آنگولار

در مقایسه با برنامه هایی که برای نمایش صفحات، نیاز به دانلود چند مگابایتی دیتا از سمت سرور دارند، برای نمایش صفحات فقط نیاز به دانلود داده مورد نیاز برای ارائه را دارد. یعنی برای نمایش هر صفحه فقط داده مورد نیاز صفحه را از سمت سرور در قالب JSON دریافت کرده و عمل Render صفحه را تماما در سمت کلاینت انجام می‌دهد. از جمله رقیبان آنگولار می‌توان بهBackbone ، Ember، Knockoutاشاره کرد.

ReactJSبراساس مدل نمایش یا view ساخته شده است این در حالی است که angular براساس MVVM است. همین دلیل باعث می شود که ReactJS نسبت به angularjs سازگاری بیشتری داشته باشد.

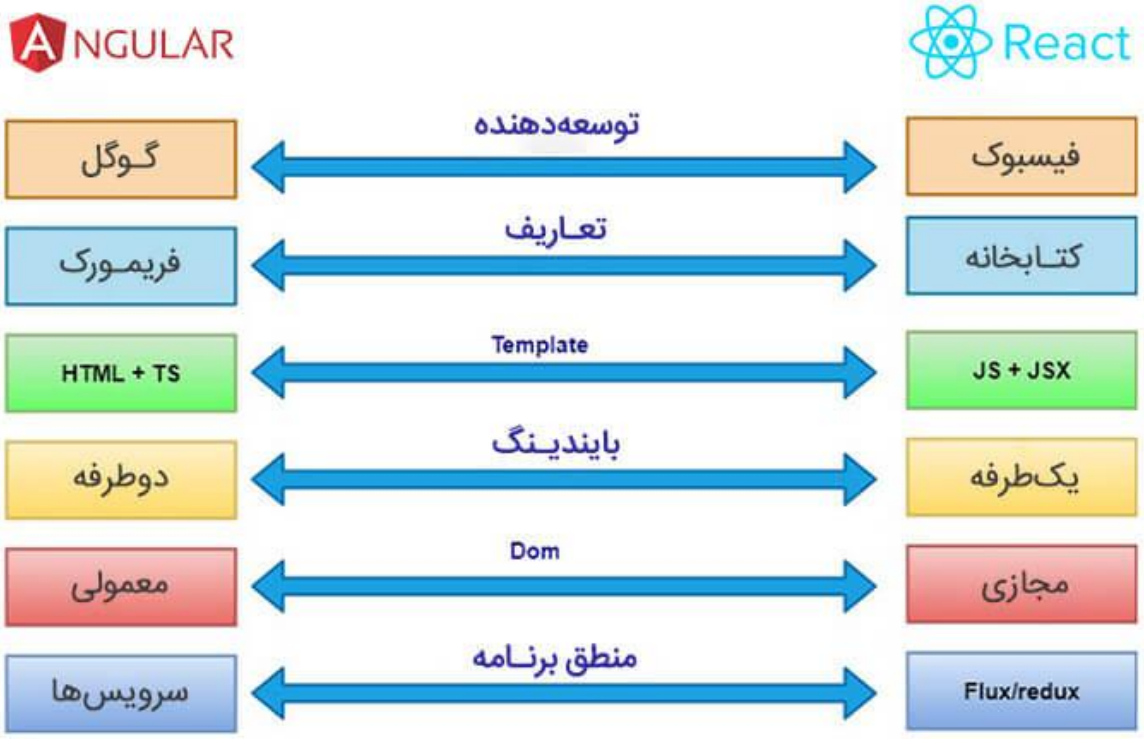
ReactJSیکی از کتابخانه های متن باز جاوا اسکریپت به حساب می آید. کتابخانه React فقط در لایه view در معماری MVC مشارکت دارد. JSXبه شما اجازه می‌دهد بتوانید نوعی از HTML را در JavaScript بنویسید.

مزایا و معایبAngularJS

مزایا

اول از همه این فریمورک کاملا براساس مدل MVC طراحی شده است. مدل MVC در واقع یک معماری نرم افزاری است که ساختار نرم افزار را به سه قسمت مدل، نما و کنترلر تقسیم می کند.MVC این قابلیت را به توسعه دهنده می دهد تا اجزای مختلف نرم افزار را از هم جدا کند و بتواند کنترل کاملتری را به صورت مجزا روی هر کدام از آنها داشته باشد. البته در حال حاضر AngularJS از MVC به معماری MVVM تبدیل شده است. که در این حالت ما می توانیم بخش منطقی اپلیکیشن را از بخش رابط کاربری آن تفکیک کنیم. در این حالت دو راه برای Data Binding وجود دارد که باعث می شود براساس منابع، اپلیکیشن بتواند بهتر به عملیات های کاربران جوابگو باشد.

از مزیت های خوب دیگر برای یادگیری این فریمورک وجود مستندات بیش از اندازه زیاد آن برای یادگیری و سوالات است. در واقع می توان گفت که AngularJS دارای جامعه پشتیبانی بسیار بزرگی است.



Technology	AngularJS	ReactJS
Develepor / Creator	Google	Facebook
Language	Java Scrip, HTML	JSX
Packaging	Medium	Strong
DOM	Regular DOM	Virtual DOM
Native Support	Native Script	React Native
Rendering	Server Side	Server Side
Size	766k	151k
Performance	Above Avg.	Faster

Features	AngularJS	React
Launch Date	2009	2013
Language	TypeScript, JavaScript	JavaScript
Market Share	0.3%	<0.1%
Model	✓	✗
View	✓	✓
Controller	✓	✗
Learning Curve	Complex	Easy
Templating	✓	✗
Failure	Run-time	Compile-time
Server-side rendering	✗	✓
DOM	✓	Virtual
Mobile Support	✓	✓
Input form validation	✓	✓ (formsy-react)

انقیاد .(Binding) فارغ از مشکلات کارایی(Performance) ، آنگولار از سه نوع روش انقیاد پشتیبانی می‌کند. انقیاد دو طرفه، انقیاد یک طرفه از طرف منبع داده (data source) به مقصد نمایش (view) و انقیاد یک طرفه از طرف لایه نمایش به منبع داده. این قابلیت در اتصال بین لایه مدل و نمایش، کمک فراوانی می‌کند همچنین با کاهش چشم گیر کد نویسی برای حفظ این اتصال، نگهداری و توسعه پروژه را آسان و سرعت می‌بخشد.

معماری MVC

در مهندسی نرم افزار MVCیا Model-View-Controllerبه یک الگوی نرم افزار گفته میشه. الگوی ساختاری MVCبه جداسازی داده های کاربر (محتویات بخش Modelاز مولفه های ارائه شده به صورت گرافیکی (بخش Viewو منطق مربوط به پردازش ورودی ها (بخش Controllerاقدام میکنه.هدف الگوی MVCصرفا یکپارچگی و ساخت یک نظم ثابت در ساختار نرم افزاره که بوسیله اون میشه نرم افزار رو مدیریت کرد و به راحتی گسترش داد.

الگوی معماری MVCبخش های مدل سازی دامنه، نمایش و منطق تجاری رو به سه بخش تقسیم میکنه که در ادامه به توضیه اون ها میپردازم.

1 – Model (مدل)

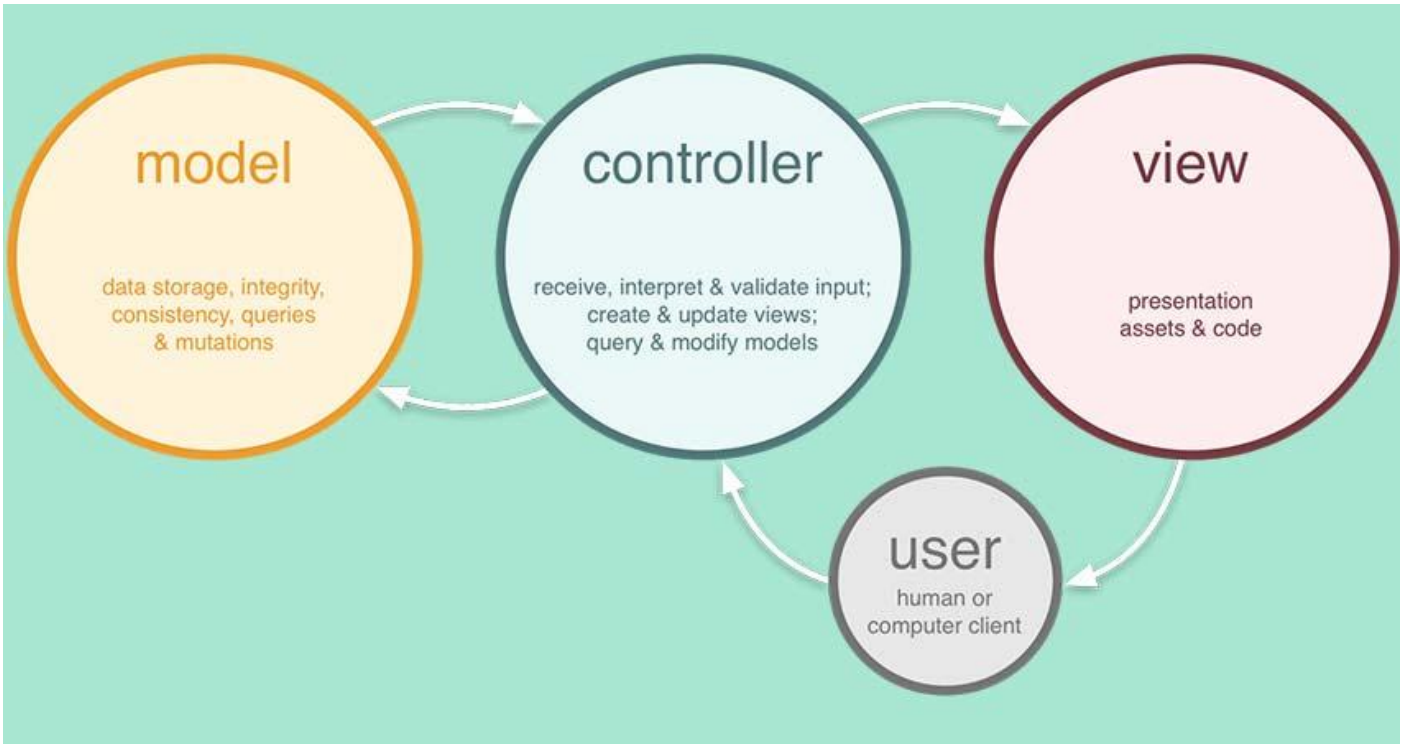
روند انتقال داده رو مدیریت میکنه و در خواست هایی که در مورد وضعیت مدل مانند تغییر فیلدی از رکورد که وجود داره رو پاسخ میده. لایه مدل عملیات هایی که موجب تغییر در وضعیت خودش میشه رو انجام میده (معمولا این تغییرات از سمت لایه کنترلر بوجود میان.)

2 – View (نمایش)

نمایش اطلاعات به کاربر رو مدیریت میکنه.

3 – Controller (کنترلر)

بخش ورودی های کاربر رو مدیریت میکنه و لایه های مدل و نمایش مربوط به اون رو فراخوانی میکنه و داده ها رو از لایه مدل به لایه نمایش اطلاعات منتقل میکنه.



استقلال در لایه مدل

طبق تعریف های گفته شده هر دو لایه کنترلر و نمایش به لایه مدل وابسته هستند اما در مورد لایه مدل این وابستگی وجود نداره و به صورت مستقل عمل میکنه. این ویژگی میتونه به تولید بخش مستقل که قابلیت تست بالایی داره منجر بشه. در نرم افزار های وب این تفکیک و جداسازی به خوبی دیده میشه ولی در نرم افزار هایی که طراحی UIو کد های پردازش و منطق در هم پیچیدگی دارن یا به اصطلاح Rich-Clientها، این موضوع دیده نمیشه و وجود نداره.

https://www.tutorialsteacher.com/mvc/mvc-architecture

Ruby	PHP	JavaScript	Python
Ruby on Rails	Laravel	Express	Django
Sinatra	Zend	Angular	Flask

آشنایی با مفهوم Model در معماری MVC

Modelرا به نوعی می‌توان به منزلهٔ مغز اپلیکیشن در نظر گرفت به طوری که اصطلاحاً Business Logic یا به عبارتی «آنچه اپلیکیشن به خاطرش توسعه یافته است» در این لایه طرح‌ریزی می‌شود. مسلماً نیاز به توضیح نیست که مثلاً در یک وب اپلیکیشن بخشی از منطق نرم‌افزار مرتبط با ارتباط با دیتابیس به منظور انجام عملیات CRUD است که تَسک‌هایی از این دست در مدل عملی می‌گردند.

سرواژهٔ CRUDبرگرفته از کلمات Update، Read، Createو Deleteاست که به ترتیب به منظور «ثبت»، «فراخوانی»، «به‌روزرسانی» و «حذف» داده‌ها از دیتابیس مورد استفاده قرار می‌گیرند.

آشنایی با مفهوم View در معماری MVC

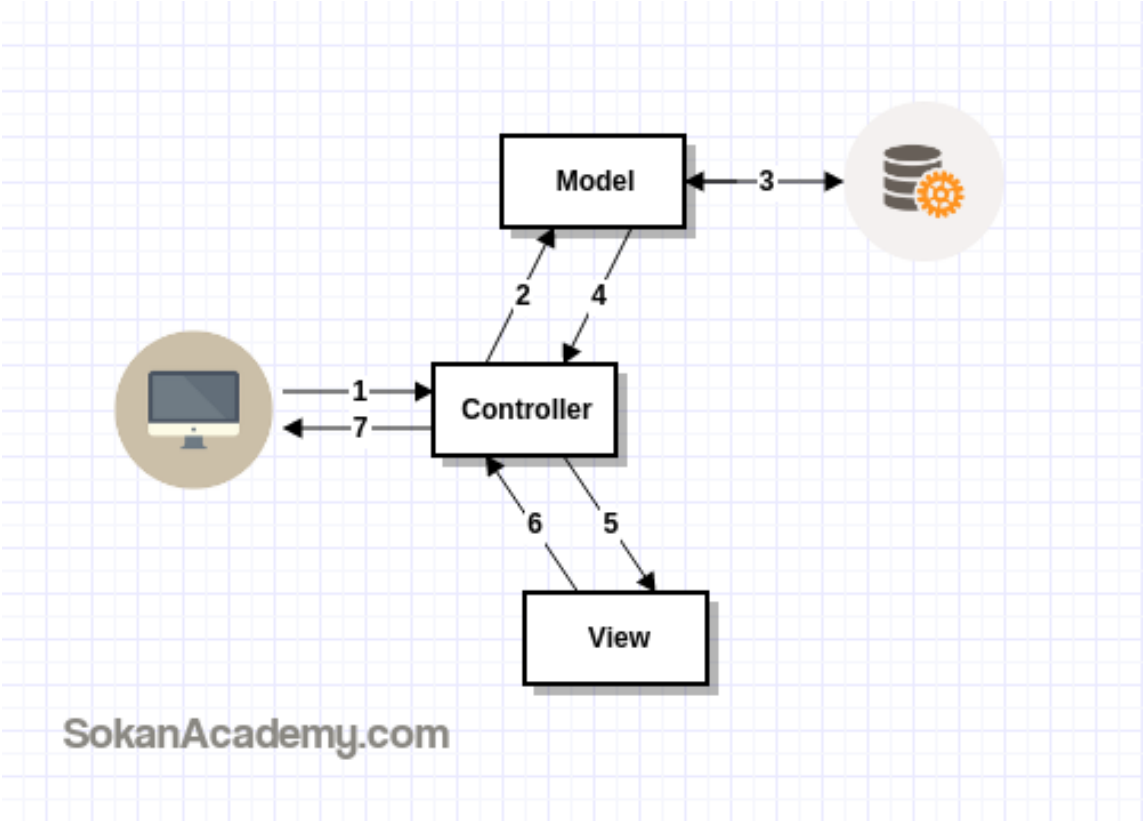
View، همان‌طور که از نام آن مشخص است، این وظیفه را دارا است تا دیتایی که در مدل ساخته و پرداخته شده را در معرض دید کاربران وب اپلیکیشن قرار دهد و به عبارتی می‌توان گفت که همان User Interfaceیا به اختصار UIاست.
ویو به طور معمول حاوی کدهای اچ‌تی‌امال و سی‌اس‌اس است و داده‌های دینامیک (پویا) نیز از روش‌های مختلفی که یکی از آن‌ها Template Engine است در ویو بارگزاری می‌شوند (به طور مثال، در فریمورک لاراول از تِمپلیت اِنجینی تحت عنوان Blade استفاده می‌شود تا در نهایت کدهای پی‌اچ‌پی داخل کدهای اچ‌تی‌امال قرار داده شده و خروجی آن‌ها در معرض دید کاربران قرار گیرند).

آشنایی با مفهوم Controller در معماری MVC

Controllerدر این معماری سه‌لایه نقش واسط را دارا است به طوری که ریکوئست‌ها را از بخش ویو گرفته و در اختیار مدل قرار می‌دهد و پس از آنکه مدل پردازش‌هایش را روی ریکوئست (درخواست) ورودی انجام داد، ریسپانس (پاسخ) را مجدد در اختیار کنترلر قرار داده و کنترلر هم پاسخ نهایی را در اختیار ویو می‌گذارد تا در معرض دید کاربران قرار دهد.

آشنایی با ارتباطات مابین View ، ModelوController

به منظور درک بهتر این موضوع، می‌توان فرم لاگین سایت‌ها را مد نظر قرار داد به طوری که در بخش ویو فرمی در اختیار کاربران قرار می‌گیرد تا بتوانند نام کاربری و رمزعبور خود را وارد کنند.



همان‌طور که در تصویر فوق مشخص است، پس از آنکه کاربر دکمهٔ سابمیت را فشرد (مرحله اول)، درخواستی از جنس POSTبرای کنترلر ارسال می‌گردد و کنترلر هم درخواست ورودی را گرفته و در اختیار مدل قرار می‌دهد (مرحلهٔ دوم) که در این فاز مدل یک کوئری به دیتابیس می‌زند (مرحلهٔ سوم) تا ببیند آیا نام کاربری و رمزعبور درست هستند یا خیر؛ خواه اطلاعات درست وارد شده باشند و خواه اشتباه، در مرحلهٔ بعد مدل پاسخی را در اختیار کنترلر قرار می‌دهد (مرحلهٔ چهارم) و کنترلر هم پاسخ دریافتی را در اختیار ویو گذاشته (مرحلهٔ پنجم) سپس ویو نیز خروجی کار را در معرض دید کاربر قرار می‌دهد (مراحل ششم و هفتم).

در همین راستا، مفهومی که ارتباط نزدیکی با این معماری دارد تحت عنوان Routing شناخته می‌شود. به طور خلاصه، منظور از این اصطلاح آن است که ما ابتدا به ساکن نیاز به تعریف یکسری Route یا URL و یا Link داریم که مشخص می‌کنند کاربر به چه لینک‌هایی در وب اپلیکیشن‌ما دسترسی دارد و هر چیزی به غیر از لینک‌هایی که از قبل تعریف شده‌اند را وارد سازد، با ارور ۴۰۴ مواجه خواهد شد.

در حقیقت، Routing مشخص می‌سازد که ویوها از چه مسیرها یا لینک‌هایی در معرض دید کاربران قرار می‌گیرند و همان لینک‌ها در نهایت به کنترلرهای مشخصی ختم می‌شوند که ریکوئست‌های کاربران را گرفته و در اختیار مدل می‌گذارند و پس از دریافت ریسپانس از مدل، مجدد نتیجهٔ نهایی را در اختیار ویو می‌گذارند.
با استفاده از معماری اموی‌سی به سادگی می‌توان بخش‌هایی از اپلیکیشن که مرتبط با یکدیگر هستند را در قالب یک ماژول یا کامپوننت قرار داد و از آنجا که در این معماری کامپوننت‌ها دارای وابستگی حداقلی به یکدیگر هستند، به سادگی این امکان در اختیار توسعه‌دهندگان قرار می‌گیرد تا از کامپوننت‌هایی که نوشته‌اند در سایر پروژه‌ها نیز استفاده نمایند.

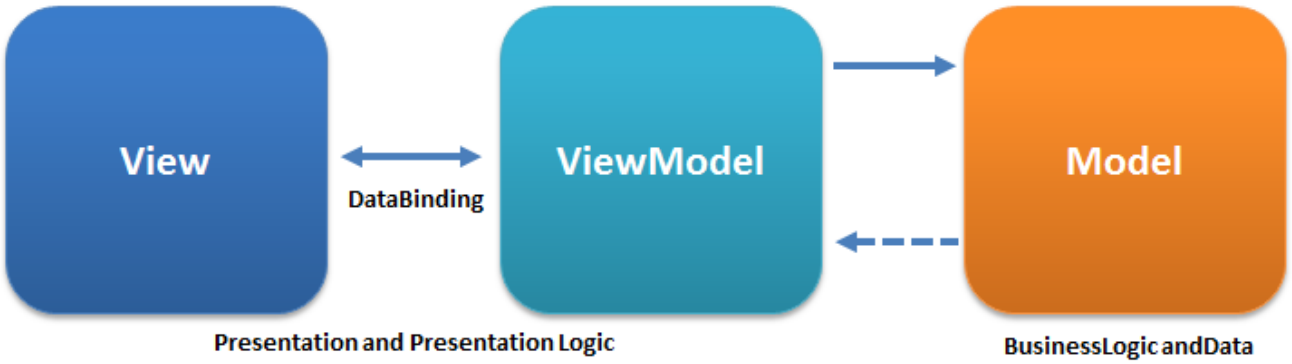
با توجه به ماهیت Loose Couplingکه در این معماری به کار گرفته می‌شود (به عبارتی وابستگی حداقلی مابین بخش‌های مختلف نرم‌افزار وجود دارد)، افزودن فیچرهای جدید و یا تغییر در کدهای قبلی به مراتب آسان‌تر خواهد شد و به عنوان آخرین مزیت معماری MVC هم می‌توان به این نکته اشاره کرد که دیتایی که کنترلر از مدل گرفته را می‌توان به روش‌های مختلفی در معرض دید کاربران قرار داد. به طور مثال، در یک وب اپلیکیشن می‌توان این دیتا را در قالب صفحات اچ‌تی‌امال رِندِر کرده و به کاربران نشان داد و یا می‌توان دیتا را به صورت جیسون در اختیار یک API قرار داده و از طریق یک اپ موبایل داده‌ها را در اختیار کاربران گذاشت (جهت آشنایی با این اصطلاح می‌توانید به آموزش APIچیست؟ مراجعه نمایید.)

در کنار تمامی مزایایی که برای این معماری برشمردیم، یکسری نقاط ضعف هم در ارتباط با اموی‌سی وجود دارد که آگاهی از آن‌ها خالی از لطف نیست. به طور مثال، با توجه به ماهیت Abstraction یا «انتزاع» که در این معماری وجود دارد، وقتی دولوپر جدیدی بخواهد روی چنین پروژه‌هایی کار کند ممکن است در ابتدا کمی سردرگم گردد اما در عین حال می‌توان گفت که این سردرگمی پس از مدتی کار با پروژه به مرور مرتفع خواهد شد.

جمع‌بندی

در پایان لازم به یادآوری است که معماری اموی‌سی تحت هیچ عنوان حاوی یکسری اصول و استانداردهای سخت‌گیرانه نیست که عدم تبعیت از آن‌ها منجر به ایجاد مشکل در اپلیکیشن گردد بلکه اموی‌سی را می‌توان به منزلهٔ یک راهنما به منظور طراحی معماری اپلیکیشن در نظر گرفت که با رعایت آن می‌توان این تضمین را ایجاد کرد که اپلیکیشنی با حداقل پیچیدگی، انعطاف‌پذیری بالا و قابل‌توسعه خواهیم داشت.

معماریMVVM



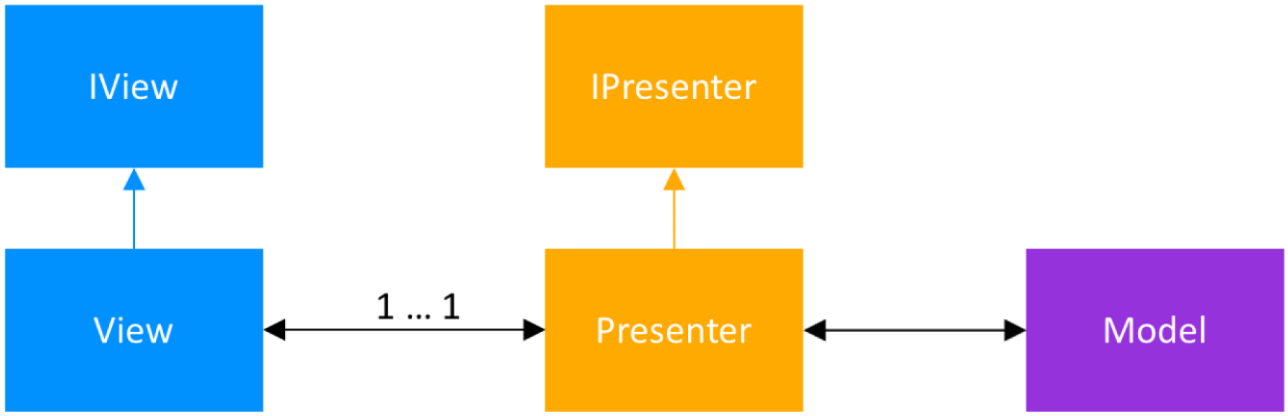
هدف MVVM این هست که با یکسری کلاس های ViewModel پیچیدگی view رو از model و پیچیدگی model از view رو پنهان کنه ، هدف این هست که در یک کلاس اکتیویتی ما داده رو مستقیما دستکاری نکنیم و همچنین با تغییر داده ها مستقیما کامپوننتی رو تغییر ندیم ، خب اینکه کارمون رو زیاد میکنه و حجم کار بالا میره ، بله درسته ، ولی تست هر بخش به تنهایی و جلوگیری از ایرادهای ناخواسته (مثل تغییر وضعیت lifecycleکه به اجبار یه فرگمنت یا اکتیویتی رو destroy میکنه و باعث از دست رفتن اطلاعات میشه) رو راحت تر میکنه.

معماری MVVM و تفاوت آن باMVP

معماری MVVM یا همون Model-View-Viewmodel همانند mvc یا mvp یکی از الگو های معماری نرم افزار است. استفاده از آن به ما کمک میکند کدی تمیز، قابل نگهداری ، قابل تست و با خوانایی بالا داشته باشیم. برای فهم کامل این سری از مقاله ها لازم است با معماری MVP آشنایی داشته باشید. همچنین سطح متوسط به بالا در برنامه نویسی اندروید هم لازم است.

به طور کلی MVVM از سه بخش تشکیل شده

ویو(view): وظیفه نمایش اطلاعات و گرفتن داده های ورودی از کاربر را دارد. اطلاعات گرفته شده از کاربر به viewmodel فرستاده می شود.
ویومدل(viewmodel): داده های دریافت شده از مدل را مدیریت می کند و به view می فرستد.
مدل(model): وظیفه مدیریت داده ها را بر عهده دارد.
در نگاه اول، به نظر می رسد شباهت بسیار زیادی بین معماری MVP و MVVM وجود دارد. انگار presenter به viewmodel تبدیل شده است. اما تفاوت آنها چیزی فراتر از تغییر نام presenter به viewmodel است. همانطور که در شکل زیر می بینید یک سری اینترفیس هم برای پیاده سازی MVP لازم است. اینترفیس IView بعد ها در presenter استفاده می شود و به طور مستقیم view را تغییر می دهد. در صورتی که در معماری MVVM این اتفاق به طور مستقیم نمی افتد یعنی viewmodel به صورت غیر مستقیم view را تغییر می دهد. نکته دیگر این است که ارتباط بین view و presenter در mvp یک به یک است، اما ارتباط بین view و viewmodel یک به چند است.



معماری MVP

معماری MVVM

اینکه در معماری MVP ، ویو به صورت مستقیم توسط presenter تغییر میکند یک سری مشکلات و پیچیدگی ها را به کد اضافه می کند. زیرا ما مجبوریم حواسمان باشد که ویو در چه وضعیتی قرار دارد و اگر در وضعیت یا onDestroy نبود اجازه به روز رسانی ویو را داریم. به همین دلیل ما مجبوریم در presenter وضعیت view را گزارش دهیم. همین باعث می شود تست کردن کد ها، نگهداری و پیچیدگی کدمان افزایش یابد. اما در معماری MVVM دیگر این مشکل وجود ندارد، چرا که دیگر viewmodel به طور مستقیم با view در ارتباط نیست. شاید برایتان سوال باشد که اگر viewmodel با view در ارتباط نیست پس چگونه view را به روز رسانی می کند؟!

چند راه حل برای به روز رسانی view توسط viewmodel پیشنهاد می شود.

دیتا بایندینگ(DataBinding)

لایو دیتا (LiveData)

ار ایکس جاوا(RxJava)

در همه راه حل های پیشنهاد شده هدف این است که یه سری observable (شی هایی هستند که زمانی که تغییر می کنند تغییرشان را به کسانی که آنها را observe (مشاهده) می کنند اعلام میکنند.) در viewmodel داشته باشیم که در view آنها را observe(مشاهده) کنیم. در آموزش های بعدی به بررسی راه حل های موجود می پردازیم.

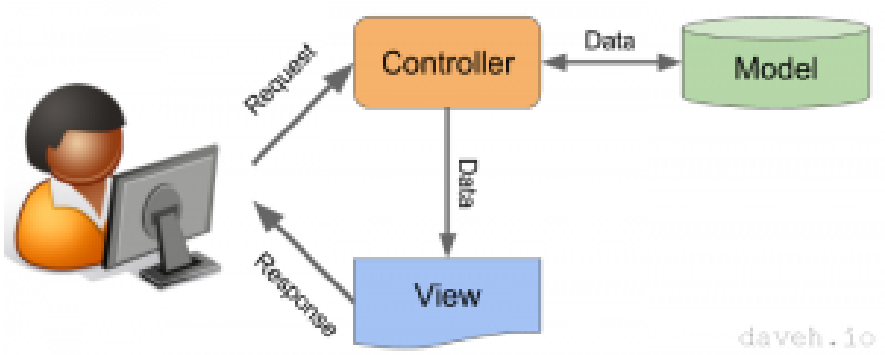
MVC

MVCدر واقع سرواژه ی Model-View-Controller می باشد. که سه مفهوم مهم در آن وجود دارد.

Modelنیز وظیفه ی Model کردنObject ها (از جمله Student یا Teacher و ...) را بر عهده دارد .را Model همچنین می توان ترکیبی از Data +State + Business logic هم دانست .ساده ترین بخش می باشد. و به هیچ Use Case یا Interaction یا حالت خاصی وابستگی ندارد.

Viewبخشی که به عنوان Interface عمل کرده وInteraction های end-user از این طریق با برنامه به انجام خواهد رسید؛ به زبان ساده تر View در واقع Representation بیزینس دیتا می باشد. در MVC می توان گفت که View حالت "dumb" را بازی می کند؛ زیرا View به هیچ وجه از Model ای که باید نمایش دهد اطلاعی ندارد؛ و همچنین از اتفاقاتی که پس از click شدن یک دکمه قرار است اتفاق بیافتد.

Controllerدر این Pattern وظیفه ی Orchestration را بر عهده دارد Controller .را معولا بخش Glue می دانند که باعث میشود که یک بخش های مختلف تشکیل دهنده ی یک Application به هم متصل شوند. بدین معنی که Controller در خواست ها را دریافت و پس از انجام عملیاتی از جمله مثلا گرفتن لیستی از دانشجویان(تعامل با Model) این لیست را به View جهت نمایش به کاربر می فرستد. پس در واقع Controller هم واسطی است میان Model و View و هم بسته درخواست دریافت شده، مشخص می کند که چه Model و چه کنترلی جهت نمایش پاسخ انتخاب شود.



در MVC ما با چند Concern مهم درباره ی Controller سرو کار داریم.
Testability:با توجه به اینکه Controller به View بسیار وابسته است؛ Test نمودن Controller با مشکل و سختی همراه خواهد بود.در MVC ما با چند Concern مهم درباره ی Controller سرو کار داریم.
Modularity & Flexibility:مجدد به دلیل tight coupling با View داشتن این دو ویژگی پس از مدتی با مشکل مواجه خواهد شد. مثلا فرض کنید شما نیاز به تغییر دادن در View خود داشته باشید؛ جهت اعمال این تغییر نیاز خواهید داشت که Controller را نیز تغییر دهید.

Maintenance:پس از مدتی معمولا کدهای بیشتر و بیشتری Business logic به Controller افزوده شده و باعث می شود که Maintain کردن آن به سختی و مشکل فراوان همراه شود.

MVP

MVPنیز acronym هست برای عبارت Model-View-Presenter. در اینجا Controller با Presenter جایگزین شده است. در MVP سعی شده است که interaction میان view و data که در مدل MVC امکان نداشت؛ نیز قابل اعمال باشد. در اینجا Presenter هم با View و هم با Model ارتباط دارد .Presenter ضمن اینکه قادر به دستکاری نمودن Model را دارد می تواند View را نیز Update کند. کاری که درMVC ؛ Controller قادر به انجام آن نبود. در MVC فقط Model مورد نیاز View جهت نمایش آن به کاربر را در اخیار View قرار می دهد.

دو نوع مختلف و متفاوت از پیاده سازی MVP وجود دارد:

Passive View:در این رویکرد View هیچ اطلاعی از Model ندارد؛ و Presenter وظیفه Update کردن Model جهت نمایش در View را بر عهده خواهد داشت. در این استراتژی View و Model کاملا از یکدیگر مستقل و جدا هستند و هیچ تعاملی میان آنها وجود ندارد .Model ممکن استeventای raise کند؛ اما در اینجا Subscriberاین event خود Presenter خواهد بود و Presenter وظیفه ی Update نمودن View را بر عهده خواهد داشت. در اینجا View همانطور که گفته شد هیچ bind مستقیمی ندارد؛ و در عوض هر Property در Viewشامل Setterای هست که توسط Presenter بهنگام نیاز پر خواهد شد. مهمترین مزیت این روش Testabilityبالای آن و clean separation ایست که میان View و Model ایجاد می کند.

Supervising Controller؛ بر خلاف استراتژی قبل در این رویکرد View کاملا و مستقیما با Model در تعامل است و عملیات bind شدن Model درون View بدون دخالت Presenter شدن قابل انجام است. اما در این حالت Presenter هنوز وظیفه ی Update کردن Model را برعهده دارد.

یکی از موارد مهمی که در MVP بدنبال آن هستیم testability بیشتر Business Logic با mock نمودن view می باشد. خوب همانطور که قابل مشاهده است؛ این مورد با Passive View راحت تر و بهتر حاصل خواهد شد.
در صورتی که با user interface های بیشتر و پیچیده تر و تعاملات بیشتر با کاربران سروکار داریم در این حالت MVPبر MVC ارجحیت بیشتری خواهد داشت.

همچنین در صورتی که نیاز به داشتن automated unit test بر روی لایه view خود داشته باشید؛ این مورد با MVPنسبت به MVC بسیار بهتر و راحت تر حاصل خواهد شد.
یکی از تفاوت های مهم دیگر MVC و MVP نیز بدین صورت است که همانطور که بالا اشاره شد؛ در MVC وظیفه ی تعیین View جهت نمایش بر عهده ی Controller می باشد. و این در حالی است که در MVP عکس این قضیه صادق است؛ و route ها به سمت View می روند. در واقع action هایی که در View در هر دو الگوی MVCو MVP وجود دارد؛ یکسان می باشند؛ با این تفاوت که این action ها در MVC به Controller و در MVP به View اشاره می کنند و سپس توسط View به Presenter مربوط delegate خواهد شد.

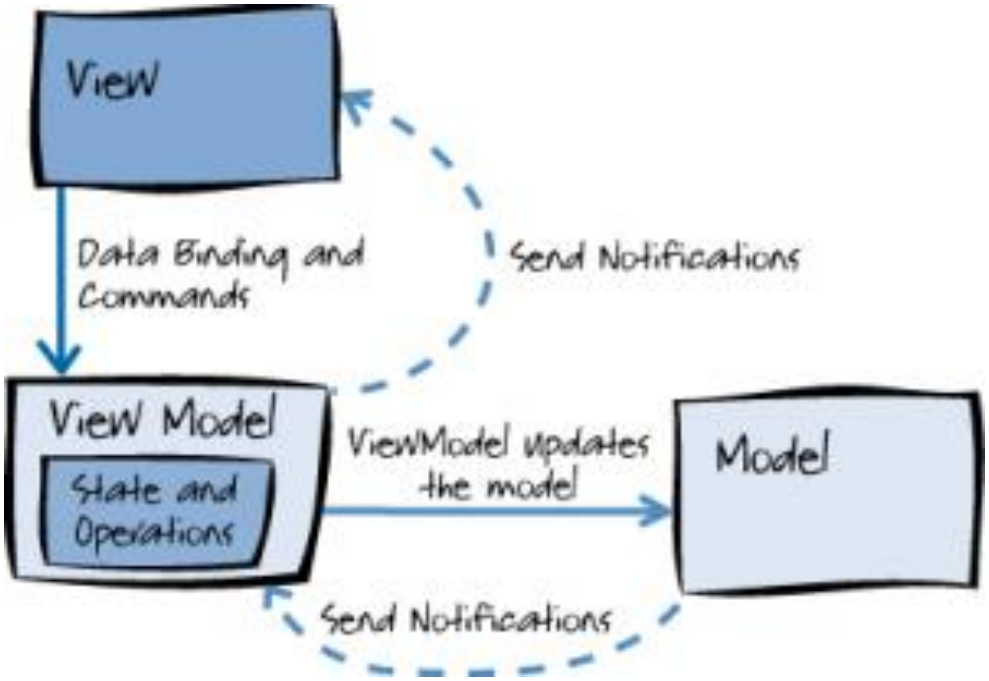
MVVM

هسته ی اصلی این الگو از الگویی بنام Presentation Modelیا Application Model که توسط آقای فاولر معرفی شد؛ گرفته شده است. در این الگو طبق تعریف آقای فاولر Presenter وجود ندارد. و View مستقیما به Presentation Model بایند می شود MVVM سپس با تاکید بیشتر بر Data Binding معرفی شد. این الگو همانند دو الگوی قبل سرواژه ی عبارت Model-View-ViewModel می باشد Model.دقیقا همان نقشی را بازی می کند که بالاتر در الگوی MVC اشاره شد.

Viewدر اینجا به یکسری object های observable بایند خواهد شد. این لایه همچنین می تواند اقدام به raiseنمودن event هایی به ازای interaction ها end user نمایند. این event ها معمولا به action هایی که درون ViewModel تعریف و پیاده سازی شده اند؛ bind می باشند. هم چنین خاصیت observable بودن object های bind شده به View در اینجا این خاصیت را می دهد؛ که در صورت تغییر و بروز رسانی object ها این تغییرات در همان لحظه به درون View نیز منعکس شود.

ViewModel

ViewModelدر واقع از یکطرف به عنوان یک wrapper بر روی Model عمل کرده و خاصیت observable بودن را به Model می دهد. و همچنین این امکان را برای View مهیا می کند که در صورت نیاز بتواند event هایی را به action هایی که درون ViewModel تعریف شده اند raise نماید. یک نکته ی مهم که در اینجا باید به آن اشاره کرد برخلاف دو الگوی قبلی هر چند در اینجا ViewModel کاملا از View بی اطلاع است؛ View اما از ViewModelآگاه و با خبر است. نکته ی دیگر اینکه در این الگو بر خلاف دو الگوی قبلی View کاملا بصورت activeعمل کرده و دارای event و data binding است. البته View در اینجا بهیچوجه state را نگه داری نمی کند. و این بر عهده ی ViewModel می باشد که متدهای و کامندهایی را جهت مدیریت state های برنامه expose.کند



ViewوViewModel در این الگو از طریق method و properties و event ها در تعامل هستند. یکی از هدف های مهم در این الگوی sync نمودن هرچه سریعتر تغییرات به درون view می باشد. و bi-directional و-two way bindingدو روشی هستند که جهت اعمال تغییرات به درون View مورد استفاده قرار می گیرند.

نتیجه گیری:

خوب همانطور که می توان انتظار داشت؛ MVP و MVVM هر دو نسبت به MVC جهت پیاده سازی یک ساختار ماژولار بهتر عمل می کنند؛ هر چند که می توان بوضوح مشاهده نمود که پیاده سازی هر دوی این الگوها در مقایسه با MVC مشکل تر خواهد بود.
برای برنامه ای به تعاملات کمتر و ساده تر MVC بهترین گزینه می تواند باشد؛ اما الگویی مثل MVVM می تواند جهت ایجاد و توسعه برنامه ای که نیازمند reactive model های بیشتر و متنوع تری جهت تعامل با کاربر باشد بوضوح بهتر و مناسبتر می باشد.

بررسی معماری فریم ورک MVPو اجزای آن

سه بخش اصلی این معماری، Presenter، Viewو Model می باشند. در واقع Presenter یک واسط بین دو بخش دیگر است. این بخش معماری، اطلاعات مختلف را از بخش Model دریافت می کند بعد از انجام عملیات مختلفی آنها را به View ارسال می کند. خب شاید بپرسید تفاوت این مدل با مدل MVC در چیست؟
پاسخ روشن است! این مدل بر خلاف مدل MVC می تواند تصمیم بگیرد که وقتی شما با View کار می کنید، چه اتفاقاتی بیافتد.

بخش View چه عملی را انجام می دهد؟ این بخش معمولاً توسط یک تابع یا دستور اجرا می شود و یک بخش ارجاع دهنده دارد که اطلاعات را به Presenter ارجاع می دهد. به زبان ساده تر، هر وقت شما در لایه کاربری و واسط کاربری دستورای را انجام می دهید، کلیک می کنید، لمس می کنید، یک متد را از لایه ی Presenter فراخوانی می کند .و در نهایت بخش Model هم با داده ها سر و کار دارد. به زبان ساده، داده هایی را که قصد داریم در View نمایش دهیم را تهیه می کند.باید بدانید که جدا کردن بخش کد نویسی و بخش منطقی برنامه اندروید از رابط کاربری کار بسیار مشکلی می باشد. ولی معماری فریم ورک MVP این کار را برای شما کمی آسان کرده است. در پروژه های برنامه نویسی اندروید گسترده و بزرگ، دسته بندی و مدیریت خوب کدها بسیار اهمیت دارد و باعث پیشرفت کار می شود. اگر این دسته بندی ها و نظم مناسب اعمال نشود، پشتیبانی و به روز رسانی اپلیکیشن بسیار سخت خواهد بود .این مدل بسیار منظم تر و تمیزتر است. به راحتی می توانید کل قسمت Presenterرا تست کنید. همچنان به راحتی اشکال یابی کنید.