تمرین های درس مهندسی نرم افزار

افشین سابقی
۹۵۰۵۰۷۱۱۳

سوال ۱

ابزار تست SELENIUM چیست؟ روش ها و ابزارهای آن کدامند؟

تست نرم افزار

تست نرم افزار قسمت نهایی از مراحل تولید یک نرم افزار است. بسیاری از پروژه های کوچک و متوسط را پس از برنامه نویسی می توان به صورت دستی تست کرد تا مشکلات احتمالی پیدا و برطرف شوند اما در پروژه های بزرگ به دلیل گستردگی نرم افزار، تست عملکرد نرم افزار بسیار زمان بر و خسته کننده است. به همین دلیل مبحث تست خودکار یا Automated Testing در اینجا اهمیت پیدا می کند.

روش های تست عملکرد

به صورت کلی برای تست عملکرد (functional test) دو روش جعبه سفید (whitebox) و جعبه سیاه (blackbox) وجود دارد. در حالت جعبه سفید، خود برنامه نویس یک روال تست را در قالب unit test برای تک تک کلاس ها و توابع برنامه می نویسد و کد را تست می کند ولی در حالت جعبه سیاه ما دیگر کاری با ساختار داخلی کد نداریم و فقط ورودی لازم را به سیستم می دهیم و انتظار داریم که سیستم خروجی درستی به ما بدهد.

وقتی ما به یکی از دوستانمان می گوییم بیا با وب سایت جدیدی که طراحی کردم کار کن و ببین مشکلی پیدا می کنی یا نه، در اصل داریم همان تست جعبه سفید را انجام می دهیم. تست جعبهٔ سیاه نیز به روشی در تست نرم‌افزار اشاره دارد که در آن فرض می‌شود اطلاعاتی در مورد جزئیات داخلی عملکرد نرم‌افزار وجود ندارد و تمرکز تست‌ها بر روی خروجی‌های مختلف در برابر ورودی‌های متفاوت است.

ابزار تستSelenium

یکی از بهترین ابزارهای تست عمل‌کرد وب سایت، Selenium است که با این نرم افزار، حین طراحی وب سایت ، نه تنها می توانید تست هایی را برای هر بخش تعریف کنید، بلکه می توانید تست ها را زمان بندی کنید و به طور خودکار انجام دهید. فرض کنید وب سایت شما ۱۰۰ ویژگی مختلف دارد، شما همه ی این ۱۰۰ ویژگی را تست و وب سایتتان را منتشر می کنید. بعد از دو روز یک باگ گزارش می شود یا اینکه می خواهید در روند فعالیت کاربر تغییر کوچکی ایجاد کنید. آیا پس از اعمال تغییرات یا رفع باگ، یا به صورت کلی حین طراحی سایت خود، می خواهید دوباره همه ی آن ۱۰۰ ویژگی را تست کنید؟ اینجاست که سلنیوم به کمک شما می آید و همه ی تست ها را به صورت خودکار انجام می دهد.

تعریف برنامهSelenium

در تعریف اولیه از برنامه Selenium باید گفت سلنیوم یک چارچوب تست نرم افزاری قابل حمل برای برنامه های تحت وب است . Seleniumامکان طراحی تست وب سایت را بدون نیاز به دانش زبان اسکریپت تست فراهم می کند. همچنین یک زبان خاص دامنه (Selenese) را برای نوشتن تست در تعدادی از زبان های برنامه نویسی وب محبوب مانند# C ، Groovy، Java، Perl، PHP، Python، Rubyو Scala فراهم می کند. پس از آن، تست ها را می توان در بسیاری از مرورگرهای وب اجرا کرد.

سلنیوم در سیستم عامل های ویندوز، لینوکس و macOS اجر می شود. این نرم افزار متن باز است که تحت مجوز آپاچی ۲,۰ منتشر می شود: توسعه دهندگان وب می‌توانند بدون پرداخت هیچ هزینه ای آن را دانلود و از آن استفاده کنند.

NUnit یک چهارچوب unit –testing (واحد تست) برای همه زبان های net. است . در حال حاضر ورژن ۳,۰ آن با بسیاری امکانات جدید بازنویسی شده و از طیف گسترده ای از پلت فرم های net. پشتیبانی میکند . در این مقاله قصد داریم یک پروژه NUnit testایجاد کنیم و با طرز کار آن آشنا شویم.

NUnit چیست ؟

NUnitیک چهارچوب unit –testing (واحد تست) برای همه زبان های net. است . در حال حاضر ورژن 3.0 آن با بسیاری امکانات جدید بازنویسی شده و از طیف گسترده ای از پلت فرم های net. پشتیبانی میکند.

سوال ۲

Postman چیست؟

Postman یک افزونه گوگل کروم است که با استفاده از آن میشود براحتی متد های یک Web API را اجرا، تست یا بررسی کرد.

این ابزار به ما کمک میکند تا به سرعت یک درخواست HTTP ایجاد و ارسال کنیم.

ذخیره درخواست ها برای استفاده های بعدی، تجزیه و تحلیل پاسخ ها، تغییر سریع محتوای یک درخواست، احراز هویت کاربر با ارسال اطلاعات، سفارشی کردن و... از امکانات خوب این افزونه است که باعث میشود در هنگام توسعه یک API زمان بسیار زیادی را صرفه جویی کنید.

پست‌من در ارتباط با برنامه هایی مانند Azure بسیار مفید است.

Postman ابزاری بسیار سبک بر مبنای وب است که از آن برای تست کردن سایر API ها استفاده می شود. کاربرد این ابزار زمانی مشهود است که برنامه نویس قبل از استفاده از یک مدل API آنرا با Postman بررسی کند. درخواست های API (حتی

درخواست های پیچیده) را می توان با Postman به راحتی ایجاد کرد و جواب آنها را به شکل ها و فرمت های مختلف در مرورگر دریافت نمود. همین کارایی باعث می شود که برنامه نویس بداند کهAPI مورد نظرش چگونه رفتار می کند و برنامه نویس در تصمیم گیری برای استفاده از آن راحت تر باشد.

Rational robot

Rational Robot یک مجموعه کامل از اجزای سازنده برای تست مایکروسافت ویندوز سرویس گیرنده / سرور و برنامه های کاربردی اینترنت است.

مولفه اصلی ربات اجازه می دهد تا تست های ضبط را به اندازه دو کلیک موس انجام دهید. پس از ضبط، ربات تست ها را به صورت جزئی از زمان انجام می دهد تا اقدامات را به صورت دستی تکرار کند.

سوال ۳

متدولوژی یا روش شناسی چیست ؟

متدولوژی خط مشی های گام به گام موسسه ها و شرکتها است که برای تکمیل یک یا چند مرحله از مراحل چرخه تکاملی به کار گرفته میشود . هر متدولوژی تکنیکها و استانداردهای خاص خود را به چرخه تکاملی تحمیل میکند.

متدولوژی ، مجموعه ای از اصول کلی مربوط به روش ها است. که در هر وضعیت مشخص باید به یک روش خاص مناسب به آن وضعیت تبدیل شود .مجموعه ای از روال ها ، فنون ، ابزار و مستنداتی که توسعه دهنگان سیستم در تلاش برای پیاده سازی یک سیستم اطلاعاتی جدید، از آنها بهره می گیرند.

یک متدولوژی ، مرکب است از مراحلی که هر یک به نوبه خود از مراحل فرعی تشکیل شده اند. با کمک این مراحل ، توسعه دهندگان سیستم می توانند در هر مرحله ابزارها و روش های مناسب آن مرحله را انتخاب کرده و پروژه‌های توسعه سیستم ها ی اطلاعاتی را برنامه ریزی ، مدیریت ، کنترل و ارزیابی می کنند.
بر اساس مفاهیم و تئوری عمومی سیستمها نگرشی شکل می گیرد که نگرش یا رویکرد سیستمی نامیده می شود . از طرفی این نگرش یک طرز تفکر است و از طرف دیگر روشی برای برخورد با مسئله است که قابلیت استفاده در حل مسائل سازمانی را بخوبی داراست.

هر گاه این رویکرد در حل مسائل سازمانی و پروژه ها بویژه پروژه های فناوری اطلاعات به کار گرفته شود به عنوان روش کلی حل مسئله ، روش شناسی ، و مسائل روانشناختی نامیده میشود.و هرگاه که برای تحلیل ، طراحی ، بهبود سیستمهای اطلاعاتی مورد استفاده قرار گیرد به عنوان متدولوژی یا روش تجزیه و تحلیل و طراحی سیستم نامیده میشود.

هر تعریفی که از متدولوژی داشته باشیم ،هر متدولوژی باید بتواند به سوالات زیر پاسخ دهد

·چگونه پروژه باید به مراحل فرعی تجزیه شود؟

·در هر مرحله چه اقدام های باید صورت گیرد؟

·در هر مرحله چه خروجی هایی باید تولید شود؟

·در چه زمانی و تحت چه شرایطی باید این وظایف انجام شوند؟

·چه محدودیت هایی باید اعمال شود؟

·چه کسانی باید درگیر شوند؟

·پروژه چگونه باید مدیریت و کنترل شود؟

·از چه ابزارهایی باید استفاده شود؟

سوال ۴

ارزیابیهای مختلف که در خلال چرخه حیات انجام می شوند عبارتند از: مراحل مهم (Milestones) بازرسی ها (Inspection) بازنگری ها (Review) سیر در کد برنامه ها (Walk troughs) مراحل مهم: در پایان هر چهار فاز RUP اتفاق می افتند و موفقیت در رسیدن به اهداف را بررسی می کنند.

این چهار مرحله مهم عبارتند از: مرحله مهم اهداف چرخه حیات (در پایان فاز Inception) مرحله مهم معماری چرخه حیات (در پایان فاز Elaboration) مرحله مهم توانایی های عملیاتی اولیه (در پایان فاز Construction) مرحله مهم انتشار محصول (در پایان فاز Transition) مراحل مهم کوچکتر (Minor Milestones) در پایان هر تکرار اتفاق می افتند و روی بررسی اهداف تکرار تمرکز دارند.

سوال ۵

یک پرونده توسعه نشان می دهد چگونه روند برای یک پروژه یا سازمان خاص اعمال می شود. این شامل جزئیاتی مانند:

که فعالیت های اختیاری و مصنوعات مورد استفاده قرار می گیرد و چه چیزی کاهش خواهد یافت

زمان بندی نسبی فعالیت ها برای هر فاز، چه ابزاری استفاده خواهد شد، و سطح رسمی که باید اعمال شود.

سوال ۶

اصول Versioning

برای Versioning درحال حاضر استاندارد رسمی وجود نداره اما اصول و قوائدی وجود دارن که با تبعیت از اونا میشه به نتیجه مطلوبی رسید. که من هم در این مقاله از اون قوائد)که شامل برخی مستنداتRFC ، قوائدGnu ، مستندات ,Eclipse سبک Semanticو (...پیروی می کنم.

شمای معمول یک Version مناسب و کامل به این صورته:

Major.Minor.Revision/Patch[-/.Build] [Architecture] Release State [Date] [Time]

توجه: واژه های داخل گیومه [] اختیاری هستند یا در بعضی از نرم افزار ها قادر به استفاده از اونها نیستید.

Major

مقدارش چیه؟ اعداد صحیح مثبت و برای انتشار عمومی از ۱ شروع میشه.

چه زمانی عددشو تغییر بدیم؟

زمانی که شکست در public API نرم افزار صورت بگیره) .در حدی که کاربر نتونه نسخه جدید رو نسخه قدیم سازگار کنه) مثل library ها یا شیوه کار با نرم افزار تغییر کنه یا دگرگونی های اساسی و گسترده ای صورت بگیره.

چه تاثیری روی بقیه مشخصه ها میذاره؟ درصورت تغییر عددMajor ، عدد Minor و عدد Revision/Patch مجدداً به ۰ ریست می شن.

مثال: در نسخه ۲,۴,۶ نرم افزار، یک شکست API صورت گرفت، پس نسخه جدید میشه ۳,۰,۰.

نکته ۱: اگر عدد Major تغییر کنه، دیگه نباید سعی کنید نرم افزار رو با نسخه های قبلیش سازگار کنید.

نکته ۲: اگر نرم افزار در اولین فاز توسعه قرار داره و هنوز در دسترس عموم قرار نگرفته، در این صورت فقط عدد Minor و عدد Revision/Patch تغییر می کنن (مثل ۰,۱,۰ ,۰,۱,۱ ,۰,۲,۳ و...) یا جدای از نسخه عمومی، یک نسخه داخلی هم برای نرم افزار درنظر گرفته میشه.

Minor

مقدارش چیه؟ اعداد صحیح مثبت و از ۰ شروع میشه.

چه زمانی عددشو تغییر بدیم؟

زمانی که تغییرات جزئی در public API نرم افزار صورت بگیره. (در حدی که کاربر بتونه با تغییراتی در نرم افزار/دیتا خودش با نسخه جدید سازگار بشه)

یا ظاهر نرم افزار تغییر پیدا میکنه.

یا یک امکان جدید مهم اضافه میشه.

یا تعدادی امکانات جدید کوچیک اضافه میشه.

یا یک ماژول/کامپونت اضافه میشه.

یا performance به مقدار قابل توجهی افزایش پیدا میکنه.

یا روی بخش عمده ای از کد دوباره کار میشه.

چه تاثیری روی بقیه مشخصه ها میذاره؟ درصورت تغییر عدد Minor ، عدد Revision/Patch مجدداً به ۰ ریست می شن.

مثال: در نسخه ۲,۴,۷ نرم افزار، یک ماژول اضافه شده، performanceهم افزایش پیدا کرده، پس نسخه جدید میشه ۲,۵,۰.

Revision/Patch

مقدارش چیه؟ اعداد صحیح مثبت و از ۰ شروع میشه.

چه زمانی عددشو تغییر بدیم؟

زمانی که تغییرات جزئی در کدها صورت بگیره.

یا باگی fix بشه.

یا مستندات نرم افزار تغییر پیدا کنن.

یا بهبود های جزیی صورت بگیره.

یا security patch انجام بگیره.

چه تاثیری روی بقیه مشخصه ها میذاره؟ اغلب هیچی

مثال: در نسخه ۱,۰,۳ نرم افزار، مستندات نرم افزار بروز شدن، تعدادی از dependency ها upgrade شدن، کمی بهبود در کدها صورت گرفت، پس نسخه جدید میشه ۱,۰,۶.

Build

این مشخصه تعیین می کنه که نرم افزار چند بار compile/build/release شده. (یا در بعضی موارد فقط خطای داخلی رفع شده)

مقدارش چیه؟ اعداد صحیح مثبت و از ۱ شروع میشه.

چه زمانی عددشو تغییر بدیم؟

در هربار که نسخه جدیدی compile یا build یا release می کنید یک واحد عددشو بالا می برید.

یا خودش بصورت خودکار در هربار compile یک واحد افزایش پیدا می کنه.

سوال ۷

مشخصات اضافی ضوابط الزامات سیستم را که در موارد استفاده در مورد مدل مورد استفاده قرار نمی گیرند، ضبط می کنند.
الزامات چنین عبارتند از:

الزامات قانونی و قانونی و استانداردهای کاربردی

ویژگی های کیفیت سیستم ساخته شده، از جمله قابلیت استفاده، قابلیت اطمینان، عملکرد، و مورد نیاز پشتیبانی

الزامات دیگر مانند سیستم عامل و محیط، الزامات سازگاری و محدودیت های طراحی

Requirements Attributes

ویژگی های مورد نیاز خواص یک مورد هستند. ویژگی ها اطلاعات اضافی مهمی در مورد یک نیاز را ضبط می کنند. این اطلاعات پس از آن می تواند مورد استفاده قرار گیرد برای پاسخ به پرسش ها در مورد وضعیت پروژه توسعه.

سوال ۸

در تجزیه و تحلیل شما باید نرم افزار را در برابر مشکلات آینده نرمافزار آزمایش کنید

در طراحی شما باید چگونگی انجام این کار را تحریک کنید و از خلاقیت و تخیل شما بستگی دارد.

سوال ۹

سوال ۱۰

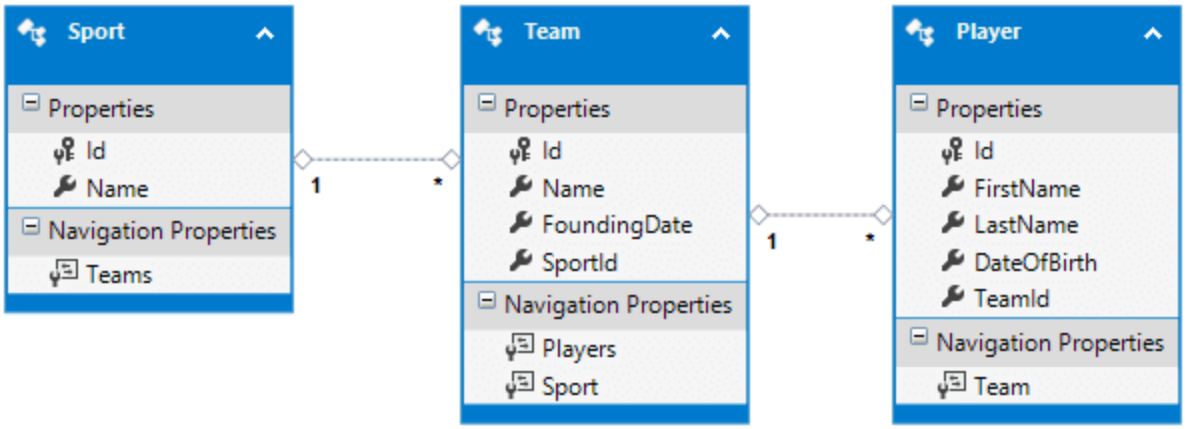
Apache Ant یکی از ابزارهای کامپایل برنامه های Java است که توسط شرکت Apache ارائه شده است.

سوال ۱۱

ما معمولا می‌خواهیم از Entity Frameworkبرای ORMخود استفاده کنیم، اما با کمی جستجو در سؤالات StackExchangeو توضیحات وبلاگ‌ها معلوم می‌شود که چگونه EF واقعا برای سیستم‌های با کارایی بالا چندان رضایت‌بخش نیست. این شکاف با مراحلی که "micro-ORMs" نامیده می‌شوند مثل Dapper.NET پر می‌شود که عملکرد مربوط به قابلیت نگهداری را به عهده می‌گیرد.
چون عملکرد در این برنامه بسیار مهم است، ما می‌خواهیم مطمئن شویم کدام یک از این ORMها بهترین شرایط را برای ما فراهم می‌کنند. بنابراین ما روی یک پروژه نمونه بر روی GitHubکار کردیم که هر یک از این سه روش دسترسی به داده را می‌گیرد تا آن‌ها را با استفاده از داده‌ها و کوئری‌های مشابه (با برخی ملاحظات، همان‌طور که در زیر مشاهده می‌کنید) امتحان کند. این مقاله به بخش‌های زیر تقسیم می‌شود:

1. متدولوژی
2. تنظیم تست
3. نتایج
4. تحلیل
5. نتیجه‌گیری

متدولوژی



این تست از یک ساختار پایگاه داده‌ای استفاده می‌کند که به صورت زیر است:

به عبارت دیگر، ورزش (Sport) تعدادی تیم (Team) دارد و یک تیم تعدادی بازیکن (Player) دارد.

ما نیاز به تعدادی داده برای تست داریم. پروژه نمونه یک بخش کامل اختصاص داده شده به تولید این داده‌ها را دارد، فقط کافی است بگویید با چه شیوه‌ای می‌توانید تعدادی ورزش، تعدادی تیم برای هر ورزش، و تعدادی بازیکن برای هر تیم در هر تست را انتخاب کنید.

آنچه که اکنون ما نیاز داشتیم مجموعه‌ای از کوئری‌ها بود که باید در هر ORM ایجاد کرده و آن را تست می‌کردیم. ما سه کوئری مختلف را انتخاب کردیم:

بازیکن توسط ID

بازیکن‌های هر تیم

تیم‌های هر ورزش (ازجمله بازیکنان)

برای هر کوئری، تست را برای همه داده‌های پایگاه داده اجرا خواهیم کرد (مثلا برای بازیکن توسط ID، هر بازیکن را با IDخودش انتخاب می‌کنیم) و به طور میانگین کل زمانی که برای اجرای کوئری صرف می‌شود (ازجمله تنظیم DB Contextیا (SqlConnectionرا برای هر اجرا در نظر می‌گیریم. سپس چندین اجرا از این مورد را برای داده‌های مشابه انجام می‌دهیم تا بتوانیم میانگین اجرای آن‌ها را محاسبه کرده و به طور واضح نشان دهیم که کدام ORMسریع‌تر است.

توجه داشته باشید که در مورد Dapper.NETو ADO.NET، یک سطر را برای هر بازیکن در کوئری GetTeamsForSportانتخاب خواهیم کرد. این یک مقایسه دقیق در برابر کوئری EFنیست، اما برای هدفی که ما داریم به خوبی کار می‌کند.

نتایج

نتایج زیر برای ۱۰ تکرار هستند، هر کدام شامل ۸ ورزش، ۳۰ تیم در هر ورزش و ۱۰۰ بازیکن برای هر تیم می‌باشد.

نتایج Entity Framework

RUN	PLAYER BY ID	PLAYERS FOR TEAM	TEAMS FOR SPORT
1	1.64ms	4.57ms	127.75ms
2	0.56ms	3.47ms	112.5ms
3	0.17ms	3.27ms	119.12ms
4	1.01ms	3.27ms	106.75ms
5	1.15ms	3.47ms	107.25ms
6	1.14ms	3.27ms	117.25ms
7	0.67ms	3.27ms	107.25ms
8	0.55ms	3.27ms	110.62ms
9	0.37ms	4.4ms	109.62ms
10	0.44ms	3.43ms	116.25ms
Average	**0.77ms**	**3.57ms**	**113.45ms**

نتایج ADO.NET

RUN	PLAYER BY ID	PLAYERS FOR TEAM	TEAMS FOR SPORT
1	0.01ms	1.03ms	10.25ms
2	0ms	1ms	11ms
3	0.1ms	1.03ms	9.5ms
4	0ms	1ms	9.62ms
5	0ms	1.07ms	7.62ms
6	0.02ms	1ms	7.75ms
7	0ms	1ms	7.62ms
8	0ms	1ms	8.12ms
9	0ms	1ms	8ms
10	0ms	1.17ms	8.88ms
Average	**0.013ms**	**1.03ms**	**8.84ms**

نتایج Dapper.NET

RUN	PLAYER BY ID	PLAYERS FOR TEAM	TEAMS FOR SPORT
1	0.38ms	1.03ms	9.12ms
2	0.03ms	1ms	8ms
3	0.02ms	1ms	7.88ms
4	0ms	1ms	8.12ms
5	0ms	1.07ms	7.62ms
6	0.02ms	1ms	7.75ms
7	0ms	1ms	7.62ms
8	0ms	1.02ms	7.62ms
9	0ms	1ms	7.88ms
10	0.02ms	1ms	7.75ms
Average	**0.047ms**	**1.01ms**	**7.94ms**

تحلیل (Analysis)

همانطور که در داده‌هایی که در بالا ذکر شده است می‌بینید، Entity Frameworkبه طور قابل توجهی کندتر از ADO.NETیا Dapper.NETمی‌باشد، که به ترتیب ۱۰-۳ برابر کندتر است.

بیایید مسأله را بازتر کنیم: به کوئری "Teams per Sport" توجه کنید؛ در این کوئری Entity Frameworkهم تیم‌ها در یک ورزش معین و هم بازیکن‌های مربوط به هر تیم (از طریق Include()) را انتخاب کرد، در حالی که کوئری‌های ADO.NETو Dapper.NETفقط داده‌های پیوست شده (joined data) را انتخاب کردند. در یک مطالعه آماری دقیق‌تر، نتایج بهتری را به دست خواهید آورد.

چیزی که جالب‌تر است این است که Dapper.NETبرای کوئری‌های پیچیده‌تر، به طور متوسط سریع‌تر از ADO.NETبود. ما معتقدیم که این موضوع مربوط به این واقعیت است که در مورد تست ADO.NETما از SqlDataAdapterاستفاده می‌کنیم، اگرچه نمی‌توانیم این موضوع را ثابت کنیم.

حتی اگر کوئری "تیم‌های هر ورزش" را هم در نظر نگیریم، هنوز EF حداقل ۳ مرتبه کندتر از Dapper.NET و ADO.NET است. داده‌ها نشان می‌دهند که حداقل از لحاظ سرعت و با این کوئری‌ها، Entity Frameworkآهسته‌ترین گزینه و Dapper.NET سریع‌ترین گزینه خواهد بود. به همین دلیل نتیجه نهایی ما ممکن است شما را شگفت‌زده کند.

نتیجه‌گیری

ما قصد داریم از Dapper.NET روی پروژه خود استفاده کنیم، در این انتخاب شکی نیست، با این حال نمی‌خواهیم توسعه را با آن آغاز کنیم، و تنها ORM مورد استفاده ما نیست. هدف این است که این پروژه را با استفاده از Entity Framework توسعه دهیم، و بعدا با استفاده از Dapper.NET در سناریوی خاصی که سیستم نیاز به افزایش عملکرد دارد پروژه را بهینه‌سازی کنیم.

بله، ما با آهسته‌ترین گزینه شروع می‌کنیم. چرا این کار را می‌کنیم؟

زیرا اشکال بزرگ استفاده از Dapper.NET این است که شما در کد خود کوئری‌های عادی SQL را دارید .اگر هر کسی هر اشتباه تایپی را انجام دهد، ما تا زمانی که تست‌های مربوط به کد را اجرا نکنیم از هیچ مشکلی باخبر نمی‌شویم. به علاوه اعضای تیم ما با EFبیشتر از Dapper.NET آشنا هستند، بنابراین زمان توسعه سریع‌تر خواهد بود.

به طور خلاصه، Dapper.NET بدون شک سریع‌تر از EFو کمی سریع‌تر از ADO.NET است، اما اکثرا ما توسعه را با EF انجام داده و در صورت نیاز آن را با Dapper.NET بهینه‌سازی می‌کنیم. ما فکر می‌کنیم این روش توازنی بین سهولت توسعه و عملکرد ایجاد می‌کند (و امیدوارم به ما اجازه دهید از هر دو روش استفاده کرده و آن را به درستی انجام دهیم).

سوال ۱۲

بر اساس تعریفی که برای **Open/Closed Principle** ارئه می‌شود, یک کلاس باید قابلیت گسترش را داشته باشد, بدون اینکه در ساختار و کد های فعلی آن تغییری ایجاد شود. مفهومی که از این تعریف بر می‌آید این است که کلاس ما باید قابلیت این را داشته باشد که امکانات جدیدی به آن اضافه شود, اما ساختار فعلی آن تغییر نکند. علت آن هم این است که ممکن است چندین کلاس دیگر به این بخش از کلاس نیاز داشته باشند و یا با آن در تعامل باشند و تغییر در ساختار فعلی, ممکن است زنجیره‌ای از تغییرات را به همراه داشته باشد و ممکن است سیستم را از حالت **stable** خارج نماید.

یکی از نمونه هایی که بسیار مشهور است و ساختار **OCP** را در آن به خوبی نشان می‌دهد, **plugin** ها هستند. وقتی یک **plugin** را به یک سیستم اضافه می‌کنیم, در حقیقت به ساختار آن, یک قابلیت جدید اضافه کرده‌ایم. اما این عمل بدون تغییر در ماهیت و ساختار فعلی سیستم انجام شده است. این قابلیتی است که **OCP** می‌تواند به پروژه های ما القا کند. در اینجا یک سیستم حداقل هایی را برای تعامل در اختیار **plugin** قرار می‌دهد اما از جزئیات آن آگاه نیست. این مفهوم در ساختار فنی آن به این معنا است که سیستم با **abstraction** آن **plugin** در تعامل است. می‌توان از این مفهوم چنین نتیجه‌ای را دریافت کرد که یک سیستم چیزی از **plugin** نمی‌داند بلکه **plugin** سیستم را می‌شناسد.

سوال ۱۳

شرکت های نرم افزاری معتبر قبل از اینکه نسخه ای از برنامه را به عنوان نسخه ی نهایی ارائه نمایند نسخه ی دیگری با نام RC)مخفف Release Candidate به معنی کاندیدای انتشار (پخش می نمایند. با توجه به اینکه در این نسخه هیچ گونه مشکل عمده ای وجود ندارد و مشکلات کوچک مانند وجود حفره های امنیتی (که این مشکل همیشه و برای همه ی نسخه های برنامه ها قابل ایجاد و بر طرف کردن کامل آن غیر ممکن است) این نسخه در اختیار شرکتهای معتبر سخت افزاری و نرم افزاری قرار می گیرد تا آنها بدون نگرانی آن را روی سیستم های خود نصب کنند و از آن بازخورد بگیرند. با این حال گاهی نسخه ی Rc نسبت به نسخه ی نهایی تغییرات جزئی (معمولا از نظر برطرف نمودن حفره های امنیتی) دارد.

برخی شرکتها از جمله مایکروسافت به دلیل اینکه به اعتبارشان خدشه ای وارد نشود در ارائه ی نسخه rc فوق العاده دقت می کند چون این نسخه از نظر شرکت به عنوان کاندیدایی برای ارائه ی نهایی می باشد، با این وجود گاهی بیش از یک نسخه ی rc ارائه می شود مانند rc2, rc3 و ... به همین دلیل گاه فاصله یزمانی بین نسخه ی rc و نهایی (Final Release) از شش ماه تجاوز می کند.

سوال ۱۴

استاندارد نرم‌افزار تدوین شده توسط دانشکده مهندسی نرم‌افزار دانشگاه کارنگی ملون آمریکا و مؤسسه SEI (Software Engineering Institute)می باشد (3)که چارچوبی است برای توصیف اجزای کلیدی یک فرآیند کارآمد جهت تولید نرم افزار و همچنین چارچوبی است برای توصیف سیر بهبود تکاملی از یک فرآیند ناکامل و نامنظم به یک فرآیند تکامل یافته و منظم