به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس آزمون نرم‌افزار

گزارش پیشرفت پروژه

مرحله دوم – انجام آزمون

نام و نام خانوادگی

فاطمه مرزانی

شماره دانشجویی

810198324

فهرست

Contents

خلاصه‌ای از گزارش امکان سنجی 4

توابع تحت آزمون 4

پیشگوی آزمون 4

روش تولید خودکار آزمایه 4

انتخاب روش ارزیابی آزمایه 5

گزارش تولید آزمایه ها 6

DealerController 6

تابع GetSuggestedDealers 6

تابع DeleteDealer 7

DealItemController 9

تابع GetDealItemsOfDeal 9

تابع GetDealItem 9

تابع PostDealItem 10

تابع PutDealItem 12

DealController 15

StatisticsController 16

تعداد آزمایه ها 19

چند نمونه از آزمایه ها 19

گزارش اجرای آزمون 24

برخی از مشکلات شناسایی شده در برنامه 24

خروجی سیستم تحت آزمون و خروجی درست 24

گزارش ارزیابی روش 25
گزارش پوشش بدست آمده 25
گزارش نتایج معیار تکمیلی انتخاب شده 26
پوشش دهی بیشتر 27
تعداد آزمایه ها پس از اعمال پوشش دهی بیشتر 31
گزارش پوشش دهی آزمایه ها پس از اعمال پوشش دهی بیشتر 32
گزارش نتایج معیار تکمیلی انتخاب شده پس از اعمال پوشش دهی بیشتر 33
نتیجه گیری و درسهای آموخته شده 34
منابع 35

خلاصه‌ای از گزارش امکان‌سنجی

توابع تحت آزمون

در فاز قبلی پروژه توابع تحت آزمون را مشخص کردیم، در این فاز سعی داشتیم تمام توابع مشخص شده را بیازماییم.

پیش‌گوی آزمون

در بخش دوم فاز گذشته پیش‌گوی آزمون[1] را Project Artifacts و پیاده‌سازی دیگر[2] و رابطه متامورفیک مشخص شده‌بود که با راهنمایی های استاد درس و با توجه به اینکه پیاده سازی‌های دیگری از پروژه وجود نداشت تصمیم بر آن شد که فقط از پیش‌گوی آزمون Project Artifacts و رابطه متامورفیک استفاده شود.

روش تولید خودکار آزمایه

روش‌های آزمون متامورفیک و تولید تصادفی آزمایه برای تولید خودکار آزمایه، انتخاب شده بودند که به منظور تولید تصادفی آزمایه از ابزار استفاده کردیم.

ابزاری برای تست خودکار برنامه های .Net است. برنامه‌نویس مشخصات[3] برنامه را در قالب ویژگی‌هایی که عملکردها[4] ، توابع[5] یا اشیاء[6] باید از آن برآورده کنند ، ارائه می دهد و سپس بررسی می‌کند که آیا این خواص در تعداد زیاد تصادفی از آزمایه ها برآورده می‌شود یا خیر و تلاش می‌کند تا با تولید آزمایه اتوماتیک مناسب مثال نقضی[7] برای آزمون موفق[8] تولید کند. استفاده از این اطمینان را به ما می‌دهد که آزمایه ها به حد کافی قوی تولید شده‌اند.

همچنین خود را به خوبی با unit testing framworks موجود مانند NUnit، xUnit و MSTest ادغام می‌کند و استفاده از آن نیز آسان است.

همچنین برای تولید آزمون متامورفیک، روابط متامورفیک تعریف کردیم و با استفاده از آن آزمایه های تصادفی تولید کردیم تا متوجه شویم که آیا این روابط برقرار هستند یا خیر.

---

[1] Test Oracle
[2] Pseudo Oracle
[3] Specification
[4] methods
[5] Functions
[6] Objects
[7] Counter example
[8] Successful Test

انتخاب روش ارزیابی آزمایه

برای ارزیابی آزمایه ها از دو روش پوشش کد و آزمون موتاسیون استفاده کرده‌ایم.

برای تعیین اینکه چه بخشی از کد پروژه توسط آزمایه ها آزمایش می شود ، می توان از ابزار پوشش کد Visual Studio استفاده کنیم. برای محافظت مؤثر در برابر باگ‌ها ، آزمایه‌ها باید بخش بزرگی از کد را تحت پوشش قرار دهد.

برای سرعت بخشیدن و ایجاد آزمون موتاسیون به تعداد قابل توجه و قابل اتکا از ابزار Stryker.NET استفاده شد. این ابزار آزمون موتاسیون را برای پروژه های net core. و .NET Framework ارائه می دهد. که به ما امکان می دهد آزمایه‌های خود را با وارد کردن باگ‌های موقت در سیستم تحت آزمون، آزمایش کنیم. همچنین به صـــورت خودکار یک گزارش کامل از اجرای آزمایه‌های جهش‌یافته تهیه می‌کند و در اختیار کاربر می‌گذارد.

گزارش تولید آزمایه‌ها

برای تولید آزمایه ها در مرحله اول نیاز بود که داده اولیه [1] تولید کنیم به همین‌منظور کلاس TestDataGenerator.cs ایجاد شده که با استفاده از آن داده اولیه تصادفی تولید می‌کند. سپس باید برای هر تابع تحت آزمون مشخص می‌کردیم که چه آزمون‌هایی باید انجام شود و Project Artifacts و روابط متامورفیک را نیز مشخص می‌کردیم و همچنین روش تولید خودکار آزمایه باید مشخص شود. می‌توان سیستم تحت آزمون را به بخش‌های زیر تقسیم بندی کرد و سپس به شرح جزیات آزمایه‌ها پرداخت:

DealerController

برای آزمودن توابع کلاس DealerController یک کلاس DealerTests ایجاد شده است که در بخش زیر به بررسی توابع تحت آزمون این کلاس می‌پردازیم:

تابع GetSuggestedDealers

توابع آزمون GetSuggestedDealers در فایل DealerTests.cs نوشته شده است.

تابع آزمون	GetSuggestedDealers_RandomInput_ShouldBeValidOkResult()		
روش تولید خود کار آزمایه	**خروجی موردنظر**	**خروجی تابع آزمون**	**passed/failed**
Random Test به عنوان ورودی تابع تحت آزمون، آزمایه ها رشته‌های تصادفی تولید می‌کنند.	OkObjectResult	در مواردی که ورودی تابع مقدار null باشد null reference exception اتفاق می افتد.	failed
تابع آزمون	GetSuggestedDealers_RandomNonNullInput_ShouldBeValidEnumerable()		
Random Test	IEnumerable<Dealer>	IEnumerable<Dealer>	passed

[1] Initial data

به عنوان ورودی تابع تحت آزمون،آزمایه-هارشته‌های تصادفی تولید می‌کنند.	خروجی که لیستی از dealer باشد.		
GetSuggestedDealers_RandomNonNullInput_ShouldNotBeEmpty()			تابع آزمون
Random Test به عنوان ورودی تابع تحت آزمون، آزمایه ها، نام dealer از داده های اولیه به طور تصادفی انتخاب می‌شود و سپس بخشی از نام منتخب به عنوان ورودی تابع تحت آزمون داده می‌شود.	AreNotEqual (dealers.Count, 0) خروجی که لیستی از dealer هست که نباید تعدادش برابر صفر شود .	AreNotEqual(dealers.Count, 0)	passed
GetSuggestedDealers_RandomNonNullInput_ShouldContainsSubName()			تابع آزمون
Random Test به عنوان ورودی تابع تحت آزمون، آزمایه ها، نام dealer از داده های اولیه به طور تصادفی انتخاب می‌شود و سپس بخشی از نام منتخب به عنوان ورودی تابع تحت آزمون داده می‌شود.	IsTrue(d.Name. Contains(subname)) خروجی که لیستی از dealer هست که نام شان شامل رشته ورودی است	IsTrue(d.Name. Contains(subname)))	passed

تابع DeleteDealer

توابع آزمون DeleteDealer در فایل DealerTests.cs نوشته شده است. در بخش زیر به بررسی توابع تحت آزمون این کلاس می‌پردازیم:

تابع آزمون	DeleteDealer_RandomSelectedDeal_ShouldNotFound()		
روش تولید خودکار آزمایه	خروجی موردنظر	خروجی تابع آزمون	passed/failed
Random Test به عنوان ورودی تابع تحت آزمون، به طور تصادفی شناسه تعدادی از dealer ها را انتخاب می‌کند.	NotFoundResult پس از اجرای تابع DeleteDealer اگر همان id را GetDealer باید این خروجی را دریافت کنیم.	NotFoundResult	passed
تابع آزمون	DeleteDealer_DuplicateRandomDelete_ShouldBeNotFoundResult()		
Random Test به عنوان ورودی تابع تحت آزمون، به طور تصادفی شناسه تعدادی از dealer ها را انتخاب می‌کند.	پس از اجرای تابع DeleteDealer اگر همان id را با DeleteDealer را مجددا فراخوانی کنیم باید خروجی BadRequestResul را دریافت کنیم.	BadRequestResul	passed
تابع آزمون	DeleteDealer_RandomDealer_ShouldBeNotFoundResult()		
Random Test به عنوان ورودی تابع تحت آزمون آزمایه ها، شناسه dealer به طور تصادفی انتخاب میشود که این شناسه بین dealer ها وجود ندارد.	NotFoundResult پس از اجرای تابع DeleteDealer باید این خروجی را دریافت کنیم.	NotFoundResult	passed

DealItemController

برای آزمودن توابع کلاس DealItemController یک کلاس DealItemsTests ایجاد شـده اسـت. در بخش زیر به بررسی توابع تحت آزمون این کلاس می‌پردازیم:

تابع GetDealItemsOfDeal

توابع آزمون GetDealItemsOfDeal در فایل DealItemsTests.cs نوشته شده است.

تابع آزمون	GetDealItemsOfDeal_SelectedRandomDeal_ShouldContainsItem()		
روش تـولـیـد خودکار آزمایه	خروجی موردنظر	خروجی تابع آزمون	passed/failed
Random Test به عنوان ورودی تابع تحت آزمون، به طور تصـادفی شـناسـه تعدادی از deal هـا را انتخاب می‌کند.	IsNotEmpty(result) چون معامله با ردیف معامله صـفر ایجاد نمی‌شـود خروجی نباید لیست خالی باشد.	IsNotEmpty(result)	passed
تابع آزمون	GetDealItemsOfDeal_SelectedRandomDeal_ShouldBelongToDeal()		
Random Test به عنوان ورودی تابع تحت آزمون، به طور تصـادفی شـناسـه تعدادی از deal هـا را انتخاب می‌کند.	IEnumerable <DealItem> خروجی بـایـد لیسـتی از ردیف معامله ها باشـد که شـناسـه معامله‌شان با شناسه انتخاب شده برابر باشد.	IEnumerable <DealItem>	passed

تابع GetDealItem

توابع آزمون GetDealItem در فایل DealItemsTests.cs نوشته شده است.

تابع آزمون	GetDealItem_RandomSelectInputId_ShouldBeValid()		
passed/failed	خروجی تابع آزمون	خروجی موردنظر	روش تولید خودکار آزمایه
passed	OkObjectResult	OkObjectResult مشخصات برگردانده شده با مشخصات dealItem انتخاب شده باید یکی باشند.	Random Test به عنوان ورودی تابع تحت آزمون آزمایه ها از بین dealItem اولیه چند dealItem به طور تصادفی انتخاب می‌کند.
تابع آزمون	GetDealItem_RandomInputId_ShouldNotNotFound()		
passed	NotFoundResult	NotFoundResult	Random Test به عنوان ورودی تابع تحت آزمون، آزمایه های تصادفی که با dealItem.Id های موجود متفاوت است تولید می‌کند.

تابع PostDealItem

توابع آزمون PostDealItem در فایل DealItemsTests.cs نوشته شده است.

تابع آزمون	PostDealItem_InvalidMaterialId_ShouldBeBadRequest()		
passed/failed	خروجی تابع آزمون	خروجی موردنظر	روش تولید خودکار آزمایه
failed	CreatedActionResult	BadRequest سیستم نباید ردیف معامله‌ای با شناسه محصولی که در پایگاه داده وجود ندارد، ایجاد کند و خروجی باید BadRequest باشد.	Random Test به عنوان ورودی تابع تحت آزمون، به طور تصادفی شناسه تعدادی از deal ها را انتخاب می‌کند. تعدادی materialId به صورت تصادفی ایجاد می‌کند که در پایگاه داده محصولات وجود ندارد. و قیمت و

تعـداد را نیز بـه صـــورت تصادفی تواید می‌کند.			
PostDealItem_ValidData_ShoulBeValid()			تابع آزمون
Random Test . به عنوان ورودی تابع تحت آزمون، به طور تصـــادفی شناسه تعدادی از deal ها را انتخاب می‌کند. تعدادی materialId به صـــورت تصـــادفی از پایگاه داده محصـولات ایجاد می‌کند. و قیمـت و تعـداد را نیز بـه صـــورت تصـــادفی توا ید می‌کند	CreatedAtActionResult	CreatedAtActionResult	passed
PostDealItem_InvalidPricePerOne_ShoulBeBadRequest()			تابع آزمون
Random Test به عنوان ورودی تابع تحت آزمون، به طور تصـــادفی شناسه تعدادی از deal ها را انتخاب می‌کند. تعدادی materialId به صـــورت تصـــادفی از پایگاه داده محصـولات ایجاد می‌کند. و قیمت منفی و تعداد را نیز به صـــورت تصـــادفی تواید می‌کند	BadRequest سـیتم نباید ردیف معامله‌ای با قیمـت منفی ایجـاد کنـد و خروجی با ید BadRequest باشد.	CreatedAtActionResult	failed
PostDealItem_InvalidQuantity_ShoulBeBadRequest()			تابع آزمون
Random Test . به عنوان ورودی تابع تحت آزمون، به طور تصـــادفی	BadRequest	CreatedAtActionResult	failed

شناسه تعدادی از deal ها را انتخاب می‌کند. تعدادی materialId به صورت تصادفی از پایگاه داده محصولات ایجاد می‌کند. و قیمت و تعداد منفی را نیز به صورت تصادفی تواید می‌کند.	سیستم نباید ردیف معامله‌ای با تعداد منفی ایجاد کند و خروجی باید BadRequest باشد		

تابع PutDealItem

توابع آزمون PutDealItem در فایل DealItemsTests.cs نوشته شده است.

تابع آزمون	PutDealItem_ValidData_ShoulBeValid()		
روش تولید خودکار آزمایه	خروجی موردنظر	خروجی تابع آزمون	passed/failed
Random Test به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند dealItem را انتخاب می‌کنیم و سپس با انتخاب تصادفی از material را و ایجاد تعداد و قیمت تصادفی dealItem جدید می‌سازیم.	NoContentResult در صورت ویرایش صحیح dealItem انتظار می رود خروجی بالا را دریافت کنیم.	NoContentResult	passed
تابع آزمون	PutDealItem_ValidData_ShoulBeUpdated()		
Random Test به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند dealItem را انتخاب می‌کنیم و سپس با انتخاب تصادفی از material را و ایجاد تعداد	در صورت ویرایش صحیح dealItem انتظار می رود پس ازفراخوانی تابع GetDealItem مقادیر dealItem به مقادیر جدید به‌روزرسانی شده باشند.	مقادیر dealItem به مقادیر جدید به‌روزرسانی شده‌اند.	passed

و قیمت تصادفی dealItem جدید می‌سازیم.			
PutDealItem_InvalidId_ShouldBeBadRequest()			تابع آزمون
Random Test . به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند dealItem را انتخاب می‌کنیم و سپس با انتخاب تصادفی از material را و ایجاد تعداد و قیمت تصادفی dealItem جدید می‌سازیم. اما یک شناسه متفاوت از DealItem انتخاب شده را به تابع تحت آزمون می-فرستیم	در صورت ارسال شناسه ردیف معامله متفاوت با شناسه ردیف معامله ای که ویرایش کرده ایم باید خروجی BadRequest دریافت کنیم.	BadRequest	passed
PutDealItem_InvalidDealItem_ShouldBeNotFound()			تابع آزمون
Random Test به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند dealItem که در دیتابیس وجود ندارد تولید می‌کنیم و سپس با انتخاب تصادفی از material را و ایجاد تعداد و قیمت تصادفی dealItem جدید می‌سازیم	در صورت ویرایش ردیف معامله ای که در پایگاه داده وجود ندارد باید خروجی NotFoundResult دریافت کنیم.	Exception خروجی یک exception در سطح پایگاه‌داده برگرداند. انتظار می‌رود که خطاهای لایه پایین تر در توابع تحت آزمون کنترل شده باشند.	failed
PutDealItem_InvalidMaterialId_ShouldBeBadRequest()			تابع آزمون
Random Test	BadRequest	NoContentResult	failed

به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند dealItem که در دیتابیس وجود ندارد تولید می‌کنیم و سپس با انتخاب تصادفی material که در پایگاه داده وجود ندارد انتخاب می‌کنیم با ایجاد تعداد و قیمت تصادفی dealItem جدید می‌سازیم	در صورت ویرایش ردیف معامله به شناسه محصولی که در پایگاه داده وجود ندارد باید خروجی BadRequest دریافت کنیم	ردیف معامله به شناسه محصولی که در پایگاه داده وجود ندارد،ویرایش می‌شود.	
PutDealItem_ChangedDealId_ShouldBeBadRequest()			تابع آزمون
Random Test به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند dealItem را انتخاب می‌کنیم و سپس با انتخاب تصادفی از material را و ایجاد تعداد و قیمت تصادفی dealItem جدید می‌سازیم. اما یک شناسه DealItem دیگر را به تابع تحت آزمون می‌فرستیم	در صورت ارسال شناسه ردیف معامله متفاوت با شناسه ردیف معامله ای که ویرایش کرده ایم باید خروجی BadRequest دریافت کنیم. به این معنا که نمی‌توان یک ردیف معامله را از یک معامله به یک معامله دیگر منتقل کرد.	NoContentResult ردیف معامله به شناسه متفاوت ویرایش می‌شود.	failed
PutDealItem_NegativePrice_ShouldBeBadRequest()			تابع آزمون
Random Test به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند dealItem را انتخاب می‌کنیم و سپس با انتخاب تصادفی از material را و ایجاد تعداد	در صورت ویرایش ردیف معامله به قیمت منفی باید خروجی BadRequest دریافت کنیم.	NoContentResult ردیف معامله به قیمت منفی،ویرایش می‌شود.	failed

و قیمت تصادفی منفی dealItem جدید می‌سازیم.			
PutDealItem_NegativeQuantity_ShouldBeBadRequest()			تابع آزمون
Random Test به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند dealItem را انتخاب می‌کنیم و سپس با انتخاب تصادفی از material را و ایجاد تعداد منفی و قیمت تصادفی dealItem جدید می‌سازیم	در صورت ویرایش ردیف معامله به تعداد منفی باید خروجی BadRequest دریافت کنیم.	NoContentResult ردیف معامله به تعداد منفی،ویرایش می‌شود.	failed
PutDealItem_NullDealItem_ShouldBeBadRequest()			تابع آزمون
Random Test به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند dealItem را انتخاب می‌کنیم و به جای ویرایش و ارسال dealItem جدید مقدارnull ارسال کنیم.	BadRequest	NullReferenceException تابع تحت آزمون قادر به ارسال خطای منطقی برای ورودی null نیست و با exception مواجه می‌شود	failed

DealController

برای آزمودن توابع کلاس DealController یک کلاس DealTests ایجاد شده است.

GetDealsOfDealer_RandomRange_ShouldBeIntegrated()			تابع آزمون
روش تولید خودکار آزمایه	خروجی موردنظر	خروجی تابع آزمون	passed/failed
Metanorphic Random Test	انتظار می‌رود خروجی تابع	خروجی تابع	passed

به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند dealer را انتخاب می-کنیم و سپس آمار کل معاملات را با تابعGetDealsOfDealer و آمار معاملات فروش را با تابع GetSalesOfDealer و آ مار معاملات خرید را با تابع GetPurchasesOfDealer بدست می‌آوریم.	GetDealsOfDealer با مجموع خروجی GetSalesOfDealer و برابر باشد.	GetDealsOfDealer با مجموع خروجی GetSalesOfDealer و برابر است.	

StatisticsController

برای آزمودن توابع کلاس StatisticsControllerکلاس StatisticsTests ایجاد شده است. در بخش زیر به بررسی توابع تحت آزمون این کلاس می‌پردازیم:

تابع آزمون	GetGeneral_InvalidDealerId_ShouldNotFound()		
روش تولید خودکار آزمایه	خروجی موردنظر	خروجی تابع آزمون	passed/failed
Random Test به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند شناسه dealer که در پایگاه داده وجود ندارد به GetGeneral داده می‌شود تا آمار کلی خریدوفروش امروز طرف حساب مشخص شده را برگرداند	انتظار می‌رود خروجی تابع با توجه به اینکه طرف حساب در پایگاه داده وجود ندارد NotFoundResult باشد.	OkObjectResult برای طرف حسابی که وجود ندارد خروجی میدهد که خرید و فروشی وجود نداشته در حالیکه باید خروجی خطا دهد که طرف حساب وجود ندارد.	failed
تابع آزمون	GetGeneralWeekPurchaseAndSalePrice_ShouldBeIntegrated()		

Metanorphic Random Test به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند شنا سه dealer که در پایگاه داده وجود دارد به GetGeneral داده می‌شود تا آمار کلی خریدوفروش امروز طرفحساب مشخص شده را برگرداند.سپس برای همان شناسه با تابع GetTwoWeeksSalesPrice مقدار فروش هفتگی به تفکیک روز محاسبه می‌شود همینطور با تابع GetTwoWeeksPurchasesPrice مقدار خرید هفتگی به تفکیک روز برای طرف حساب مورد نظر محاسبه می‌شود.	انتظار می‌رود مقدار خرید روزانه برابر با مقدار خرید امروز که در گزارش هفتگی نمایش داده می‌شود یکسان باشد همچنین مقدار فروش روزانه با مقدار فروش امروز در گزارش هفتگی برابر باشد.		passed
GetWeeklyTwoWeeklyPurchaseAndSalePrice_ShouldBeIntegrated()			تابع آزمون
Metanorphic Random Test به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند شنا سه dealer که در پایگاه داده وجود دارد به GetWeeklyPurchaseAndSalePricel داده می‌شود تا آمار کلی خریدوفروش هفتگی طرفحساب مشخص شده را به تفکیک روز برگرداند.سپس برای همان شناسه با تابع GetTwoWeeksSalesPrice مقدار فروش هفتگی به تفکیک روز محاسبه می‌شود همینطور با تابع GetTwoWeeksPurchasesPrice مقدار خرید هفتگی به تفکیک روز برای طرف حساب مورد نظر محاسبه می‌شود.	انتظار می‌رود چون بین یک هفته اخیر و دو هفته اخیر رابطه زیر مجموعه او مجموعه برقرار است. مقدار فروش در هر یک از روز های هفته اخیر برابر مقدار فروش همان روز که توسط تابع دو هفته اخیر گزارش داده شده است باشد. همچنین مقدار خرید در هر یک از روز های هفته اخیر برابر مقدار خرید همان روز که توسط تابع دو هفته اخیر گزارش داده شده است باشد.		passed
GetTwoWeeklyMonthlyPurchaseAndSalePrice_ShouldBeIntegrated()			تابع آزمون

Metanorphic Random Test به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند شناسه dealer که در پایگاه داده وجود دارد به GetTwoWeeksSalesPrice داده می‌شود تا آمار کلی فروش دو هفته اخیر طرفحساب مشخص شده را به تفکیک روز برگرداند. سپس برای همان شناسه با تابع GetTwoWeeksPurchasesPrice مقدار فروش دو هفته اخیر به تفکیک روز محاسبه می‌شود همینطور با تابع GetMonthlyPurchaseAndSalePrice مقدار خرید ماهانه به تفکیک روز برای طرف حساب مورد نظر محاسبه می‌شود.		انتظار می‌رود چون بین یک دو هفته اخیر و ماه اخیر رابطه زیر مجموعه او مجموعه برقرار است. مقدار فروش در هر یک از روز های دو هفته اخیر برابر مقدار فروش همان روز که توسط تابع ماه اخیر گزارش داده شده است باشد. همچنین مقدار خرید در هر یک از روز های دو هفته اخیر برابر مقدار خرید همان روز که توسط تابع ماه اخیر گزارش داده شده است باشد.	passed
تابع آزمون	GetMaterialAmountOverTime_ShouldBeIntegrated()		
Metanorphic Random Test به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند شناسه material که در پایگاه داده وجود دارد به تابع GetMaterialAmountOverTime تا لیستی از مقدار موجودی کالای مشخص شده برای طرف حساب را در بازه زمانی شروع تا پایان برگرداند که هر آیتم لیست برابر موجودی کالا در یک تاریخ مشخص است. می‌توانیم برای هر یک از کالا دو مقدار شروع و دو مقدار پایان متفاوت در نظر گرفت به طوری که بازه زمانی اول شامل بازه زمانی دوم نیز شود.		با رابطه زیر مجموعه و مجموعه بودنی که برای ورودی های شروع تا پایان تاریخ ها در نظر گرفتیم انتظار می‌رود موجودی کالا در روزهای مشخص بازه زمانی کوچک تر نیز دقیقا برابر با موجودی کالا در همان روز ها در بازه زمانی بزرگتر باشد.	passed

تعداد آزمایه‌ها

برنامه تحت آزمون مجموعه سی و پنج آزمایه نوشته شد که سیزده تا از آن‌ها با عدم موفقیت اجرا شد.

چند نمونه از آزمایه‌ها

```
     [Test]
        public void GetSuggestedDealers_RandomNonNullInput_ShouldContainsSubName()
        {
            //init
            var dealersResult = sut.Dealers();
            var givenDealers =
dealersResult.GetObject<PagedResult<Dealer>>().Queryable.ToArray();
            var dealersArb = ControllerHelper.ChooseFrom(givenDealers).Where(i =>
i.Name != null).ToArbitrary();

            Prop.ForAll(dealersArb, (dealer) =>
            {
                //execution
                var subName = dealer.Name.Substring(0, Gen.Choose(0,
dealer.Name.Length).Sample(0, 1).Single());
                var result = sut.GetSuggestedDealers(subName);

                //assertion
                var dealers = result.GetObject<IEnumerable<Dealer>>().ToList();

                foreach (var d in dealers)
                {
                    Assert.IsTrue(d.Name.ToLower().Contains(subName.ToLower()), $"
Expected: {d.Name} contains: {subName}");
                }

            }).QuickCheckThrowOnFailure();
        }
```

به عنوان ورودی تابع تحت آزمون، آزمایه ها، نام dealer از داده‌های اولیه به طور تصادفی انتخاب می‌شود و سپس بخشی از نام منتخب به عنوان ورودی تابع تحت آزمون داده می‌شود. خروجی که لیستی از dealer هست که نام‌شان شامل رشته ورودی است.

```
     [Test]
        public void DeleteDealer_DuplicateRandomDelete_ShouldBeNotFoundResult()
        {
            Prop.ForAll<int[]>((i) =>
            {
                //init
                var dealersResult = sut.Dealers();
                var givenDealers =
dealersResult.GetObject<PagedResult<Dealer>>().Queryable.ToArray();

                if (givenDealers.Length != 0)
                {
```

```
                    var dealer =
ControllerHelper.ChooseFrom(givenDealers).Sample(0, 1).Single();

                    //execution
                    var result1 = sut.DeleteDealer(dealer.Id).Result;
                    Assert.IsInstanceOf<OkObjectResult>(result1);
                    var result2 = sut.DeleteDealer(dealer.Id).Result;
                    //assertion
                    Assert.IsInstanceOf<NotFoundResult>(result2);

                }


            }).QuickCheckThrowOnFailure();
        }
```

به عنوان ورودی تابع تحت آزمون، به طور تصادفی شناسه تعدادی از dealer ها را انتخاب می‌کند. پس از اجرای تابع DeleteDealer اگر همان id را باDeleteDealer را مجددا فراخوانی کنیم باید خروجی NotFoundResult دریافت کنیم.

```
    [Test]
        public void GetDealItemsOfDeal_SelectedRandomDeal_ShouldBelongToDeal()
        {
            //init
            var dealsResult = dealController.Deals(1,
int.MaxValue).GetObject<PagedResult<Deal>>().Queryable.ToArray();
            var dealsArb = ControllerHelper.ChooseFrom(dealsResult).ToArbitrary();

            Prop.ForAll(dealsArb, (deal) =>
            {
                //execution
                var result =
sut.GetDealItemsOfDeal(deal.Id).GetObject<IEnumerable<DealItem>>();

                //assertion
                foreach (var r in result)
                {
                    Assert.AreEqual(r.DealId, deal.Id);
                }
            }).QuickCheckThrowOnFailure();
        }
```

به عنوان ورودی تابع تحت آزمون، به طور تصادفی شناسه تعدادی از deal ها را انتخاب می‌کند، خروجی باید لیستی از ردیف معامله ها باشد که شناسه معامله‌شان با شناسه انتخاب شده برابر باشد.

```
    [Test]
        public void PostDealItem_InvalidMaterialId_ShouldBeBadRequest()
        {
            //init
            var materialsResult = materialController.GetMaterials(1,
int.MaxValue).GetObject<PagedResult<Material>>().Queryable.Select(x =>
x.Id).ToList();
```

```
            var deals = dealController.Deals(1,
int.MaxValue).GetObject<PagedResult<Deal>>().Queryable.ToArray();

            var materialIdArb = Arb.Generate<int>().Where(i =>
!materialsResult.Contains(i)).ToArbitrary();
            var dealsGen = ControllerHelper.ChooseFrom(deals);


            Prop.ForAll(materialIdArb, (materialId) =>
            {
                var deal = dealsGen.Sample(0, 1).Head;


                var dealItem = new DealItem
                {
                    DealId = deal.Id,
                    MaterialId = materialId,
                    PricePerOne = Gen.Choose(1, 1000000).Sample(0, 1).Head,
                    Quantity = Gen.Choose(1, 1000).Sample(0, 1).Head
                };

                //execution
                var result = sut.PostDealItem(dealItem).Result;
                //assertion
                Assert.IsInstanceOf<BadRequestResult>(result);

            }).QuickCheckThrowOnFailure();
        }
```

به عنوان ورودی تابع تحت آزمون، به طور تصـــادفی شـــناســـه تعدادی از deal ها را انتخاب می‌کند. تعدادی materialId به صورت تصادفی ایجاد می‌کند که در پایگاه داده محصولات وجود ندارد. و قیمت و تعداد را نیز به صورت تصادفی تواید می‌کند. سیتم نباید ردیف معامله‌ای با شناسه محصولی که در پایگاه داده وجود ندارد، ایجاد کند و خروجی باید BadRequest باشد.

```
     [Test]
        public void PutDealItem_ChangedDealId_ShouldBeBadRequest()
        {
            //init
            var materialsResult = materialController.GetMaterials(1,
int.MaxValue).GetObject<PagedResult<Material>>().Queryable.ToArray();
            var deals = dealController.Deals(1,
int.MaxValue).GetObject<PagedResult<Deal>>().Queryable.ToArray();
            var materialsGen = ControllerHelper.ChooseFrom(materialsResult);
            var dealsGen = ControllerHelper.ChooseFrom(deals);

            Prop.ForAll<int[]>((i) =>
            {
                //execution
                var deal = dealsGen.Sample(0, 1).Head;
                var dealItems =
sut.GetDealItemsOfDeal(deal.Id).GetObject<IEnumerable<DealItem>>().ToArray();
                var dealItem = ControllerHelper.ChooseFrom(dealItems).Sample(0,
1).Head;
                var newDealId = dealsGen.Sample(0, 1).Head.Id;
                var material = materialsGen.Sample(0, 1).Head;
                dealItem.MaterialId = material.Id;
```

```
                dealItem.DealId = newDealId;
                var price = Gen.Choose(1, 1000000).Sample(0, 1).Head;
                dealItem.PricePerOne = price;
                var quantity = Gen.Choose(1, 1000).Sample(0, 1).Head;
                dealItem.Quantity = quantity;
                var result = sut.PutDealItem(dealItem.Id, dealItem).Result;
                Assert.IsInstanceOf<BadRequestResult>(result);

            }).QuickCheckThrowOnFailure();
        }
```

به عنوان ورودی تابع تحت آزمون، ابتدا به صـــورت تصـــادفی چند dealItem را انتخاب می‌کنیم و سپس با انتخاب تصادفی از material را و ایجاد تعداد و قیمت تصادفی dealItem جدید می سازیم. اما یک شناسه DealItem دیگر را به تابع تحت آزمون می‌فرستیم.

در صورت ارسال شناسه ردیف معامله متفاوت با شناسه ردیف معامله ای که ویرایش کرده ایم باید خروجی BadRequest دریافت کنیم.

به این معنا که نمی‌توان یک ردیف معامله را از یک معامله به یکه معامله دیگر منتقل کرد.

```
    [Test]
       public void GetDealsOfDealer_RandomRange_ShouldBeIntegrated()
       {
           //init
           var dealersResult = dealerController.Dealers();
           var givenDealers =
dealersResult.GetObject<PagedResult<Dealer>>().Queryable.ToArray();
           var dealersArb =
ControllerHelper.ChooseFrom(givenDealers).ToArbitrary();
           Prop.ForAll(dealersArb, (dealer) =>
           {
               var deals = sut.GetDealsOfDealer(dealer.Id, 1,
int.MaxValue).Result.GetObject<PagedResult<Deal>>().Queryable.ToList();
               var sales = sut.GetSalesOfDealer(dealer.Id, 1,
int.MaxValue).GetObject<PagedResult<Deal>>().Queryable.ToList();
               var purchases = sut.GetPurchasesOfDealer(dealer.Id, 1,
int.MaxValue).GetObject<PagedResult<Deal>>().Queryable.ToList();
               var union = sales.Union(purchases).ToList();

               foreach (var item in deals)
               {
                   var u = union.Single(i => i.Id == item.Id);


                   Assert.AreEqual(u.SellerId, item.SellerId);
                   Assert.AreEqual(u.BuyerId, item.BuyerId);
                   Assert.AreEqual(u.DealPriceId, item.DealPriceId);
                   Assert.AreEqual(u.DealPaymentId, item.DealPaymentId);
                   Assert.AreEqual(u.DealTime, item.DealTime);
                   union.Remove(u);
               }

               Assert.IsEmpty(union);

           }).QuickCheckThrowOnFailure();
       }
```

به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند dealer را انتخاب می‌کنیم و سپس آمار کل معاملات را با تابعGetDealsOfDealer و آمار معاملات فروش را با تابع GetSalesOfDeale و آمار معاملات خرید را با تابع GetPurchasesOfDealer بدست می‌آوریم.

```
    [Test]
       public void GetWeeklyTwoWeeklyPurchaseAndSalePrice_ShouldBeIntegrated()
       {
           var dealersResult = dealerController.Dealers();
           var givenDealers =
dealersResult.GetObject<PagedResult<Dealer>>().Queryable.Select(i =>
i.Id).ToArray();
           var dealersArb =
ControllerHelper.ChooseFrom(givenDealers).ToArbitrary();
           Prop.ForAll(dealersArb, (dealerId) =>
           {
               var weekly =
sut.GetWeeklyPurchaseAndSalePrice(dealerId).Result.GetObject<IEnumerable<DatePurcha
seAndSalePriceModel>>();

               var twoWeekPurchasePrice =
sut.GetTwoWeeksPurchasesPrice(dealerId).Result.GetObject<IEnumerable<AmountOverDate
Model>>();
               var twoWeekSalesPrice =
sut.GetTwoWeeksSalesPrice(dealerId).Result.GetObject<IEnumerable<AmountOverDateMode
l>>();

               foreach (var daily in weekly)
               {
                   var dayPurchase = twoWeekPurchasePrice.Where(p =>
((DateTime)p.Date).Date == ((DateTime)daily.Date).Date).Single();
                   var daySale = twoWeekSalesPrice.Where(s =>
((DateTime)s.Date).Date == ((DateTime)daily.Date).Date).Single();

                   Assert.AreEqual(daily.PurchasesAmount, dayPurchase.Amount);
                   Assert.AreEqual(daily.SalesAmount, daySale.Amount);
               }

           }).QuickCheckThrowOnFailure();
       }
```

به عنوان ورودی تابع تحت آزمون، ابتدا به صورت تصادفی چند شناسه dealer که در پایگاه داده وجود دارد به GetWeeklyPurchaseAndSalePricel داده می شود تا آمار کلی خرید وفروش هفتگی طرفحساب مشخص شده را به تفکیک روز برگرداند. سپس برای همان شناسه با تابع GetTwoWeeksSalesPrice مقدار فروش هفتگی به تفکیک روز محاسبه می شود همینطور با تابع GetTwoWeeksPurchasesPrice مقدار خرید هفتگی به تفکیک روز برای طرف حساب مورد نظر محاسبه می شود. انتظار می رود چون بین یک هفته اخیر و دو هفته اخیر رابطه زیر مجموعه او مجموعه برقرار است. مقدار فروش در هر یک از روز های هفته اخیر برابر مقدار فروش همان‌روز که توسط تابع دو هفته اخیر گزارش داده شده است باشد. همچنین مقدار خرید در هر یک از روز های هفته اخیر برابر مقدار خرید همان‌روز که توسط تابع دو هفته اخیر گزارش داده شده است باشد.

گزارش اجرای آزمون

برای تولید آزمایه ها ابتدا داده های اولیه صحیح[1] ایجاد شد سپس برای هر یکی از توابع سعی بر آن شد که دامنه ورودی های تابع را به بخش های مختلفی افراز کنیم و بو سیله و آزمون متامورفیک آزمایه ای برای هر یک از افراز ها نوشته شود سپس با توجه به آزمون های انجام شده با استفاده از ارزیابی پوشش کد بخشی‌هایی از برنامه که تحت پوشش آزمایه‌ها قرار نگرفتند شناسایی شد.

بخش بزرگی از مشکلات مرتبط با بررسی صحت داده[2] ورودی برای ایجاد شی جدید یا ویرایش شی بوده است. و بخش دیگری از مشکلات برنامه به علت عدم پاسخ[3] مناسب به کاربر نرم‌افزار می‌باشد.

برخی از مشکلات شناسایی شده در برنامه

- عدم هندل کردن exception هنگام درخواست لیست طرف حساب‌هایی که نامشان شامل مقدار null میباشد.
- امکان ایجاد و ویرایش ردیف معامله با کالایی که ثبت نشده است.
- امکان ایجاد و ویرایش ردیف معامله با قیمت منفی.
- امکان ایجاد و ویرایش ردیف معامله با تعداد کالای منفی.
- عدم هندل کردن exception و ارسال پیام مناسب به کاربر هنگام درخواست ویرایش صورت حسابی که در پایگاه داده وجود ندارد.
- عدم هندل کردن exception و ارسال پیام مناسب به کاربر هنگام درخواست ویرایش صورت حساب به مقدار null.
- عدم ارسال پیام مناسب برای کاربر (Not found) هنگام درخواست گزارش خرید و فروش روزانه،هفتگی،ماهانه برای کاربر ثبت نشده در پایگاه‌داده.

خروجی سیستم تحت آزمون و خروجی درست

مقایسه خروجی سیستم تحت آزمون و خروجی درست در بخش گزارش تولید آزمایه‌ها در جدول مرتبط با هر زیرسیستم آورده شده‌است.

[1] Valid Seed Data
[2] Data Validation
[3] Response

گزارش ارزیابی روش

گزارش پوشش بدست آمده

در ابتدا تمام توابع آزمون که در بخش گزارش تولید آزمایه‌ها مفصلا شرح داده شد با استفاده ابزار پوشش کد Visual Studio محاسبه و در جدول زیر نمایش داده می‌شود.

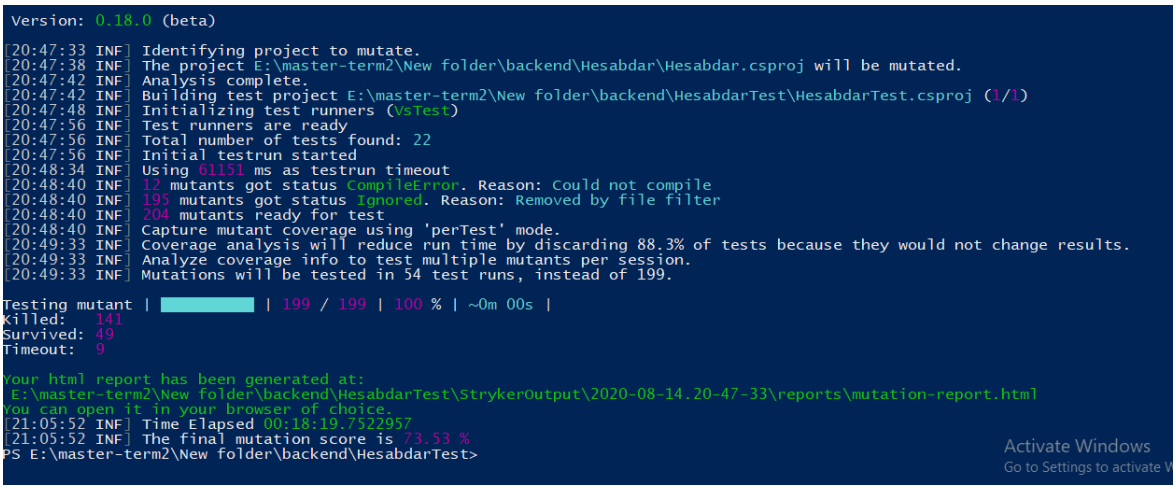
Function Name	Number Of Not Covered Blocks	Percent Of Not Covered Blocks	Number Of Covered Blocks	Percent Of Covered Blocks
GetSuggestedDealers	0	0%	24	100%
DeleteDealer	3	9.68%	28	90.32%
GetDealItemsOfDeal	0	0%	18	100%
GetDealItem	3	11.54%	23	88.46%
PutDealItem	4	16%	21	84%
PostDealItem	3	18.75%	13	81.25%
DealItemsOfMaterial	0	0%	85	100%
DeleteDealItem	3	9.68%	28	90.32%
GetDealsOfDealer	0	0%	85	100%
GetSalesOfDealer	4	6.35%	59	93.65%
GetPurchasesOfDealer	4	6.35%	59	93.65%
GetGeneral	2	1.35%	146	98.65%
GetTwoWeeksSalesPrice	3	2.61%	112	97.39%
GetTwoWeeksPurchasesPrice	2	1.80%	109	98.20%
GetWeeklyPurchaseAndSalePrice	2	0.97%	205	99.03%
GetMonthlyPurchaseAndSalePrice	2	1%	198	99%
GetMaterialAmountOverTime	7	1.89%	363	98.11%
Average		5%		95%

همانطور که مشـــاهده می فرمایید آزمایه ها با میانگین 95 درصـــد کد توابع تحت آزمون را پوشـــش می‌دهند و می‌توان نتیجه گرفت آزمایه‌ها به ن سبت خوبی موثر ه ستند اما هنوز بخ شی ازکد در صورت داشتن باگ شناسایی و آزموده نشده اند.

گزارش نتایج معیار تکمیلی انتخاب شده

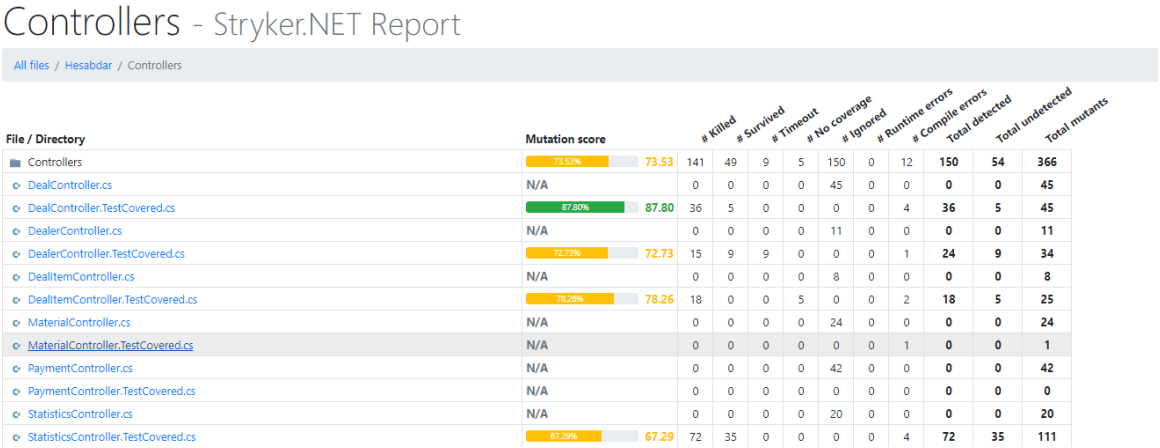
برای ازیابی روش با آزمون موتاسیون از ابزار Stryker.NET استفاده کردیم این ابزار آزمون موتاسیون را برای پروژه های NET Core و .NET Framework ارائه می دهد. به ما امکان می دهد آزمایه‌های خود را با وارد کردن باگ‌های موقت آزمایش کنیم.برای این کار ابتدا توابع تحت آزمون را در فایل های جداگانه با نام TestCovered.cs قرار دادیم و با config کردن Stryker فقط این توابع را mutate کرده‌ایم.

پس از اجرا آزمون موتاسیون 199 آزمایه جهش یافته تولید می‌شود که 141 آنها Killed می‌شوند49 از آن‌ها Survived می‌شوند.



ابزار Stryker.NET یک گزارش کامل از آزمایه های جهش یافته تولید می‌کند که در آدرس

/backend/HesabdarTest/StrykerOutput/2020-08-14.20-47-33/reports/mutation-report.html



File / Directory	Mutation score	# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
Controllers	73.53	141	49	9	5	150	0	12	150	54	366
DealController.cs	N/A	0	0	0	0	45	0	0	0	0	45
DealController.TestCovered.cs	87.80	36	5	0	0	0	0	4	36	5	45
DealerController.cs	N/A	0	0	0	0	11	0	0	0	0	11
DealerController.TestCovered.cs	72.73	15	9	9	0	0	0	1	24	9	34
DealItemController.cs	N/A	0	0	0	0	8	0	0	0	0	8
DealItemController.TestCovered.cs	78.26	18	0	0	5	0	0	2	18	5	25
MaterialController.cs	N/A	0	0	0	0	24	0	0	0	0	24
MaterialController.TestCovered.cs	N/A	0	0	0	0	0	0	1	0	0	1
PaymentController.cs	N/A	0	0	0	0	42	0	0	0	0	42
PaymentController.TestCovered.cs	N/A	0	0	0	0	0	0	0	0	0	0
StatisticsController.cs	N/A	0	0	0	0	20	0	0	0	0	20
StatisticsController.TestCovered.cs	67.29	72	35	0	0	0	0	4	72	35	111

پیوست شده است.

پوشش‌دهی بیشتر

با تمدید شدن زمان تحویل پروژه در نظر گرفتیم پو شش دهی کدها را افزایش دهیم به همین منظور بخش‌هایی از کد که توسط آزمایه‌ها پوشش داده نشده‌بودند بررسی و توابع زیر اضافه شدند:

تابع آزمون	PostDealItem_NullDealItem_ShouldBeBadRequest()		
passed/failed	**خروجی تابع آزمون**	**خروجی موردنظر**	**روش تولید خودکار آزمایه**
failed	Falsifiable در تابع `GetSuggestedDealers(string text)` ورودی تابع تبدیل به حروف کوچک می‌شود `text = text.ToLower();` درصورتیکه ورودی تابع null باشد،خطا رخ می‌دهد.	BadRequestResult	Random Test به عنوان ورودی تابع تحت آزمون، آزمایه ها ردیف معامله null تولید می‌کنند.
تابع آزمون	GetDealsOfDealer_NotExistDealer_ShoudBadRequest()		
passed	BadRequestResult	BadRequestResult در صورت درخواست لیست معامله های طرف حسابی که در پایگاه داده وجود ندارد باید خروجی بالا را دریافت کنیم.	Random به عنوان ورودی تابع تحت آزمون، آزمایه ها طرف حساب هایی که در پایگاه داده وجود ندارد،تولید می-کنند.
تابع آزمون	GetSalesOfDealer_NotExistDealer_ShoudBadRequest()		
passed	BadRequestResult	BadRequestResult	Random

به عنوان ورودی تابع تحت آزمون، آزمایه ها طرف حساب هایی که در پایگاه داده وجود ندارد،تولید می-کنند.	در صورت درخواست لیست معامله های فروش طرف حسابی که در پایگاه داده وجود ندارد باید خروجی بالا را دریافت کنیم.		
GetPurchasesOfDealer_NotExistDealer_ShoudBadRequest()			تابع آزمون
Random به عنوان ورودی تابع تحت آزمون، آزمایه ها طرف حساب هایی که در پایگاه داده وجود ندارد،تولید می-کنند.	BadRequestResult در صورت درخواست لیست معامله های خرید طرف حسابی که در پایگاه داده وجود ندارد باید خروجی بالا را دریافت کنیم.	Random به عنوان ورودی تابع تحت آزمون، آزمایه ها طرف حساب هایی که در پایگاه داده وجود ندارد،تولید می-کنند.	passed
GetSalesOfDealer_NullDealerId_ShoudBadRequest()			تابع آزمون
به عنوان ورودی تابع تحت آزمون، آزمایه ها طرف حساب null،تولید می‌کنند.	BadRequestResult در صورت درخواست لیست معامله های فروش طرف null باید خروجی بالا را دریافت کنیم.	BadRequestResult	passed
GetPurchasesOfDealer_NullDealerId_ShoudBadRequest()			تابع آزمون
به عنوان ورودی تابع تحت آزمون، آزمایه ها طرف حساب null،تولید می‌کنند.	BadRequestResult در صورت درخواست لیست معامله های خرید طرف null باید	BadRequestResult	passed

	خروجی بالا را دریافت کنیم.		
GetGenerall_NullDealerId_ShouldEqualToIdOne()			تابع آزمون
به عنوان ورودی تابع تحت آزمون، ابتدا صورت حساب null به GetGeneralداده می‌شود تا آمار کلی خریدوفروش امروز طرف حساب مشخص شده را برگرداند	انتظار می‌رود اگر شناسه صورت حساب null ارسال شد، آمار خرید و فروش امروز برای شناسه 1 که ادمین تعریف شده محاسبه شود بنابراین آمار خرید و فروش امروز برای صورت حساب null و شناسه یک باید برابر باشد.	Assertion is not violated	passed
GetWeeklySalesAndPurches_NullDealerId_ShouldEqualToIdOne()			تابع آزمون
به عنوان ورودی تابع تحت آزمون، ابتدا صورت حساب null به WeeklySalesAndPurchesدا ده می‌شود تا آمار کلی خریدوفروش هفتگی طرف حساب مشخص شده را به تفکیک روز برگرداند	انتظار می‌رود اگر شناسه صورت حساب null ارسال شد، آمار خرید و فروش هفتگی برای شناسه 1 که ادمین تعریف شده محاسبه شود بنابراین آمار خرید و فروش هفتگی برای صورت حساب null و شناسه یک در تاریخ های یکسان باید برابر باشد.	Assertion is not violated	passed
GetMonthySalesAndPurches_NullDealerId_ShouldEqualToIdOne()			تابع آزمون
به عنوان ورودی تابع تحت آزمون، ابتدا صورت حساب null به MonthlySalesAndPurches	انتظار می‌رود اگر شناسه صورت حساب null ارسال شد، آمار	Assertion is not violated	passed

داده می‌شود تا آمار کلی خریدوفروش ماهانه طرف حساب مشخص شده را به تفکیک روز برگرداند	خرید و فروش ماهانه برای شناسه 1 که ادمین تعریف شده محاسبه شود بنابراین آمار خرید و فروش ماهانه برای صورت حساب null و شناسه یک در تاریخ های یکسان باید برابر باشد.		
GetTwoWeeklyPurchesPrice_NullDealerId_ShouldEqualToIdOne()			تابع آزمون
به عنوان ورودی تابع تحت آزمون، ابتدا صورت حساب null به TwoWeeklyPurchesPriceدا ده می‌شود تا آمار کلی خرید دو هفته اخیر طرف حساب مشخص شده را به تفکیک روز برگرداند	انتظار می‌رود اگر شناسه صورت حساب null ارسال شد، آمار خرید دو هفته اخیر برای شناسه 1 که ادمین تعریف شده محاسبه شود بنابراین آمار خرید دو هفته اخیر برای صورت حساب null و شناسه یک در تاریخ های یکسان باید برابر باشد.	Assertion is not violated	passed
GetTwoWeeklySalesPrice_NullDealerId_ShouldEqualToIdOne()			تابع آزمون
به عنوان ورودی تابع تحت آزمون، ابتدا صورت حساب null به TwoWeeklyPurchesPriceدا ده می‌شود تا آمار کلی فروش دو هفته اخیر طرف حساب مشخص شده را به تفکیک روز برگرداند	انتظار می‌رود اگر شناسه صورت حساب null ارسال شد، آمار فروش دو هفته اخیر برای شناسه 1 که ادمین تعریف شده محاسبه شود بنابراین آمار خرید دو هفته اخیر برای صورت	Assertion is not violated	passed

	حساب null و شناسه یک در تاریخ های یکسان باید برابر باشد.		
GetMaterialAmountOverTime_NullItems_ShouldBeEqualToIdOne			تابع آزمون
تابع GetMaterialAmountOverTime این اجازه را به کاربر می دهد که به جای ورودی های تاریخ آغاز و پایان و شناسه طرف حساب مقدار null دریافت کند. در این آزمایه برای این مقادیر مقدار null می‌فرستیم.	انتظار می‌رود اگر شناسه صورت حساب null ارسال شد، تابع شناسه صورت حساب را برابر یک (شناسه ادمین) قرار دهد. و اگر تاریخ شروع null ارسال شد، تاریخ امروز محاسبه شود. و اگر تاریخ پایان null ارسال شد، تاریخ امروز محاسبه شود.	Assertion is not violated	passed

تعداد آزمایه‌ها پس از اعمال پوشش‌دهی بیشتر

پس از افزودن آزمایه های فوق تعداد آزمایه‌ها به 47 تغییر یافت و تعداد 14 تا از آن‌ها با عدم موفقیت اجرا شدند.

گزارش پوشش‌دهی آزمایه‌ها پس از اعمال پوشش‌دهی بیشتر

Function Name	Number Of Not Covered Blocks	Percent Of Not Covered Blocks	Number Of Covered Blocks	Percent Of Covered Blocks
GetSuggestedDealers	0	0%	24	100%
DeleteDealer	3	9.68%	28	90.32%
GetDealItemsOfDeal	0	0%	18	100%
GetDealItem	3	11.54%	23	88.46%
PutDealItem	4	16%	21	84%
PostDealItem	3	18.75%	13	81.25%
DealItemsOfMaterial	0	0%	85	100%
DeleteDealItem	3	9.68%	28	90.32%
GetDealsOfDealer	0	0%	85	100%
GetSalesOfDealer	0	0	63	100%
GetPurchasesOfDealer	0	0	63	100%
GetGeneral	0	0	148	100%
GetTwoWeeksSalesPrice	0	0	111	100%
GetTwoWeeksPurchasesPrice	0	0	111	100%
GetWeeklyPurchaseAndSalePrice	0	0	207	100%
GetMonthlyPurchaseAndSalePrice	0	0	200	100%
GetMaterialAmountOverTime	0	0	370	100%
Average		3.8%		96.2%

همانطور که مشاهده می‌فرمایید پوشش‌دهی کد از 95 درصد به 96.2 درصد بهبود یافت بخش‌هایی از کد که پوشش داده نشده‌اند مرتبط قسمت هایی از کد زیر هستند که مربوط راستی آزمایی مقادیر ورودی هستند که بر اساس قوانین راستی آزمایی در صورت نقض قوانین این قسمت از کد پوشش داده می‌شود ولی به دلیل عدم تعریف قوانین راستی آزمایی در پروژه، امکان پوشش این بخش وجود ندارد.

```
if (!ModelState.IsValid)
{
    return BadRequest(ModelState);
}
```

گزارش نتایج معیار تکمیلی انتخاب شده پس از اعمال پوشش دهی بیشتر

پس از اجرا آزمون موتاسیون 199 آزمایه جهش یافته تولید می‌شود که 143 آنها Killed می‌شوند47 از آن‌ها Survived می‌شوند

```
[21:26:26 INF] Identifying project to mutate.
[21:26:31 INF] The project E:\master-term2\New folder\backend\Hesabdar\Hesabdar.csproj will be mutated.
[21:26:36 INF] Analysis complete.
[21:26:36 INF] Building test project E:\master-term2\New folder\backend\HesabdarTest\HesabdarTest.csproj (1/1)
[21:26:46 INF] Initializing test runners (VsTest)
[21:26:55 INF] Test runners are ready
[21:26:55 INF] Total number of tests found: 31
[21:26:55 INF] Initial testrun started
[21:27:56 INF] Using 95931 ms as testrun timeout
[21:28:03 INF] 12 mutants got status CompileError. Reason: Could not compile
[21:28:03 INF] 195 mutants got status Ignored. Reason: Removed by file filter
[21:28:03 INF] 204 mutants ready for test
[21:28:03 INF] Capture mutant coverage using 'perTest' mode.
[21:29:26 INF] Coverage analysis will reduce run time by discarding 88.4% of tests because they would not change results.
[21:29:26 INF] Analyze coverage info to test multiple mutants per session.
[21:29:26 INF] Mutations will be tested in 54 test runs, instead of 199.

Testing mutant | ########## | 199 / 199 | 100 % | ~0m 00s |
Killed:   143
Survived: 47
Timeout:  9

Your html report has been generated at:
 E:\master-term2\New folder\backend\HesabdarTest\StrykerOutput\2020-08-14.21-26-26\reports\mutation-report.html
You can open it in your browser of choice.
[21:51:46 INF] Time Elapsed 00:25:19.6746460
[21:51:46 INF] The final mutation score is 74.51 %
PS E:\master-term2\New folder\backend\HesabdarTest>
```

ابزار Stryker.NET یک گزارش کامل از آزمایه های جهش یافته تولید می‌کند که در آدرس

backend\HesabdarTest\StrykerOutput\2020-08-14.21-26-26\reports\mutation-report.html

پیوست شده‌است.

All files - Stryker.NET Report

All files

File / Directory	Mutation score	# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
All files	74.51% 74.51	143	47	9	5	195	0	12	**152**	**52**	**411**
Hesabdar	74.51% 74.51	143	47	9	5	158	0	12	**152**	**52**	**374**
Program.cs	N/A	0	0	0	0	2	0	0	**0**	**0**	**2**
Startup.cs	N/A	0	0	0	0	35	0	0	**0**	**0**	**35**

همانظور که مشاهده می‌فرمایید پس از افزودن توابع آزمون بیشتر mutation score از 73.53 به 74.51 افزایش یافته‌است.

نتیجه گیری و درس‌های آموخته شده

با توجه به اینکه آزمون تصـادفی با ابزار تا حد زیادی این اطمینان را به کاربر می‌دهد که آزمایه ها قوی و در مناطق بحرانی انتخاب شوند و هم چنین پو شش کد قابل توجهی تولید می‌کند، می‌توان نتیجه گرفت که با افراز دامنه ورودی به بخش های مناسب می‌توان نتیجه مطلوب تری گرفت.

همچنین روش آزمون متامورفیک میتواند آزمایه های یکپارچه تولید کند و باعث قوی‌تر شـــدن آزمایه ها در شناسایی باگ‌های احتمالی می‌شود.

بیشتر مشکلات آزمون این نرم‌افزار به علت عدم وجود مستندات کافی از برنامه نرم‌افزاری ناشی می‌شود با وجود مستند پروژه ایجاد پیش‌گوی آزمون دقیق تر و هزینه آزمون نرم‌افزار کمتر خواهد بود.

پروژه درس آزمون نرم‌افزار این فرصـــت را برای بنده فراهم کرد که علاوه بر مطالب درس با ابزار های آزمون خود کار نرم‌افزار مانند QuickCheck و همینطور Stryker.NET که یک ابزار قوی برای ایجاد آزمون موتاسـیون در تعداد بالا و سـرعت بالا می‌باشـد و هزینه ایجاد آزمون موتاسـیون دسـتی را کاهش می‌دهد ،آشنا شوم و مهارت جدیدی برای ایجاد برنامه های درست کسب کنم.

همچنین بررسـی پوشـش دهی‌کدها به منظور ارزیابی کیفیت آزمایه‌ها تولید شـده تجربه ی ارزشـمندی برایم بود.

در پایان وظیفه ی خود می‌دانم از استاد ارجمندم دکتر رامتین خسروی نهایت تشکر را داشته باشم که با وجود محدودیت‌های آموزش مجازی این درس را به نحو احسن تدریس کردند.

منابع

https://docs.microsoft.com/en-us/visualstudio/test/using-code-coverage-to-determine-how-much-code-is-being-tested?view=vs-2019

https://www.nuget.org/packages/dotnet-stryker/0.11.0

https://channel9.msdn.com/Events/Ignite/New-Zealand-2016/M360

https://github.com/stryker-mutator/stryker-net

https://codeburst.io/code-coverage-in-net-core-projects-c3d6536fd7d7

https://github.com/mvakili/hesabdar