طراحی و پیاده سازی آزمون خودکار برای نرم افزار گمانیک در شرکت آدانیک

دانشجو: امیرحسین قربانی

استاد راهنما: دکتر احسان خامس‌پناه

دانشکده مهندسی برق و کامپیوتر، دانشگاه تهران

مقدمه

آزمون نرم‌افزار به عنوان یک ابزار حیاتی در ارزیابی کیفیت نرم‌افزار شناخته می‌شود؛ که نه تنها بر کارایی و سرعت پاسخگویی تأکید دارد، بلکه به درستی عملکرد نیز توجه می‌کند. در نرم‌افزارهایی با تعداد زیاد کاربر یا حجم داده بالا، اهمیت آزمون نرم‌افزار بیشتر می‌شود. این روش باعث افزایش اعتماد به نرم‌افزار، شناسایی سریع مشکلات و جلوگیری از تأثیرات منفی بر تجربه کاربری می‌شود.

در این پروژه قصد داریم تا برای سرویس‌های زیر در نرم‌افزار گمانیک آزمون واحد و یکپارچه‌ی خودکار طراحی و پیاده سازی کنیم:

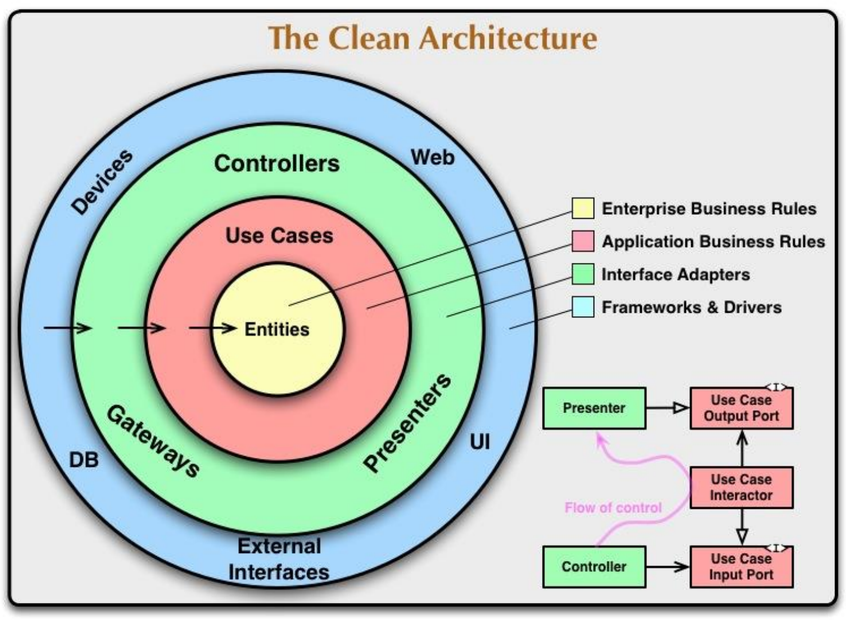
- سرویس Domain
- سرویس Web Server

پس از طراحی و پیاده سازی و اجرای آزمون‌ها می‌خواهیم به نتایج زیر برسیم:

- دستیابی به آزمون‌هایی که تکرار پذیر باشند
- آماده سازی زیرساخت مناسب برای ایجاد آزمون برای سرویس‌های دیگر پروژه
- دستیابی به بخش‌های حساس و حیاتی پروژه که قابلیت آزمون دارند

طراحی و ابزار پیشنهادی

معماری پروژه‌ی گمانیک از نوع Clean Architecture می‌باشد که با توجه به ارتباط اجزای مختلف پروژه با هم باید تست‌های مناسب طراحی شود.



با توجه به معماری بالا در سرویس Domain که نگه‌دارنده‌ی entityهای ما در نرم‌افزار است، برای بخش‌های زیر آزمون واحد نوشته شد:

- Domain (موجودیت‌ها)
  - Filter
  - Format
  - DataComponent
- DTO
- Utils

همچنین برای سرویس Web Server که پخش کننده‌ی درخواست‌های سامانه میان سرویس‌های مختلف است در موارد زیر تست نوشته شده است:

- Entity: دریافت و ایجاد موجودیت‌های سیستم
- Authentication: ورود و خروج و اضافه شدن کاربران به سیستم

برای پیاده سازی آزمون‌ها از ابزار ScalaTest استفاده شده است زیرا گمانیک به طور کامل در زبان Scala نوشته شده است.

نتایج

در زیر پوشش ایجاد شده توسط تست‌های نوشته شده در سرویس Domain را مشاهده می‌کنید. این نتایج توسط ابزار Scoverage تولید شده است که پس از اجرای آزمون‌ها میزان پوشش آن‌ها را مشخص می‌کند.

Lines of code:	9601	Files:	128	Classes:	344	Methods:	879		
Lines per file:	75.01	Packages:	50	Classes per package:	6.88	Methods per class:	2.56		
Total statements:	11031	Invoked statements:	2755	Total branches:	1455	Invoked branches:	176		
Ignored statements:	0								
Statement coverage:	24.98 %			Branch coverage:	12.10 %				

پوشش کلی تست‌ها

Format.scala	29	1	4	4	100.00 %	0	0	100.00 %	
Format.scala	39	2	6	6	100.00 %	2	2	100.00 %	
Format.scala	49	1	6	6	100.00 %	2	2	100.00 %	
Format.scala	59	1	13	13	100.00 %	2	2	100.00 %	
Format.scala	20	1	4	4	100.00 %	0	0	100.00 %	

پوشش تست‌های بخش Format

DataComponent.scala	154	11	49	28	57.14 %	6	3	50.00 %	
DataComponent.scala	160	1	3	3	100.00 %	0	0	100.00 %	
DataComponent.scala	168	1	3	3	100.00 %	0	0	100.00 %	

پوشش تست‌های بخش DataComponent

جمع بندی

با در نظر گرفتن تمام محدودیت‌ها، توانستیم در مجموع تعداد ۴۳۰ تست برای دو سرویس Domain و Web Server بنویسیم که با توجه به تعداد بیشتر تست‌ها در Domain، بتوانیم میزان پوشش آن‌ها در سرویس را بررسی کنیم.

بزرگ‌ترین محدودیت برای اجرای این پروژه پیدا کردن بهترین ابزار و پکیج‌های همخوان با ویژگی‌های پروژه بود. محدودیت بعدی برای نوشتن آزمون‌ها در سیستم گمانیک، پیچیده بودن فرایندها و در هم تنیدگی سناریوها است که باعث می‌شود در بسیاری از موارد جز آن چه که پیاده شد نتوان خروجی قابلverify شدن از USECASEهای آن دریافت کرد.

مراجع اصلی

1. **R.C. Martin, "The Clean Architecture," The Clean Code Blog, August 13, 2012. [Online]. Available: blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html**
2. **B. Venners, "ScalaTest," Artima, Inc., 2024. [Online]. Available: https://www.scalatest.org/**
3. **Scoverage Contributors, "scoverage - Scala Code Coverage Tools," GitHub repository, Last updated January 2024. [Online]. Available: https://github.com/scoverage.**