شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
3bc695b	هندل کردن نامناسب اورها	d80cf13	تعدادی کلاس جدید برای هندل کردن بهتر flow اضافه شد مانند `NotEnoughCreditExcep tion` و ...
3bc695b	کد تکراری در انواع `Order` ها	d80cf13	اضافه کردن یک سوپر کلاس
3bc695b	استفاده از کد های unsafe و ریترن کردن مقدار نال در بسیاری از جاها	d80cf13	اضافه کردن شرط ها و استفاده از مکانیزم `Error Handling`
3bc695b	استفاده بسیار زیاد از `instance of` و casting	d80cf13	استفاده بهتر از شی گرایی برای `Iceberg Order`
3bc695b	عدم وجود یک entry point ثابت برنامه برای order-های ورودی	d80cf13	تغییر متد `validation` برای داشتن یک entry point ثابت
3bc695b	عدم رعایت encapsulation برای `Request`	d80cf13	
3bc695b	تغییر `Order Handler` برای ساخت و هندل کردن `Request`	d80cf13	با اینکار کمی از قابل دسترس بودن `Request` در بقیه جاها کاهش یافت
3bc695b	تغییر `Security` برای کار کردن با مدل جدید `Request`	d80cf13	
3bc695b	عدم رعایت encapsulation برای `Order Handler`	d80cf13	تغییر accessibility متدها
3bc695b	یکسان سازی متد `deleteOrder` در	d80cf13	

		برای `Security` consistency	
3bc695b	پراکندگی در منطق برنامه	d80cf13	تغییر credit برای `Broker` در جاهای زیادی رخ میداد و تلاش شد یکسان سازی انجام شود
3bc695b	عدم رعایت encapsulation برای `Order Status`	d80cf13	در متدهای `snapshot` این مقدار تغییر میکرد که از لحاظ منطقی خیلی مناسب نبود و تغییر داده شد
3bc695b	مشکل وابستگی در تغییر دادن کد به خاطر اینکه پس از صدا زدن `decreaseQuantity` باید حتما متدهای دیگری صدا زده میشد که Safe نبود	d80cf13	تغییر ساختار `decreaseQuantity`
3bc695b	طولانی بودن و داشتن وظیفه بسیار زیاد `Matcher`	d80cf13	کمی از flow ّرنامه تا حد امکان به بقیه کلاس ها delegate شد
3bc695b	عدم تمیز دادن حالت `Order` در حالت اتمام از بقیه حالات	d80cf13	اضافه کردن `OrderStatus.DONE`
3bc695b	داشتن flow نامناسب برای rollback کردن	d80cf13	منتقل کردن flow برنامه به داخل کلاس های `Order` و `Trade`
3bc695b	متدهای طولانی داخل `Trade`	d80cf13	
3bc695b	کمبود تست ها برای چک کردن منطق کلی برنامه	1cee5c3 تا b472f7	این موضوع در کامیت های مختلف حل شده، در این کامیت بخش های اولیه تست های مورد نیاز اضافه شده، در این کامیت ها در جاهای نیاز

			تغییراتی در ساختار کد داده شد که از حوصله این متن خارج است
3bc695b	در ساختار تست ها duplication بسیاری وجود داشت که سعی شد با یک منطق ساخت سناریو هندل شود	281348f	
3bc695b	تغییر نام `OrderStatus` ها برای خوانایی بیشتر	a3fc190	
d80cf13	پس از تغییر ساختار بررسی `peakSize` در جای درستی انجام نمی شد	1f6d34c	
1cee5c3 تا b472f7	شکستن تست ها به بخش های کوچک تر	18b1c66 تا 9f5a168	هر تست در حال بررسی چند چیز بود و از حالت "unit test" بودن خارج شده بود
05d0d34	داشتن duplication زیاد پس از اضافه کردن SLO	9545e7c	
f5c0248	اضافه کردن سناریو های بیشتر برای جلوگیری از کد تکراری در قسمت تست کردن	cf1a555d	
1a98371	ساختار غیر یکپارچه SLO از بقیه کلاس ها	d705e28	
9918791	اضافه کردن یک لایه جدید `ApplicationService` برای جدا کردن بهتر ساختار برنامه	ecba35b	پس از اضافه کردن SLO-ها به این نتیجه رسیدیم که لایه بندی کد مرز مشخصی ندارد و برای همین یک لایه جدید که به فهم flow برنامه کمک کند اضافه شد

9918791	اضافه کردن `ApplicationServiceResponse` برای یکسان سازی با لایه جدید	35de13	
35de13	منتقل کردن flow مربوط به `validatoin` به داخل لایه جدید برای خوانایی	ef5ad3e	
9918791	حذف دوپلیکیشن ها با کمک ساختار جدید	9545e7c	
9918791	درخواست برای `replenish` در زمان `queue` شدن برای کاهش وابستگی و افزایش portability	14244d2 و 934f543c	
3bc695b	انتقال `Matcher` به داخل `Security`	dbfe1e85	با این کار میتوان وظایف و کارهای مربوط به `Security` را راحت تر و بهتر هندل کرد و نیازی به دسترسی زیاد `Matcher` به بقیه ساختار کد نیست
c38bdd6	ریفکتور حذف یک `Order` و تغییر منطق کاهش `Credit` برای خوانایی	79e11f6	
40c060a	تغییر نام پکیج `security_stats` به `stats` برای اینکه پس از توسعه نرم افزار موجودیت های این پکیج تغییر یافت و نام مناسبی نبود	9d710a7	
7379800	تغییر منطق محاسبه قیمت	c733dba	

	بازگشایی برای خوانایی بهتر		
d3c5e7e	استفاده از stream به جای for-ها برای خوانایی و performance	855cb35	
f505be2	تغییر ساختار هندل کردن ریکوئست ها برای هندل کردن مشکل `InvalidRequestException`	1dee649	در حالت قبلی در حالت کست کردن در صورت دادن ارور برنامه کرش میکرد که در صورتی که این مشکل الان رخ دهد ارور را به لایه بالایی میدهیم
7379800	هندل کردن درست ارور ها در زمان دسترسی به فیلدی ار ریکوئست که وجود ندارد در `ApplicationServiceResponse`	368dbc4	در بعضی جاها بدون توجه به حالت تایپ ها به یکدیگر کست میشدند که در این کامیت به درستی هندل شدند
f9f44c2	تمیز کردن و شکستن تابع های `AuctionBehave` به توابع کوچکتر	89c959e	
f9f44c2	تمیز کردن و شکستن تابع های `ContinuousBehave` به توابع کوچکتر	a8563b6	
f9f44c2	تمیز کردن و شکستن تابع های `Security` به توابع کوچکتر	ce06231	بعضی از بخش ها شکسته شدن و در تابعی قرار داده شدن تا flow برنامه بهتر دیده شود
f9f44c2	تمیز کردن و شکستن تابع های `Matcher` به توابع کوچکتر	4ebb216	
5a15c97	تصحیح تعدادی غلط املایی	41c8490 و 9d62628	

بهبود طراحی دامین

لایه ApplicationServices

یکی از مشکلات طراحی که سبب درک سخت کد مخصوصا برای افراد غیر فنی و کارفرمایانی که متخصص حوزه دامین هستند می‌شد، مشخص نبودن خدماتی بود که اپلیکیشن ارائه می‌داد. این خدمات بر اساس فیلدهای داخلی ریکوئست و یا نوع سفارشی موقتی (temp order) که به security فرستاده می‌شد در مسیر اجرا مشخص می‌شدند. برای رفع این مسئله و افزایش خوانایی کد تصمیم گرفتیم تا لایه‌ای بین Security و OrderHandler قرار دهیم تا تمامی entry point های سرویس‌های مختلفی که برنامه ارائه می‌دهد از هم جدا شود. این کار هم سبب شد که لیستی از خدماتی که اپلیکیشن ارائه می‌کند داشته باشیم و هم خوانایی و تریس (trace) کردن کد به ازای دریافت هر سرویس را به شدت افزایش داد. این بهبود به صورت موازی با بهبودی که در مورد بعد به آن اشاره خواهد شد داده شده و به عبارتی این دو مکمل یکدیگر بوده‌اند. شروع این تغییرات از کامیت ecba35b بوده و حدودا تا کامیت ecba35b ادامه داشته است.

وظیفه و اینترفیس مشخص برای هر لایه

یکی از مواردی که می‌توان آن را جزو المان‌های اصلی طراحی TinyME دانست لایه‌ای بودن معماری آن است که این المان با اضافه کردن Application Services تشدید نیز شد. همچنین از اصلی ترین دلایل استفاده از لایه‌های مختلف در کد جدا کردن وظایف مختلف و ایجاد استقلال در انجام این وظایف است، که نداشتن اینترفیس مشخص بین بخش‌ها و ماژول‌های مختلف و مشخص نبودن وظیفه دقیق هر لایه در جهت عکس این هدف حرکت می‌کرد. برای مثال دقیقا مشخص نبود که ولیدیشن وظیفه کیست و هر تکه از آن در بخشی جدا انجام می‌شد. یا security به عنوان نتیجه MatchResult بر می‌گرداند که مشخصا نتیجه match کردن را در خودش نگه می‌دارد، ولی در خیلی مواقع خبری از match به ازای یک کار خاص در security نیست. پس ساختار داده‌هایی از قبیل SecurityResponse و ApplicationServicesResponse در جهت ثابت و استاندارد کردن نوع خروجی هر لایه به کد اضافه شد. همچنین تا جای امکان وظایف هر لایه به صورت مشخص تعریف شد که این وظایف به شرح زیر است.

- OrderHandler : تشخیص سرویس درخواستی کاربر از روی Request ورودی و صدا زدن آن سرویس. منتشر (publish) کردن رخدادهایی (event) که در مسیر ارائه خدمت مد نظر رخ داده

است. این رخدادها به عنوان بخشی از ApplicationServicesResponse به دست OrderHandler می‌رسید.

- ApplicationServices : انجام ولیدیشن‌های مخصوص به آن سرویس خاص، فراهم آوردن مقدمات ارائه خدمت مثل پیدا کردن سکیوریتی مورد نظر، ساخت ورودی مناسب سکیوریتی از روی Request و در آخر ترجمه گزاره‌های آماری سکیوریتی (SecurityStats) که بخشی از خروجی سکیوریتی یا همان SecurityResponse است به رخدادهای قابل گزارش با توجه به سرویس درخواست شده.
- Security : اجرای نهایی درخواست. پیاده‌سازی منطق نهایی اجرا که در این مسیر از Matcher نیز کمک می‌گیرد.

تمامی مراحل توسعه برنامه سعی شد تا به این هدف نزدیک شویم و دامنه کامیت‌هایی که ما را به این هدف نزدیک کردند بسیار گسترده و پراکنده است. ولی بسیاری از بخش‌های مربوط به آن در بازه بین دو کامیت ecba35b و 18d1d36 انجام شده است.

ماژول بندی منطق برنامه

موجودیت‌های Shareholder، Broker و Order را می‌توان پر تنش ترین موجودیت‌های TinyME در نظر گرفت که ویژگی‌های مختلف آنها مانند quantity، position، credit و حتی جایگاه قرارگیری Order در OrderBook تحت تاثیر بسیاری از سرویس‌ها و فرایندهای جاری در برنامه تغییر می‌کنند. این امر سبب می‌شود که منطق تغییر این فیلدها در جای جای کد پخش باشند. این پراکندگی باعث کاهش شدید خوانایی کد و همچنین افزایش احتمال داشتن کد تکراری می‌شود. برای رفع مشکل ذکر شده تصمیم بر این شد تا ماژول‌های مجزایی برای هندل کردن این موضوع به کد اضافه شود. این ماژول‌ها عبارت اند از PositionControl، CreditControl و QuantityControl. تمامی منطق تغییر این فیلدها در ماژول‌های ذکر شده قرار گرفت و به عنوان یک سرویس در اختیار مابقی بخش‌های کد گذاشته شد تا از آنها استفاده کنند. توسعه این ماژول‌ها نیز موازی با توسعه مورد بعدی انجام شده که بازه کامیت‌های مربوط به این دو مورد در مورد بعدی آمده است.

ماژول‌های MatchingControl

یکی از مشکلات واضح در کد بخش Matching این بود که این کلاس در فرایند execution داشت تعداد زیادی کار که لزومان وظیفه انجام دادن آنها با او نیز نبود را انجام می‌داد. در نگاه اول حل این مسئله بسیار ساده به نظر می‌رسد "کارهایی که به او مربوط نیست را بدهیم یکی دیگر انجام دهد". ولی مشکل اصلی آنجا بود که روند اجرای فرایند execution بر اساس نتایج این کارهای نامرتبط با Matcher ممکن بود که تغییر کند. پس نمی‌توانستیم موجودیتی کاملا مستقل از Matcher داشته باشیم که این فعالیت‌ها را هندل کند، به عبارتی Matcher برای انجام فرایند execution به آن نیاز داشت ولی لازم نبود تا از جزئیات کار کرد آن با خبر باشد. بنابراین Interface ای با نام MatchingControl ساخته شد که به ازای هر مرحله از چرخه اجرای فرایند execution متدی داشت تا هم چک کردن‌های مربوط بر هر مرحله را انجام دهد و Matcher را از وضعیت موجود مطلع کند و هم فعالیت‌های ضروری در روند execution که ربطی به Matcher نداشت، مثل مدیریت credit خریدار و فروشنده را انجام دهد. دو کلاس AuctionMatchingControl و ContinuousMatchingControl این اینترفیس را implement کردند تا Matcher در هر دو روند auctionExecution و continuousExecution تفاوتی احساس نکند. منطقا این MatchingControl ها از ماژول‌های پایه‌ای که در مورد قبلی به آنها اشاره شد استفاده می‌کنند. البته استفاده از آن ماژول‌ها صرفا در MathcingControl ها خلاصه نمی‌شود و کارایی آنها بیشتر از صرفا این بخش است. کامیت‌هایی که منجر به اضافه شدن دو مورد اخیر هستند را می‌توانید در بازه کامیت‌های بین 40c060a و b6f183 مشاهده کنید.

استفاده از State Design Pattern در Security

بعد از اضافه شدن حالت auction به سکیوریتی‌ها عملا به ازای تمامی فعالیت‌هایی که سکیوریتی قبلا در حالت continuous انجام می‌داد یک متد جدید اضافه شد که پیاده‌سازی همان فعالیت در حالت auction در آن قرار داشته باشد. این اتفاق سبب شد که فایل Security پر از if و متدهایی با نام‌های بسیار شبیه به هم شود که تمیزی آن را کاهش می‌داد و فایل مورد بحث را شلوغ می‌کرد. برای حل این موضوع از state design pattern کمک گرفتیم و فعالیت‌های کلی که یک سکیوریتی می‌تواند انجام دهد را در اینترفیسی به اسم SecurityBehave قرار دادیم و دو کلاس AuctionBehave و ContinuousBehave آن را implement کردند. در نهایت اشاره‌گری به SecurityBehave در Security قرار دادیم و نام آن را currentBehave گذاشتیم. با استفاده از این تکنیک رفتاری که سکیوریتی باید در مواجهه با درخواست انجام بدهد به صورت run time و با توجه به نوع currentBehave مشخص می‌شود.

بهبود طراحی تست‌ها

ابزار کمکی AssertingPack

برای بزرگ نشدن تست‌ها و جلوگیری از داشتن assert های فراوان در یک تابع برای تست کردن یک مورد خاص، یک کلاس داخلی به نام AssertingPack در فایل SecurityTest قرار دادیم تا به کمک آن بتوانیم واحدهای معنادار در دامین را به راحتی assert کنیم. به عبارتی این ابزار یک کتابخانه بسیار کوچک و شخصی سازی شده برای assert کردن بخش‌های مختلف در TinyME است.

ابزار کمکی ScenarioGenerator

برای جلوگیری از وجود کد تکراری و همچنین پایبندی به این اصل که در هر تابع تست فقط یک چیز تست شود. ابزار کمکی به نام ScenarioGenerator در فایل SecurityTest اضافه شده که وظیفه آن قرار دادن وضعیت کلی سیستم در سناریو مدنظر است و در اول توابع تست ابتدا سناریو مدنظر از این ابزار صدا زده می‌شود و بعد از قرار گیری سیستم در وضعیت مناسب یک بخش معنادار از دامین به کمک ابزار مورد قبل assert می‌شود.

بهبود بر اساس ماژول‌های جدید اضافه شده

در قسمت قبلی (بهبود طراحی دامین) اشاره شد که ماژول‌های جدیدی به منظور جمع آوری منطق تغییر فیلدهای مختلف در یک جای مشخص به برنامه اضافه شد. حال برای افزایش سرعت پیدا کردن خطا در سیستم و بهبود کیفیت unit test ها می‌توانیم به ازای این ماژول‌های جدید فایل‌های تست مستقل نیز داشته باشیم. با اضافه کردن این فایل‌ها می‌توانیم از حجم فایل SecurityTest کم کنیم و چک کردن هر بخش را به فایل مختص به خود منتقل کنیم.