به نام خدا

دانشگاه پیام نور استان تهران

مرکز / واحد تهران شمال

گروه فنی مهندسی

پروژه کارشناسی

رشته مهندسی کامپیوتر

گرایش نرم افزار

عنوان پروژه:

پروژه دفترچه تلفن در وب

استاد راهنما:

استاد علی رضوي ابراهیمی

تهیه کننده:

زینب جلیلوند(935121562)

خرداد1400

کلیه حقوق مادي مرتبط برنتایج مطالعات، ابتکارات و

نوآوري هاي ناشی از این پروژه متعلق به:

"دانشگاه پیام نور استان تهران مرکز تهران شمال"

می باشد.

چکیده:

پروژه دفترچه تلفن با استفاده از جدیدترین تکنولوژي هاي مایکروسافت در زمینه برنامه نویسی وب و با پیاده سازي تمامی مفاهیم اصلی یک پروژه، امکان استفاده به عنوان پروژه آموزشی براي دانشجویان نرم افزار میتواند مورد استفاده قرار گیرد .

پیاده سازي تمامی مفاهیم و ارتباط برنامه با دیتابیس از جدیدترین تکنولوژي هاي مایکروسافت که به صورت منبع باز) **opensource**(موجود هستند، استفاده شده است که در ادامه در فصل هاي مختلف هر کدام از تکنولوژي ها و مفاهیم را بررسی خواهیم کرد.

فهرست

فصل اول:

مقدمه:

هدف اصلی در این پروژه استفاده از جدیدترین و به روز ترین تکنولوژي هاي مایکروسافت در زمینه برنامه نویسی وب براي پیاده سازي مفاهیم اصلی اي که در هر پروژه برنامه نویسی باید وجود داشته باشد است .**Framework** استفاده شده در پروژه

Net Core 2.1. است .**Framework** وب مورد استفاده **Asp.Net Core** است و همچنین در سطح بالاتر

از **Asp.NetCore MVC** استفاده شده است و **ORM** مورد استفاده در پروژه **2.1.0**

EntityFramework Core است.

هدف کلی:

هدف کلی از انجام این پروژه، استفاده از تکنولوژي هاي مایکروسافت براي پیاده سازي یک پروژه وب از صفر تا صد به صورتی که تمامی مفاهیم اصلی یک پروژه اعم از معماري و اصول **SOLID**، لایه بندي پروژه ،**ORM** و ارتباط برنامه با دیتابیس و ... به صورتی پیاده سازي شده باشد که بتواند براي آموزش پروژه محور مورد استفاده گردد. به همین علت برنامه دفترچه تلفن پیاده ساري شده است که قابل درک براي همه باشد و دامنه وسیعی نداشته باشد.

به صورت کلی براي انجام یک پروژه برنامه نویسی در وب از صفر تا صد، به کد نویسی در دو بخش مختلف نیاز داریم .

یک بخش برنامه نویسی سمت کاربر) **Client Side**(است که تقریبا با نام **FrontEnd** معروف است. در این بخش از برنامه نویسی با استفاده از زبان **HTML** و **Css** و **Java Script** و یا استفاده از **Library** هاي مختلف مانند **Bootstrap**، برنامه نویسی سمت کاربر انجام میشود و شامل قسمتی است که کاربر در مرورگر خود هنگام کار با سیستم مشاهده میکند. در این پروژه تمرکز اصلی پروژه بر روي این قسمت از برنامه نویسی نیست و قالب طراحی شده پنل ادمین با استفاده از یک قالب آماده از سایت **SBAdmin 2** استفاده شده است. قالب طراحی شده سایت دانلود شده و شخصی سازي شده تا در این پروژه مورد استفاده قرار گیرد. همچنین در یک قسمتی از سایت براي انتخاب تگ هاي مرتبط با هر کاربر از قطعه کدي با عنوان **Select 2** استفاده شده است.

بخش دیگر، برنامه نویسی سمت سرور (**Server Side**) است که به بخش **BackEnd** معروف است. در این بخش با استفاده از یک زبان برنامه نویسی مانند سی شارپ و همچنین یک **Framework** توسعه وب، کد نویسی انجام میشود تا بتوان درخواست هاي وارد شده از سمت کاربر را دریافت کرد، پردازش هاي مورد نیاز را انجام داد و همچنین پاسخ مورد نیاز کاربر را در قالب یک ساختار وب به کاربر نمایش داد. تمرکز اصلی این پروژه در قسمت **BackEnd** و همین قسمت کد نویسی است. زبان برنامه نویسی مورد استفاده نیز سی شارپ است. در قسمت **BackEnd** پروژه مفاهیم بسیار زیادي مورد بحث و استفاده قرار میگیرد که در فصل هاي بعدي به تشریح آنها خواهیم پرداخت.

براي انجام این پروژه سعی شد تا از استاندارد ترین و به روز ترین معماري ها و تکنولوژي ها استفاده شود تا پروژه واقعا جنبه آموزشی داشته باشد. یعنی از تکنولوژي ها و معماري هایی استفاده شده است که در همه جاي دنیا اکثرا مورد تایید است و همچنین به روز است. به عنوان نمونه معماري مورد استفاده در پروژه **Onion Architecture** است.

در سال هاي گذشته قالبا معماري سه لایه مورد تایید و استفاده قرار میگرفت که معماري سه لایه به این گونه بود که برنامه به سه بخش جدا تقسیم میشود. عنوان بخش اول **Data Access Layer** بود که در این بخش ارتباط برنامه با دیتابیس صورت میگرفت و عملیات **CRUD** (**Create, Read, Update, Delete**) روي دیتابیس انجام میگرفت و پایین ترین لایه برنامه بود. این گونه ارتباط برنامه از دیتابیس از قسمت هاي دیگر برنامه جدا میشد. همچنین این لایه شامل کلاس هاي دامنه و موجودیت هاي پروژه نیز بود که به ازاي آنها در دیتابیس یک جدول وجود دارد.

عنوان لایه بالاتر **Business Logic Layer** بود که تمام منطق برنامه در این لایه پیاده سازي میشد. این لایه واسط بین لایه بالایی و لایه **Data Access** بود که درخواست وارده از لایه بالایی را دریافت میکرد و منطق کاري متناسب با درخواست را بر روي درخواست آمده انجام میداد و با همکاري با لایه **Data Access** عملیات مورد نظر را انجام میداد و پاسخ مورد نظر را به درخواست دهنده برمیگرداند. به طور کلی تمام منطق پروژه در این لایه پیاده سازي میشد.

بالاترین لایه در این معماري **Presentation Layer** است. این لایه وظیفه ارتباط با کاربر را برعهده داشت.

درخواست هاي کاربر را دریافت میکرد و با داده هاي مورد نیاز به لایه **Business** واگذار میگرد و پاسخ را دریافت میکرد و به کاربر نمایش میداد.

زمانی که این معماري مطرح شد مورد استقبال برنامه نویسان قرار گرفت زیرا بخش هاي مختلف هر برنامه را از یکدیگر جدا میکرد و وابستگی لایه ها به هم مشخص بود و درهم تنیدگی کد را کمتر میکرد. با وجود این معماري وقتی که نیاز داشتیم که بعضی از منطق کاري برنامه را تغییر دهیم، فقط تغییرات مورد نیاز را در لایه **Business Logic** انجام میدادیم.

اما در این معماري به شدت وابستگی به لایه **Data Access** وجود داشت. پس از پیشرفت تکنولوژي ها و گذر زمان کم کم استفاده از این معماري کمتر شد و معماري هاي دیگر جایگزین شد. یکی از این معماري ها ، **Onion Architecture** بود که در این پروژه مورد استفاده قرار گرفته است. در این معماري وابستگی برنامه به دیتابیس و ارتباط برنامه با دیتابیس از بین رفته است به این گونه که حتی بدون ارتباط برنامه برنامه با دیتابیس هم پروژه بالا می آید.

همچنین براي پیاده سازي مفاهیم اصلی **AAA**)**Authentication, Authorization, Accounting**(از امکانات خود مایکروسافت به نام **Asp.Net Identity** استفاده شده است که امکانات زیادي به ما میدهد و خود دیتابیس مورد نیاز براي **AAA** را ایجاد میکند.

در تمامی بخش ها سعی شده است که از تکنولوژي هایی مورد استفاده قرار گیرد که هم بسیار کارآمد باشد و همچنین پر استفاده در ایران و دنیا باشد.

جمع بندي:

به طور کلی این پروژه براي پیاده سازي مفاهیم به طور کلی انجام شده است. همچنین براي پیاده سازي مفاهیم و تکنولوژي ها سعی بر آن بود که طوري پیاده سازي انجام گیرد که جنبه آموزشی بالایی داشته باشد که در فصل بعد بررسی خواهیم کرد.

فصل دوم:

مقدمه:

آموزش پروژه محور همواره مورد بحث بوده است. عده اي اين نوع آموزش را خوب می دانند و عده اي ديگر در اين روش ضعف هايی بيان ميكنند. به نوعی هر دو گروه دلايل قانع كننده اي براي اثبات صحبت هاي خود دارند. اما دليلی كه باعث شد تا من اين اين پروژه آموزشی را انجام دهم، اين بود كه واقعا با يادگيري صرفا زبان برنامه نويسی، انجام كار برناممه نويسی واقعا كار دشواري است. بايد با پيچيدگی هاي پروژه بيشتر درگير شد كه در ادامه فصل به اين موضوع خواهيم پرداخت.

هدف كلی:

آموزش پروژه محور هميشه مورد بحث بوده است و پروژه هاي زيادي در اين زمينه انجام شده است و در قالب كلاس هاي آموزشی موسسه هاي آموزشگاهی برنامه نويسی و تا حدودي كمتر در دانشگاه ها هم انجام ميشود. در دانشگاه بيشتر مفاهيم پايه و مهم علم نرم افزار بيان ميشود و برنامه نويسی خيلی با جزئيات كامل مورد بررسی قرار نميگيرد. در اينجا سعی شده است تا بيشتر با جزئيات و پيچيدگی هاي قسمت هاي مختلف پروژه درگير شد. از شروع تا پايان و پابليش كردن پروژه.

به عنوان مثال سايت **StackOverflow** كه يک سايت بسيار محبوب و پر استفاده در بين جامعه نرم افزار و برنامه نويسی در دنيا است، در سال 1602 چيزي نزديک 1 ميليارد بازديد داشته است. اگر اين سايت از **ORM** هاي مطرح دنيا استفاده ميكرد ،با همان تجهيزات سخت افزاري چيزي نزديک 02 روز براي پردازش درخواست هاي كاربران در سمت ديتابيس كم می اورد .

از اين رو خود توسعه دهندگان تيم **StackOverflow** بر آن شدند تا **ORM** اختصاصی خود را توسعه دهند و آنرا با نام **Dapper** منتشر كنند كه از نظر سرعت اجراي درخواست ها از 46 تا 96 درصد سريعتر نسبت به **ORM** هاي ديگر كار ميكند ولی خب سرعت توسعه كمتري نسبت به **ORM** هاي ديگر دارد.

در آموزش هاي دانشگاه غالبا دانشجويانی كه علاقه به برنامه نويسی دارند با اين پيچيدگی ها و دغدغه ها رو به رو نميشوند. از اين رو بعد از فارغ التحصيلی به راحتی نميتوانند وارد بازار كار شوند و غالبا در مصاحبه ها رد ميشوند. به عنوان مثال اصول **SOLID** را نميشناسند كه در خيلی از مصاحبه ها از اين اصول سوال پرسيده ميشود. يا با **DI Container** ها آشنايی ندارند.

همچنین با **Design Pattern** ها و **Anti Pattern** ها و **Best Practice** ها در برنامه نویسی شناختی ندارند .

در این پروژه سعی شده تا بیشتر مفاهیم کلیدي و مهم و مورد سوال در مصاحبه هاي استخدام کاري پیاده سازي شود تا آشنایی با این مفاهیم پیدا کنند تا بتوانند سوال هاي استخدامی مصاحبه ها را پاسخ دهند و وارد بازار کاري شوند.

همچنین دانشجویان باید سعی داشته باشند که فقط مفاهیم کار نکنند، چون مفاهیم بدون تمرین پیاده سازي آنها خیلی کاربرد خاصی براي اشخاص ندارد چون بین پیاده سازي و مفاهیم کلی فرق وجود دارد و باید اشخاص بتوانند در پیاده سازي نیز موفق باشند و بتوانند مفاهیم تئوري را به خوبی پیاده سازي کنند.

همچنین خیلی خوب است تا دانشجویان روند و دلیل تغییرات تکنولوژي هاي مختلف را بدانند. یعنی اینکه چرا یک تکنولوزي مطرح شد، مورد استقبال و استفاده قرار گرفت و سرانجام بعد از تایمی پاسخگوي تمامی نیاز ها نبود و از بین رفت. به عنوان مثال بعد از مطرح شدن مبحث برنامه نویسی وب، ابتدا ماکروسافت تکنولوژي اي براي برنامه نویسی وب نداشت و داشت بازار را از دست میداد و همه برنامه نویسان سمت جاوا و پلتفرم هاي مختلف دیگر میرفتند. از این رو ماکروسافت تکنولوزي **asp.net web forms** را مطرح کرد تا برنامه نویسان **Windows Forms** بتوانند با کمترین درگیري براي یادگیري مباحث زیاد، از برنامه نویسی ویندوز به برنامه نویسی وب کوچ کنند. اما بعد از مدتی و با رشد بیشتر و سریعتر تکنولوزي هاي مختلف و همچنین رشد خیلی زیاد برنامه نویسی **asp.net web forms**، **Front end** دیگر پاسخگوي تمامی نیاز هاي برنامه نویسان نبود و ماکروسافت مجددا داشت بازار رقابتی را از دست میداد. از همین براي براورده کردن نیازمندي هاي برنامه نویسان، از **Asp.net MVC** رونمایی کرد که باعث شد تا مجددا برنامه نویسان را به سمت خود جذب کند. بعد از عوض شدن مدیرعامل ماکروسافت، آقاي **Satya Nadella**، کل **net Framework.** را منبع باز کرد و همچنین **net Core.**

را نیز به صورت منبع باز منتشر کرد. در ادامه براي پاسخگویی نیازمندي هاي **Asp.net Core MVC**، **net.**

Core را نیز معرفی نمود که تغییري بزرگ در ماکروسافت است. پس از منبع باز شدن کد هاي فریم وورک، ماکروسافت با سرعت خیلی بیشتري رشد کرد چون حالا تعداد محدود برنامه نویسان خود ماکروسافت نبودند که توسعه میدادند بلکه تمامی برنامه نویسان دنیا در توسعه بیشتر ماکروسافت کمک کردند که سرعت رشد آنرا چند برابر کرد.

نتيجه گيري:

در اين پروژه به طور كلی سعی شده است تا اكثر مفاهيمی پايه اي و مهمی كه غالبا در مصاحبه هاي استخدامی سوال ميشود ،پياده سازي شود تا بيشتر با پيچيدگی هاي پروژه و برنامه نويسی واقعی اشنا شد و باعث رشد برنامه نويسی دانشجويان شد.

در فصل بعد تا حدودي تكنولوژي هاي مورد استفاده بيان ميگرد

فصل سوم:

مقدمه:

در این بخش به بررسی تکنولوژي ها و ابزار هاي استفاده شده در پروژه و مفاهیم اصلی به کار برده شده در پروژه میپردازیم.

چوا این پروژه خیلی بزرگ نیست، برخی از مفاهیم در آن پیاده سازي شده است که به تشریح آنها میپردازیم.

هدف کلی:

ابتدا با معماري استفاده شده در پروژه شروع میکنیم. معماري و لایه بندي کد ها باعث خوانایی بیشتر کد و توسعه آسان تر و

رعایت **Separation of Concern** و ... میشود. در گذشته از معماري سه لایه معروف استفاده میشد که کمی درباره آن صحبت کردیم. در معماري سه لایه وابستگی ما به دیتابیس بسیار زیاد است، به گونه اي که بدون دیتابیس برنامه ما هرگز کار نخواهد کرد. از آنجایی که دیتابیس مورد استفاده و نحوه اتصال برنامه به دیتابیس یک تکنولوژي است و تکنولوژي ها تغییر میکنند و تکنولوژي هاي جدید جایگزین تکنولوژي هاي قدیمی تر میشوند، پس دیتابیس و نحوه اتصال برنامه به دیتابیس نیز ممکن است در طول زمان تغییر کند. به همین خاطر یک معماري دیگري پیشنهاد شد که وابستگی برنامه را از دیتابیس به سمت بیزینس و منطق برنامه هدایت کرد. چون بیزینس و منطق برنامه همیشه وجود دارد و ربطی به تکنولوژي ندارد. معماري مطرح شده ، **Onion Architecture** نام دارد. در این معماري پایین ترین لایه در برنامه، لایه **Domain** است که دامنه اصلی برنامه در آن وجود دارد. عموما هر نیازمندي اي که براي پیاده سازي نیاز داریم با عنوان **Infrastructure**، بالاتر از لایه **Domain** قرار میگیرد. در لایه بالاتر سرویس ها وجود دارند که درخواست هاي کاربران با استفاده از لایه **Domain** پردازش میکنند. عموما هر نیازمندي اي که براي پیاده سازي نیاز داریم با عنوان **Infrastructure**، بالاتر از لایه **Domain** قرار میگیرد. سپس در لایه بالاتر **Data Access** را داریم که وظیفه ارتباط برنامه را با دیتابیس بر عهده دارد. همچنین در بالاترین لایه نیز **End Point** ما قرار دارد که همان رابط کاربري است که کاربر میتواند به کمک آن از برنامه استفاده کند.

در این پروژه **End Point** ما **Asp.Net Core MVC** است و از دیتابیس **Sql Server** و **ORM** **EntityFrameworkCore** و رویکرد **Code First** استفاده شده است .**Di Container** استفاده شده در پروژه، خود **Di Container** موجود در **Asp.Net Core MVC** است. قالب استفاده شده براي **End Point**، قالب آماده موجود در اینترنت با نام **SB2 Admin** است که به صورت رایگان در اختیار استفاده کنندگان قرار می گیرد. همچنین در **UI** در قسمت انتخاب تگ ها از **Select 2** نیز استفاده شده است. همچنین براي **AAA** (**Authentication , Authorization , Accounting**(از **Asp.Net Identity** خود ماکروسافت استفاده شده است. همچنین از **Repository Pattern** نیز استفاده شده است.

در ادامه به بررسی بیشتر برنامه می پردازیم و درباره لایه هاي متفاوت و **ORM** با جزئیات بیشتري صحبت خواهیم کرد.

قبل از شروع بررسی بیشتر لایه هاي برنامه، به بررسی مختصر **Dipendency Injection** که یکی از اصول **Solid** هست میپردازیم. در برنامه نویسی ما سعی داریم برنامه اي بدون وابستگی بنویسیم که همچین چیزي غیر ممکن است. بنابراین تلاش میکنیم وابستگی را به گونه اي در کد مدیریت کنیم که کمترین مشکل را بوجود بیاورد. به همین منظور وابستگی هاي خود را از پیاده سازي به سطح **Abstraction** نزدیک میکنیم. یعنی به جاي اینکه به پیاده سازي متد ها در کلاس ها وابسته باشیم و مستقیما از آنها استفاده کنیم، وابستگی خود را به سطح اینترفیس ها نزدیک میکنیم. به این گونه که هنگام استفاده از متدي، از اینترفیس متد استفاده میکنیم سپس در یک جایی از برنامه، مشخص میکنیم که هنگام ساخت شی جدید از اینترفیس پیاده سازي از کدام کلاسی که آن اینترفیس را پیاده سازي کرده است، مورد استفاده قرار گیرد که این کار مشخص نمودن وابستگی هاي در **Di Container** انجام میشود. حال هر زمان که نیاز داشته باشیم تا پیاده سازي جدید از اینترفیس داشته باشیم،

فقط کافی است در **Di Container** برنامه، ایجاد ساخت شی از کلاس مورد نظر جدیدي که اینترفیس را پیاده سازي میکند استفاده کرد. این مختصري از **Dependenci Injection** بود.

در لایه **Domain**، در قسمت **Core** کلاس هاي دامنه اصلی خود را داریم که پایین ترین لایه ما را تشکیل میدهند. همچنین در قسمت **Contarct** تمامی اینترفیس هاي مورد نیاز در تمامی قسمت هاي برنامه براي انجام عملیات مختلف از قبیل ایجاد اینترفیس **Repository** ها یا **UnitOfWork** یا پیاده سازي لاجیک و منطق برنامه و ... در این قسمت تعریف میشود. در این لایه کلاس هاي موجودیتی با عنوان **Data Transfer Object** (**DTO**) نیز تعریف میشود. **DTO** ها فقط وظیفه انتقال آبجکت ها به لایه هاي بالاتر را دارند براي اینکه وابستگی ها در پیاده سازي به حداقل برسد، کلاس موجودیت را به همان شکل در تمامی برنامه استفاده نمیکنیم، بلکه از کلاس هایی با همان نام کلاس موجودیت و اضافه شدن پسوند **DTO**، آبجکت ها را به یکدیگر **map** میکنیم و به **End Point** خود منتقل میکنیم. در **End Point** هم براي هر عملیات مورد نیاز، یک کلاس مدل با نام همان موجودیت کلاس با اضافه شدن پسوند **ViewModel** از آبجکت ها استفاده میکنیم. به طور کلی در طول یک درخواست تا پاسخ برگردد، ابتدا در قالب کلاس **View Model** ورودي هاي کاربر را به لایه پایین تر انتقال میدهیم و آنرا به کلاس هاي **DTO** نگاشت میکنیم و **DTO** ها را به کلاس موجودیت اصلی خود نگاشت میکنیم و در دیتابیس ذخیره میکنیم یا هنگام خواندن از دیتابیس در قالب کلاس هاي موجودیت اصلی از دیتابیس داده واکشی میکنیم و در لایه بالاتر به **DTO** ها نگاشت میکنیم و در **UI** خود نیز کلاس هاي **DTO** را به **View Model** ها نگاشت میکنیم و براي استفاده کاربر قرار میدهیم تا کمترین وابستگی را در لایه هاي مختلف داشته باشیم.

در لایه بالاتر که لایه **Service** نام دارد، اینترفیس هاي تعریف شده در **Contract** که مربوط به منطق برنامه هستند را بهعنوان **Application Service** پیاده سازي میکنیم که در **End Point** ما، براي پردازش درخواست هاي ورودي درکنترلر، اینترفیس هاي **Service** خود را فراخوانی میکنیم که با استفاده از پیاده سازي انجام شده در لایه سرویس، پاسخ را آماده کرده و به کنترلر برمیگرداند و از آنجا هم پاسخ مورد نیاز کاربر نمایش داده میشود.

لایه بالاتر **Data Access** است که در این لایه از برنامه به دیتابیس خود متصل خواهیم شد و عملیات **CRUD** خود را با استفاده از این لایه بر روي دیتابیس خود انجام میدهیم. در این لایه **ORM** نیز قرار دارد که بوسیله آن به دیتابیس وصل خواهیم شد. همانطور که گفته شد رویکرد استفاده شده در پروژه **Code First** است که با استفاده از کلاس هاي موجودیت هاي دامنه اصلی ما، دیتابیس ما ساخته خواهد شد و عملا درگیر ساخت دیتابیس نخواهیم شد. براي اینکار باید یک کلاس براي ارتباط برنامه با دیتابیس ایجاد کنیم که از کلاس **DbContext** ارث بري میکند و در آن پراپرتی هاي از جنس **<DbSet<T>** که **T** همان کلاس هاي موجودیت ما هستند ایجاد کنیم. به ازاي هر پراپرتی از جنس کلاس دامنه ما در **Context**، در دیتابیس ما یک جدول با آن نام ایجاد خواهد کرد. همچنین براي پیاده سازي روابط بین موجودیت ها میتوان از **Navigation Property** ها استفاده نمود و یا کلاس هاي واسطی براي پیاده سازي انواع روابط یک به یک و یک به چند و چند به چند ایجاد کرد. همچنین **Connection String** خود را ست میکنیم. پس از ایجاد زیر ساخت هاي لازم ،با استفاده از **Package Manager Console**، میتوان **Migration** زد تا دیتابیس ما از روي موجودیت هاي ما ایجاد گردد. همچنین براي به روز رسانی جدول هاي دیتابیس نیز، ابتدا باید کلاس هاي دامنه خود را به روز رسانی کرد یا اگر جدول جدیدي نیاز داریم، پراپرتی **<DbSet<T>** آنرا به **Context** اضافه نمود و مجددا **Migration** زد تا دیتابیس به روز رسانی گردد. دستور هاي **Migration** نیز به شرح زیر است:

ابتدا باید در پنجره **Package Manager Console** پروژه اي که کلاس هاي **Context** در آن قرار دارند را انتخاب کرد، سپس ابتدا با دستور **add-migration migrationName -ContextName**، یک **migration** جدید ایجاد کرد و سپس با نوشتن دستور **Update_database –ContextName**، دیتابیس خود را به روز رسانی کرد. اگر در یک پروژه بیش از یک کلاس **Context** دارید، باید براي **–ContextName** نیز نام کانتکس خود را بنویسید در غیر این صورت نیازي به **–ContextName** نیست.

EF به طور پیش فرض در هر کلاس، اگر پراپرتی اي با نام **Id** یا **Id+** نام کلاس وجود داشته باشد، آنرا به عنوان کلید اصلی در نظر میگیرد و کلید خارجی ها را با توجه به روابط تعریف شده در کلاس هاي موجودیت ها مشخص میکند. در صورتی که نیاز داشته باشیم تا پیکربندي خاصی را براي جدول هاي خود در نظر بگیریم، مانند کلید اصلی یا کلید خارجی خاصی داشته باشیم یا اینکه محدودیت خاصی یا دیتا تایپ خاصی را براي پراپرتی ها مشخص کنیم، میتوانیم از دو روش **Attribute**

Base یا **Fluent API** استفاده کنیم. در روش **Attribute Base**، بالاي هر پراپرتی که نیاز به پیکربندي آن داریم ،**Attribute** مورد نیاز با مقدار مورد نیاز را قرار میدهیم. در روش **Fluent API** کلاسی براي هر کلاس موجودیتی که نیاز به پیکربندي آنرا داریم، ایجاد میکنیم و از اینترفیس **IEntityTypeConfiguration<T>** ارث بري میکنیم و در این کلاس تابع **Configure** اینترفیس را پیاده سازي میکنیم . براي پراپراتی ها با استفاده از متد هاي **EF** پیکربندي مورد نیاز خود را انجام میدهیم.

همچنین در این لایه پیاده سازي اینترفیس هاي **Repository** خود را نیز انجام میدهیم که در کلاس هاي **Repository** خود متد هاي اتصال برنامه به دیتابیس را پیاده سازي میکنیم. عملا **Repository** ها نقش یک لایه میانجی بین دیتا مدل ها و دامین مدل هاي ما را دارند که این ارتباط را مدیریت میکنند و اجازه نمیدهند که از هر قسمت برنامه که برنامه نویس نیاز داشت مستقیما به دیتابیس وصل شود و براي اتصال به دیتابیس تنها راه ارتباطی همین **Repository** هاي ما هستند. به ازاي هر موجودیت نیز یک کلاس **Repository** داریم که وظیفه ارتباط با دیتابیس براي آن موجودیت را بر عهده دارد.

به طور کلی توصیه با دانشجویانی که قصد یادگیري کار با **ORM** ها را دارند، این است که ابتدا **ADO.Net** یاد بگیرند و با **Ado** برنامه خود را به دیتابیس وصل کنند تا با پیچیدگی ها و سختی هاي کار با **Ado** آشنا شوند تا در ادامه بتوانند درک بهتري از **ORM** ها و هزینه هاي که هنگام کار با **ORM** ها را متحمل میشوند بیشتر درک کنند. یادگیري نحوه کار داخلی **ORM** خوب است چون دید عمیق تري نسبت به **ORM** ها میتوان پیدا کرد و سپس درک بهتري از میکرو **ORM** ها مانند **Dapper** میتوان پیدا کرد.

بالاترین لایه برنامه نیز **End Point** ما است که کاربر از آن استفاده میکند. قالب آماده استفاده شده **SB 2 Admin** است که میتوان رایگان آنرا از سایت **SB Admin 2** دانلود کرد و استفاده کرد که در این قسمت باید قسمت هاي مشترک ویو را جدا کرد تا **Layout** را ایجاد کرد تا کد نویسی کمتري داشته باشیم که خیلی در این قسمت وارد جزئیات برنامه نویسی سمت کاربر نمیشویم.

در اینجا از **Asp.Net Core MVC** استفاده شده است که معماري **End Point** ما است. معماري **Model , View , Controller**(که پیش تر درباره آن صحبت کردیم، فقط معماري لایه **End Point** ما است.

در این لایه **AAA** وجود دارد که همانند لایه **Data Access** برنامه، یک کلاس **Context** دارد تا اطلاعات کاربران و میزان دسترسی آنها را ذخیره کرد. در کنترلر ها، بالاي هر کنترلر یک **Attribute** مشخصی قرار میگیرد تا میزان دسترسی کاربران به بخش هاي مختلف مشخص گردد. به عنوان مثال بالاي کنترلري که [**Authorize**(**Roles** = "**admin**")] این **Attribute** قرار داشته باشد، تنها کاربرانی که نقش **admin** دارند قادر خواهند بود تا از متد هاي این کنترلر استفاده کتند .همه این قابلیت ها به صورت تو کار در **Asp.Net Identity** تعریف شده است و برنامه نویس هاي زیادي در پروژه هایخود از آن استفاده میکنند. همچنین در کلاس **Startup** پروژه نیز وابستگی هاي خود را معرفی نموده و همانطور که درقسمت بررسی **Di Container** گفته شد، از **Di Container** خود **Net Core**. استفاده میکنیم که در کلاس **Startup** مشخص نمودیم که براي هر اینترفیس تعریف شده، پیاده سازي از نوع کدام کلاس استفاده شود. کلاس هاي **View Model** ما هم در **Model** قرار میگیرد.

نتیجه گیري:

در این فصل به صورت کوتاه تکنولوژي ها و معماري استفاده شده مورد بررسی قرار گرفت و در کد پروژه نیز از هر قسمت و بخش یک نمونه ایجاد گردیده است. به طور کلی دنیاي برنامه نویسی بسیار وسیع است و این قسمت تنها بخشی از بسیار کوچکی از این دنیا است. تکنولوژي هاي استفاده شده در این پروژه تقریبا به روز هستند اما امروزه دنیاي برنامه نویسی به سمت **Polyglot Programming** پیش میرود یعنی در پروژه تنها با استفاده از یک زبان و یک فریم وورک و یک دیتابیس پروژه ها انجام نمیشود، بلکه سمت معماري سرویس گرا پیش میرود که در آن میکروسرویس هایی وجود دارند که میتوانند با هر زبان و فریم وورکی کد نویسی شده و به صورت **json** با میکروسرویس هاي دیگر صحبت کنند. به عنوان مثال زبان برنامه نویسی پایتون یادگیري نسبتا آسان تري نسبت به سی شارپ دارد و در قسمت هایی که یک میکروسرویس مورد کوچکی مورد نیاز است میتوان با پایتون سریع تر و راحت تر کدنویسی کرد. یا به عنوان مثال **No Sql** ها(**Not Only Sql** پیشرفت زیادي داشتند و واقعا امروزه اینگونه نیست که در یک پروژه با یک دیتابیس رابطه مانند **Sql Server** پروژه خود را پیش ببریم بلکه استفاده زیادي از **Redis** که یک **No Sql** محبوب است یا همینطور **Mongo Db** یا دیتابیس هاي **No Sql**

دیگر بیشتر شده است. از این رو برنامه نویسان نیاز دارند تا دانش خود را وسیع تر کنند اما خوبی **PolyglotProgramming** این است که در عوض برنامه نویسان نیاز ندارند تا تکنولوژي هاي دیگر را خیلی عمیق یاد بگیرند زیرا استفاده خیلی عمیقی از آنها نخواهند کرد.

من خیلی علاقه زیادي داشتم تا از **Rest API** ها نیز در این پروژه استفاده کنم تا توضیحی هم درباره آنها داشته باشیم که بسیار پر کاربرد هستند و پیچیدگی هاي شدید تر برنامه نویسی **WCF** را ندارند، تا مقایسه اي از **Rest API** ها **WCF** را نیز داشته باشیم که به دلیل کمبود وقت موفق به پیاده سازي این قسمت نشدم.

فصل چهارم:

مقدمه:

در فصل هاي قبل پيرامون پروژه و تكنولوژي هاي استفاده شده در آن صحبت كرديم. در اين فصل پيرامون تاثير پروژه حين آموزش برنامه نويسی كمی صحبت خواهيم كرد تا تاثير آنرا در آموزش برنامه نويسی ببينيم تا دانشجويان بتوانند در حين تحصيل آمادگی بيشتري براي ورود به بازار كار پيدا كنند.

هدف كلی:

اكثر درس هاي مرتبط با برنامه نويسی كه دانشجويان رشته نرم افزار در دانشگاه پاس ميكنند مربوط به حل مسئله و مباحث مهم و پايه علم نرم افزار است كه بسيار نيز خوب استو چون تكنولوژي ها اغلب عمر زيادي ندارند و از بين ميروند و تكنولوژي هاي جديدتر و بهتر جايگزين ميشوند. به همين خاطر بايد مباحث پايه را كامل ياد گرفت و اين نيز در دانشگاه ها اولويت قرار دارد.

مباحثی كه در اين پروژه با هم بررسی كرديم غالبا در تمامی شركت هايی كه نياز به برنامه نويس دارند استفاده ميشود. به همين خاطر يادگيري و تمرين همچين مباحثی باعث ميشود تا دانشجويان خيلی بهتر در مصاحبه هاي استخدامی به سوالات پاسخ بدهند و موفق شوند كار را بدست بياورند. تقريبا در تمامی شركت ها از **ORM** ها استفاده ميشود. اصول **Solid** رعايت ميشود و. ...

همچين مباحث بيشتري وجود دارد كه امكان توضيح آنها در اين گزارش وجود ندارد، مانند نحوه كار **ORM** **EntityFramework** كه بسيار مفيد است دانستن نحوه كار اين **ORM**، زيرا در توسعه ميتوان با دانش نحوه كار **ORM**، از نظر كارايی و سرعت پردازش درخواست ها كد نويسی بهتري انجام داد كه در غالب اين گزارش امكان توضيح آن وجود ندارد. كه در چنين حالتی در كلاس درسی امكان بررسی كامل اين موضوع ها وجود دارد.

بررسی همچين پروژه اي براي دانشجويان باعث ميشود تا ديد بهتري نسبت به پروژه هاي واقعی كه در حال حاضر وجود دارد ،پيدا كنند و بعد اينكه براي اولين كار برنامه نويسی درخواست دادند بتوانند از پس مصاحبه بربيايند و هنگام شروع اولين كار خود خيلی سردرگم در پروژه نباشند. من در دوران كارآموزي خود وقتی يک پروژه واقعی ديدم، حجم زياد كد ها را كهديدم كاملا سردرگم بودم. بعد از مطالعات شخصی بسياري كه داشتم توانستم ديد بهتري نسبت به پروژه ها داشته باشم .از اينرو به ذهنم آمد تا اين مباحث را در پروژه بررسی كنم و در فصل بعدي نيز پيشنهادي براي اين موضوع ارائه بدهم.

جمع بندي:

در اين فصل به طور خلاصه پيرامون مزاياي اين نوع آموزش صحبت كرديم و گفتيم كه آموزش پروژه محور ديد بهتري به دانشجويان ميدهد تا هم در مصاحبه كاري بتوانند پاسخ هاي بهتري بدهند و هم هنگام كار بتوانند راحت تر با پروژه اي كه روي آن كار ميكنند، توسعه بدهند. در فصل بعدي پيشنهادي براي اضافه شدن اين نوع آموزش به درس ها نيز ارائه ميدهم.

فصل پنجم:

در این فصل پیشنهادي براي اضافه شدن این نوع آموزش حداقل در دروس اختیاري یا در دروس عملی به درس هاي دانشجویان برنامه نویسی مطرح میکنیم.

از آنجا که برنامه نویسی امروزه بیشتر در غالب برنامه نویسی **BackEnd** و **FrontEnd** مطرح میشود، می توان یک درس عملی براي دانشجویان در نظر گرفت که به صورت دلخواه دانشجویان امکان انتخاب درس عملی **BackEnd** یا **FrontEnd** را داشته باشند که دانشجویان به هر کدام از زمینه ها که بیشتر علاقه دارند و قصد ادامه در آن زمینه را دارند، در همان زمینه این درس را پاس کنند. در این درس نیز با توجه به تکنولوژي هاي روز که بیشترین استفاده را دارند به دانشجویان آموزش داده شود تا دانشجویان بتوانند با نحوه انجام یک پروژه از صفر تا صد آشنا شوند. براي دانشجویان **FrontEnd** یک پروژه طراحی سایت به عنوان مثال و براي دانشجویان استفاده از یک قالب آماده براي رابط کاربري و انجام یک پروژه مانند همین پروژه انجام شده با جزئیات بیشتر و دقیق تر.

تکنولوژي هاي مورد استفاده هم میتوان از تکنولوژي هایی که کاربرد بیشتري دارند استفاده نمود تا دانشجویان بتوانند از تجربه این درس براي مصاحبه ها و استخدام استفاده کنند. همچنین در طول این درس میتوان به دانشجویان آموزش داد تا مصاحبه کاري ها را چگونه پاسخ دهند و چگونه از پس سوالات فنی بربیایند. اغلب در درس هاي ساختمان داده و طراحی الگوریتم و ... مباحث مهم و مفاهیم پایه و کاربردي علم نرم افزار بیان میشود اما در بعضی مصاحبه ها علاوه بر این ها اززز مفاهیمی که

بالا بیان شد نیز سوالات مختلفی پرسیده میشود مانند **Dependency Injection** از اصول **Solid** و ... که

دانشجویان با داشتن دانش کافی در این زمینه ها بتوانند راحت تر وارد بازار کار شده و پیشرفت کنند. امروزه نیروي برنامه نویس خوب در دنیا کمتر شده و از طرفی نیاز دنیا به برنامه نویسان روز به روز رشد میکند تا این حد که براي تمامی نیاز هاي مردم اپلیکیشن هایی توسعه داده میشود تا بتوانند نیاز هاي خود را به صورت اینترنتی تامین کنند. یا با پیشرفت علم اینترنت اشیا) **IOT**(، روز به روز نیاز دنیا به برنامه نویسان بیشتر خواهد شد و زمینه خوبی براي کار کردن و درآمد هاي خوب است. وقتی استفاده از اینترنت اشیا بیشتر شود و قدرت این علم بیشتر به نمایش گذاشته شود، قاعدتا دنیا بیشتر از این علم استفاده خواهد کرد. ممموضوع دیگري که درباره برنامه نویسی مورد توجه خود من قرار گرفته است، این است که در دنیا انتخاب مدیران در شرکتهاي مختلف براي بخش هاي

مختلف غالبا از بین کسانی که با علم نرم افزار آشنایی زیادي دارند، است. زیرا این اشخاص بانحوه پیشبرد پروژه ها و چگونگی انجام پروژه آشنایی زیادي دارند.

به طور کلی هرچه بهتر دانشجویان براي وارد شدن به بازار کار تربیت شوند، صد در صد آمار بیکاري کمتر میشود و جامعه سالم تري خواهیم داشت. به عنوان یک پیشنهاد شاید بتوان بعد از لیسانس براي مقطع فوق لیسانس دانشجویان را از یکدیگر تفکیک کرد. به این گونه که دانشجویانی که قصد تحصیل در زمینه هاي تحقیقاتی و پژوهشی و همچنین قصد ادامه تحصیل براي مقطع دکترا دارند، از دانشجویانی که بعد از فوق لیسانس قصد وارد شدن به بازار کار دارند را از هم تفکیک کرد تا براي دانشجویان دسته اول بیشتر واحد هاي پژوهشی و تحقیقاتی در نظر گرفته شود و براي دانشجویان دسته دوم بیشتر واحد هاي عملی و کارآموزي و انجام پروژه در نظر گرفت تا بهتر براي وارد شدن به بازار کار آماده شوند و رشد بهتري در هر دو زمینه داشته باشیم. زیرا کسانی که قصد ورود به بازار کار را دارند فقط با مطالعه هاي تئوري نمیتوانند به راحتی وارد بازار کار شوند و تا وقتی چیزي را عملی انجام نداده باشند، به صورت تئوري نمیتوانند رشد خیلی خوبی داشته باشند.

Faculty of Payam e Noor Tehran

Department of Technical Engineering

B.Sc. Final Project Report

Title of the Report:

Phonebook in Web MVC

Under Supervision of:

Mr. Ali Razavi Ebrahimi

By:

Zeynab jalilvand(935121562)

<16/05/2021>