



دانشکده مهندسی

پایان‌نامه‌‌ی کارشناسی

گرایش نرم‌افزار

**پیاده‌سازی سیستم منشی دیجیتال**

نگارش:

**سینا ابراهیمی**

استاد راهنما:

**جناب آقای مهندس داوود محمد‌پور**

پاییز ۹۵

تقدیم به   
مقدس‌ترین واژه ها در لغت نامه دلم، **مادر** مهربانم که زندگیم را مدیون مهر و عطوفت آن می دانم.   
**پدر**، مهربانی مشفق، بردبار و حامی.   
**خواهر و برادرم** همراهان همیشگی و پشتوانه های زندگیم.

تمامی دوستانم که در مراحل مختلف زندگی حامی من بوده اند.

و با تشکر فراوان از استاد گرامیم، جناب مهندس **محمدپور**

استادی که سپیدی را بر تخته سیاه زندگیم نگاشت.

فهرست

[چکیده 1](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028364)

[فصل اول 2](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028365)

[1.۱: مقدمه 3](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028366)

[۲.۱: جامعه انسانی 3](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028367)

[۳.۱: رفتارهای مشابه در جوامع 4](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028368)

[۴.۱: تشخیص رفتار مشابه 4](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028369)

[فصل دوم 6](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028370)

[۱.۲: تشریح مفاهیم اولیه 7](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028371)

[۱.۱.۲: متن‌باز 7](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028372)

[۲.۱.۲: پایگاه‌داده‌ای NoSQL 8](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028373)

[۳.۱.۲: یادگیری ماشین 9](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028374)

[۴.۱.۲: چارچوب 10](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028375)

[۲.۲: تکنولوژی‌های استفاده شده در پروژه 10](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028376)

[۱.۲.۲: لینوکس 10](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028377)

[۲.۲.۲: Redis 12](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028378)

[۳.۲.۲: پایتون 16](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028379)

[۵.۲.۲ جنگو 21](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028380)

[۴.۲.۲: محیط یکپارچه توسعه نرم‌افزار 22](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028381)

[۵.۲.۲: Bootstrap 23](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028382)

[فصل سوم 25](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028383)

[۱.۳: معماری 26](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028384)

[۱.۱.۳: هستهٔ مرکزی 26](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028385)

[۲.۱.۳: وب‌سایت 26](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028386)

[۳.۱.۳: پایگاه‌داده 26](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028387)

[۴.۱.۳: پویش‌گر 27](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028388)

[۵.۱.۳: برنامه‌های خارجی 27](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028389)

[۲.۳: پیاده‌سازی 28](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028390)

[۱.۲.۳: وب‌سایت 29](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028391)

[۲.۲.۳: پایگاه‌داده 42](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028392)

[خلاصه فصل 42](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028393)

[فصل چهارم 43](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028394)

[منابع 45](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028395)

فهرست اشکال

[شکل 1: معماری سیستم 28](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028430)

[شکل 2: usecase انجام فرآیند دانلود 28](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028431)

[شکل 3: صفحه موفقیت جنگو 30](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028432)

[شکل 4: صفحه progress 38](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028433)

[شکل 5: صفحه لاگین برای admin 40](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028434)

[شکل 6: صفحه index برای admin 41](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028435)

[شکل 7: صفحه اضافه کردن دانلود برای admin 41](file:///C:\Users\ebine\Desktop\Document.docx#_Toc462028436)

## چکیده

سیستم منشی دیجیتال با هدف گرامیداشت و ارزش نهادن به مفهوم زمان ایده پردازی شده است. هدف اصلی این سیستم توانایی برنامه ریزی و تبلیغ غیر مستقیم برای صاحبان مشاغل است. مشتری ها در این سیستم توانایی جستجوی مشاغل مختلف را دارند و می توانند از آنها بر حسب نیاز خودشان وقت بگیرند. در این سیستم نقش های صاحب شغل، همکار و کاربر عادی تعریف شده اند که هر یک سطح دسترسی خاصی به سیستم دارند.

سعی بر این بود تا بتوانم سیستمی پیاده سازی نمایم که برای کاربران نهایی بسیار ساده بوده و قابلیت اعلان نیز بصورت داخلی وو اتوماتیک در این سیستم پیاده سازی شود.

# فصل اول

مفاهیم اولیه و تکنولوژی

**1 .1: مقدمه**

نسل جدید زبان‌های برنامه‌نویسی به این دلیل تولید می‌شود که زبان‌های قدیمی‌تر دارای امکانات محدود بوده و یا قدرت استفاده از تکنولوژی‌های فعلی را بصورت مطلوب ندارند.ASP.Net نسل جدیدی از ASP بوده که توسط شرکت Microsoft عرضـه گردیده‌است.ASP.Net اولین سیستم جامع برای برنامه‌نویسی تحت‌وب (اینترنت) است که از مراحل سطح پایین آن یعنی زبان ماشین تا بالاترین سطح آن‌که برنامه‌نویسی ویژوال می‌باشد برای استفاده در اینترنت و شبکه های محلی طراحی شده‌است. بزرگترین مزیت آن در برابر سیستم‌های دیگر، امکانات اینترنت آن است. از ASP.Net می‌توان در طراحی و تولید سایت‌های وب اینترانت کوچک یک شرکت و نیز سایت‌های وب تجاری خیلی بزرگ استفاده نمود.

مهمترین نکته‌ای که در طراحی این محصول در نظر گرفته‌شده‌است، استفاده‌ی آسان، کارآیی بالا و نیز قابلیت فوق‌العاده این نرم‌افزار است.ASP.NET MVC بخشی از چارچوب کاری برنامه های وب ASP.NET application framework است، MVC یکی از دو نوع مدل برنامه‌سازی ASP.NET است.  
در واقع ASP.NET MVC پیاده‌سازی مدل MVC به کمک ابزارهای توسعه نرم افزاری Microsoft در بستر وب ( ASP.NET ) است .

## 2.1: معرفی ASP.NET

زمانی كه شركت مایكروسافت نسخه‌ی اولیه‌ی .Net Framework را به نمایش گذاشت، برای اكثر طراحان و پیاده‌كنندگان مشخص شد كه می‌بایست در انتظار تحولات چشمگیری در این عرصه باشند.ASP3 امكانات گسترده و انعطاف لازم را به منظور ایجاد سایت‌های پویا در اختیار علاقه‌مندان قرار می‌داد. تاكنون صدها كتاب و مقاله با موضوع ASPنوشته شده‌است. حاصل تمامی تلاش‌های انجام شده در این رابطه، تسهیل در امر طراحی و پیاده‌سازی وب‌سایت‌های پویا و برنامه‌های وب بود. چیزی كه ASP نداشت، یك Framework برنامه‌نویسی بود. هر چیزی كه در ASP انجام می‌شد با رویكرد كدنویسی بود و برنامه‌نویسان برای انجام هر كاری ملزم به نوشتن كدهای مورد نیاز بودند. ASP.Net با هدف غلبه بر محدودیت فوق طراحی شده‌است.

ASP.Net یك Framework برنامه‌نویسی است كه بر روی CLR3 ایجاد می‌شود و می‌توان از آن به منظور ایجاد برنامه‌های قدرتمند وب استفاده كرد. یكی از اهداف اساسی طراحی ASP.Net برنامه‌نویسی ساده‌تر و با سرعت بیشتر از طریق كاهش حجم كدهای مورد نیازی است كه برنامه‌نویسان ملزم به نوشتن آن‌ها می‌باشند. برنامه‌نویسی اعلانی، مجموعه‌ای گسترده از كنترل‌های سرویس‌دهنده به همراه رویدادهای مربوطه، یك كتابخانه كلاس بزرگ و ابزارهای پیاده‌سازی كاملا حرفه‌ای نظیر Visual Studio.Net، از جمله ویژگی‌های شاخص ASP.Netمحسوب می‌شود. کنترل‌های سرویس‌دهنده، معماری Postback، حمایت از حالت دید[[1]](#footnote-1) و استفاده از كد ترجمه شده و مدل برنامه‌نویسی مبتنی بر رویداد از مهم‌ترین دست‌آورد‌های ASP.Net محسوب می‌شوند كه نوید ایجاد و اشكال‌زدایی سریع برنامه‌های وب را در اختیار پیاده‌كنندگان قرار می‌دهد.

**مقایسه ASP.Net و ASP Classic**

ASP.Net نسل بعدی ASP Classic است. اما این یك پیشرفت تكاملی است بطوریكه این دو فناوری تقریباً از یكدیگر متفاوت‌اند. صفحات ASP با زبان‌های دستورالعمل‌نویسی[[2]](#footnote-2) مانندVB Script یا Java Script ایجاد می‌شوند.

اما در ASP.Net ما یك فرایند كامل برنامه‌نویسی با زبان‌های Visual Basic یا C# داریم. همچنین در ASP Classic تنها پنج كلاس استانداردSession ،Response ، Request، Server و Application وجود دارد. حال آن كه ASP.Net می‌توان از بیش از 4500 كلاس استاندارد موجود در بدنه‌ی .Net بهره ‌جست. هم‌چنین علی‌رغم قدرت و امكانات زیاد و متعدد ASP.Netاستفاده از آن در مقایسه با ASP Classic بسیار آسان‌تر است. به عنوان مثال با استفاده از چند ابزار در یك صفحه‌ی ASP.Net، می‌توان یك صفحه بسیار پیچیده HTMLبه دست‌آورد كه ساخت آن در ASP Classic نیاز به ‌چند روز كار دارد.

## ۱.2.1: معرفی IIS

سرویس Internet Information Service یا به اختصار همان IIS وب سرور قدرتمند، قابل انعطاف، امن و با قابلیت کاربری آسان و محصول شرکت Microsoft می‌باشد که امکانات بسیار زیادی از قبیل Media Streaming، میزبانی وب، Application و… را در اختیار وب‌سایت‌های مربوطه قرار می‌دهد.

وب سرور IIS برای ایجاد، مدیریت و میزبانی وب[[3]](#footnote-3) مورد‌استفاده قرار می‌گیرد. این نرم‌افزار از امکاناتی مانند HTTP ،HTTPS ،SMTP ،FTP،SFTP و NNTP به خوبی پشتیبانی می‌نماید و به صورت کامل با ویندوز سرور سازگار می‌باشد.

نسخه‌های قدیمی‌تر IIS دارای آسیب پذیری‌های بسیار زیادی بودند، که معروف‌ترین آن‌ها مربوط به کد‌های مخرب Code Red Worm در سال ۲۰۰۱ می‌باشد. گرچه تا به حال هیچ‌گونه گزارشی مبنی بر وجود حفره‌های امنیتی در نسخه‌های ۶ و ۷ این نرم‌افزار منتشر نشده‌است اما Microsoft برای اطمینان هرچه بیشتر کاربرانش به طور کلی سعی در تغییر ساختار امنیتی نرم‌افزار سرویس‌دهنده وب خود در نسخه 7.5 نموده است.

یکی از این امکانات، قابلیتی تحت عنوان Web Service Extension از نسخه IIS 6.0 اضافه گردیده که این نرم‌افزار را از اجرای هر برنامه ثالثی بدون اجازه سرویس‌دهنده منع می‌نماید.

تمامی اجزای نرم‌افزار مذکور در نسخه IIS 7 به صورت پودمانی[[4]](#footnote-4) تغییر یافته است، یعنی هر یک از قابلیت‌های آن می‌توانند به صورت منحصر‌بفرد نصب و یا حذف شوند.

برای برنامه‌نویسی وب در Visual Studio، ابتدا باید IIS را نصب کرد و بعد از آن اقدام به نصب Visual Studio شود. البته فارغ از این که روی چه سیستم عاملی از Microsoft، Visual Studio نصب شده باشد، خود ابزار Visual Studio امکانات IIS را فراهم خواهد کرد. لازم به ذکر است که Microsoft با استفاده از زبان C++ این سرویس‌دهنده وب را توسعه داده است.

در پروژه ذیل از IIS 10.0 استفاده شده‌است که لازم به ذکر است این نسخه از IIS به همراه Windows 10 و Windows Server 2016 در تابستان 2015 عرضه شده‌است.

## ۲.2.1: معرفی Framework

Framework مجموعه منسجم از کلاس‌ها و توابع از پیش تعریف شده‌است که قابلیت‌های بالقوه گوناگون از یک زبان برنامه‌نویسی را در خود دارد و بدین ترتیب کاربر نهایی را قادر می‌سازد که از امکانات یک زبان استفاده کند بدون اینکه درگیر مسائل پیچیده و وقت گیر آن شود، لذا همان طور که یک کلاس یا تابع با هدف جلوگیری از تکرار و افزایش سرعت کار، تعریف می‌شود، Framework از این هم فراتر رفته و علاوه بر افزایش سرعت، مواردی مثل توسعه‌پذیری و ساده‌سازی را هم مدنظر دارد**.**

.NET Framework از دو بخش اصلی، کتابخانه .net[[5]](#footnote-5) وCLR [[6]](#footnote-6) تشکیل شده‌است.

کتابخانه .Net مجموعه‌ای از کلاس‌های آماده از پیش نوشته شده‌است که در تمامی زبان‌های مبتنی بر .Net قابل استفاده هستند. اما CLR وظیفه اجرا و مدیریت برنامه‌های تحت Net. را دارد. کنترل و مدیریت اجرای برنامه‌ها، مدیریت حافظه و کنترل و مدیریت امنیت در برنامه‌های Net. از وظایف CLR هست.

Framework‌ها در واقع یک هسته اصلی می‌باشند که هرگونه اضافه کردن، توسعه یا حذف اشیا می‌تواند به راحتی و در مقابل فایل‌های کتابخانه‌ای و وابسته به هسته اصلی این Framework در نظر گرفته شود.

هم‌چنین ایجاد یک برنامه کاربردی تحت‌وب بر مبنای یک Framework باعث می‌گردد تا توسعه‌دهندگان و برنامه‌نویسان علاوه بر تجارب خود بتوانند از تجارب سایر توسعه‌دهندگان نیز سود جسته و به‌کار بندند؛ مشخص بودن نوع معماری به کار رفته نیز باعث می‌گردد تا توسعه‌دهندگان مختلف بتوانند به راحتی سیستم خود را توسعه دهند.

## 3.1: معرفی معماری MVC

نام MVC ازModel-View-Controller گرفته شده و هدف اصلی آن جدا سازی اجزای تشکیل دهنده برنامه است، بخصوص برنامه‌هایی که دارای واسط کاربری گرافیکی هستند. (GUI یا همان Graphical User Interface)

Mدر MVC معادل مدل[[7]](#footnote-7) است و حاوی اطلاعاتی است که نهایتا در اختیار کاربر قرار خواهد‌گرفت. Model وظیفه کار با پایگاه‌داده و دیگر اشیا را بر عهده دارد.

Vدر MVC معادل دید[[8]](#footnote-8) است و حاوی نتیجه‌ای است که کاربر نهایتا در مرورگر خواهد‌دید. view در واقع User Interface برنامه است و وظیفه ارتباط با کاربر نهایی را بر عهده دارد.

C در MVC معادل کنترل‌گر[[9]](#footnote-9) است وظیفه کنترل View و Model و نحوه ارتباط آن دو را با هم بر عهده دارد.

در واقع MVC بر روی معماری‌های چندلایه‌ای، جهت جداسازی قسمت‌های مختلف برنامه و به طور دقیق تر، جدا کردن بخش‌های منطقی برنامه اعم از داده‌ها[[10]](#footnote-10)، مجوز‌های دسترسی[[11]](#footnote-11)، چک کردن صحت داده‌ها و‌.‌.‌. از لایه ﻧﻤﺎﯾﺶ [[12]](#footnote-12) یا در واقع همان لایه‌ای که مستقیما با کاربر نهایی در ارتباط است (مانند فرم‌ها، اجزا و‌.‌.‌.) قرار‌می‌گیرد. الگوی طراحی MVC در لایه ﻧﻤﺎﯾﺶ در معماری سه لایه استفاده می‌شود و هدف نهایی آن جداسازی مفاهیم در لایه ﻧﻤﺎﯾﺶ به منظور خواناتر کردن کد و بالا بردن قابلیت نگهداری[[13]](#footnote-13) آن است.

این الگوی طراحی در سایر Framework ها مثل Ruby on Rails و Django و Zend هم به‌کار برده شده‌است.

پس بر اساس توضیحات فوق می‌توانیم هر یک از بخش‌های MVC یعنی View، Model، Controller را به شکل زیر تعریف کنیم:

## ۱.3.1: معرفی Model

مدل امکان دسترسی به داده‌های ذخیره شده جهت نمایش به کاربر یا دریافت و ذخیره‌سازی داده‌هایی که در اختیار آن قرار می‌گیرد را فراهم می‌کند و نقش پلی بین بخش "دید" و "کنترل‌گر" را در درون سیستم ایفا می‌کند. یک جنبه بسیار مهم مدل آن است که باید کور باشد، یعنی هیچ‌گونه ارتباط و آگاهی از آنچه که بر سر داده‌ها در بخش‌های دید و کنترل‌گر می‌افتد، نداشته باشد. Model، نه بخش‌های دیگر را فراخوانی می‌کند و نه منتظر پاسخ از آن‌ها می‌ماند. تنها هدف آن، ذخیره‌سازی داده‌ها در رسانه فیزیکی (اعم از فایل یا بانک اطلاعاتی) یا آماده‌سازی اطلاعات برای تحویل به بخش‌های دیگر است. البته مدل را نمی‌توان به سادگی در حد پایگاه‌داده‌ها یا یک معبر به سایر سیستم‌های نگهداری اطلاعات (مثل فایل) کوچک شمرد. مدل باید به عنوان دروازه‌بان داده‌ها عمل کند. سوالی نپرسد اما تمامی درخواست‌هایی را که از کانال صحیح وارد می‌شوند، بپذیرد. در اغلب موارد، مدل به عنوان پیچیده‌ترین بخش MVC، نقش هسته مرکزی آن را نیز ایفا می‌کند زیرا بدون آن ارتباطی بین عناصر دیگر یعنی "دید" و "کنترل‌گر" برقرار نخواهد‌شد.

در واقع بار اصلی معماری MVC بر عهده این بخش است. این بخش می‌تواند با داده‌ها در ارتباط باشد. هر‌چند منظور از داده حتما ارتباط با پایگاه‌های داده هم‌چون SQL Server، Access، MySQL و... نیست، حتی منبع داده‌ها در بخش مدل می‌تواند یک آرایه از اعداد و یا هر چیز دیگری باشد. هم‌چنین مدل وظیفه چک کردن داده‌ها جهت صحت درستی داده‌ها را بر عهده دارد (در این زمینه همکاری بیشتری با بخش Controller دارد).

بطور کلی می‌توان گفت Model شامل سه بخش اصلی می‌باشد :

* Data Model: هر نرم‌افزاری که با بانک اطلاعاتی تعامل دارد.
* Business Model: برای به‌کارگیری Data Model و ذخیره اطلاعات در بانک اطلاعاتی می‌باشد.
* View Model: اطلاعات را از سمت کنترل‌گر به سمت view ارسال می‌کند.

## ۲.3.1: معرفی View

این بخش که در واقع همان بخش لایه‌ی ﻧﻤﺎﯾﺶ در معماری 3 لایه می‌باشد، وظیفه برقراری ارتباط با کاربر نهایی و گرفتن داده از کاربر و نمایش داده‌های آماده برای کاربر از طریق برقراری ارتباط با دو بخش دیگر یعنی مدل و کنترل‌گر است. پوشه View صفحاتی از نوع view و partial view را ذخیره می‌کند که هر کدام از آن‌ها برای عمل[[14]](#footnote-14) مشخصی در کنترل‌گر تعریف شده‌است. پوشه مربوط به view شامل یک پوشه به‌نام shared folder می‌باشد که صفحات و [User Control](http://barnamenevisan.org/Articles/Article4350.html)‌هایی که در داخل آن وجود دارد به طور مشترک در دسترس همه‌ی کنترل گرها قرار می‌گیرد. هر در‌خواستی که از طرف action method ها برای یک صفحه‌ی  view یا partial view فرستاده می‌شود در صفحه‌ی extension که در داخل shared folder قرار دارد نیز بررسی می‌شود.

در‌واقع نکته مهمی که در بخش View باید مدنظر داشت این است که این لایه مسئول کنترل صحت داده‌های وارد شده از طریق کاربر و هم‌چنین مسئول صحت داده‌های نشان داده به کاربر نیست. در واقع این بخش با داده‌های خام کار می‌کند. یک مثال ساده خیلی از برنامه‌نویسان، هنگاهی که در فرم Login برنامه کاربر کلمه عبور خود را وارد می‌کند، در همان فرم Login اقدام به چک کردن رمز عبور مبنی بر صحت آن می‌کنند، که این عمل در معماری MVC قابل قبول نیست. در واقع در معماری MVC در فرم Login هنگامی که کاربر کلمه عبور را وارد کرد و دکمه Login یا ورود را زد، کلمه عبور داده شده بدون هیچ‌گونه اعمالی اعم از Encrypt کردن و‌.‌.. به بخش‌های دیگر فرستاده می‌شود و فقط یک نتیجه ساده مبنی بر این که کاربر اجازه ورود دارد یا خیر را از بخش‌های دیگر دریافت می‌کند که بر اساس آن اجازه ورود کاربر به برنامه داده می‌شود. در واقع View مکانی است که داده‌هایی که از Model درخواست شده‌اند، به نمایش در‌آمده و خروجی نهایی آن توسط کاربر دیده می‌شود.

در روش استاندارد کاربرد MVC، عنصر View بخشی از سیستم است که کد Html در آن تولید و نمایش داده‌می‌شود. هم‌چنین View محیطی را برای انجام عملیات توسط کاربر فراهم می‌کند و سپس، این عملیات به Controller منتقل می‌شوند تا به آن‌ها پاسخ مناسب داده‌شود. مثال ساده دیگر روش دکمه‌ای است که توسط View تولید می‌شود وقتی کاربر بر روی آن کلیک می‌کند، عملیاتی در Controller اتفاق می‌افتد. برخی باورهای اشتباه درباره عنصر View به خصوص در بین توسعه‌دهندگان وب که از MVC استفاده می‌کنند، برای مثال، بسیاری از آن‌ها به اشتباه فکر می‌کنند که View هیچ‌گونه ارتباط مستقیمی با Model ندارد و تمام داده‌های مورد نیاز جهت نمایش توسط View، از طریق Controller ارسال می‌شود. در حقیقت، این روند با اصول MVC ناسازگاری کامل دارد متاسفانه این روند تا جایی رواج داشته است که در کتاب‌های آموزش MVC و حتی در برخی از Framework‌های معروف مثل CakePHP نیز از آن استفاده شده‌است. به علاوه توصیف View به عنوان یک قالب نیز اشتباه است هر چند، متاسفانه این باور نیز نه به صورت فردی بلکه در بین انبوهی از توسعه‌دهندگان و نویسندگان رایج است و منجر به یادگیری MVC به صورت اشتباه می‌شود.

## ۳.3.1: معرفی Controller

این بخش همان‌طور که از اسم آن مشخص است یک بخش کنترل کننده می‌باشد، و در واقع واسطی بین بخش Model وView می‌باشد. در معماری MVC روند کلی برنامه به این شکل است که کاربر تقاضای خود را از طریق واسط‌های برنامه‌نویسی (نظیر Form‌ها و User Control‌ها و...) از برنامه (از بخش View) درخواست می‌کند. بخش View درخواست‌ها را به بخش Controller فرستاده و این بخش با برقراری ارتباط با بخش Model درخواست‌های کاربر را پردازش کرده و پس از پایان پردازش زمانی که خروجی درخواست داده شده آماده‌گردید بخش Controller بخش View را آگاه می‌سازد تا خود را براساس تغییرات جدید که به اصطلاح در معماری MVC به آن حال Model می‌گویند، به‌روز سازد. در واقع چیزی که باعث می‌شود که بخش Controller به بخش View اطلاع دهد که باید حالت جدید Model را دریافت کند و خود را به‌روز کند این است که بخش View باید قبلا خودش را در بخش Model به اصطلاح Register کرده باشد که البته Register کردن توسط Controller انجام می‌گیرد. نحوه Register کردن بخش View به معماری آن محیط و هم‌چنین قابلیت‌های آن زبان بستگی دارد.

اﮔﺮ در ﻻﯾﻪ ﻣﺪل ﯾﮏ ﭘﺎﯾﮕﺎه داده ﺑﺎﺷﺪ ﮐﺎرﺑﺮ اﻃﻼﻋﺎﺗﯽ را در راﺑﻂ ﮐﺎرﺑﺮ وارد ﻣﯽ ﮐﻨﺪ ﺗﺎ ﭘﺎﯾﮕﺎه داده را ﺟﺴﺘﺠﻮ کند. ﺑﻪ اﯾﻦ اﻃﻼﻋﺎت ﭘﺎراﻣﺘﺮﻫﺎی داده‌ای گفته می‌شود. ﺑﯿﺸﺘﺮ اﯾﻦ ﭘﺎراﻣﺘﺮﻫﺎ ﺑﻪ ﺷﮑﻞ ﻋﺪدی می‌باشند. اﮔﺮ ﻻﯾﻪ ﻣﺪل از ﯾﮏ ﮐﺘﺎﺑﺨﺎﻧﻪ ﺗﻮاﺑﻊ ﺗﺸﮑﯿﻞ ﺷﺪه ﺑﺎﺷﺪ ﻧﺮم‌اﻓﺰار ﺑﺮای اﻧﺠﺎم ﮐﺎری ﻏﯿﺮ از ﺟﺴﺘﺠﻮ از ﭘﺎﯾﮕﺎه داده ﻧﻮﺷﺘﻪ ﺷﺪه است. ﻓﺮض ﻣﯽ‌ﮐﻨﯿﻢ ﮐﻪ اﯾﻦ ﮐﺎر ﺑﻪ روش‌های ﻣﺨﺘﻠﻔﯽ اﻧﺠﺎم ﺷﻮد ﮐﻪ ﺑﻪ ﻫﺮ ﮐﺪام از اﯾﻦ روش‌ها ﯾﮏ رﻓﺘﺎر ﻧﺮم‌اﻓﺰار ﮔﻔﺘﻪ می‌شود. ﻫﺮ رﻓﺘﺎر ﺑﻪ وﺳﯿﻠﻪ ﯾﮏ ﺗﺎﺑﻊ ﮐﻪ در ﻗﺴﻤﺖ ﻣﺪل اﺳﺖ ﭘﯿﺎده‌ﺳﺎزی ﻣﯽ‌ﺷﻮد. ﺑﻪ‌ازای ﻫﺮ رﻓﺘﺎر ﯾﮏ اﻧﺘﺨﺎب در راﺑﻂ ﮐﺎرﺑﺮ وﺟﻮد دارد ﮐﻪ ﮐﺎرﺑﺮ ﺑﺎ ‌آن‌ها روش اﻧﺠﺎم ﮐﺎر را ﻣﺸﺨﺺ می‌کند. مثلا در ﺑﺮﻧﺎﻣﻪ WinZip ﺑﺮای ﻓﺸﺮده‌ﺳﺎزی ﯾﮏ ﻓﺎﯾﻞ دو اﻧﺘﺨﺎب وﺟﻮد دارد. ﯾﮑﯽ ﺑﺮاﺳﺎس اﯾﻨﮑﻪ ﻓﺸﺮده‌ﺳﺎزی زودﺗﺮ اﻧﺠﺎم ﺷﻮد و دﯾﮕﺮ اﯾﻨﮑﻪ ﻓﺎﯾﻞ ﻓﺸﺮده ﺷﺪه ﺣﺠﻢ کم‌تری داﺷﺘﻪ ﺑﺎﺷﺪ. ﺑﻪ ازای ﻫﺮ ﮐﺪام از اﯾﻦ انتخاب ها ﯾﮏ ﺗﺎﺑﻊ وﺟﻮد دارد ﮐﻪ ﮐﺎر ﭘﯿﺎده‌ﺳﺎزی را اﻧﺠﺎم می‌دهد. ﯾﺎ ﺑﺮﻧﺎﻣﻪ Visual Studio ﺑﺮﻧﺎﻣﻪ‌ﻧﻮﯾﺲ ﺑﺮای ﮐﺎﻣﭙﺎﯾﻞ ﮐﺪ ﺑﺮﻧﺎﻣﻪ دو اﻧﺘﺨﺎب دارد. اول اﯾﻨﮑﻪ ﺑﺮﻧﺎﻣﻪ ﺑﻪ ﺷﮑﻠﯽ ﮐﺎﻣﭙﺎﯾﻞ ﺷﻮد ﮐﻪ ﮐﺪ ﺗﻮﻟﯿﺪ ﺷﺪه از ﺣﺠﻢ ﮐﻤﺘﺮی ﺑﺮﺧﻮردار ﺑﺎﺷﺪ و دوم این‌که ﮐﺪ ﺗﻮﻟﯿﺪ ﺷﺪه سریع‌تر اﺟﺮا شود. ﺑﺮای ﻫﺮ ﮐﺪام از اﯾﻦ انتخاب‌ها ﯾﮏ ﺗﺎﺑﻊ وﺟﻮد دارد ﮐﻪ ﮐﺎر ﮐﺎﻣﭙﺎﯾﻞﮐﺮدن ﺑﺮﻧﺎﻣه را اﻧﺠﺎم می‌دهد. ﺑﻪ اﯾﻦ اﻧﺘﺨﺎب‌ها ﮐﻪ رﻓﺘﺎر ﻧﺮم‌اﻓﺰار را ﺗﻌﯿﯿﻦ می‌کند ﭘﺎراﻣﺘﺮ ﮐﻨﺘﺮﻟﯽ ﮔﻔﺘﻪ ﻣﯽ‌ﺷﻮد. ﻻﯾﻪ کنترل با کمک این پارامترهای کنترلی ﻧﺘﺎﯾﺞ را از ﻻﯾﻪ ﻣﺪل درﯾﺎﻓﺖ ﮐﺮده و ﺑﻪ ﻻﯾﻪ ﻧﻤﺎﯾﺶ می‌فرستد ﺗﺎ ﮐﺎرﺑﺮ آن‌ها را ﺑﺒﯿﻨﺪ.

## 4.3.1: معرفی MVC 5

چارچوب ASP.NET MVC یک چارچوب نرم افزاری تحت‌وب است که الگوی MVC (Model، View، Controller) را پیاده‌سازی می‌کند.

مهم‌ترین ویژگی‌های MVC 5 که در این پروژه مورد استفاده قرار دادیم:

* استفاده از سیستم Routing موجود در زیرساخت ASP.NET برای نمایش URLهایی بدون پسوند. برای مثال به‌جای این‌که آدرس localhost/Account/SignIn.cshtml را داشته‌باشیم آدرس localhost/Account/SignIn را خواهیم‌داشت.
* مدیریت بهتر قسمت‌های مختلف سایت در پوشه‌های جداگانه: برای مثال View‌های هر مدل به طور اتوماتیک در پوشه مخصوص خود جای می‌گیرد و این امر ذاتاً باعث زیباتر شدن پروژه و قابلیت خوانایی بیشتر می‌شود.
* مقداردهی خودکار مدل متناظر با یک View در ASP.NET MVC به این صورت که در Controller مربوطه متد مربوط به آن View به صورت اتوماتیک ایجاد می‌شود.
* کنترل بهتر بر روی اعتبارسنجی اطلاعات دریافتی
* امکان استفاده از فرم ها و Viewهای Razor به جای موتور وب فرم‌ها
* سازگاری کامل با JQuery وJQuery Ajax و انواع Framework‌های JavaScript
* امکانات فشرده‌سازی css و js
* استفاده از سیستم کدسازی خودکار به‌نام Scaffolding

## 5.3.1: معرفی Razor

Razor یک Syntax برنامه‌نویسی ASP.NET برای ساختن صفحات‌وب پویا (همان فرمت cshtml) به کمک C# و یا Visual Basic .NET است. به این صورت که یک صفحه وب برای مرورگر نوشته شده‌است، کدهای مبتنی بر سرور می‌توانند محتوای پویا ایجاد کنند. وقتی صفحه وبی فراخوانی می‌شود، سرور کدهای سمت سرور داخل صفحه را قبل از برگرداندن صفحه به مرورگر اجرا می‌کند.

این کدها عملیات پیچیده‌ای را، مانند دستیابی به پایگاه‌داده، انجام می‌دهند.

Razor بر مبنای ASP.NET است، و برای ایجاد کاربردهای وب طراحی گردیده است و قابلیت‌های علامت‌گذاریASP.NET سنتی را دارد، اما استفاده از آن و یادگیری آن آسان‌تر است. Razor اولین بار در Visual Studio 2010 عرضه شد.

Razor یک View Engine با Syntax ساده است که اولین بار به عنوان بخشی از MVC 3 عرضه شد. لازم به ذکر است که فایل‌های تولید شده توسط این View Engine با فرمت .cshtml ذخیره می‌شوند.

برای مثال اگر در قسمت کنترلر یک لیست از پایگاه داده بخوانیم و آن را داخل یک ViewBag بریزیم، در داخل View خود می توانیم آن ViewBag را توسط یک foreach فراخوانی کنیم و لیست مورد نظرمان را به کاربر نشان بدهیم. توانایی ترجمه دستوراتی مانند foreach و if در فایل html که اکثرا هم با نماد @ شروع می‌شوند از جمله قابلیت‌های موتور Razor می‌باشد.

## 6.3.1: معرفی Entity Framework

Entity Framework با ارائه یک مدل مفهومی که با پایگاه‌داده و برنامه کار می‌کند، به عنوان یک واسط عمل می‌کند و به توسعه دهنده این قابلیت را می‌دهد که با داده‌‌های Database هم‌چون Objectهایی در زبان C# یا VB کار کند و به‌جای درگیر شدن مستقیم با ساختار پایگاه‌داده و برنامه‌نویسی مستقیم بر روی داده‌های آن، بر روی تعدادی موجودیت[[15]](#footnote-15) که از روی پایگاه‌داده ساخته شده‌اند، پرس‌وجو[[16]](#footnote-16) بنویسد، عملیات (Create,Read,Update,Delete) CRUD را انجام دهد، رابطه‌های بین موجودیت‌ها را مدیریت کند و حتی از روابط ارث‌بری بین موجودیت‌ها بهره ببرد. مجموعه این موجودیت‌ها و روابط بین آن‌ها، مدل مفهومی ما را در Entity Framework تشکیل می‌دهند که با نام EDM (Entity Data Model) شناخته می‌شود.

به‌طور کلی در EF سه حالت مدل سازی موجود است:

* **Database First Modeling:** در این روش مدل ما از روی یک پایگاه‌داده موجود ایجاد می‌شود و می‌توان از طریق Entity Data Model Designer در Visual Studio تغییرات لازم را بر روی مدل انجام داد و یا در پایگاه داده تغییرات را اعمال کرده و در Model Designer مدل خود را Update کنیم.
* **Model First Modeling:** در این روش ابتدا مدل توسط برنامه نویس در محیط Entity Data Model Designer ایجاد می‌شود، سپس به‌طور خودکار پایگاه‌داده و کدها و اسکریپت‌های موردنیاز از روی مدل ساخته می‌شوند.
* **Code First Modeling :** در این روش کلاس‌های معادل موجودیت ها (جداول) توسط برنامه نویس نوشته می‌شوند (این کلاس ها POCO-Plain OLD CLR Objects نامیده می‌شوند) سپس EF بطور خودکار پایگاه‌داده و مدل را از روی این کلاس ها می‌سازد.

در واقع در Entity Framework با پایگاه‌داده و جداول آن مانند اشیائی برخورد می‌کنیم که مکانیزم‌های زیر برایشان فراهم شده‌است:

* انجام عملگرهای پایه CRUD ( Delete، Update، Read،Create)
* مدیریت آسان رابطه‌های یک به یک، یک به چند و چند به چند
* قابلیت داشتن روابط ارث‌بری بین Entity ها
* قابلیت تبدیل اشیاء پایگاه‌داده به کلاس ها (مدل ها)

مزایای استفاده از Entity Framework به شرح زیر است:

* دسترسی به داده‌ها در یک زبان سطح بالا با استفاده از دستوراتی مانند Select و Where.
* مدل Conceptual را می‌توان با استفاده از روابط بین Entity ها بیان نمود. (تبدیل به مدل)
* مدیریت آسان‌تر داده‌ها، مانند افزودن، حذف و به‌روز رسانی
* هم‌چنین در ASP.NET و MVC 5 Framework Microsoft مکانیزم‌هایی برای آسان‌سازی ایجاد CRUD در IDE فراهم نموده است که می‌توان با چند کلیک به سادگی برای یک مدل خود Controller و چند View مربوط به عملیات CRUD را ایجاد نمود و در وقت صرفه‌جویی کرد. (این ویژگی را Scaffolding می‌نامند.)

لازم به ذکر است که در این پروژه ابتدا پایگاه داده طراحی شد و سپس با استفاده از روش Database First دسترسی به پایگاه داده ها را با استفاده ازEntity Framework فراهم نمودیم.

## 4.1: معرفی تکنولوژی Bootstrap

افزایش روزافزون استفاده از ابزارهای مختلف در طراحی سایت و هم‌چنین تلاش طراحان سایت برای ایجاد سایتی کاربرپسند و زیبا موجب این شده‌است که استفاده از طراحی‌های پیش ساخته در طراحی سایت امروزه بیشتر متداول شود‌. از جمله امکانات Bootstrap می‌توان به چارچوب آن اشاره نمود؛ این چارچوب یا Framework با تمامی مرورگرهای استاندارد همخوانی داشته و حتی در نسخه‌های قدیمی‌تر مانند اینترنت اکسپلورر 8 نیز ظاهر زیبای خود را حفظ میکند. از نسخه دوم Bootstrap به بعد طراحی واکنش‌گرا یا responsive نیز در آن لحاظ شد که موجب نمایش مناسب صفحه در تلفن‌های هوشمند و تبلت‌ها می‌گردد. هم‌چنین می‌توان به متن باز بودن آن اشاره نمود. تا به اینجا نیز نسخه سوم این ابزار کاربردی منتشر شده‌است و ما در این پروژه از نسخه‌ی Bootstrap3.3.6 استفاده کرده‌ایم.

## 1.4.1: Bootstrap چیست؟

Bootstrap مجموعه‌ای از ابزارهای رایگان برای ایجاد صفحات‌وب و نرم افزارهای تحت‌وب است که شامل دستورات Html ،CSS و توابع JavaScript جهت تولید و نمایش فرم‌ها، دکمه‌ها، سربرگ‌ها، ستون‌ها و سایر المان‌های مورد نیاز طراحی وب می‌باشد.

Bootstrap در ابتدا توسط Mark Otto و Jacob Thornton در جهت ایجاد یک چارچوب ظاهری مشخص و یکسان در ابزارهای توییتر طراحی و نوشته شد. قبل از شروع این پروژه نمونه‌های زیادی با همین رویکرد ایجاد شده‌بود که همگی با سرنوشتی مشابه و عدم استقبال طراحان وب دنیا مواجه شده‌بودند. به دلیل وجود مشکلات اساسی در نمونه‌های دیگر، سازنده اصلی توییتر یا همان مارک اتو تصمیم به ساخت یک سیستم داخلی و قدرتمند برای خود را با نام Bootstrap گرفت.

## 2.4.1: Bootstrap به زبان ساده

بزرگترین مشکل طراحان وب و کدنویسان قدیمی، ایجاد ظاهری زیبا و مناسب است. اصول کدنویسی و ایجاد زیربنا و ساختار مناسب برای یک سایت بسیار مهم و پیچیده است ولی نمایش صحیح خروجی کار و ایجاد یک فضای کارپسند نیز اهمیت بسیار بالایی خواهد‌داشت.Bootstrap قصد دارد که خلا میان طراحی و کدنویسی را از میان برداشته و کدنویسان را ترغیب به استفاده از طراحی‌های پیش‌فرض و استاندارد نماید. به همین منظور دستورات CSS و توابع jQuery مورد نیاز را برای شما فراهم کرده است تا شما بتوانید با استفاده از دستورات پیشفرض و رعایت اصول متناسب با طراحی Bootstrap زمان راه‌اندازی یک پروژه را تا حد زیادی کاهش داده و خروجی آن را متناسب با استانداردهای روز دنیا پیش ببرید.

## 3.4.1: مزایای Bootstrap

بزرگترین مزیت Bootstrap این است که دارای مجموعه رایگانی از ابزارها برای ایجاد صفحات‌وب انعطاف‌پذیر و responsive می‌باشد.

به‌علاوه، با استفاده از اطلاعات رابط برنامه‌نویسی (API) درBootstrap می‌توانید اجزاء واسط پیشرفته مانند scroll spy و تکمیل‌کننده خودکار کلمات (type ahead) را بدون نیاز به نوشتن حتی یک خط کد جاوا اسکریپت ایجاد نمایید. اما Bootstrap مزایای دیگری دارد که در زیر به آن اشاره می‌نماییم:

**Mobile First:** در طراحی Bootstrap از رویکرد موبایل در الویت استفاده می‌شود. یعنی درطراحی سایت قبل از اینکه به رایانه فکر کنیم ابتدا سایت را برای موبایل طراحی کنیم. در واقع دو نوع رویکرد برای طراحی سایت داریم: رویکرد Desktop محور، رویکرد Mobile First.

**صرفه جویی در زمان:** می‌توان با استفاده از قالب ها و کلاس‌های از پیش طراحی شده Bootstrap زمان و انرژی کم‌تری برای طراحی صرف کرده و بیش‌تر بر روی جنبه‌های دیگر پروژه متمرکز شد.

**ویژگی‌های** **: responsive** با استفاده از Bootstrap می‌توان به راحتی طراحی‌های responsive ایجاد کرد. ویژگی‌های responsive باعث می‌شوند که صفحات‌وب در دستگاه‌های مختلف و وضوح تصویر متفاوت به درستی و به صورت مناسب و بدون نیاز به هیچ‌گونه تغییر در کدگذاری، نمایش داده شوند.

**طراحی منسجم و یکپارچه :**  تمامی مؤلفه‌های Bootstrap از قالب‌های طراحی مشترک از طریق یک کتابخانه مرکزی استفاده می‌کنند. بنابراین طرح و پیکربندی صفحات‌وب در طول توسعه و طراحی، ثابت و یکپارچه باقی می‌مانند.

**سهولت استفاده:**  استفاده از Bootstrap بسیار ساده است به‌طوری که هر شخص با دانش و اطلاعات اولیه و پایه‌ای از HTML و CSS می‌تواند از آن استفاده کند.

**سازگار با مرورگرها :**  Bootstrapبا کلیه مرورگرهای پیشرفته و جدید مانند Mozilla Firefox، Google Chrome، Safari،Internet Explorer و Opera سازگار است.

## 5.1: معرفی زبان JavaScript

زبان برنامه‌نویسی اسکریپت مبتنی بر اشیاء است که توسط Netscape تولید شده‌است. این زبان، یک زبان شی‌گراست.

این زبان می‌تواند هم به‌صورت ساخت‌یافته و هم به‌صورت شی‌گرا مورد استفاده قرار گیرد. در این زبان اشیاء با اضافه شدن متدها و خصوصیات پویا به اشیاء خالی ساخته می‌شوند.

کاربرد گسترده این زبان در سایت‌ها و صفحات اینترنتی می‌باشد و به کمک این زبان می‌توان به اشیاء داخل صفحات HTML دسترسی پیدا کرد و آن‌ها را تغییر داد. به همین علت برای پویانمایی در سمت کاربر، از این زبان استفاده می‌شود.

JavaScript به یکی از زبان‌های برنامه‌نویسی پر طرفدار در وب تبدیل شده‌است. هر چند ابتدا بسیاری از برنامه‌نویسان حرفه‌ای این زبان را کم ارزش تلقی می‌کردند چون مخاطبین آن نویسندگان صفحات‌وب و آماتورهای این‌چنینی بودند. ظهور Ajax بار دیگر JavaScript را در معرض توجه قرار داد و برنامه‌نویسان حرفه‌ای بیشتری را به خود جذب نمود. نتیجه‌ی این تغییر، ازدیاد Framework و کتابخانه‌های جامعی در این زمینه (مانند jQuery و...)، بهبود شیوه‌های رایج برنامه‌نویسی در JavaScript و افزایش کاربرد JavaScript خارج از وب است.

## 1.5.1: معرفی jQuery

jQuery یک کتابخانه سبک وزن چند مرورگری است که برای ساده سازی نوشتن اسکریپت های سمت کاربر(Client) در Html طراحی شده و امروزه محبوب ترین کتابخانه‌ی JavaScript در حال استفاده است.

jQuery نرم افزاری متن باز و رایگان است. زبان jQuery به گونه ای طراحی شده است که عمل هدایت به پرونده را آسان‌تر کرده باشد. می توان با آن حرکات انیمیشن ایجاده کرده و از رویدادهای صفحه استفاده کرد و مهم تر از همه می توان نرم افزارهایی مبتنی بر Ajax را ایجاد نموده و توسعه داد.

jQuery هم‌چنین این اختیار را به برنامه نویسان می دهد که افزونه هایی برای کتابخانه JavaScript ایجاد کنند. جدا از این‌ها، jQuery به توسعه دهندگان این اختیار را می‌دهد که تکه برنامه‌های سطح پایین مبادله‌ای (مانند ارتباط مرورگر با کاربر) و یا انیمیشن و حتی افکت‌های پیشرفته و سطح بالا و اشیاء فرضی را ایجاد کنند. به‌کارگیری همه‌ی این اجزای jQuery کمک می‌کند تا صفحات وب قدرتمند و پویا، ساده تر ایجاد شوند.

jQuery شامل ویژگی‌های زیر است:

* دسترسی به عناصر موجود در پرونده‌ها و تغییر در آن‌ها.
* کنترل آسان و قدرتمندتر رویدادها (Events).
* تغییر در CSS.
* ایجاد افکت و حرکات انیمیشنی.
* توسعه‌ی افزونه‌ها.
* تولید برنامه‌های کوچک سودمند.

## 2.5.1: معرفی دیگر ابزارهای JavaScript استفاده شده

در این پروژه برای زیباسازی و هم‌چنین سهولت دسترسی برای کاربر از چند ابزار JavaScript متن باز که توسط توسعه دهندگان دیگر توسعه داده شده بود، استفاده کردیم که در ذیل معرفی می‌شوند:

Clockpicker

زمانی که از کاربر می‌خواهیم زمان رزرو وقت را وارد کند یا زمانی که صاحب شغل یا همکار زمان کاری خودش را مشخص کند، از این ابزار استفاده می‌کنیم تا زیبایی و دسترسی پذیری سایت بالا برود.

نمونه استفاده:

PersianDatePicker

زمانی که صاحب شغل یا همکار می‌خواهند روز خاصی را در تقویم کاری خود قرار بدهند یا کاربر تاریخ رزرو خود را مشخص می‌کند هم از این ابزار استفاده می‌کنیم و تاریخ شمسی را از کاربر می‌گیریم ولی در BackEnd آن تاریخ را به تاریخ میلادی (مشخصاً نوع DateTime در .Net) تبدیل می‌کنیم تا به همان صورت در پایگاه داده آن را ذخیره کنیم.

Select2

این ابزار را وقتی که صاحب شغل یا همکار می‌خواهد روزهای هفتگی یک ماه آینده خود را به‌صورت دسته‌ای وارد کند استفاده کرده‌ایم. بدین صورت که او روزهای هفته‌ی دلخواه خود را انتخاب می‌کند و ما توسط این ابزار آن را به شکل یک Multiple ListBox به کاربر نشان می‌دهیم و مقادیر آن را در BackEnd می‌خوانیم تا برای ذخیره در پایگاه داده آنها را آماده کنیم.

هم‌چنین در پروژه های وب MVC بصورت پیش‌فرض از برخی ابزارهای اسکریپتی مانند respond.js، modernizr.js، jQuery.Validate و... استفاده می‌شود که بیشترین کاربردهای آن‌ها اعتبارسنجی فرم های ورودی توسط کاربر (نشان دادن هشدار و...)، نمایش صحیح HTML5 و CSS3 در مرورگرهای مختلف است.

## 6.1: Json چیست؟

JSON مخفف JavaScript Object Notation می‌باشد. اجازه دهید با یک مثال بیشتر به توضیح JSON بپردازیم. در مثال زیر شیء user دارای سه رکورد اطلاعات است که شامل نام و نام خانوادگی افراد می‌باشد.

{

“user”:[

{“firstName”:”John” , “lastName”: “Malkovich”},

{“firstName”:”Peter” , “lastName”: “Dinklage”},

{“firstName”:”Anna” , “lastName”: “Karina”}

]

}

JSON جهت نگهداری و انتقال اطلاعات متنی به‌کار می‌رود، تقریبا شبیه XML است ولی نسبت به XML کم‌حجم‌تر و در نتیجه سریع‌تر است و با آن راحت‌تر می‌توان کار کرد.

عموماً از JSON در جاوا اسکریپت استفاده می‌شود ولی از آن‌جایی که ساختار خوب و نسبتاً کم‌حجمی دارد در بسیاری از زبان‌های برنامه‌نویسی دیگر و مخصوصاً در وب سرویس‌ها جهت تبادل اطلاعات به‌صورت متنی استفاده می‌گردد.

یکی از بزرگترین مزیت‌های JSON این است که می‌توانیم به‌صورت مجموعه‌ای از اشیا در کدهای JavaScript به اشیاء داخلی آن دسترسی پیدا کنیم.

## 1.6.1: شباهت‌های JSON با XML

## JSON به‌صورت متن خالص است.

## JSON به راحتی توسط JavaScript قابلیت خوانده شدن دارد.

## JSON به راحتی توسط AJAX قابل انتقال است.

## JSON ساختار درختواره دارد و می‌تواندتقسیم به هزاران شاخه بشود.

## در JSON کلیه‌ی قسمت‌های داخل ساختار قابل خواندن است و نیاز به توضیح اضافه ندارد.

## 1.6.1: تفاوت‌های JSON با XML

## تگ انتها برای هر رکورد ندارد.

## رکوردها کوتاه‌تر هستند.

## سرعت خواندن و نوشتن بیشتر است.

## به‌راحتی بدون هیچ کتابخانه‌ای در JavaScript قابلیت کارکرد با آن وجود دارد.

## از آرایه‌ها استفاده می‌کند.

## هیچ کلمه‌ی رزرو شده‌ای در آن تعریف نشده است. (نگرانی بابت کلمات رزرو شده در زبان‌های برنامه نویسی را اینجا نداریم.)

## 3.6.1: چرا JSON؟

در نرم‌افزارها و سایت‌هایی که از AJAX استفاده می‌کنیم کار کردن با JSON به مراتب سریع‌تر و آسان‌تر از XML است.

وقتی با XML کار می‌کنیم:

اول باید اطلاعات XML را دریافت کنیم، سپس از XML DOM برای بررسی و جستجوی XML استفاده کنیم. بعد از آن مقادیر مورد نر خود را استخراج کنیم.

ولی وقتی با JSON کار می‌کنیم:

اول باید اطلاعات JSON را دریافت کنیم سپس با استفاده از تابع eval() در JavaScript از مقادیر آن به‌راحتی استفاده کنیم.

## 3.6.1: قوانین JSON

قوانین استفاده از JSON بسیار شبیه به JavaScript است ولی به مهم‌ترین آن‌ها در ساختار JSON اشاره می‌کنیم:

مقادیر به‌صورت نام فیلد/مقدار در ساختار درج شده و بین علامت “ قرار می‌گیرند هم‌چنین بین نام فیلد یا خاصیت و مقدار آن از : استفاده می‌شود.

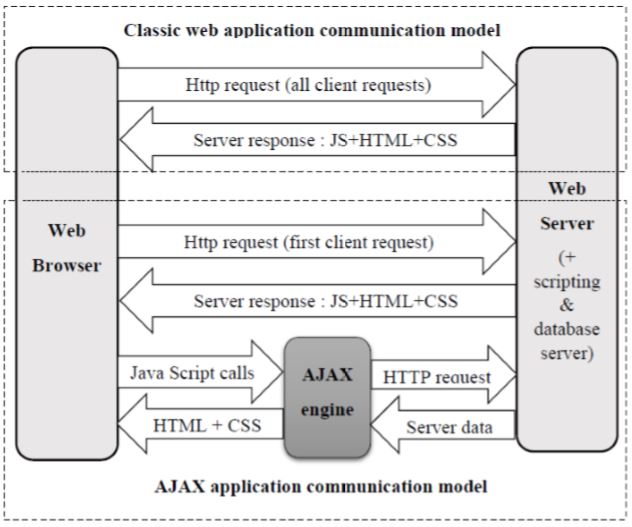
بین رکوردها از کاما (,) استفاده می‌شود. از } و { نیز برای ابتدا و انتهای رکوردها و اشیاء استفاده می‌شود. و در مقداردهی‌های یک شیء مانند user در مثال بالا از [ و ] استفاده می‌شود.

## 7.1: تکنولوژی AJAX

واژه AJAXبه معنی ترکیب نامتقارن JavaScript و XML است. ماهیت صفحات وب و پروتکل Http به گونه‌ای است که به‌طور معمول وقتی در حال وب‌گردی هستیم، به ازای هر کنش و واکنش میان ما و سایتی که در حال کار کردن با آن هستیم، کل صفحه وب از نو به‌روزرسانی (refresh) می‌شود. Ajax فناوری جدیدی است که تغییر محسوسی را در این سناریو به‌وجود می‌آورد، به این ترتیب که به‌جای بارگذاری مجدد کل صفحه، فقط قسمتی تغییر می‌کند که قرار است اطلاعات جدید را به نمایش درآورد و کلیه عملیات ارسال اطلاعات و دریافت نتایج در پشت صحنه انجام میشود. در نتیجه هیچ‌گاه صفحه سفید و خالی وب در فواصل کنش و واکنش‌های هنگام کار با مرورگر دیده نمی شود و احساسی مشابه تجربه کار با یک نرم‌افزار Desktop به کاربر دست می‌دهد. جادوی AJAX چیزی نیست جز یک فکر بکر و آن هم ترکیب JavaScript و XML در قالب یک موجود نرم‌افزاری جدید. در این پروژه برای مثال در قسمت رزرو وقت از این تکنولوژی استفاده شده است.

## 1.7.1: چگونگی کارکرد AJAX

در برنامه‌های کلاسیک تحت وب ارتباط بین مرورگر و سرور به‌صورت مستقیم و با استفاده از درخواست Http برقرار می‌شود. در مدل کلاسیک با هر درخواست صفحه از طرف کاربر، سرور کلیه کدهای HTML و CSS را یک مرتبه به سمت مرورگر ارسال می‌کند. پس از اینکه کاربر فرم را پر و آن را به سمت مرورگر برمی‌گرداند و این روند ادامه پیدا میکند. اما در مدلی که از AJAX استفاده می‌شود و آن هم در پاسخ به اولین درخواست است، تعدادی فایل JavaScript هم که همان موتور AJAX است به همراه کدهای HTML و CSS که ساختار صفحه را تشکیل می‌دهند، بارگذاری می‌شوند. کلیه‌ی درخواست های بعدی به‌صورت JavaScript بوده و به موتور AJAX ارسال می‌شود. این موتور درخواست اطلاعات را به‌صورت آسنکرون (نامتقارن) به سرور می‌فرستد. بنابراین تنها بخشی از صفحه که مورد نیاز کاربر است مبادله می‌شود. در نهایت موتور AJAX اطلاعات را بدون بارگذاری کل صفحه نمایش می‌دهد. انتقال تنها اطلاعات ضروری به جای کل صفحه بین سرور و مرورگر باعث می‌گردد که میزان پاسخ‌گویی رابط کاربری بالا رود. در شکل زیر به خوبی ارتباط بین مرورگر و سرور و هم‌چنین تفاوت‌های موجود در دو مدل ذکر شده نمایش داده شده است.



## 2.7.1: مشکلات AJAX

گرچه این روش مزایای بسیاری دارد و شکاف بین برنامه‌های تحت ویندوز و برنامه‌های تحت وب بسیار کمتر کرده است، اما مشکلاتی نیز دارد که هنگام کار به این روش، باید به آنها توجه کرد. یکی از مشکلاتی که در مورد AJAX عنوان می‌شود، این است که کاربرد کلید Back در مرورگرها را مختل کرده و این برخلاف عادت کاربران در محیط وب است.البته برای حل مسئله راه‌حل‌هایی پیشنهاد شده که مجال بحث در مورد آن‌ها نیست.

مشکل دیگری که در مورد سیستم‌های بر اساس AJAX وجود دارد استفاده AJAX از JavaScript است و این مسئله که کاربر می‌تواند اجازه‌ی اجرای JavaScript در مرورگر خود را ندهد. پس باید پیش از بارگذاری موتور AJAX از اجازه داشتن اجرای JavaScript روی مرورگر کاربر اطمینان پیدا کرد. مسئله‌ی دیگری که باید به آن توجه داشت، این است که کدهای JavaScript برنامه را هرکسی به‌راحتی می‌تواند ببیند. در نتیجه بهتر است قسمت‌هایی از برنامه را که مربوط به امنیت یا منطق پردازشی برنامه می2شود حتما سمت سرور نگه داشت.

به‌نظر می‌رسد ساخت سیستمی که ترکیب متوازنی از پردازش‌های سمت سرور و یک موتور AJAX قوی در طرف کاربر (Client) باشد ، می‌تواند راه حل بسیار مناسبی باشد.

## 6.1: Font Awesome چیست؟

فونتی است که دارای آیکون‌های بسیار زیادی می‌باشد که با فراخوانی آن در قالب می‌توانید از این آیکون‌ها بهره ببرید‌. از ویژگی‌های مهم این آیکون ها می‌توان به سرعت بارگذاری زیاد سایت و در نتیجه افزایش سرعت و عدم نیاز بهsprite اشاره کرد. استفاده از این فونت دو روش دارد:

روش اول این است که کد زیر را در قالب مورد نظر فراخوانی کنید و سپس کد آیکون هارو در قسمت‌های مورد نظر قرار بدهید.

**نمونه کد:**

<link rel="stylesheet" href="~/Content/css/font-awesome.min.css">

<i class="fa fa-plus-circle"></i>

روش دوم استفاده از CSS است که برای این‌کار نیاز دارید فونت ها را به قالبcss خود اضافه کنید. لازم به ذکر است که ما در این پروژه از روش اول و از نسخه‌ی Font Awesome 4.6.1 استفاده کرده‌ایم. این تکنولوژی بُرداری است و از کیفیت آن کاسته نمی‌شود.

## 7.1: معرفی IDE

محیط یکپارچه توسعه نرم‌افزار (Integrated Development Environment) عبارت‌است‌از محیطی عمدتا گرافیکی که تمام یا شماری از ابزارهای لازم برای توسعه نرم‌افزار (بخش‌هایی یا تمام زنجیره ابزار توسعه) را خود دارد. در IDE دسترسی به ابزارها و اعمال آن‌ها در پروژه جاری تسهیل شده‌است.

امکاناتی که به‌طور معمول در IDE ها وجود دارد:

1. ویرایش و نوشتن کد به ‌صورت پیشرفته با استفاده از امکانات پیشنهاد دهنده اتوماتیک که با نوشتن حرف اول یک دستور نام کامل دستورهایی که وجود دارد لیست می‌شود.
2. نمایش کدها به صورت رنگی و امکان تمییز متغیرها و کامنت‌ها از Syntax زبان.
3. کمک به رفع عیب‌های نرم‌افزار و حل مشکلات آن (Debugging).

در این پروژه هم ما از Visual Studio 2015 استفاده نمودیم تا بتوانیم از امکاناتی هم‌چون MVC 5، Entity Framework و هم‌چنین Razor Syntax در پروژه بهره‌مند شویم.

## 1.7.1: معرفی Visual Studio

Visual Studio نام مجموعه برنامه‌نویسی شرکتMicrosoft است که دارای چند زبان برنامه‌نویسی است.

نرم‌افزار Visual Studio، نرم‌افزاری توسعه‌یافته برای برنامه‌نویسان رایانه است که توسط شرکت Microsoft تولید شده‌است. تمرکز اصلی این نرم‌افزار از اولین نسخه‌های آن تا‌کنون بر روی خصوصیت IDE بودن آن است که به برنامه‌نویس اجازه می‌دهد تا برنامه‌های کاربردی مستقل، وب‌سایت، برنامه‌های کاربردی وب و یا سرویس‌های وب را که بر روی تعدادی از ‌Platformهای پشتیبانی شده توسط Microsoft .NET Framework (البته برای تمام نسخه‌های بعد از Visual Studio 6) هم‌چنین ‌Platformهایی مانند Microsoft Windows servers and workstations, Pocket PC Smartphones و World Wide Web browsers اجرا می‌شوند را به راحتی ایجاد نماید. از‌جمله قابلیت‌های اضافه شده که در آخرین ویرایش ارائه شده‌است می‌توان بهSilverlight اشاره کرد که یک Web Application framework بسیار پیشرفته جهت توسعه نرم‌افزارهای کاربردی تحت‌وب می‌باشد.

Visual Studio یک مجموعه از برنامه‌هایی است که ارتباط بسیار نزدیک با‌هم دارند که Microsoft آن را به توسعه‌دهندگان و برنامه‌نویسان برنامه‌های کاربردی اهدا نمود تا آن‌ها را وادار نماید در محیطی توسعه‌یافته بر روی Platform‌های ویندوز و Net. به ساخت برنامه‌های خود بپردازند. Visual Studio می‌تواند برای نوشتن برنامه‌های کنسولی، ویندوزی، سرویس‌های ویندوز، برنامه‌های کاربردی موبایل، برنامه‌های کاربردی ASP.NET و سرویس‌های وب ASP.NET بنا به انتخاب شما همراه با زبان‌هایی مانند C++, C#, VB.NET, J استفاده شود.

با Visual Studio واقعاً چه کارهایی می‌توان انجام داد؟ در زیر تعدادی از کاربردهایی را که برای تولید آن‌ها می‌توان از Visual Studio استفاده نمود معرفی گردیده‌اند:

: **Console Applications** این کاربرد برای اجرای خطوط دستور البته بدون محیط گرافیکی استفاده می‌شود که از این کاربرد برای برخی از ابزارهای کوچک یا برای اجرا شدن کدها توسط دیگر کاربردها استفاده می‌شود. این دستورها در خط فرمان اجرا می‌شود.

**Windows Forms Applications**: برای برنامه‌های کاربردی ویندوزی که با استفاده از .NET Framework نوشته می‌شوند.

**Windows Services:** سرویس‌ها برنامه‌های کاربردی هستند که در پس زمینه ویندوز اجرا می‌شوند.

**ASP.NET Applications:** یک تکنولوژی قدرتمند که برای طراحی و ساخت صفحات‌وب پویا استفاده می‌شود. (در این پروژه از این نوع Application استفاده نموده ایم.)

**ASP.NET Web Services:** مدل سرویس‌های وب را بطور کامل فراهم نموده تا شما به راحتی و با سرعت سرویس‌های وب را تولید نمایید.

**Windows Mobile Applications**: که می‌تواند بر روی ابزارهایی که شامل Framework هستند مانند Pocket PCها و هم‌چنین ‌Cell Phoneهایی کهMicrosoft Smartphone Platform بر روی آن‌ها اجرا می‌شود، اجرا گردد.

**MFC/ATL/Win32 applications:** شما هم‌چنان می‌توانید برنامه‌های سنتی MFC، ATL یا برنامه‌های Win32را با استفاده از C++ ایجاد نمایید. این برنامه‌ها برای اجرا به .NET Framework نیاز ندارند اما نمی‌توانند از مزایای .NET Framework نیز بهره‌ای ببرند.

**Visual Studio Add-ins:** شما می‌توانید از خود Visual Studio برای ساخت توابعی جدید و قابل اضافه شدن به خود Visual Studio استفاده نمایید.

**کاربردهای دیگر:** Visual Studio هم‌چنین شامل پروژه‌هایی برای توسعه برنامه‌های کاربردی شما، کار با Database ها، ساخت گزارش‌ها و‌.‌.‌. می‌باشد.

لازم به ذکر است در این پروژه از ویرایش Visual Studio Enterprise استفاده شد که در ادامه به برخی ویژگی‌های آن می‌پردازیم.

## 2.7.1: معرفی Visual Studio Enterprise

نسخه Enterprise تمامی قابلیت‌های نسخه Professional را دارد و البته امکانات بیشتری از جمله قابلیت توسعه‌دهی پایگاه‌داده، همکاری تیمی، قابلیت معماری سیستماتیک، سیستم متریک (شامل میزان بهره‌وری CPU، RAM و Network و...)، ابزارهای تست و Report را در داخل خود جای داده‌است.

لازم‌به‌ذکر است نسخه Professional تمامی ‌قابلیت‌های ویرایش Standard را دارد و از Remote Debugging، SQL Server Developer Edition، برنامه‌نویسی موبایل، Server Explorer و‌.‌.‌. برخوردار است. از نسخه 2008 به بعد امکان توسعه‌ی برنامه‌های Office نیز در IDE گنجانده شده‌است.

## 8.1: معرفی پایگاه‌داده مورد استفاده

از آنجایی که شرکتMicrosoft سعی در تولید یک Platform برنامه‌نویسی نموده است، ما بر‌آن شدیم تا از این Platform به بهترین شکل ممکن استفاده کنیم و با توجه به اینکه پروژه توسط ASP.NET طراحی و اجرا می‌شد، از پایگاه‌داده Microsoft SQL Server استفاده کردیم.

## 1.8.1: معرفی Microsoft SQL Server

Microsoft SQL Server یک نرم‌افزار سیستم مدیریت بانک‌های اطلاعاتی است که توسط شرکت Microsoft توسعه داده می‌شود. برخی از ویژگی‌های این سیستم مدیریت پایگاه‌داده‌ها به این شرح است:

* بانک اطلاعاتی رابطه‌ای
* امکان استفاده از Triggerها، Viewها و Stored Procedureها
* پشتیبانی از XML
* بسیار قدرتمند و بدون محدودیت حجم و تعداد رکورد
* پشتیبانی از Full Text Search برای سرعت در بازیابی اطلاعات و استفاده از زبان طبیعی در جستجوها (Query ها)

لازم به ذکر است که در این پروژه از MSSQL Server 2016 بهره بردیم.

## 9.1: ساختار پوشه پروژه

هرکدام از اجزایMVC Framework در پوشه‌ای جداگانه قرار گرفته‌است که این امر سادگی جداسازی بخش‌های مختلف و هم‌چنین زیبایی بصری و خوانایی بیشتر را امکان پذیر می‌سازد. پوشه پروژه در ASP.NET MVC شامل پوشه‌های زیر است:

**App\_Data:** در این پوشه که یک پوشه بسیار محافظت شده از لحاظ امنیتی می‌باشد فایل‌های بسیار مهمی نظیر فایل .mdf و .ldf بانک اطلاعاتی قرار می‌گیرند.

**App\_Start:** در این پوشه فایل‌هایی نظیر RouteConfig که برای تنظیم آدرس‌دهی در پروژه استفاده می‌شود و یا فایل BundleConfig که برای آدرس‌دهی Component‌های مختلف Javascript مانند JQuery و CSS مانند Bootstrap استفاده می‌شود، نگهداری می‌شوند.

**bin:** در این پوشه تمامی فایل‌های اجرایی Visual Studio و ASP.NET قرار دارند.

**Content:** این پوشه نیز حاوی فایل‌های CSS و هم‌چنین تصاویر مورد نیاز در پروژه است.

**Controllers:** در این پوشه تمامی Controller‌های مبتنی بر مدل طراحی MVC نگهداری می‌شوند. لازم‌به‌ذکر است پسوند فایل‌های Controller در ASP.NET، .cs می‌باشد.

**Files:** در این پوشه فایل‌های ارسالی کاربران و دیگر تصاویر استفاده شده در پروژه ذخیره می‌شوند.

**Fonts:** در این پوشه فایل‌های فونت‌های مختلف استفاده‌شده در پروژه از جمله Font Awesome و IRAN Sans قرار گرفته‌است.

**Models:** در این پوشه تمامی Model‌های مبتنی بر مدل طراحی MVC نگهداری می‌شوند. لازم به‌ذکر است پسوند فایل‌های Model در ASP.NET، .cs می‌باشد. با توجه به استفاده ما از Entity Framework و تبدیل اشیاء بانک اطلاعاتی به اشیاء قابل ترجمه در MVC توسط قابلیت‌های Entity Framework هر جدول پایگاه‌داده ما اینجا به یک مدل (یک فایل .cs) تبدیل شده‌است به علاوه مدل‌هایی که بعدها برای ساده‌سازی به پروژه افزده‌ایم. هم‌چنین در این پوشه یک فایل.edmx هم وجود دارد که نمودار رابطه‌ای اشیاء پایگاه‌داده است و می‌توانیم در آن به پایگاه داده برای تغییرات دسترسی پیدا کنیم.

**obj:** در این پوشه هم اطلاعاتی نظیر logهای اجرای پروژه و فایل‌های Cache برای سرعت بخشیدن به اجرای پروژه نگهداری می‌شوند.

**Scripts:** در این پوشه فایل‌های JavaScript مورد‌استفاده در پروژه که با پسوند .js ذخیره شده‌اند قرار می‌گیرند.

**Security:** در این پوشه برخی Controllerهای از‌پیش‌آماده MVC برای مدیریت امنیت وب‌سایت قرار گرفته‌است. برای مثال SessionPersister وظیفه ذخیره کردن Sessionها در هر زمانیکه برای تغییر یا مقداردهی فراخوانی می‌شوند را بر عهده دارد.

**Utilities:** در این پوشه که بعدا به پروژه اضافه کردیم توابعی برای تبدیل تاریخ شمسی‌ای که از ورودی دریافت می‌کنیم به نوع استاندارد DateTime و بالعکس پیاده سازی شده است که برای سهولت هرچه بیشتر دسترسی و ذخیره نوع DateTime استفاده شده اند.

**Views:** در این پوشه تمامی View‌های مبتنی بر مدل طراحی MVC نگهداری می‌شوند. لازم‌به‌ذکر است پسوند فایل‌های View در ASP.NET، .cshtml می‌باشد و به ازای هر Controller یک پوشه درست می‌شود که View‌های مربوط به آن Controller را در خود جای داده‌است. هم‌چنین در پوشه Shared و در فایل \_layout.cshtml نمای کلی Viewهای وب‌سایت را به کمک قابلیت‌های بسیار Razor، ساخته‌شده است. فایل \_Header.cshtml هم برای مدیریت View در قسمت بالایی صفحات ایجاد شده‌است. اصولا هر زمان که می‌خواهیم از قسمتی از View در چند جای پروژه استفاده کنیم در پوشه Shared یک Partial View می‌سازیم و در آینده به راحتی از آن در بقیه Viewهای خود استفاده می‌کنیم. برای مثال برای نشان دادن Input مناسب برای گرفتن تاریخ شمسی از کاربر در پوشه EditorTemplates یک Partial View به نام DateTime.cshtml ساختیم تا هرجا که برای نوع DateTime برای کاربر یک Input نشان دادیم این Partial View را ببیند. هم چنین برای نمایش تاریخ شمسی به کاربر هم همین رویه را در پوشه DisplayTemplates اجرا کردیم تا هر زمان به کاربر خواستیم نوع DateTime را نشان بدهیم تاریخ میلادی داخل سرور را به تاریخ شمسی تبدیل کرده و به‌صورت شمسی به کاربر نمایش دهیم.

**فایل Web.Config:** تنظیمات پیکربندی مربوط به پایگاه‌داده و فایل‌های پیکربندی هسته در این فایل نگهداری می‌شوند.

**فایل Appointer.sln:** این فایل، همان فایل اجرایی Visual Studio است که با اجرای آن به محیط Visual Studio رفته و می‌توانیم تغییرات خود را در پروژه ایجاد نماییم.

**پوشه packages:** در این پوشه تمامی تکنولوژی‌هایی که خود MVC در پروژه به‌صورت خودکار استفاده می‌کند وجود دارند. فایل‌های پکیج‌های nuget و فایل‌های dll و ... از جمله فایل‌هایی است که به‌صورت خودکار در این پوشه توسط MVC ذخیره شده‌اند.

زمان كمیاب ترین و بنابراین پرارزش ترین دارایی انسان است. زمان تنها دارایی غیرقابل جایگزینی است كه انسان در اختیار دارد. از بدو تولد، شمارش معكوس مصرف جبران ناپذیر دارایی زمان (عمر انسان) آغاز می شود. عقب ماندن از جهان را فقط با توقف زمان می توان جبران كرد. که امری غیر ممکن است.

**10.1: خلاصه**

در این فصل به معرفی ASP.net، تکنولوژی Bootstrap و زبان JavaScript پرداختیم. در ادامه به معرفی معماری MVC و مفاهیم اصلی آن پرداخته‌شد و هم‌چنین توضیح مختصری در رابطه با Font Awesome و IDE و پایگاه‌داده مورد استفاده در پروژه شرح داده‌شد. هم‌چنین توضیحات کوتاهی در مورد پیکربندی پوشه‌ی حاوی فایل‌های مختلف پروژه دادیم.

# فصل دوم

لزوم انجام پروژه

# فصل دوم

تحلیل و طراحی

**1.۲: مقدمه**

در این فصل به تحلیل و بررسی سیستم منشی دیجیتال می‌پردازیم.

در این تحلیل و بررسی، ابتدا نیازمندی‌های ابتدایی سیستم را از نظر گذرانده‌ایم. سپس موارد کاربردی سیستم و کاربران آن از طریق نمودار مورد کاربرد نشان داده شده است. علاوه بر این نمودار، ترتیب انجام عملیات سیستم توسط نمودار توالی کار برای نقش‌ها (بازیگران) مختلف سیستم ارائه شده است و در پایان نیز به معرفی کلی پایگاه داده سیستم منشی دیجیتال می‌پردازیم.

هر سیستمی می‌تواند دو جنبه داشته باشد: ایستا و پویا.

یک سیستم تنها زمانی کامل محسوب می‌شود که هر دو جنبه را به‌طور کامل پوشش دهد.

**1.1.۲: مدل‌های فرایند نرم‌افزار[[17]](#footnote-17)**

فرایند نرم‌‌افزاری مجموعه‌ای از فعالیت‌هاست که هدف آن توسعه یا تکمیل نرم‌افزار است. بر اساس نوع نرم‌افزاری که قصد توسعه دادن آن را داریم و هم‌چنین طبیعت پروژه و کاربرد آن مدل خاصی انتخاب می‌شود.

**2.1.۲: مدل آبشاری**

مدل آبشاری یک مدل ترتیبی توسعه و تولید نرم‌افزار است و درآن مراحل تولید به شکل یک جریان مداوم متمایل به سمت پايین است (همانند یک آبشار) که شامل فازهای تحلیل خواسته‌ها، طراحی، پیاده‌سازی، آزمودن و تست کردن، یکپارچه‌سازی و دادن محصول به بازار می‌شود. اغلب گفته می شود ریشه ی اصطلاح آبشاری از مقاله‌ای گرفته شده است که توسط وینستون و. رویس در سال ۱۹۷۰ نوشته شده است. مدیریت و مراحل تکمیل پروژه در این متدولوژ‌ی به سادگی قابل پیاده‌سازی است. زیرا در مرحله اول که مرحله بررسی نیازمندیهای پروژه می باشد، مشتری و تیم برنامه نویسی طی چند جلسه به بررسی نیازمندی‌ها و خواسته‌های پروژه می‌پردازند. سپس نوبت به مرحله طراحی می‌رسد، در مرحله‌ی طراحی افراد طرح کلی پروژه را می‌ریزند و جزئیات پیاده‌سازی مشخص می شود. پس از مرحله‌ی طراحی تیم برنامه نویسی خود را برای پیاده‌سازی آماده می کند. در این مرحله همه‌ی قسمت های کد, پیاده‌سازی می شوند, در انتهای این مرحله, ما مرحله یکپارچه‌سازی را خواهیم داشت یکی از مشکل‌ترین قسمت های انجام پروژه‌ها در این مرحله می باشد زیرا تنوع و گسترد‌گی کار کامل در این مرحله نقش دارد، هر چه میزان گستردگی کار بالاتر باشد سختی یکپارچه‌سازی نیز بیشتر خواهد بود.

**مزایای مدل آبشاری:**

1-فهم این مدل ساده تر است.

2-از نظر تولید مستندات بهتر هستند.

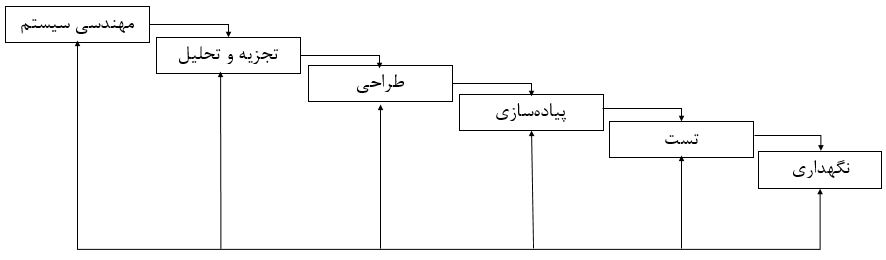
3-مراحل قابل کنترل و بررسی هستند.

**معایب مدل آبشاری:**

1- کار تیم‌های پایانی وابسته به اتمام کار تیم‌های ابتدایی می‌باشد.

2-بررسی بازخورد سیستم توسط کاربران نهایی دارای پروسه‌ی مجزاست و در انتهای کار می‌باشد.

3- گذر از مراحل بصورت نوبتی و وقت‌گیر می‌باشد.



**3.1.۲: تکنیک‌های نسل چهارم**

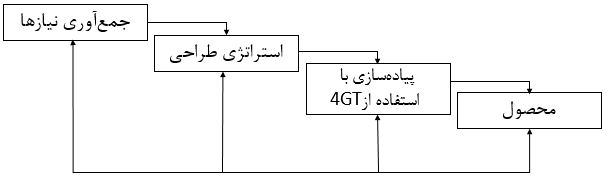
تکنیک‌های نسل چهارم شامل محدوده‌ی وسیعی از ابزارهای نرم‌افزاری هستند که همگی یک ویژگی مشترک دارند: مهندس نرم‌افزار را قادر می‌سازند تا برخی از خصوصیات نرم‌افزار را در سطح بالایی تعریف کند. سپس این ابزار بر اساس تعریف داده شده، کد نرم افزار را تولید می‌کنند.

در این شکی نیست که هر چه سطحی که نرم‌افزار در آن تولید شود بالاتر باشد سرعت ساخت برنامه بیشتر خواهد بود و روش 4GT نیز بر این نکته تمرکز دارد که نرم‌افزار را با استفاده از زبانی خاص (نزدیک‌تر به زبان طبیعی، مثلا با استفاده از انتخاب ویژگی‌ها توسط کاربر(مانند Scaffolding در ASP.NET MVC)) و یا علائم گرافیکی بتوان تشریح نمود.

در محیط‌های توسعه نرم‌افزار کنونی که از 4GT حمایت می‌کنند ابزارهای زیر (و یا بعضی از آن‌ها) وجود دارد:

* زبان‌های غیر رویه‌ای برای Query گرفتن از پایگاه داده (مانند Entity Framework و Linq در Visual Studio)
* تولید گزارش
* دستکاری داده‌ها
* صفحات تعریف و محاوره‌ای
* تولید کد
* قابلیت‌های گرافیکی بالا
* تولید خودکار کد HTML در ابزارهای حرفه‌ای ساخت وب‌سایت (مانند Razor در Visual Studio)

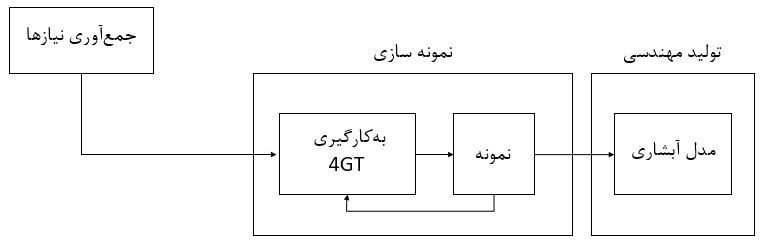
ترتیب عملیات در الگوی 4GT به‌صورت زیر است:



**4.1.۲: ترکیب الگوها**

بسته به کاربرد می‌توان برای توسعه یک نرم‌افزار از ترکیب مدل‌های قبلی استفاده نمودمثلا اگر خصوصیات نرم‌افزار را نتوان به درستی شناخت ابتدا با روش 4GT (4th Generation Techniques) یک نمونه‌سازی سریع انجام می‌گیرد. پس از مشخص شدن جزئیات برای داشتن یک محصول مطمئن و باکیفیت از مدل آبشاری استفاده شود زیرا این مدل شروع هر فاز با تست و اطمینان از فاز قبلی آغاز می‌شود).

شکل زیر فرایند دقیق این کار را نشان می‌دهد.



لازم به ذکر است در این پروژه ما از این مدل فرایند نرم‌افزار استفاده کردیم.

**5.1.۲: نمودارهای رفتاری**

اين گونه نمودار ها با استفاده از اجزای خود بر هم کنش اشياء درون سيستم را نشان مي دهند و رفتار سيستم را توصيف می كنند. در واقع نمودارهای کلاس یک دید ایستا از کلاس ها در سیستم فراهم می کنند ولی نمودارهای رفتاری یک دید پویا فراهم می‌نمایند و نشان می‌دهند که سیستم و کلاس های آن در آینده چگونه تغییر می کنند.

دید ایستا به تحلیل‌گر کمک می کند که با مشتری ارتباط برقرار کند. دید پویا کمک می‌کند که یک تحلیل‌گر با یک تیم از توسعه دهندگان ارتباط برقرار نماید و به توسعه‌دهندگان کمک می‌کند که برنامه ها را ایجاد نمایند.

نمودارهای رفتاری تصویری از جنبه‌ی پویای سیستم ارائه می دهد. اگر بخواهیم جنبه‌ی پویای یک سیستم را دقیق‌تر توضیح دهیم، باید بگوییم که جنبه‌‌‌ی پویای سیستم همان بخش‌های در حال تغییر و حرکت سیستم می باشد.

به طور کلی، UML پنج نوع نمودار رفتاری ارائه می‌دهد که در زیر آن‌ها را مشاهده می کنید:

1. نمودار مورد کاربرد (Use case diagram)

2. نمودار توالی (sequence diagram)

3. نمودار همکاری (collaboration diagram)

4. نمودار حالت (State diagram)

5. نمودار فعالیت (Activity diagram)

به‌علت این که دو مورد از نمودارها بیشتر مورد توجه قرار می‌گیرند و ضمناً بقیه نمودارها نیز از روی آن‌ها قابل استخراج هستند در ادامه به بررسی مواردی از دو نمودار مورد کاربرد و توالی خواهیم پرداخت.برای مثال نمودار همکاری، همان نمودار توالی است بدون این که فاکتور زمان و ترتیب انجام کارها در آن اهمیت داشته باشد. یا مثلا نمودار فعالیت از لحاظ تحلیلی ارزش چندانی ندارد چون نمودار توالی تقریباً آن را پوشش می‌دهد.

## 2.۲: نیازمندی‌ها

## خروجی فرایند مهندسی سیستم تعریفی از یک سیستم کامپیوتری یا محصول است. در این مرحله نیز این مشکل وجود دارد که چگونه مطمئن شویم که تعریف ارائه شده از سیستم نیازهای مشتری را برطرف می‌کند و انتظارات او را رفع می‌سازد. برای این منظور نیازمند به طی فرایند مهندسی محصول هستیم. این فرایند مکانیزم‌های مناسب را فراهم می‌آورد تا تشخیص دهیم مشتری چه می‌خواهد، نیازهای تحلیل چیست، یک راه حل معقول کدام است و ابهامات نیازمندی‌ها در کجاست.

## مهندسی نیازمندی ها شامل مراحل زیر است:

## استخراج نیازمندی‌ها: اهداف سیستم و یا محصول تعیین می‌گردد و نیز این‌که چه چیزی انجام گیرد، سیستم و یا محصول چگونه نیازهای تجاری را رفع می‌کند و بر اساس پایه ای روزانه کار می‌کند.

## تحلیل و مذاکره‌ی نیازمندی‌ها: هنگامی که نیازمندیها جمع آوری شدند عمل تحلیل روی آن‌ها انجام می‌گیرد. تحلیل، نیازمندی‌ها را در زیر دسته‌هایی خاص طبقه‌بندی می‌کنند، ارتباط هرکدام را با دیگری بررسی نموده، جامعیت و ابهامات آن‌ها را تست و نیازمندی‌ها را بر اساس نیاز مشتری اولویت بندی می‌کند.

## تعریف نیازمندی‌ها: قالب استانداردی برای نمایش نیازمندی‌ها که جامعیت آن‌ها حفظ شود ایجاد می‌گردد.

## مدل‌سازی سیستم: بر اساس تعریف ایجاد شده از سیستم، یک مدل از آن ساخته می‌شود.

## اعتبارسنجی نیازمندی‌ها: نیازمندی‌ها برای وجود ابهامات مورد آزمایش و بررسی دقیق قرار می‌گیرند.

## مدیریت نیازمندی‌ها: مجموعه‌ای از فعالیت‌ها را تعریف می‌کند که باعث می‌شوند تیم پروژه بتواند تعیین، کنترل و ردگیری نیازمندی‌ها و تغییرات آن‌ها را در هر زمان مدیریت کند.

## هنگامی‌که نیازمندی‌ها تعیین شدند، جدول ردیابی تشکیل می‌شود. این جدول هرکدام از نیازمندی‌های تعریف‌شده را به یک یا چند جنبه از سیستم یا محیط ربط می‌دهد. به گونه‌ای که نباید در آخر هیچ‌کدام از جنبه‌های سیستم بدون نیازمندی(های) متناظر بماند و هم‌چنین نیازمندی‌ای باقی نماند که به جنبه‌ی سیستم متناظر متصل نشده باشد.

## 1.2.۲: نیازمندی‌های سیستم منشی دیجیتال

|  |  |
| --- | --- |
| ردیف | نیازمندی |
| ۱ | سیستم باید دارای یک تقویم فارسی برای رزرو وقت باشد. (همگی زمان‌های نمایش داده شده به کاربران باید شمسی باشند) |
| ۲ | سیستم باید دارای صفحه‌ی ثبت نام مناسب باشد. |
| ۳ | سیستم باید قابلیت رزرو وقت برای کاربر از صاحب شغل در زمان‌های کاری صاحب شغل را داشته باشد. (بدون برخورد زمانی رزروهای کاربران با یکدیگر) |
| ۴ | سیستم باید دارای صفحه‌ی ورود کاربران باشد. (اهراز هویت) |
| ۵ | سیستم باید دارای قابلیت جستجوی شغل بر اساس عنوان شغل، نوع شغل و شهر را داشته باشد. |
| ۶ | سیستم باید طوری طراحی شود که برای کاربر ساده ترین شکل ممکن را داشته باشد. (UI و UX مناسب و درخور) |
| ۷ | سیستم باید قابلیت های پیشرفته‌ی وارد کردن زمان‌های کاری به‌صورت دسته ای یا تکی را به صاحبان مشاغل بدهد. . (بدون افزودن زمان کاری مشتری قادر به رزرو وقت نخواهد بود) |
| ۸ | سیستم باید قابلیت ویرایش اطلاعات کاربری و تغییر نقش را به کاربر بدهد. |
| ۹ | سیستم باید قابلیت لغو ملاقات را هم برای مشتری و هم برای صاحبان مشاغل فراهم نماید. |
| 10 | سیستم باید به مشتری این امکان را بدهد که اطلاعات مربوط به شغل مورد نظر و هم‌چنین زمان‌های ملاقات صاحب شغل و همکاران آن شغل را مشاهده کند. |
| ۱۱ | سیستم باید به صاحب شغل این امکان را بدهد که با تعریف یک کد ثبت نام، آن را به همکاران خود بدهد تا آن‌ها با استفاده از آن بتوانند در سایت برای همکار شدن در آن شغل ثبت نام کنند. |
| ‍۱۲ | سیستم باید به صاحب شغل توانایی حذف همکاران را بدهد. |
| ‍13 | سیستم باید به صاحب شغل و همکار این قابلیت را بدهد که بتوانند زمان‌های کاری‌ای که قبلاً به سیستم اضافه کرده اند را تک تک حذف (لغو) کنند. |
| ‍14 | سیستم باید به صاحبان شغل و همکاران این امکان را بدهد که برای خود چندین سرویس (شامل نام سرویس، زمان سرویس‌دهی و شرح مختصر) تعریف نمایند تا مشتری‌ها به‌راحتی سرویس خود را انتخاب نمایند. (بدون افزودن سرویس مشتری قادر به رزرو وقت نخواهد بود) |

## 3.۲: کاربران سیستم

## نمودار مورد کاربرد کاربران قبل از وارد شدن به سیستم به‌صورت زیر است:

## C:\Users\ebine\AppData\Local\Microsoft\Windows\INetCacheContent.Word\User bsi.png

## در این پروژه 4 نوع کاربر (یا همان actor) داریم:

## لازم به ذکر است ‌که در طراحی شیء گرا رابطه ارث بری is-a بین actorها را می‌توان تعریف کرد. در شکل زیر این رابطه را با استفاده از پیکان‌هایی به سمت کاربر سیستم (که در بالای آن‌ها مثلث‌های توخالی وجود دارند) تعریف نموده‌ایم.

## 

## 1.3.۲: کاربر ]عادی[ (User)

## محدوده عملیات کاربر عادی محدود به دو کنترلر HomeController و MainController می‌باشد. کاربر عادی می‌تواند از بین مشاغل جستجو کند و از JobCorp مورد نظر در آن شغل وقت بگیرد. هم‌چنین می‌تواند لیست Reservationهای خودش را به راحتی مشاهده کند. هم‌چنین می‌تواند از بین لیست Reservation خود یکی از آن‌ها را انتخاب نموده و آن را کنسل کند یا جزئیات آن را مشاهده کند.

## C:\Users\ebine\AppData\Local\Microsoft\Windows\INetCacheContent.Word\User (1).png

## 2.3.۲: همکار (JobCorp)

## یک همکار تمامی امکانات یک کاربر عادی از جمله امکان رزرو و مشاهده لیست Reservation را خواهد داشت. هم‌چنین می‌تواند زمان‌های کاری خودش را برای شغلی که در آن ثبت شده است را وارد کند و لیست Appointments (را که مخصوص قرارهای کاری‌اش است) را ببیند. هم‌چنین توانایی کنسل کردن قرارهای کاری را دارد و می‌تواند به راحتی جزئیات آن را مشاهده کند.

## شکل زیر نمودار مورد کاربرد مخصوص کارهایی است که اکتور همکار می‌تواند انجام دهد. در بالاترین پیکان سمت چپ منظور از include این است که شاید JobCorp صاحب شغل باشد. اگر JobCorp خود صاحب شغل باشد نیازی به Enroll کردن ندارد. هم‌چنین چند قابلیت دیگر نیز دارد که در نمودار بعدی فقط آن‌ها را نشان خواهیم داد. تمامی کارهای این نمودار جز Enroll کردن، توسط صاحب شغل هم قابل انجام است.

## C:\Users\ebine\AppData\Local\Microsoft\Windows\INetCacheContent.Word\JobCorp (1).png

## 3.3.۲: صاحب شغل (JobOwner)

## صاحب شغل تمام امکانات نقش‌های همکار و کاربر عادی را دارد. هم‌چنین توانایی عوض کردن مشخصات شغل مورد نظر و مدیریت همکاران(از جمله حذف همکاران و تعریف کلید ورود برای همکاران جدید) را داراست.

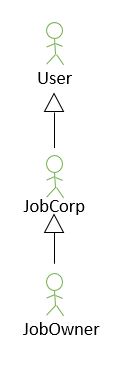
## شکل زیر نمودار مورد کاربرد مخصوص کارهایی است که اکتور صاحب شغل می‌تواند انجام دهد. البته این کارها به جز کارهایی است که در نمودار بالا معرفی شده‌اند. همان‌طور که پیش‌تر گفتیم صاحب شغل تمامی عملیات همکار را می‌تواند انجام دهد. وجود include قبل از "مشاهده لیست همکاران" نشانگر این است که اگر صاحب شغل هنوز همکاری نداشته باشد نمی‌تواند لیستی را هم مشاهده کند.

## C:\Users\ebine\AppData\Local\Microsoft\Windows\INetCacheContent.Word\JobOwner (1).png

## 4.3.۲: مدیر (Admin)

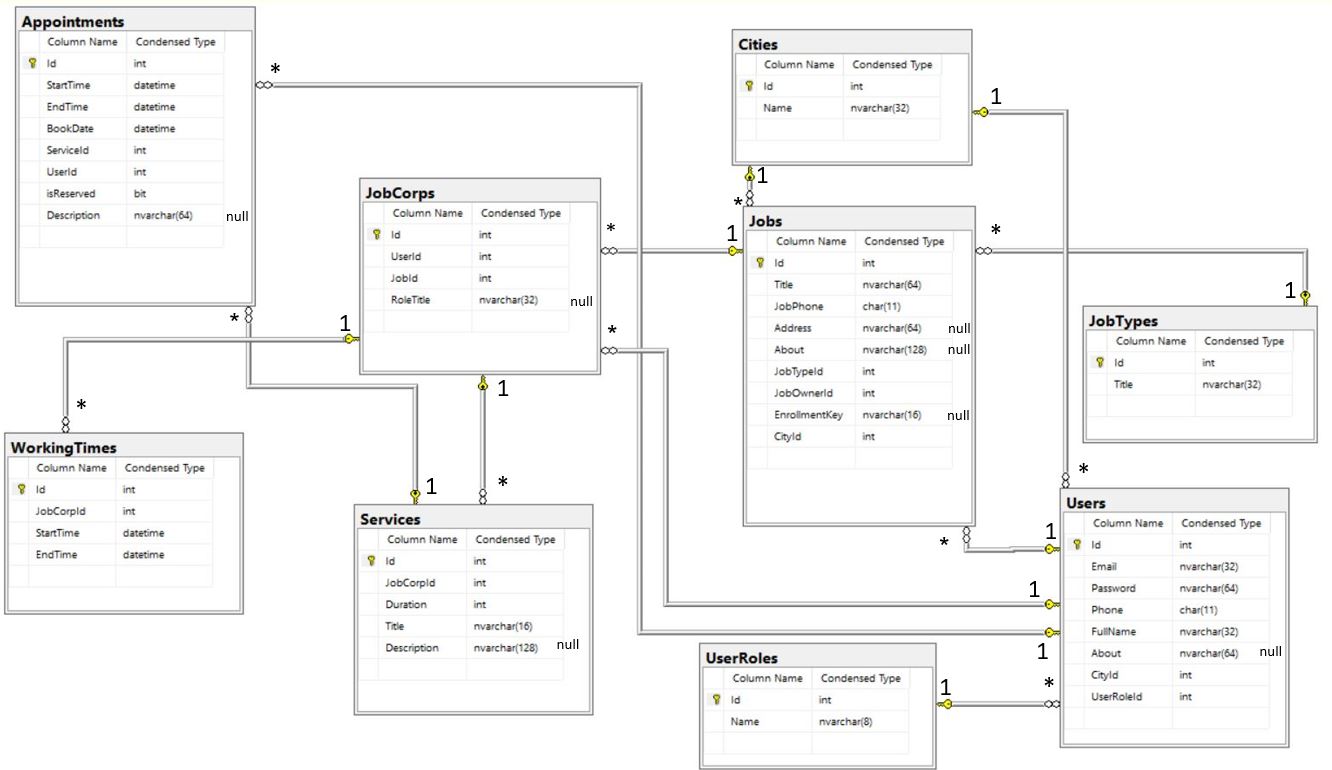
## مدیر نقش نظارت بر سایت را برعهده دارد و می‌تواند نوع شغل و شهرهای جدید تعریف کند.

در این سیستم نقش‌ها سلسله مراتب دارند و برای مثال یک JobCorp می‌تواند مانند یک کاربر عادی از JobCorp دیگری وقت رزرو کند ولی کاربر عادی مثلا نمی‌تواند وقت کاری خود (جزو شرح وظایف JobCorp) را به سیستم بیفزاید. شکل زیر بهتر مفهوم این سلسله مراتب را تبیین می‌نماید. ضمنا نماد پیکان با مثلث توخالی همان رابطه is-a می‌باشد. یعنی برای مثال یک JobCorp یک User می‌باشد و تمام ویژگی‌ها و قابلیت‌های یک User را دارد.



**4.۲: پایگاه داده**

در زیر نمودار ER پایگاه داده به شکل مبسوط آورده شده است:

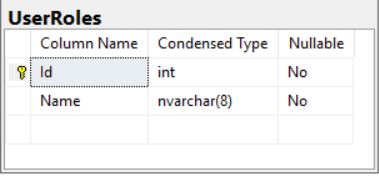


**1.4.۲: جدول‌های پایگاه داده**

حال تمامی جدول‌های واقع شده در پایگاه داده را مورد بررسی قرار می‌دهیم:

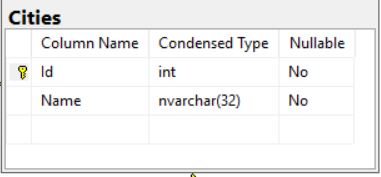
1. **جدول UserRoles:**

در قسمت Name، اسم نقش کاربر نوشته می‌شود. ضمنا این جدول تنها شامل چهار مقدار User، JobOwner ، JobCorp و Admin می‌باشد. در جدول Users کلید خارجی این جدول به نام UserRoleId وجود دارد.



1. **جدول Cities:**

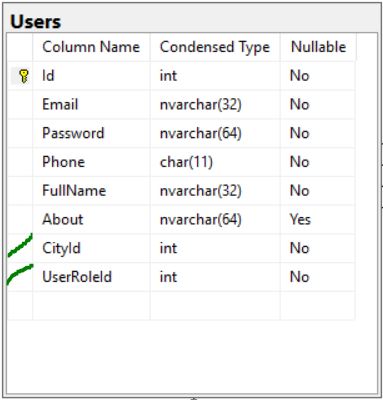
در قسمت Name، نام شهر نوشته می‌شود. در جدول‌های Users و Jobs کلید خارجی این جدول به نام CityId وجود دارد.



1. **جدول Users:**

در قسمت Email، ایمیل کاربر، در قسمت Password رمز عبور کاربر، در قسمت Phone شماره موبایل کاربر، در قسمت FullName نام و نام خانوادگی کاربر، در قسمت About کاربر می‌تواند توضیحاتی در مورد خود نیز بدهد (فیلد Nullable می‌باشد). هم‌چنین دو کلید خارجی CityId و UserRoleId نیز به ترتیب به دو جدول Cities و UserRoles متصل می‌شوند.

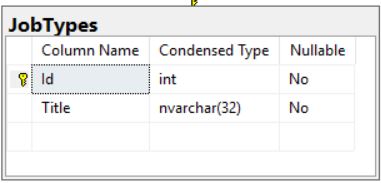
جدول‌های Jobs، JobCorps و Appointments هم با کلید خارجی UserId به این جدول متصل می‌شوند.



1. **جدول JobTypes:**

در قسمت Title عنوان نوع شغل مورد نظر قرار می‌گیرد. (برای مثال پیرایش، تجارت یا پزشکی)

در جدول Jobs کلید خارجی این جدول به نام JobTypeId وجود دارد.

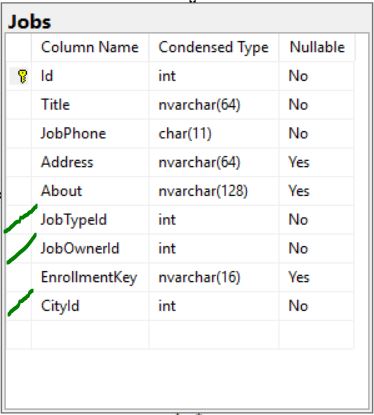


1. **جدول Jobs:**

در قسمت Title عنوان شغل، در قسمت JobPhone شماره تلفن محل کار، در بخش Address که Nullable هم می‌باشد آدرس محل کار، در بخش About شرح مختصری درباره‌ی شغل مورد نظر می‌تواند داده شود. (Nullable) هم‌چنین EnrollmentKey نیز دربردارنده رمز ورود برای همکاران دیگر در شغل است که ابتدا به‌صورت پیش‌فرض Null فرض می‌شود و سپس صاحب شغل می‌تواند در صورت تمایل آن را اضافه کند.

ضمنا سه کلید خارجی CityId و JobOwnerId و JobTypeId نیز به ترتیب به دو جدول Cities و Users و JobTypes متصل می‌شوند. هر موجودیت Job تنها یک صاحب شغل دارد که شناسه کلید خارجی JobOwnerId که به جدول Users متصل شده آن را تعیین می‌کند.

جدول‌JobCorps هم با کلید خارجی JobId به این جدول متصل می‌شوند.

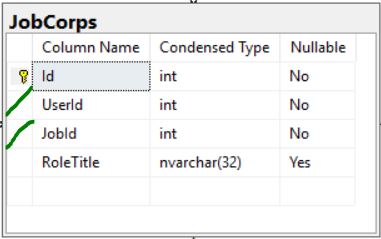


1. **جدول JobCorps:**

در قسمت RoleTitle همکار می‌تواند نقشش را بنویسد. به‌صورت پیش‌فرض به JobOwner در RoleTitle مقدار صاحب شغل و برای JobCorpای بعدا اضافه شده مقدار همکار درنظر گرفته می‌شود.

هم‌چنین دو کلید خارجی JobId و UserId نیز به ترتیب به دو جدول Jobs و Users متصل می‌شوند.

جدول‌های Services و WorkingTimes هم با کلید خارجی JobCorpId به این جدول متصل می‌شوند.

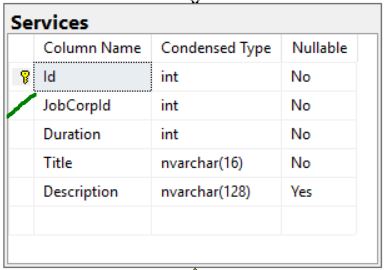


1. **جدول Services:**

در قسمت Title همکار مورد نظر عنوان خدمت خود را می‌نویسد و در قسمت Description که Nullable هم هست می‌تواند درباره آن توضیح دهد(قیمت بگذارد و ...). در قسمت Duration هم وقتی که برای آن خدمت خواهد گذاشت را می‌نویسد (به دقیقه).

هم‌چنین این جدول با کلید خارجی JobCorpId نیز به جدول JobCorps متصل می‌شود.

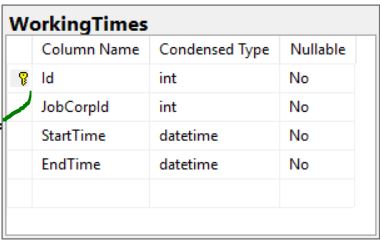
جدول‌ Appointments هم با کلید خارجی ServiceId به این جدول متصل می‌شود.



1. **جدول WorkingTimes:**

در قسمت StartTime که نوع datetime نیز دارد شروع زمان کاری یک روز قرار می‌گیرد.(لازم به ذکر است که در یک روز می‌توان چند زمان کاری هم داشت.)(مثال به زبان ساده: دوشنبه 27 آذر 1395 ساعت 10:00) و قسمت EndTime هم به همین‌گونه است با این تفاوت که پایان زمان کاری را مشخص می‌کند.(ادامه مثال بالا: دوشنبه 27 آذر 1395 ساعت 14:00)

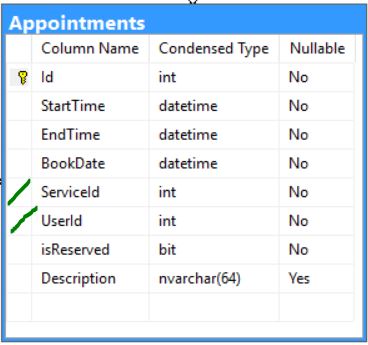
این جدول با کلید خارجی JobCorpId نیز به جدول JobCorps متصل می‌شود.



1. **جدول Appointments:**

در قسمت StartTime کاربر زمان شروع ملاقاتش را ثبت می‌کند. زمان EndTime را بعدا در کنترلر خود مساوی StartTime+Service.Duration می‌کنیم. BookDate زمان رزرو وقت توسط کاربر است. بیت isReserved هر وقت 0 شود بدین معناست که قرار کنسل شده است.(موقع کنسل شدن قرار آن را از رکوردهای Appointments حذف نمی‌کنیم.) در غیر این‌صورت بیت isReserved یک خواهد بود. قسمت Description نیز Nullable است و کاربر می‌تواند توضیحاتی به همکاری که از او وقت گرفته بدهد.

هم‌چنین دو کلید خارجی ServiceId و UserId نیز به ترتیب به دو جدول Services و Users متصل می‌شوند. (این که User از کدام JobCorp وقت گرفته به راحتی از Service انتخابی قابل تشخیص خواهد بود.)



(این فصل فعلا ناقص است. در آینده کلاس دیاگرام برنامه، دو سه نمودار توالی برای مورد کاربردهای مهم سیستم به این فصل افزوده خواهد شد.)

# فصل دوم

پیاده سازی

## ۱.۳: مقدمه

با توجه به این که در طراحی سیستم منشی دیجیتال از معماری MVC استفاده شد و در پیاده سازی آن از ASP.NET MVC 6 بهره جستیم، در بخش پیاده سازی به تشریح هر سه قسمت معماری MVC یعنی مدل‌ها، کنترلرها و Viewها می‌پردازیم:

لازم به ذکر است که در این پروژه ابتدا پایگاه داده طراحی شد و سپس با استفاده از روش Database First دسترسی به پایگاه داده ها را با استفاده ازEntity Framework فراهم نمودیم.

## 2.۳: بخش مدل‌ها

نظر به این‌که در این پروژه از Entity Framework و روش Database First استفاده نموده ایم دو نوع مدل را اینجا تشریح خواهیم کرد:

* مدل‌های استخراج شده از پایگاه داده
* مدل‌های دیگر

ابتدا به بررسی مدل‌هایی می‌پردازیم که از پایگاه داده استخراج شده‌اند:

**۱.2.۳: مدل‌های استخراج شده از پایگاه داده**

در این پروژه سعی کردیم با استفاده از قابلیت های Entity Framework پایگاه ‌داده ای را که برای سیستم قبل از پیاده‌سازی طراحی کرده‌ بودیم را به سادگی روی پروژه پیاده‌سازی نماییم. در این عملیات هر جدول در پایگاه‌داده تبدیل به یک کلاس مدل می‌شود و هر فیلد جدول پایگاه‌داده هم به یک attribute به همراه setterها و getterهایش در همان کلاس تبدیل می‌شود.

ضمناً همه این کلاس ها زیرمجموعه فایل Appointer.edmx می‌شوند که وقتی می‌خواستیم پایگاه‌داده را به مدل تبدیل کنیم توسط Entity Framework ساخته شده‌بود. ضمنا یک فایل دیگر به اسم Appointer.edmx.sql هم داریم که دستورات sql درون آن قرار دارند.

برای رعایت اصل ایجاز این کلاس‌ها و Attributeهای آن‌ها را در اینجا تشریح نمی‌کنیم چون در فصل قبل و در طراحی پایگاه داده به آن‌ها پرداخته بودیم.

در اینجا تنها به یک مثال از این کلاس‌ها می‌پردازیم و آن را تشریح می‌کنیم:

کلاس User.cs که از جدول Users استخراج شده است به‌صورت زیر توسط Entity Framework تولید شده است:

namespace Appointer.Models

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Web.Mvc;

public partial class User

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public User()

{

this.Appointments = new HashSet<Appointment>();

this.JobCorps = new HashSet<JobCorp>();

this.Jobs = new HashSet<Job>();

}

public int Id { get; set; }

[Required(ErrorMessage = "وارد کردن ایمیل الزامی است")]

[EmailAddress(ErrorMessage = "ایمیل درست وارد کنید")]

[StringLength(32)]

[DataType(DataType.EmailAddress)]

[Display(Name = "ایمیل")]

public string Email { get; set; }

[Required(ErrorMessage = "وارد کردن رمز عبور الزامی است")]

[DataType(DataType.Password)]

[Display(Name = "رمز عبور")]

[StringLength(64, ErrorMessage = "رمز عبور باید حداقل 6 کاراکتر باشد", MinimumLength = 6)]

public string Password { get; set; }

[DataType(DataType.PhoneNumber)]

[Display(Name = "شماره موبایل")]

[Required(ErrorMessage = "وارد کردن شماره موبایل الزامی است")]

[StringLength(11)]

[RegularExpression(@"^09(1[0-9]|3[1-9]|2[1-9])-?[0-9]{7}$", ErrorMessage = "شماره

موبایل خود را با فرمت زیر وارد کنید: 09127404062")]

public string Phone { get; set; }

[Required(ErrorMessage = "این فیلد باید پر شود")]

[Display(Name = "نام و نام خانوادگی")]

public string FullName { get; set; }

[Display(Name = "درباره شما")]

public string About { get; set; }

[Display(Name = "شهر")]

public int CityId { get; set; }

[Display(Name = "وظیفه")]

public int UserRoleId { get; set; }

public String RoleName

{

get

{

return Enum.GetName(typeof(MyUserRole), (MyUserRole)UserRoleId);

}

}

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",

"CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Appointment> Appointments { get; set; }

public virtual City City { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",

"CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<JobCorp> JobCorps { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",

"CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Job> Jobs { get; set; }

public virtual UserRole UserRole { get; set; }

}

}

در همان ابتدای تعریف کلاس و در تابع public User() به معرفی دیگر کلاس‌هایی پرداخته شده که با استفاده از کلید خارجی به کلاس User متصل هستند. برای مثال در خط اول (this.Appointments = new HashSet<Appointment>();) به معرفی کلاس Appointment پرداخته شده تا اگر در آینده این کلاس از کلاس User استفاده نمود با استفاده از این خط به این کلاس دسترسی پیدا کند. هم‌چنین وجود سطر زیر که در آخر کلاس آورده شده هم برای این امر ضروری است:

public virtual ICollection<Appointment> Appointments { get; set; }

هم‌چنین در آخر کلاس به تعریف جدول‌هایی پرداخته می‌شود که در جدول User به سمت آن‌ها کلید خارجی وجود دارد. برای مثال سطر:

public virtual UserRole UserRole { get; set; }

کلاس UserRole را به کلاس User معرفی می‌کند تا در آینده بتوان از محتویات آن کلاس در کلاس User استفاده کرد.

سپس به معرفی فیلدهای مختلف جدول Users پرداخته شده است. در بالای تعریفِ بعضی از فیلدها نوعی حاشیه نویسی انجام شده است که بعدها در طراحی Viewها به ما کمک شایانی خواهند کرد. برای مثال در بالای فیلد Phone این حاشیه‌نویسی‌ها انجام شده‌اند که تک تک به معرفی آن‌ها می‌پردازیم:

[DataType(DataType.PhoneNumber)]

این حاشیه‌نویسی باعث می‌شود تا کامپایلر بفهمد این فیلد را باید از نوع شماره تلفن بگیرد و نوع دیگری را نپذیرد تا در آیند و در ذخیره‌ی آن در پایگاه داده به مشکل نخوریم.

[Display(Name = "شماره موبایل")]

این حاشیه‌نویسی باعث می‌شود تا هر وقت در View از دستور @Html.LabelFor(model => model.Phone) استفاده کردیم به کاربر نهایی عبارت "شماره موبایل" نشان داده شود.

[Required(ErrorMessage = "وارد کردن شماره موبایل الزامی است")]

این حاشیه‌نویسی باعث می‌شود تا هر وقت کاربر برای مثال در فرم ثبت نام این فیلد را خالی بگذارد و بخواهد فرم را submit کند با این خطا مواجه شود.

[StringLength(11)]

این حاشیه‌نویسی باعث می‌شود تا حداکثر طول رشته‌ای که از کاربر گرفته می‌شود را مشخص کنیم. با توجه به این که در ایران شماره‌های موبایل 11 رقمی هستند و نیاز نیست برای این فیلد بیشتر جا در نظر بگیریم تصمیم گرفتیم کاربر نتواند بیشتر از این تعداد رقم را نتواند وارد کند تا در ثبت مشخصات خود سهل انگاری نکند.

[RegularExpression(@"^09(1[0-9]|3[1-9]|2[1-9])-?[0-9]{7}$", ErrorMessage = "لطفا شماره خود را این گونه وارد کنید: 09127404062")]

با استفاده از این حاشیه نویسی از عبارات منظم استفاده کرده‌ایم بدین صورت که چون در ایران همه شماره‌های موبایل با 09 آغاز می‌شوند و پس از این دو رقم 2،1 یا 3 داریم که بعد از آنها یک رقم دلخواه می‌تواند بیاید (رقم چهارم از چپ) و بعد از این رقم‌ها که برای بررسی شماره موبایل ضروری بودند 7 رقم دیگر داریم که دلخواه هستند. این عبارت منظم این‌گونه پیاده سازی می‌شود: ^09(1[0-9]|3[1-9]|2[1-9])-?[0-9]{7}$

(لازم به ذکر است که این عبارات منظم استاندارد شده باید با کاراکتر ^ آغاز شده و با کاراکتر $ پایان یابند.)

در قسمت دوم این حاشیه نویسی (ErrorMessage) اگر کاربر رشته مورد نظر را طوری که ما خواسته بودیم وارد نکند با استفاده از توابع jquery ساخت‌یافته خود MVC پس از رفتن به فیلدهای دیگر به کاربر خطا را نشان می‌دهد تا شماره موبایل را درست وارد کند.

لازم به ذکر است تمام این حاشیه نویسی‌ها که کار ما را در بخش View بسیار ساده‌تر می‌کنند از کتابخانه‌ی System.ComponentModel.DataAnnotations استفاده می‌کنند که در همان ابتدا آن را به کلاس‌ها افزودهایم. هم‌چنین به‌علت پشتیبانی نشدن زبان فارسی در Encoding پیش‌فرض Visual Studio باید این فایل را پس از انجام این تغییرات با Encoding زیر ذخیره نمود:

Unicode (UTF-8 without signature) - Codepage 65001

در این قسمت می‌توانیم برای کلاس صفات دیگری (به جز فیلدهای جدول پایگاه داده) را نیز تعریف کنیم تا در پیاده‌سازی به ما کمک کنند. برای مثال در کلاس Appointment.cs صفت زیر را تعریف کردیم تا بعدا در View بتوانیم به راحتی از تقویم فارسی استفاده نماییم:

[Display(Name = "تاریخ")]

public DateTime myDate { get; set; }

بعدا در کنترلر می‌توانیم روی چنین صفاتی اعمال مختلف انجام دهیم تا آنها را به شکلی که ‌می‌خواهیم در یکی (یا چند تا) از فیلدهای پایگاه داده ذخیره کنیم.

نکات کافی در مورد این نوع از مدل ها گفته شد. حال به بررسی دیگر مدل‌ها خواهیم پرداخت:

### **2.2.۳: مدل های اصلی**

مدل حساب کاربری (Account Model)

این مدل شامل تمامی اطلاعات مربوط به حساب‌کاربری است. در این مدل از اطلاعات مدل User.cs استفاده کرده‌ایم و با استفاده از متدهایی نظیر findAll (که برای تایید کاربران توسط مدیر گروه مورد استفاده قرار می‌گیرد) سعی در راحت‌تر کردن کار AccountController داشته‌ایم.

کلاس AccountModel مانند یک Interface عمل می‌کند چون تابع Constructor آن خالی است و از تعدادی متد به شرح زیر تشکیل شده‌است:

* public User find(string email) : این تابع وظیفه یافتن کاربران بر اساس ایمیل آن‌ها را بر عهده دارد.
* public User findById(int id) : این تابع وظیفه یافتن کاربران بر اساس id آنها را بر‌عهده دارد.
* public User signIn(string email, string fullname, string password): این تابع دو متغیر وارد شده در صفحه SignIn را می‌گیرد و بررسی می‌کند که آیا کاربری با این مشخصات وجود دارد یا نه. در فیلد اول صفحه SignIn کاربر می‌تواند ایمیل یا نام خود را وارد کند که در صورت برابری با رمز عبور وارد شده در رکورد جدول Users در پایگاه داده می‌تواند وارد شود. با استفاده از یک Query ساده‌ی Entity Framework می‌توانیم این کار را انجام دهیم:

return db.Users.Where(acc => (acc.Email.Equals(email) || acc.FullName.Equals(fullname)) && acc.Password.Equals(password)).FirstOrDefault();

* public User signup(User user) : این تابع کاربر را به سیستم اضافه کرده و در پایگاه‌داده ذخیره می‌کند.
* public User EditUser(User user) : در این تابع ویژگی هایی را که کاربر تغییر داده را در پایگاه‌داده تغییر می‌دهیم و در آن ذخیره می‌کنیم.
* public User DeleteUser(User user) : در این تابع کل یک سطر مربوط به یک کاربر را از پایگاه‌داده حذف می‌کنیم و وضعیت پایگاه‌داده را ذخیره می‌کنیم.
* public List<User> findAll() : این تابع لیست تمام کاربران را برمی‌گرداند.
* public List<City> GetCityList() : این تابع با یک Query ساده نام شهرها را از جدول Cities به‌صورت لیست برمی‌گرداند. از این تابع مثلا در صفحه SignUp استفاده می‌کنیم تا لیست شهرها را در قالب یک DropDown List به کاربر نمایش دهیم.
* public List<UserRolePersian> GetUserRolesList() : این تابع لیست تمام نقش‌های مجاز کاربران را برمی‌گرداند. بدیهی است که نباید نقش مدیر هنگام ثبت نام به کاربری نشان داده شود.

مدل HourMinute

این مدل با داشتن تنها یک Attribute به نام public TimeSpan hm {set; get;}دارد و از آن تنها در کنترلر داشبورد JobCorp و در اکشن AddWorkingTime که برای ذخیره‌ی دسته‌ای زمان‌های کاری صاحبان مشاغل و همکاران به کار می‌رود استفاده شده است تا بتوانیم لیستی از زمان‌ها را در اختیار داشته باشیم. لازم به ذکر است نوع TimeSpan برای ذخیره زمان به کار برده می‌شود. مثلا رشته 23:30 را گرفته و آن را به ساعت و دقیقه دلخواه .NET ترجمه می‌کند تا بعدا در نوع DateTime نیز به راحتی بتوانیم آن را به تاریخ مورد نظرمان بیفزاییم.

مدل ClockRangeModel

این مدل با داشتن تنها دو Attribute ساعت شروع و ساعت پایان، برای ذخیره‌ی زمان‌هایی که صاحب شغل یا همکار به‌صورت دسته ای برای زمان‌های کاری خود وارد می‌کند به کار می‌رود. به این صورت که در مدل اصلی WorkingTime یک لیست از این مدل تعریف کرده‌ایم تا بازه‌های زمانی که کاربر تعریف می‌کند را داخل این لیست نگه داریم و سپس در کنترلر آن را به تاریخ (یا تاریخ‌های) انتخاب شده الحاق کرده و به‌صورت استاندارد خودمان در پایگاه داده آن را ذخیره می‌کنیم.

مدل LittleWorkingTime

این مدل همانطور که از نامش برمی‌آید زیرمجموعه‌ای از مدل WorkingTime است که جزو مدل‌هایی است که Entity Framework از روی پایگاه داده ساخته است. از این مدل زمانی استفاده می‌شود که کاربر وارد صفحه رزرو وقت از یک صاحب شغل بشود. وقتی کاربر تاریخ مورد نظرش را انتخاب می‌کند از طریق JSON تاریخ را گرفته و پردازش می‌کنیم تا زمان‌های کاری‌ای که صاحب شغل در آن روز خاص ثبت کرده به کاربر نشان دهیم. هم‌چنین لیست دیگر ملاقات‌های آن روز را به کاربر نشان می‌دهیم تا زمانی را انتخاب نکند که با بقیه زمان‌ها تداخل داشته باشد.

مدل LittleJob

این مدل همانطور که از نامش برمی‌آید زیرمجموعه‌ای از مدل Job است که جزو مدل‌هایی است که Entity Framework از روی پایگاه داده ساخته است. از این مدل زمانی استفاده می‌شود که موقع ثبت نام کاربر نقش خود را همکار انتخاب می‌کند. سپس ما او را به صفحه‌ای هدایت می‌کنیم که او کدی که صاحب شغل به او داده را وارد کند. در زیر این فیلد متنی دکمه‌ای به نام "بررسی" گذاشته‌ایم که کاربر با فشردن آن کد شغلی را به‌صورت JSON و با استفاده از AJAX به سمت سرور می‌فرستد. سرور هم با استفاده از JSON شیء LittleJob را که اطلاعات حیاتی شغلی که EnrollmentKey آن برابر با کد ارسالی از طرف کاربر باشد را روی صفحه نمایش می‌دهد. اگر هم که کد وارد شده اشتباه باشد به کاربر اطلاع می‌دهیم. در صورت مطابقت EnrollmentKey با کد وارد شده کاربر می‌تواند به عنوان همکار آن شغل در سیستم ثبت نام نماید.

مدل MyUserRole

این مدل حاوی یک Enum است که نقش‌های کاربران در پایگاه داده را در آن تعریف کرده و به هریک عددی اختصاص داده ایم. با توجه به این که در پایگاه داده به همین صورت (انگلیسی) نام این نقش‌ها را ذخیره کرده‌ایم، این مدل را صرفاً برای ایجاد ارتباط بین پایگاه داده و View کاربر تعریف کرده ایم و با استفاده از مدل بعدی این اعداد که به هر نقش تخصیص داده شده را به اسامی فارسی تخصیص می‌دهیم. در View با استفاده از یک Cast این تبدیل را انجام می‌دهیم و در DropDown List نقش کاربر (در صفحه ثبت نام) نقش فارسی را به کاربر نمایش می‌دهیم که البته با گذاشتن یک شرط در کنترلر نقش مدیر یا Admin را از اسامی نمایش داده شده خارج می‌کنیم. Enum داخل این مدل به‌صورت زیر تعریف می‌شود:

public enum MyUserRole

{

User = 1,

JobOwner = 2,

JobCorp = 3,

Admin = 4

}

مدل UserRole\_Persian

این مدل را در بالا توضیح دادیم. Enum داخل این مدل به‌صورت زیر تعریف می‌شود:

public enum UserRole\_Persian

{

کاربر = 1,

کارفرما = 2,

همکار = 3,

مدیر = 4

}

‌

در این قسمت، مدل‌های سیستم را یک بررسی اجمالی نمودیم. مدل‌ها قلب تپنده‌ی پروژه هستند. حال نوبت بخش کنترلر است که وظیفه‌ی اصلی آن ایجاد ارتباط بین بخش مدل و View می‌باشد:

## 3.۳: بخش کنترلر‌ها

حال به بررسی کنترلرها می‌پردازیم:

### **۱.3.۳: Account Controller**

1. **اکشن SignUp:**

در قسمت GET این اکشن لیست‌ شهرها، لیست انواع شغل‌ها و لیست نقش‌های کاربر را در ViewBagهای مربوطه می‌گذاریم تا در View بتوانیم آن‌ها را فراخوانی کنیم. در قسمت POST این اکشن نیز اطلاعات را در جدول Users پایگاه داده ذخیره می‌کنیم و سپس با توجه به نقش انتخابی کاربر او را به صفحه بعدی هدایت می‌کنیم.

1. **اکشن SignIn:**

در قسمت POST این اکشن ابتدا چک میکنیم که اطلاعات با هم مطابقت داشته باشند و در صورت مطابقت چند شرط را چک میکنیم تا مسیر هدایت کاربر را مشخص کنیم. ابتدا با استفاده از Session نقش کاربر، نقشش را می‌فهمیم. سپس با استفاده از Queryهای زیر و استفاده کردن از آن‌ها در شرط‌ها کاربر را به صفحاتی هدایت می‌کنیم که بهترین صفحه‌ی پیشنهادی سیستم برای او هستند. مثلا اگر یک JobCorp هنوز سرویسی اضافه نکرده باشد او را به کنترلر Services اکشن Create هدایت می‌کنیم تا برای خود سرویسی بسازد تا مشتری بتواند از او وقت بگیرد. یا مثلا کاربر عادی اگر قبلا وقتی گرفته باشد او را به صفحه‌ی Reservations از کنترلر Main می‌فرستیم تا ببیند چه وقت‌هایی را رزرو کرده است. اگر هنوز وقتی نگرفته باشد، او را به صفحه جستجوی مشاغل (Main/Jobs) هدایت می‌کنیم.

var hasReservationsToday = db.Appointments.Any(p => p.UserId == SessionPersister.UserId && p.StartTime >= DateTime.Now && p.StartTime <= dt);

var hasReservations = db.Appointments.Any(p => p.UserId == SessionPersister.UserId);

var hasAppointmentsToday = db.Appointments.Any(a => a.Service.JobCorp.UserId == SessionPersister.UserId && a.StartTime >= DateTime.Now && a.StartTime <= dt);

var hasAppointments = db.Appointments.Any(p => p.Service.JobCorp.UserId == SessionPersister.UserId);

var hasServices = db.Services.Any(p => p.JobCorp.UserId == SessionPersister.UserId);

var hasWorkingTimes = db.WorkingTimes.Any(p=>p.JobCorp.UserId==SessionPersister.UserId);

نام متغیرها برای درک کار آن‌ها کافی است. لازم به ذکر است دو Query اول برای چک کردن داشتن وقت ملاقات‌های کاربر عادی و چهار Query بعدی برای کارهای مختلف JobCorpها طراحی شده اند.

1. **اکشن SignOut:**

در این اکشن Sessionها را برابر Null می‌کنیم و سپس دوباره کاربر را به صفحه SignIn می‌فرستیم.

1. **اکشن Edit کاربر:**

بسیار شبیه به اکشن SignUp عمل می‌کند. با این تفاوت که در متد POST آن به‌جای ذخیره کردن یک رکورد در پایگاه داده رکورد مربوط به کاربر مورد نظر را تغییر می‌دهیم.

### **2.3.۳: Home Controller**

این کنترلر برای صفحات اصلی وبسایت که در همه وبسایت‌ها موجود است View متناظرشان را برمی‌گرداند. کاربری که برای بار اول به وبسایت بیاید به صفحه Index این کنترلر هدایت خواهد شد. لازم به ذکر است در اکشن‌های این کنترلر به جز برگرداندن View متناظر کار دیگری صورت نمی‌گیرد.

1. **اکشن Index (صفحه اصلی وبسایت)**
2. **اکشن About**
3. **اکشن Contact**

### **3.3.۳: Job Controller**

این کنترلر را به‌صورت مستقیم و از طریق Scaffolding مدل Job که از جدول Jobs در پایگاه داده درست شده بود ساختیم. چون اکشن‌های این کنترلر همان CRUD هست دیگر آن‌ها را لازم به توضیح بیشتر نمی‌بینیم. ضمنا لازم به ذکر است که تمامی اکشن‌های این کنترلر تنها برای صاحب شغل قابل دسترسی بوده و دسترسی بقیه نقش‌ها به اکشن‌های این کنترلر مسدود شده است.

1. **اکشن Index**
2. **اکشن Create**
3. **اکشن Edit**
4. **اکشن Delete**
5. **اکشن JobCorpsList**

در داخل یک ViewBag لیست همکاران شغل مورد نظر را می ریزیم و به View می‌فرستیم. لازم به ذکر است که در این اکشن هیچ کاربر دیگری به‌جز صاحبان مشاغل حق ورود ندارد و در همان ابتدای اکشن آن‌ها را به اکشن مناسبی هدایت می‌کنیم.

1. **اکشن DeleteJobCorp**

در صورت انتخاب شدن یک همکار در View بالا و زدن دکمه حذف همکار، Id همکار را به این اکشن می‌فرستیم و سپس با توجه به id رکورد او را از جدول JobCorps در پایگاه داده حذف می‌کنیم و نقش کاربری او را هم به کاربر عادی تغییر می‌دهیم. البته عملیات تغییر در پایگاه داده پس از تایید کاربر انجام می‌شود، یعنی پس از اکشن بعدی!

1. **اکشن DeleteJobCorpConfirmed**

همانطور که توضیح داده شد در این اکشن رکورد این همکار را از جدول JobCorps در پایگاه داده حذف می‌کنیم و نقش کاربری او را هم به کاربر عادی تغییر می‌دهیم.

### **4.3.۳: Service Controller**

این کنترلر را نیز به‌صورت مستقیم و از طریق Scaffolding مدل Service که از جدول Services در پایگاه داده درست شده بود ساختیم. چون اکشن‌های این کنترلر همان CRUD هست دیگر آن‌ها را لازم به توضیح بیشتر نمی‌بینیم.

1. **اکشن Index**
2. **اکشن Create**
3. **اکشن Edit**
4. **اکشن Delete**

### **5.3.۳: JCDashboard Controller**

## نام این کنترلر مخفف JobCorp Dashboard است. در واقع چون بیشتر کارهایی که همکار و صاحب شغل انجام می‌دهند یکسان است، لذا تصمیم گرفتیم برای صرفه‌جویی در زمان و رعایت مولفه‌های مهندسی نرم‌افزار فعالیت‌های مشترک این دو نقش را در این کنترلر پیاده‌سازی نماییم.

## اکشن‌های این کنترلر عبارتند از:

1. **اکشن Index:**

در این اکشن قرار ملاقات‌های صاحب کار (یا همکار) را از پایگاه داده خوانده و به View متناظر می‌فرستیم.

1. **اکشن EnrollJob:**

وقتی کاربری در صفحه ثبت نام گزینه همکار را انتخاب می‌کند او را به این View هدایت خواهیم کرد. در این اکشن اگر کاربر کد صحیحی وارد کند او را به عنوان همکار به شغل مربوطه اضافه خواهیم کرد و سپس او را به اکشن AddWorkingTime از همین کنترلر هدایت می‌کنیم.

1. **اکشن AddWorkingTime:**

در قسمت HttpGet این اکشن زمان‌های کاری جاری کاربر را از پایگاه داده می‌خوانیم و به View می‌فرستیم. و در قسمت HttpPost این اکشن سعی در ذخیره کردن زمان‌های انتخابی کاربر داریم.

هسته این اکشن از یک حلقه for تشکیل شده است. با توجه به این که در View از کاربر روزهای هفته به‌صورت دسته‌ای دریافت می‌شوند و بعد از آن کاربر امکان وارد کردن ساعت شروع و پایان روزهای کاری را نیز به‌صورت دسته‌ای دارد، باید در داخل این اکشن با ساز و کاری اطلاعات دریافتی را مدیریت کنیم. این حلقه به این صورت پیاده‌سازی می‌شود:

for (int i = 0; i < 31; i++)

{

t = DateTime.Now;

t = t.AddDays(i);

DoW = (int)t.DayOfWeek;

if (WeekDayId.Contains(DoW))

{

if (i == 0)

{

if (ts <= DateTime.Now.TimeOfDay)

{

t = t.AddDays(7);

for (int j = 0; j < wd.Range.Count; j++)

{

stimelist.Add(new HourMinute{

hm = TimeSpan.Parse(wd.Range[j].StartHour) });

etimelist.Add(new HourMinute{

hm = TimeSpan.Parse(wd.Range[j].EndHour) });

list.Add(new WorkingTime{

JobCorpId = jc.Id,

StartTime = t.Date + stimelist[j].hm,

EndTime = t.Date + etimelist[j].hm });

}

continue;

}

else //when the start hour is valid

{

for (int j = 0; j < wd.Range.Count; j++)

{

stimelist.Add(new HourMinute{

hm = TimeSpan.Parse(wd.Range[j].StartHour) });

etimelist.Add(new HourMinute{

hm = TimeSpan.Parse(wd.Range[j].EndHour) });

list.Add(new WorkingTime{

JobCorpId = jc.Id,

StartTime = t.Date + stimelist[j].hm,

EndTime = t.Date + etimelist[j].hm });

}

}

}

if ((i == 7) && (ts <= DateTime.Now.TimeOfDay)) {

continue;

}

if ((i == 0) && (ts >= DateTime.Now.TimeOfDay)) {

continue;

}

for (int j=0; j<wd.Range.Count; j++){ //\*\*\*\*

stimelist.Add(new HourMinute{

hm = TimeSpan.Parse(wd.Range[j].StartHour) });

etimelist.Add(new HourMinute{

hm = TimeSpan.Parse(wd.Range[j].EndHour) });

list.Add(new WorkingTime{

JobCorpId = jc.Id,

StartTime = t.Date + stimelist[j].hm,

EndTime = t.Date + etimelist[j].hm });

}

}

}

حال به شرح این قطعه کد کلیدی می‌پردازیم. ابتدا زمان حال را در متغیر t که از نوع DateTime است قرار می‌دهیم. سپس (در هر بار تکرار حلقه) i روز به t اضافه می‌کنیم. در متغیر DoW که مخفف روزهای هفته می‌باشد شکل عددی روز هفته متغیر t را قرار می‌دهیم. در اولین شرط خود چک می‌کنیم که آیا DoW در این تکرار از حلقه ددر آرایه‌ای که از سمت View به کنترلر پاس داده شده(آرایه‌ای شامل اعداد 0 تا 6 که نشانگر روزهای هفته هستند و کاربر انتخاب کرده است) وجود دارد یا نه. حال استثناها را بررسی می‌کنیم. اگر i با صفر برابر باشد بدین معنی است که همین امروز هم جزو انتخاب‌های روز هفته‌ی کاربر می‌باشد. سپس اگر ساعت انتخابی کاربر قبل از ساعت فعلی امروز باشد 7 روز به t اضافه می‌کنیم و سپس عمل ذخیره را در list خود که از نوع WorkingTime می‌باشد، انجام می‌دهیم و با دستور continue حلقه را از ابتدا فراخوانی می‌کنیم. در غیر این‌صورت(اگر ساعت انتخابی کاربر بعد از ساعت فعلی امروز باشد) بدون مشکل t را ذخیره می‌کنیم. سپس با دو شرط چک می‌کنیم اگر قبلا این دو شرط را اعمال کرده باشیم با دستور continue به ابتدای حلقه برویم تا دیگر نگران افزونگی داده در پایگاه داده نباشیم.

در نهایت به for آخر می‌رسیم که با \*\*\*\* مشخص شده است. این حلقه کار اصلی ذخیره را بر عهده دارد. پس آن را با دقت بیشتری بررسی می‌کنیم. در شرط j<wd.Range.Count در داخل for چک می‌کنیم که تعداد تکرارهای حلقه بیش از تعداد بازه‌های زمانی که کاربر وارد کرده نشود. لیست stimelist شامل زمان‌هایی خواهد بود که زمان شروع انتخابی توسط کاربر هستند. همینطور لیست etimelist هم شامل زمان‌های پایان انتخابی توسط کاربر می‌باشد. هم‌چنین قبلاً متذکر شدیم که لیست list شامل یک لیست از نوع WorkingTime می‌باشد. (در نهایت همین لیست را به پایگاه داده ارسال خواهیم کرد تا به‌صورت دسته‌ای ذخیره شود.) با توجه به جدول WorkingTime در پایگاه داده برای هر رکورد در لیست list متغیرهای JobCorpId ، StartTime و EndTime را ذخیره می‌کنیم. برای مثال با دستور StartTime = t.Date + stimelist[j].hm تاریخ t را به زمان شروع شماره j انتخابی کاربر الحاق می‌کنیم.

1. **اکشن AddWorkingDate:**

در قسمت HttpGet این اکشن زمان‌های کاری جاری کاربر را از پایگاه داده می‌خوانیم و به View می‌فرستیم. و در قسمت HttpPost این اکشن سعی در ذخیره کردن زمان‌های انتخابی کاربر داریم. این اکشن را کاربر زمانی استفاده می‌کند که به جای انتخاب زمان کاری دسته‌ای بخواهد تنها یک روز را به زمان‌های کاری‌اش بیفزاید. بنابراین در View یک تقویم فارسی به کاربر نشان ‌می‌دهیم و پس از این که کاربر اطلاعات را به سمت سرور فرستاد سعی می‌کنیم آن روز را به ساعت ابتدا و انتهای انتخاب شده توسط کاربر الحاق کنیم. دقیقا همانطور که در اکشن قبلی این کار را کردیم با این تفاوت که دیگر لازم نیست از لیستی استفاده کنیم و تنها با چند متغیر کار خود را پیش می‌بریم:

wd.JobCorpId = jc.Id;

TimeSpan ts = TimeSpan.Parse(wd.start);

wd.StartTime = wd.myDate.Date + ts;

TimeSpan es = TimeSpan.Parse(wd.end);

wd.EndTime = wd.myDate.Date + es;

wd همان مخفف WorkingDate است و قرار است آن را در پایگاه داده ذخیره کنیم. ابتدا JobCorpId آن را ست می‌کنیم. در متغیرهای ts و es به ترتیب زمان انتخابی ابتدایی و انتهایی توسط کاربر را قرار می‌دهیم و سپس با الحاق تاریخ انتخاب شده توسط کاربر با این دو متغیر آن‌ها را در wd.StartTime و wd.EndTime ذخیره می‌کنیم تا بتوانیم موجودیت wd را در قالب یک رکورد در پایگاه داده ذخیره کنیم.

1. **اکشن ModifyWorkingDate:**

در این اکشن کلیه‌ی زمان‌های کاری را برای صاحب شغل (یا همکار) لیست می‌کنیم و به View می‌فرستیم. در View جلوی هر رکورد از زمان‌های کاری یک دکمه حذف قرار داده ایم که کاربر با فشردن آن، رکورد مورد نظر را پاک می‌کند. بدین ترتیب با Viewهای AddWorkingDate و ModifyWorkingDate نیز کاربر می‌تواند استثناها را نیز مدیریت کند.

1. **اکشن AppointmentList:**

در این اکشن لیست تمامی قرار ملاقات‌ها را برای صاحب شغل یا همکار از پایگاه داده می‌گیریم و سپس به View ارسال می‌کنیم.

1. **اکشن ExpiredAppointmentList:**

مانند Viewی بالاست با این تفاوت که قرار ملاقات‌های قبل از تاریخ فعلی را لیست می‌کند.

1. **اکشن FutureAppointmentList:**

مانند Viewی بالاست با این تفاوت که قرار ملاقات‌های بعد از تاریخ فعلی را لیست می‌کند یعنی قرار ملاقات‌های آینده را به کاربر نشان می‌دهد.

1. **اکشن AppointmentDetails:**

جزئیات هر قرار ملاقات اعم از نام طرف ملاقات کننده، شماره تلفن، زمان ملاقات و ... در اینجا به View فرستاده می‌شود و هم‌چنین در View نیز دکمه کنسل ملاقات کاربر را به View بعدی منتقل می‌کند.

1. **اکشن AppointmentCancellation:**

در کنسل قرار ملاقات‌ها آن رکورد را از پایگاه داده حذف نمی‌کنیم، بلکه بیت isReserved را برابر صفر می‌گذاریم تا در آینده به کاربر و صاحب شغل آن را طور دیگری نشان دهیم. هم‌چنین در View بعد از تایید کنسل کردن قرار ملاقات به اکشن زیر می‌رویم تا عملیات تغییر در پایگاه داده را در آن انجام دهیم.

1. **اکشن AppointmentCancellationConfirmed:**

همان‌طور که توضیح داده شد در این اکشن بیت isReserved را در رکورد مربوطه‌اش صفر می‌کند و در پایگاه داده ذخیره می‌کند. سپس کاربر را به صفحه لیست قرارهای کاری اش می‌فرستد.

### **6.3.۳: Main Controller**

## این کنترلر مسئول کنترل نیازهای کاربر عادی است. نیازهایی مانند رزرو وقت، جستجوی مشاغل و مشاهده‌ی لیست Reservation برای کاربر از جمله نیازهایی هستند که در این کنترلر پیاده‌سازی شده‌اند.

## اکشن‌های این کنترلر عبارتند از:

1. **اکشن Index:**

در این اکشن صفحه‌ی جستجوی مشاغل را به کاربر نشان می‌دهیم.

1. **اکشن Jobs:**

این اکشن هم صفحه‌ی جستجوی مشاغل را به کاربر نشان می‌دهد با این تفاوت که می‌توان با لینک دادن هم جستجو را انجام داد. برای مثال در صفحه Index از کنترلر Home لینک نوع شغل‌های مختلف گذاشته شده و با فشردن آن لینک، کاربر مستقیما به صفحه Jobs منتقل شده و تنها نوع شغل مورد نظر خود را خواهد دید. Query یا همان فیلتر مورد استفاده در کنترلر این‌گونه پیاده‌سازی شده است:

Query = db.Jobs.Where(m=>

(string.IsNullOrEmpty(title) ? true : m.Title.Contains(title)) && (string.IsNullOrEmpty(city) ? true : m.City.Name == city) && (string.IsNullOrEmpty(jobtype) ? true : m.JobType.Title == jobtype));

1. **اکشن JobDetails:**

کاربر می‌تواند جزئیات مربوط به هر کدام از مشاغلی که انتخاب می‌کند را مشاهده نماید.

1. **اکشن JobCorpsList:**

کاربر پس از کلیک بر روی شغل مورد نظر که از قسمت جستجو یافته است به این صفحه هدایت می‌شود که لیست همکاران شغل مورد نظر در آن نمایش داده می‌شود. کاربر با کلیک بر روی هر کدام از همکاران به صفحه رزرو وقت از او وارد می‌شود. هم‌چنین در این صفحه لینکی به اکشن JobDetails وجود دارد.

1. **اکشن Reserve:**

در متد GET این اکشن با توجه به JobCorpId همکار انتخابی لیست سرویس‌ها و زمان‌های کاری او را به View می‌فرستیم. در متد POST با توجه به تاریخ انتخابی کاربر باید تشخیص دهیم که زمان انتخاب شده توسط او در بازه‌ی کاری همکار هست یا خیر و هم‌چنین اگر کسی وقت دیگری در آن زمان گرفته باشد با پیغام خطا کاربر را مطلع سازیم. Query های زیر این کارها را انجام می‌دهند:

var isInWorkingTimes = db.WorkingTimes.Any(p => (p.StartTime <= ap.StartTime) && (p.EndTime >= ap.EndTime)&&(ap.StartTime >= DateTime.Now)&&(p.JobCorpId == s.JobCorpId));

var isInOtherAppointments = db.Appointments.Any(m => (ap.StartTime >= DateTime.Now) && (m.Service.JobCorpId==s.JobCorpId)&&(m.StartTime<ap.EndTime)&&(m.EndTime>ap.StartTime));

1. **اکشن ChooseReserve:**

برخی سختی‌ها در پیاده‌سازی اکشن بالا باعث شد این اکشن را برای رزرو کاربر استفاده کنیم. در اکشن بالا تاریخ شمسی را با استفاده از JsDatePicker از کاربر می‌گرفتیم. ولی با توجه به این که کاربر می‌خواهد تنها روزهای کاری همکار مورد نظر را ببیند تصمیم گرفتیم به جای نشان دادن تقویم به کاربر تنها روزهایی که او کار می‌کند را به کاربر نشان دهیم. هم‌چنین پس از انتخاب تاریخ (با استفاده از DropDownList) به کاربر با استفاده از Ajax (اکشن JsonInfo) مشخصات شغل و هم‌چنین زمان دقیق کاری در آن روز توسط همکار را نشان می‌دهیم. هم‌چنین اگر کس دیگری در آن زمان وقت گرفته باشد به کاربر نشان می‌دهیم تا بداند که نمی‌تواند در آن زمان وقت رزرو کند. Query چک کردن isInWorkingTimes و isInOtherAppointments را هم که در اکشن بالا تعریف کردیم، دوباره اینجا مورد استفاده قرار داده‌ایم.

1. **اکشن JsonInfo:**

در View و در قسمت جاوا اسکریپت با استفاده از کد زیر این اکشن را فراخوانی می‌کنیم:

$(document).ready(function () {

var url = '@Url.Action("JsonInfo")';

$('#selDate').change(function () {

$.getJSON(url, { wtid: $('#selDate').val() }, function (response) {

if (response == false) {

$(".ajaxRes").html("<span style='color: red;'>زمان کاری وجود ندارد</span>");

}

else{

$(".ajaxRes").html(response.dow + "<span> از </span>" +

response.StartTime + "<span> تا </span>" +

response.EndTime + "<br/>" + response.JobTitle +

"<br/>" + response.JobCorp);

if (response.ap.length > 0) {

$(".ajaxRes").append("<hr><span>دیگر قرارهای این روز: </span><br/>");

}

for (var j = 0; j < response.ap.length ; j++) {

$(".ajaxRes").append(response.ap[j].StartHour + " تا " +

response.ap[j].EndHour + "<br/>");

}

}

});

});

});

حال wtid که همان روز انتخابی کاربر (WorkingTime همکار مورد نظر) است را به JsonInfo می‌فرستیم و با استفاده از مدل LittleWorkingTime که قبلا در قسمت مدل‌ها توضیح دادیم، اشیائی که می‌خواهیم با استفاده از AJAX به View بفرستیم را مقداردهی می‌کنیم. لازم به ذکر است لیست ap را که دربرگیرنده قرار ملاقات‌های JobCorp در زمان‌کاری مورد نظر است را با استفاده از Query زیر به دست می‌آوریم:

lwt.ap = (from s in db.Services

from a in db.Appointments

from w in db.WorkingTimes

where w.Id == wt.Id && w.StartTime <= a.StartTime && w.EndTime >= a.EndTime

&& w.JobCorpId == s.JobCorpId && s.Id == a.ServiceId

select new Item{ StartTime = a.StartTime, EndTime = a.EndTime }).ToList();

حال باید اشیاء را با استفاده از AJAX به View بفرستیم. پس شیء lwt را که یک instance از مدل LittleWorkingTime است را به عنوان شیء ارسالی می‌فرستیم. دستور مورد نظر Json را به View ارسال می‌کند:

return Json(lwt, JsonRequestBehavior.AllowGet);

1. **اکشن ReserveEdit:**

در متد GET این اکشن لیست سرویس‌ها و زمان‌های کاری همکار مورد نظر را لود کرده و به View می‌فرستیم. در متد POST نیز اطلاعات ویرایش شده را می‌خوانیم و سپس در پایگاه داده آن‌ها را تغییر می‌دهیم.

1. **اکشن ReserveDetails:**

جزئیات هر قرار ملاقات اعم از عنوان شغل، شماره تلفن شغل، مشخصات همکاری که از آن وقت گرفته شده است، زمان ملاقات و ... در اینجا به View فرستاده می‌شود و هم‌چنین در View نیز دکمه کنسل ملاقات کاربر را به View بعدی منتقل می‌کند.

1. **اکشن ReserveCancellation:**

در کنسل قرار ملاقات‌ها آن رکورد را از پایگاه داده حذف نمی‌کنیم، بلکه بیت isReserved را برابر صفر می‌گذاریم تا در آینده به کاربر و صاحب شغل آن را طور دیگری نشان دهیم. هم‌چنین در View بعد از تایید کنسل کردن قرار ملاقات به اکشن زیر می‌رویم تا عملیات تغییر در پایگاه داده را در آن انجام دهیم.

1. **اکشن AppointmentCancellationConfirmed:**

همان‌طور که توضیح داده شد در این اکشن بیت isReserved را در رکورد مربوطه‌اش صفر می‌کند و در پایگاه داده ذخیره می‌کند. سپس کاربر را به صفحه لیست قرارهایش می‌فرستد.

1. **اکشن Reservations:**

در این اکشن لیست تمامی قرارهای کاربر را از پایگاه داده می‌گیریم و سپس به View ارسال می‌کنیم.

1. **اکشن ExpiredAppointmentList:**

مانند Viewی بالاست با این تفاوت که قرار‌های قبل از تاریخ فعلی را لیست می‌کند.

1. **اکشن FutureAppointmentList:**

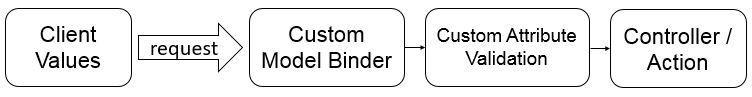
مانند Viewی بالاست با این تفاوت که قرارهای بعد از تاریخ فعلی را لیست می‌کند یعنی قرارهای آینده را به کاربر نشان می‌دهد.

### **4.۳: برخی پیاده‌سازی‌های کاربردی سیستم**

### **1.4.۳: نحوه تبدیل تاریخ شمسی به میلادی در سیستم**

**-اطلاعات از سمت کاربر چگونه به سرور می‌رسد؟**

زمانی که یک کاربر مقداری را به سمت سرور می‌فرستد با استفاده از یک request آن را به سرور می‌فرستد (با استفاده از متدهایی مانند POST، GET و JSON) مقدار وارد شده به Custom Model Binder می‌رود. وظیفه Custom Model Binder تبدیل مقدار داده شده به اشیائی است که در .NET تعریف شده اند می‌باشد. پس در همین مرحله باید تبدیل به تاریخ میلادی را انجام دهیم. پس از این مرحله نوبت به Custom Attribute Validation است که کار اعتبارسنجی اطلاعات را انجام می‌دهد و در صورت درست بودن اطلاعات (true شدن مقدار ModelState.IsValid) اطلاعات را به سمت کنترلر و اکشن مربوطه هدایت می‌کند. لازم به ذکر است Custom Attribute Validation هر دو Validation در سمت Client و Server را انجام می‌دهد. مثلا استفاده از اعتبارسنجی Regular Expressionها در سمت Client و اعتبارسنجی مقادیری که به سرور فرستاده می‌شوند توسط خود Server انجام می‌شوند.



برای توضیح دادن این تبدیل مثال کلی زیر را در نظر بگیرید:

در یکی از مدل‌های خود مثلا WorkingTime یک متغیر DateTime به صورت زیر تعریف میکنیم:

public DateTime myDate { get; set; }

برای میلادی کردن تاریخ شمسی باید از Custom Model Binder استفاده کنیم. برای نوشتن آن یک پوشه جدید در پروژه به نام Utilities ایجاد کرده و سپس در آن پوشه‌ای به نام ModelBinders ایجاد می‌کنیم. سپس کلاس PersianDateTimeModelBinder.cs را در آن ایجاد می‌کنیم. این کلاس حتما باید از اینترفیس IModelBinder ارث بری کند. در این کلاس مقادیری که کاربر به عنوان تاریخ شمسی وارد کرده را گرفته و سپس چک می‌کنیم اگر مقدار داده شده توسط کاربر اشتباه باشد ModelState.Error را اضافه می‌کنیم تا سرور خطا دهد. در غیر این‌صورت ModelState.IsValid را برابر با true می‌کنیم. اگر تاریخ وارد شده توسط کاربر مشکلی نداشت آن را به تاریخ میلادی تبدیل می‌کنیم و سپس برمی‌گردانیم. پس از این در پارامتر ورودی اکشن خود تاریخ میلادی نخواهیم داشت و تاریخ به میلادی تبدیل شده است.

کد این کلاس بدین صورت پیاده سازی شده است. لازم به ذکر است در برخی جاها از کلاس Utilities که معرفی خواهیم کرد، استفاده کرده‌ایم:

public class PersianDateTimeModelBinder : IModelBinder

{

public object BindModel(ControllerContext controllerContext, ModelBindingContext bc){

var valueResult = bc.ValueProvider.GetValue(bc.ModelName);

var modelState = new ModelState { Value = valueResult };

object actualValue = null;

try{

if (valueResult.AttemptedValue.IsPersianDateTime() == false){

var metadata = bindingContext.ModelMetadata;

var displayName = metadata.DisplayName ?? metadata.PropertyName ??

bc.ModelName.Split('.').Last();

modelState.Errors.Add(string.Format("{0} را به درستی وارد کنید", displayName));

}

else{

var datetime = Convert.ToDateTime(valueResult.AttemptedValue);

var miladi = datetime.ToMiladiDateTime();

actualValue = miladi;

}

}

catch (FormatException e) {

modelState.Errors.Add(e);

}

bindingContext.ModelState.Add(bindingContext.ModelName, modelState);

return actualValue;

}

}

یک کلاس دیگر در پوشه Utilities به نام Utility.cs می‌سازیم. در این کلاس سه متد کاربردی را پیاده سازی کرده‌ایم که تمامی آن‌ها Extension Method هستند(به علت static بودن توابع و استفاده از this در متغیرهای ورودی آن‌ها).یعنی می‌توانیم این متدها را بعد از هر property فراخوانی کنیم و نیازی به new کردن این کلاس نخواهیم داشت. متد اول با استفاده از متد PersianCalender که از توابع .NET است استفاده کرده و تاریخ میلادی را به شمسی تغییر می‌دهد. متد دوم دقیقا برعکس متد اول است. و متد سوم نیز با استفاده از عبارات منظم تعیین می‌کند که رشته ارسالی تاریخ شمسی معتبری هست یا خیر.

پیاده سازی این توابع به صورت زیر است:

public static DateTime ToPersianDateTime(this DateTime datetime) {

var pc = new PersianCalendar();

return new DateTime(pc.GetYear(datetime), pc.GetMonth(datetime),

pc.GetDayOfMonth(datetime),0,0,0);

}

public static DateTime ToMiladiDateTime(this DateTime datetime){

var pc = new PersianCalendar();

return pc.ToDateTime(datetime.Year, datetime.Month, datetime.Day, 0, 0, 0, 0);

}

public static bool IsPersianDateTime(this string datetime) {

return Regex.IsMatch(datetime,@"^(13\d{2}|[1-9]\d)/(1[012]|0?[1-9])/([12]\d|3[01]|0?[1-9])$");

}

حال که CustomModelBinder خود را نوشته‌ایم باید آن را به پروژه معرفی کنیم. این معرفی را در کلاس Global.asax انجام می‌دهیم. لذا خط زیر را در این کلاس اضافه می‌کنیم تا این ModelBinder به پروژه اضافه گردد:

ModelBinders.Binders.Add(typeof(DateTime), new PersianDateTimeModelBinder());

با این معرفی هرجا که به سمت سرور نوع DateTime ارسال شود کار اعتبارسنجی آن متغیر را کلاس PersianDateTimeModelBinder انجام می‌دهد.

حال تمام اعتبارسنجی سمت سرور برای تاریخ شمسی را ساختیم. پس از انجام این اعتبارسنجی، در کنترلر و View خود اعمال همیشگی را انجام می‌دهیم. مثلا می‌توانیم در قسمت JavaScript صفحه خودمان هم یک عبارت منظم برای input تاریخ شمسی در نظر بگیریم تا سمت client هم اعتبارسنجی کنیم. برای زیباسازی بیشتر پروژه یک jquery component تاریخ شمسی را استفاده می‌کنیم تا کاربر بتواند به راحتی تاریخ خود را انتخاب کند. این ابزار را در قسمت 2.5.1 فصل اول معرفی کرده بودیم.

### **1.4.۳: نحوه ست کردن متغیرهای Session توسط SessionPersister**

برای دسترسی به برخی متغیرها در صفحات html نیاز به چند Session داریم. مانند نام و نام خانوادگی کاربر که در بالای همه صفحات به کاربر نشان دهیم. همینطور id کاربر و نقش کاربر را هم در متغیرهای Session جداگانه نگهداری می‌کنیم تا در اکشن‌های کنترلرها از آن‌ها استفاده نماییم.

کلاس SessionPersister را در پوشه Security به‌صورت استاتیک تعریف می‌کنیم. یکی از توابع این کلاس به‌صورت زیر پیاده‌سازی می‌شود:

public static string UserRole

{

get

{

if (HttpContext.Current == null)

return string.Empty;

var sessionvar = HttpContext.Current.Session["userRole"];

if (sessionvar != null)

return sessionvar as string;

return null;

}

set

{

HttpContext.Current.Session["userRole"] = value;

}

}

که مشخصا مقدار SessionPersister.UserRole را برابر Session[“userRole”] می‌کند. این تابع را بعدها در کنترلرها استفاده کرده‌ایم. لازم به ذکر است مثلا در اکشن SignIn کاربر اینگونه این Session را مقداردهی می‌کنیم:

SessionPersister.UserRole = user.UserRole.Name;

بعداً به طور مثال در کنترلر JCDashboard و در ابتدای متد GET اکشن AddWorkingTime باید چک کنیم که اگر کاربر، صاحب شغل یا همکار نباشد او را به جای دیگری هدایت کنیم تا به آن صفحه دسترسی نداشته باشد. مانند کد زیر:

if (SessionPersister.UserRole.ToString() == "User"){

return RedirectToAction("Index", "Home");

}

کد بالا در صورتی که کاربر عادی بخواهد وارد اکشن AddWorkingTime شود او را به صفحه Index از کنترلر Home هدایت می‌کند. لذا صفحه AddWorkingTime هیچ‌گاه برای او لود نخواهد شد.

منابع

مهندسی نرم افزار پوران پژوهش

http://www.tahlildadeh.com/ArticleDetails/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%D9%86%D9%85%D9%88%D8%AF%D8%A7%D8%B1%D9%87%D8%A7%DB%8C-%D8%B1%D8%A7%DB%8C%D8%AC-%D8%AF%D8%B1-UML

<https://fa.wikipedia.org/wiki/%D8%AC%DB%8C%E2%80%8C%DA%A9%D9%88%D8%A6%D8%B1%DB%8C>

فصل اول کتاب مهندسی نرم افزار پوران پژوهش پیمان تیموری (پرسمن)

1. View State [↑](#footnote-ref-1)
2. Scripting Languages [↑](#footnote-ref-2)
3. Web Hosting [↑](#footnote-ref-3)
4. modular [↑](#footnote-ref-4)
5. .NET Framework Class Library [↑](#footnote-ref-5)
6. Common Language Runtime [↑](#footnote-ref-6)
7. Model [↑](#footnote-ref-7)
8. View [↑](#footnote-ref-8)
9. Controller [↑](#footnote-ref-9)
10. Data [↑](#footnote-ref-10)
11. Permission [↑](#footnote-ref-11)
12. Presentation Layer [↑](#footnote-ref-12)
13. Maintainability [↑](#footnote-ref-13)
14. Action [↑](#footnote-ref-14)
15. Entity [↑](#footnote-ref-15)
16. Query [↑](#footnote-ref-16)
17. Software Process Models [↑](#footnote-ref-17)