اسلامیه	نسخه اولیه (پس از چند مرحله ویرایش و تکمیل)
۲			
۳			
۴			