

تمرین سوم آزمون نرم افزار - پاییز 1404

استاد: دکتر خامس پناه

مجید صادقی نژاد - 810101459

محمد مهدی صمدی - 810101465

بخش تئوری

توضیحات مربوط به شناسایی code smells در اینجا آورده شده است.

• فایل AdminController:

a. این کد مشکل Large Class را دارد، یعنی حجم کد زیادی در آن قرار دارد. در واقع این کلاس به

چند entity مختلف پاسخگو است. این کلاس 15 متود دارد که همه شان به یک business domains مربوط نیستند. در نتیجه فهم این کد سخت است، برای حالات نامربوط به هم تغییر مجبور به تغییرات زیادی هستیم، testability پایین تری دارد زیرا مثلاً برای یک تست ساده باید تمام سرویس ها را Mock کنیم. برای رفع این مشکل با ساختن یک کلاس جداگانه برای متود های هر کدام به کلاس هایی با cohesion بالاتری می رسیم. مثلاً یک کلاس که مربوط به http request های بین order برای admin باشد، یکی دیگر برای product و ...

b. مشکل دیگر این است که Controller صرفاً باید ورودی request را بگیرد، در صورت لزوم validation انجام دهد، سرویس مربوطه را کال کند و خروجی دهد. اما این کنترلر در بعضی متودها، لاجیک هم پیاده سازی کرده است. این مورد در unit test نوشتن مشکل را می شود. زیرا مثلاً برای تست یکی از لاجیک های حتی خیلی ساده باید ستاپ کامل انجام شود. برای رفع آن می توان لاجیک را به صورت کامل به سرویس ها منتقل کرد.

• فایل AdminServices:

a. یک مورد شاید کم اهمیت که به نظر می رسد نام گذاری بد است. مثلاً findByAdminEmail. یا این مورد که اسم متود ها هماهنگی ندارند. مثلاً اسم یکی GetAdmin (باید Get خالی می بود) و دیگری Update (باید UpdateAdmin می بود) است. اگر قرار است استاندارد برای اسم گذاری رعایت شود باید روی همه آنان یک چیز رعایت شود که اینجا نشده است. این مورد شاید ساده

بنظر برسد ولی برای کسی که می‌خواهد کدها را بخواند، وجود یک استاندارد واحد کمک بزرگی است. برای رفع آن باید اسم‌ها را refactor کنیم. این مورد تاثیری روی unit test ندارد.

b. این سرویس تقریباً هیچ لاجیکی ندارد و فقط متود های repository را کال می‌کند. در واقع وجودش اضافی است. البته با توجه به معماری برنامه نمی‌توان آن را حذف کرد اما می‌توان لاجیک کنترلر را به آن منتقل کرد تا حداقل از لحاظ مفهومی درست باشد. این باعث می‌شود نوشتن تست برای این سرویس تا حدی بی‌معنی شود و تمام تست‌ها به لایه دیگری منتقل شوند.

- فایل `UserService`:

a. متود `validateLoginCredentials`، مشکل `Feature Envy` را دارد. در واقع غیر بهینه پیاده‌سازی شده است زیرا یک `equality search` ساده تبدیل شده به یک کوئری برای گرفتن کل آیتم‌ها و سپس فور زدن روی آن. مثلاً برای یک تست ساده باید `UserRepository.findAll` را Mock کنیم. برای رفع آن می‌توان مستقیماً کوئری زد (یا کوئری خالص یا با استفاده از orm البته من جاوا بلد نیستم شاید orm ندارد).

b. در متود `getUser` یک متغیر optional (که انگار به صورت temp رفتار می‌کند) ساخته شده و بلافاصله مقدار آن `get` شده که کار بیهوده‌ای است. باید مانند متود `getUserByEmail` مستقیماً متغیر ساخته شود. این مورد تاثیری روی unit test ندارد.

- فایل `MealPlanService`:

a. متود `generateWeeklyPlan` بیش از حد طولانی است و `redability` کد را کاهش داده است. مثلاً می‌توان بخش `input validation` را یک متود جدا کرد.

b. کلاس `ProductScore` به صورت `private` با فیلد های `private` تعریف شده اما در خروجی متود های کلاس استفاده شده است. که یعنی وقتی یک کلاس از این سرویس استفاده می‌کند، شاید نیاز داشته باشد فرمت خروجی گرفته شده را بداند که در این صورت به آن دسترسی ندارد. اینکه تماماً `private` است باعث می‌شود نوشتن unit test ممکن نباشد.

- فایل `ProductServices`:

a. همان ایرادی که به `getUser` برای `UserService` وارد بود، به `getProduct` اینجا هم است.

b. در متود `getProductByName` قطعه کدی که در خطوط 55-59 قرار دارند می‌تواند به ریترن کردن `product` خلاصه شود. این مشکل تاثیری در unit test ندارد.

c. در متود `updateProduct`، ایف خط 41 اشتباه است زیرا بر اساس همان آیدی سرچ را انجام داده‌ایم و شرط همواره برقرار است. پس باید حذف شود. این ایف باعث می‌شود یک unit test برای آن بنویسیم که همواره پاس می‌شود و تست الکی است.