BAB III
PELAKSANAAN KERJA PROFESI

3.1 Bidang Kerja

Selama melaksanakan kerja profesi di ADW Consulting, praktikan ditempatkan pada Divisi *Business Development* bagian *product development*. Praktikan ditempatkan bertanggung jawab terhadap segala aktivitas pengembangan seluruh produk digital. Praktikan menjalankan tugas sebagai programmer tim produk Iproc pada bagian Iproc 2go. Adapun tugas dan tanggung jawab programmer secara umum di ADW Consulting antara lain:

a. Melakukan analisis detail terhadap sebuah aplikasi.
b. Melakukan perencaaan detail pemrograman terhadap sebuah aplikasi.
c. Membuat program dengan teknologi yang sudah ditentukan oleh perusahaan.
d. Menguji program dari suatu aplikasi yang telah dibuat.
e. Membuat dokumentasi teknis terhadap suatu aplikasi yang telah dibuat.
f. Melakukan penelitian terhadap teknologi baru yang berhubungan dengan pekerjaan.
g. Melaksanakan pekerjaan sebagai bagian dari tim.
h. Membuat laporan secara berkala kepada atasan langsung.

Karena keterbatasan waktu yang dimiliki praktikan untuk menyelesaikan pekerjaan sebagai programmer, tidak semua tugas dan tanggung jawab dapat diselesaikan sepenuhnya. Namun, praktiktikan tetap mendapatkan pembelajaran secara langsung tentang lingkungan dan proses kerja sebagai programmer. Praktikan sebagai programmer telah melaksanakan tahapan dalam siklus hidup pengembagan sistem (SDLC), antara lain perencanaan, desain, implementasi, uji coba dan serta pemeliharaan.

3.2 Pelaksanaan Kerja

Praktikan bekerja sebagai karyawan profesional selama 53 hari, dari 5 Februari 2024 hingga 30 April 2024. Sebagai programmer, praktikan terlibat dalam proyek untuk membuat aplikasi pengadaan barang dan jasa berbasis android. Produk utama akan digunakan untuk klien yang telah menggunakan aplikasi pengadaan berbasis web sebelumnya. Pengembangan aplikasi tersebut telah dimulai sebelum praktikan melaksanakan kerja profesi di ADW Consulting dan saat ini telah sampai pada proses pengembangan pengujian otomatis, meskipun begitu praktikan tetap memahami keseluruhan dari alur pengembagan aplikasi pengadaan barang dan jasa berbasis android tersebut.

Aplikasi pengadaan barang dan jasa, juga disebut sebagai *e-procurement* sendiri merupakan perangkat lunak yang digunakan untuk mengelola proses pengadaan mulai dari perencanaan hingga pembayaran. Aplikasi ini bertujuan untuk meningkatkan transparansi dan akuntabilitas, meningkatkan akses ke pasar, meningkatkan daya saing bisnis, meningkatkan efisiensi proses pengadaan, mendukung pemantauan dan audit, dan memberikan informasi terkini.

Pengujian aplikasi merupakan tahap yang krusial dalam siklus pengembangan aplikasi. Tahap ini bertujuan untuk memastikan fungsionalitas aplikasi berfungsi dengan semestinya. Secara garis besar, pengujian aplikasi dilakukan untuk memastikan bahwa hasil yang diharapkan sesuai dengan hasil yang sebenarnya, serta untuk memastikan bahwa aplikasi bebas dari kesalahan dan bug.

Pengujian yang dilakukan dalam proyek pengembangan aplikasi Iproc 2go menggunakan metode STLC (*Software Testing Life Cycle*). STLC merupakan rangkaian proses dalam pengujian aplikasi yang bertujuan memastikan aplikasi sesuai dengan standar kualitas yang telah ditentukan. STLC terdiri dari enam tahapan, yakni: Analisa kebutuhan (*Requirement Analysis*), Perencanaan Pengujian (*Test Planning*), Pengembangan kasus pengujian (*Test Case Development*), Pengaturan lingkungan pengujian (*Environment setup*), Menjalankan pengujian (*Test Execution*), dan Penutupan kasus pengujian (*Test Case Clousure*).

3.2.1. Analisa Kebutuhan (*Requirement Analysis*)

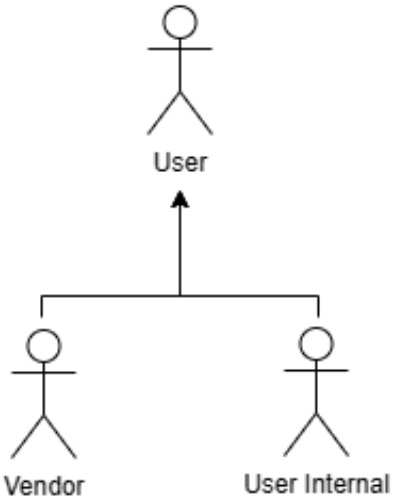
Tahap Analisa kebutuhan merupakan proses melakukan analisa terhadap persyaratan seperti apa yang akan diuji. Pada fase ini tim penjaminan mutu memahami persyaratan seperti apa yang akan diuji. Jika ada yang kurang atau tidak dapat dipahami, maka tim penjaminan mutu akan bertemu dengan pemangku kepentingan untuk lebih memahami pengetahuan rinci tentang persyaratan. Aktivitas yang berlangsung pada tahap analisa kebutuhan antara lain:

a. Meninjau dokumen persyaratan perangkat lunak (SRD) dan dokumen terkait lainnya.
b. Mewawancarai pemangku kepentingan untuk mengumpulkan informasi tambahan
c. Mengidentifikasi ambiguitas atau ketidakkonsistenan dalam persyaratan
d. Mengidentifikasi potensi risiko atau masalah yang mungkin berdampak pada proses pengujian

Berdasarkan tahapan analisa di atas yang telah dilakukan, terdapat beberapa komponen utama yang perlu diperhatikan sebelum melakukan percencaan pengujian. Kompenen – komponen tersebut antara lain:

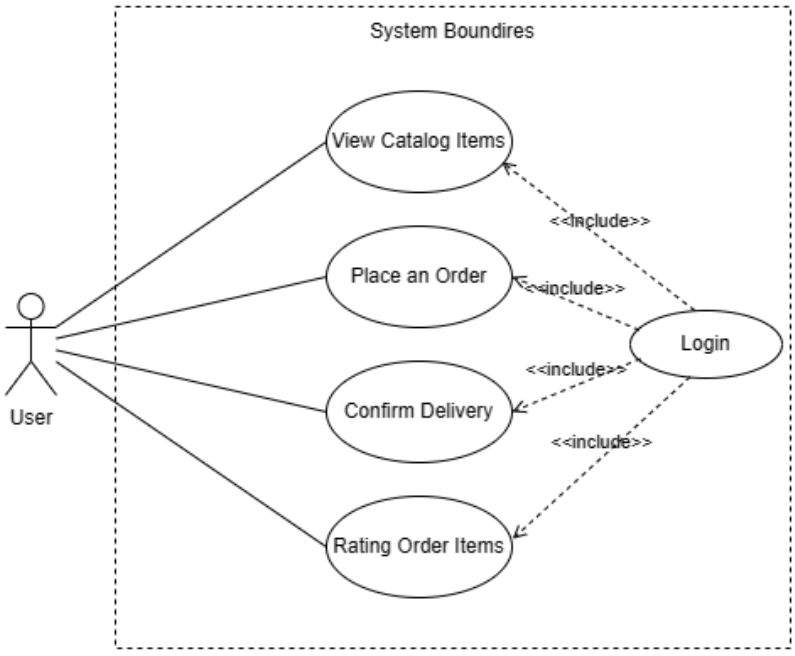
3.2.1.1. *Use Case*

Use case merupakan gambaran ringkas mengenai bagaimana sebuah sistem berfungsi dari perspektif pengguna. Diagram *use case* sendiri merupakan representasi visual yang menunjukkan secara fungsional bagaimana pengguna dapat berinteraksi dengan sistem, serta bagaimana sistem merespons aksi dari pengguna atau aktor yang terlibat. Aktor pada *use case* digeneralisasi berdasarkan beberapa *role*, yaitu:



Gambar 3.1 User Generalition

Adapun *use case* terkait fitur *ecatalog* pada aplikasi Iproc 2Go sebagai berikut.



Gambar 3.2 Use Case Iproc 2GO pada fitur Ecatalog

Deskripsi *use case* mencakup semua detail yang diperlukan untuk membuat diagram *use case*. Dengan membuat deskripsi *use case*, pengguna dapat menjelaskan secara

terperinci tentang setiap ***use case*** individu yang diperlukan. Adapun deskripsi *use case* dari gambar 3.1 sebagai berikut.

Table 3.1 Use Case Description Melihat Katalog Barang

Use Case Name	Menelusuri Katalog Barang	**ID**	UC1	**Importance Level**	High
Primary Actor	Admin Utama			**Use Case Type**	Detail, Essential
Stakeholder and Interest :					
User Internal	Dapat menelusuri daftar barang yang tersedia				
Brief Description :					
Dalam use case ini diuraikan bagaimana user internal dapat menelusuri katalog barang dengan menggunakan fitur pencarian barang.					
Trigger :					
Saat User membuka halaman eCatalog.					
Preconditions :					
1. User sudah login					
2. Memiliki akses terhadap proses eCatalog					
Normal Flow :					
Actor Actions	System Responses				
1. User Internal membuka menu eCatalog	1.1. Sistem menampilkan daftar ecatalog				
2. User Internal melakukan pencarian Katalog berdasarkan nama barang / Jasa atau menggunakan filter lainnya.	2.1. Sistem menampilkan daftar pencarian ecatalog berdasarkan kata kunci atau filter yang diteriapkan				
3. User Internal membuka Katalog Barang yang dipilih	3.1. Sistem menampilkan halaman Detail Produk				
Alternative Flow :					
Actor Actions	System Responses				
Exceptional Flow :					
1E. Jika terjadi kegagalan pada saat memuat data Katalog Barang, maka aplikasi akan menampilkan pesan error dan alasan kegagalannya					

Table 3.2 Use Case Description Melakukan Pemesanan

Use Case Name	Pemesanan Barang / Jasa	**ID**	UC2	**Importance Level**	High
Primary Actor	User Internal			**Use Case Type**	Detail, Essential
Stakeholder and Interest :					
User Internal	Dapat melakukan Pemesanan				
Brief Description :					
Didalam use case ini diuraikan bagaimana user internal dapat melakukan proses pemesanan barang atau jasa yang dipilih oleh user.					
Trigger :					
Saat User menekan tombol tambah keranjang pada halaman detail eCatalog					
Preconditions :					
1. User sudah login					
2. Memiliki akses terhadap proses eCatalog					
Normal Flow :					
Actor Actions	System Responses				
1. User Internal membuka menu eCatalog	1.1. Sistem menampilkan daftar ecatalog				
2. User Internal melakukan pencarian Katalog berdasarkan nama barang / Jasa atau menggunakan filter lainnya.	2.1. Sistem menampilkan daftar pencarian ecatalog berdasarkan kata kunci atau filter yang diteriapkan				
3. User Internal membuka Katalog Barang yang dipilih	3.1. Sistem menampilkan halaman Detail Produk				
4. User Internal menambahkan Barang/Jasa ke dalam keranjang	4.1. Sistem menampilkan pesan berhasil menambahkan ke dalam keranjang				
5. User Internal membuka menu keranjang	5.1. Sistem menampilkan halaman Keranjang				
6. User Internal melakukan proses checkout barang yang	6.1. Sistem menampilkan halaman Checkout				

ada di keranjang	6.2. Sistem menampilkan pesan berhasil checkout pesanan
Alternative Flow :	
Actor Actions	System Responses
Exceptional Flow :	
1E. Jika terjadi kesalahan saat memuat data Katalog Barang, aplikasi akan menampilkan pesan error yang menjelaskan alasan kegagalan tersebut.	
2E. Jika persediaan barang kosong, aplikasi akan menampilkan notifikasi bahwa persediaan telah habis.	
3E. Jika tidak ada kurir yang tersedia untuk vendor dan lokasi tujuan, aplikasi akan menampilkan pesan bahwa pengiriman tidak dapat ditemukan.	

Table 3.3 Use Case Description Melakukan Konfirmasi Pengiriman

Use Case Name	Melakukan Konfirmasi Pengiriman	**ID**	UC3	**Importance Level**	High
Primary Actor	User Internal			**Use Case Type**	Detail, Essential
Stakeholder and Interest :					
User Internal	Dapat melakukan konfirmasi pengiriman pesanan				
Brief Description :					
Didalam use case ini diuraikan bagaimana user internal dapat melakukan konfirmasi pengiriman pesanan					
Trigger :					
Saat User membuka detail todo pengiriman pesanan					
Preconditions :					
1. User sudah login					
2. Memiliki akses terhadap proses eCatalog					
Normal Flow :					
Actor Actions	System Responses				
1. User Internal membuka	1.1. Sistem menampilkan halaman daftar				

menu todo	pekerjaan
2. User Internal melakukan pencarian To do Penermaan berdasarkan nomor Pemesanan dan filter status	2.1. Sistem menampilkan daftar pencarian pekerjaan berdasarkan kata kunci atau filter yang diteriapkan
3. User Internal membuka To do Penerimaan yang dipilih	3.1. Sistem menampilkan halaman Detail Pekerjaan
4. User Internal memproses penermaan barang / jasa	4.1. Sistem menampilkan dialog action proses
	4.2 Sistem menampilkan loading progress
	4.3. Sistem menampilkan pesan berhasil memproses data
Alternative Flow :	
Actor Actions	System Responses
Exceptional Flow :	
1E. Jika terjadi kegagalan pada saat memuat formulr Penerimaan Pesanan, maka aplikasi akan menampilkan pesan error dan alasan kegagalannya	
2E. Jika terjadi kegagalan pada saat menyimpan formulr Penerimaan Pesanan, maka aplikasi akan menampilkan pesan error dan alasan kegagalannya	

Table 3.4 Use Case Description Melakukan Penilaian Pesanan

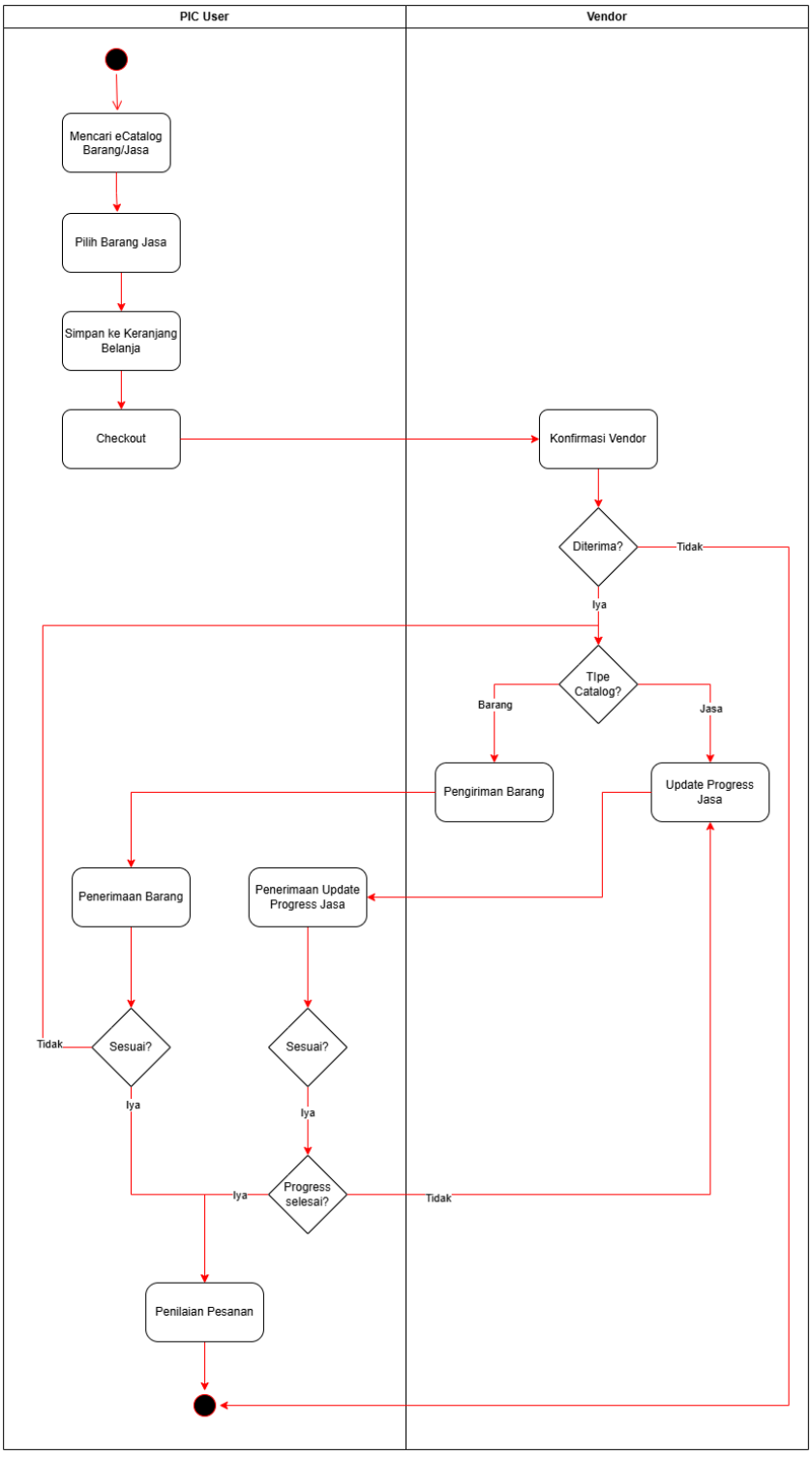
Use Case Name	Melakukan Penilaian Pesanan	**ID**	UC4	**Importance Level**	High
Primary Actor	User Internal			**Use Case Type**	Detail, Essential
Stakeholder and Interest :					
User Internal	Dapat melakukan evaluasi atau penilaian terhadap pesanan.				
Brief Description :					
Didalam use case ini diuraikan bagaimana user internal dapat melakukan evaluasi terhadap pengiriman pesanan.					

Trigger :	
Saat User membuka detail todo penilaian pesanan	
Preconditions :	
1. User sudah login	
2. Memiliki akses terhadap proses eCatalog	
Normal Flow :	
Actor Actions	System Responses
1. User Internal membuka menu todo	1.1. Sistem menampilkan halaman daftar pekerjaan
2. User Internal melakukan pencarian To do Penilaian berdasarkan nomor Pesanan dan filter status	2.1. Sistem menampilkan daftar pencarian pekerjaan berdasarkan kata kunci atau filter yang diteriapkan
3. User Internal membuka To do Penilaian yang dipilih	3.1. Sistem menampilkan halaman Detail Pekerjaan
4. User Internal memproses penilaian pesanan	4.1. Sistem menampilkan dialog action proses
	4.2 Sistem menampilkan loading progress
	4.3. Sistem menampilkan pesan berhasil memproses data
Alternative Flow :	
Actor Actions	System Responses
Exceptional Flow :	
1E. Jika terjadi kegagalan pada saat memuat formulr penilaian pesanan, maka aplikasi akan menampilkan pesan error dan alasan kegagalannya	
2E. Jika terjadi kegagalan pada saat menyimpan formulr penilaian pesanan, maka aplikasi akan menampilkan pesan error dan alasan kegagalannya	

3.2.1.2. Activity Diagram

Activity diagram digunakan untuk merepresentasikan perilaku dalam proses bisnis yang tidak bergantung pada objek

tertentu. Diagram ini lebih kompleks daripada *data flow diagrams* karena mampu menangani aktivitas paralel, konkurensi, dan pengambilan keputusan kompleks. Secara esensial, *activity diagram* adalah alat yang sangat fleksibel dalam pemodelan, dapat digunakan untuk menggambarkan berbagai hal, dari alur kerja bisnis tingkat tinggi yang melibatkan banyak *use case*, hingga detail spesifik dari sebuah *use case* individu, atau bahkan dari sebuah metode individu. Diagram aktivitas ini dibuat berdasarkan *use case diagram* dan deskripsi *use case* yang telah diuraikan sebelumnya.



Gambar 3.3 Activity Diagram fitur Ecatalog

Berikut adalah penjelasan dari *activity diagram* tersebut

1. User Internal melakukan pencarian katalog berdasarkan nama barang / jasa atau menggunakan filter lainnya.
2. User internal membuka katalog barang / jasa yang dipilih
3. User internal menambahkan barang / jasa ke dalam keranjang
4. User internal membuka menu keranjang dan melakukan perubahan pada kuantitas barang apabila diperlukan.
5. User internal melakukan proses *checkout* barang yang ada di keranjang. Pada proses ini memilih sumber *budget* yang diambil, kantor tujuan pengiriman, ppn pada barang atau jasa yang akan di proses, serta jasa pengiriman yang digunakan untuk katalog yang bertipe barang.
6. Vendor melakukan konfirmasi pesanan
    - Jika vendor menerima pesanan user, maka vendor akan memproses pesanan yang dibuat oleh user.
        - Jika item katalog pada pesanan yang dibuat berupa katalog barang, maka vendor akan melakukan pengiriman barang
            1. User akan melakukan penerimaan barang dari barang yang dikirim oleh vendor
                - Jika barang yang dikirim tidak sesuai, maka user internal akan melakukan retur barang
                    1. Vendor akan melakukan penerimaan retur barang.
                - Jika barang yang dikirim telah sesuai, maka dilanjutkan ke proses berikutnya.
            2. User melakukan penilaian pesanan
        - Jika item katalog pada pesanan yang dibuat berupa katalog jasa, maka vendor akan melakukan update progress jasa
            1. User akan melakukan konfirmasi update progress jasa dari yang dilakukan oleh vendor

- Jika progress pengerjaan tidak sesuai, maka progress akan dikirimkan kembali ke vendor
- Jika progress pengerjaan sesuai yang dikirim telah sesuai, maka akan ada pengecekan terkait penyelesaian progress jasa.
    - Jika semua sudah dikerjakan oleh vendor, maka dilanjutkan ke proses berikutnya.
    - Jika masih ada progress yang belum diselesaikan oleh vendor, maka vedor dapat mengirimkan update progress jasa selanjutnya.

2. User melakukan penilaian pesanan

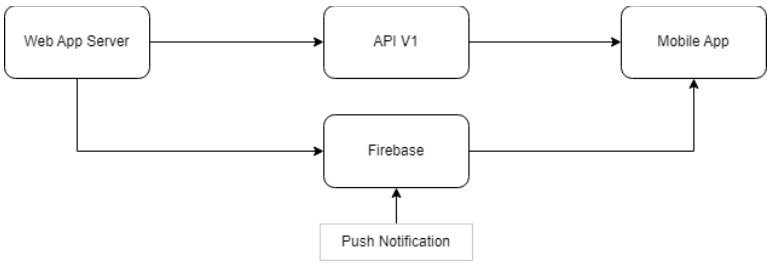
- Jika vendor menerima pesanan user, maka dilanjutkan ke proses berikutnya.

7. Seluruh proses selesai dan aktivitas selesai dilakukan.

3.2.1.3. Application Architecture

a. Data Communication Flow

Data Communication Flow adalah proses transfer data antara beberapa komponen dalam suatu sistem. Pada aplikasi mobile, aliran komunikasi data terdiri dari beberapa komponen yaitu *web app server*, *api v1*, *firebase* dan *mobile app*. Alur komunikasi datanya seperti berikut

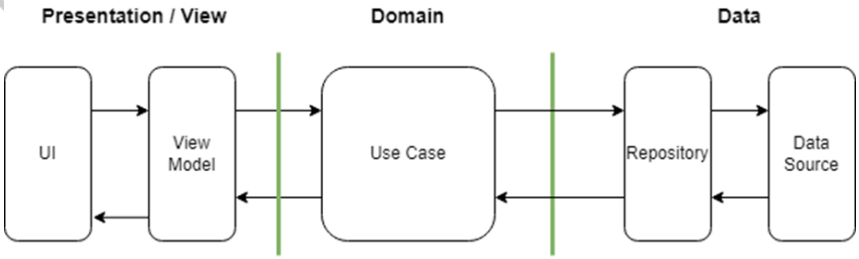


Gambar 3.4 Data Communication Flow Diagram

Web App Server menyimpan data dan diakses melalui API V1. *Mobile App* mengirimkan permintaan ke API V1 untuk mengambil atau memperbarui data pada *Web App Server*. *Firebase* digunakan untuk mengirim notifikasi dari *Web App Server* ke *Mobile App*

b. *Architecture & Design Pattern (MVVM)*

Aplikasi menerapkan struktur folder yang mendukung *clean architecture* dengan membagi layer menjadi 3 yaitu *data, domain* dan *presentation/view*. Alur aliran datanya seperti berikut



Gambar 3.5 Arsitektur aplikasi Iproc 2GO

- *Data Layer*

  Merupakan lapisan yang berfungsi untuk menampung semua sumber data baik *local* maupun *network* dan *repository* sebagai satu pintu untuk mengakses data dari berbagai sumber data tersebut.

- *Domain Layer*

  Merupakan lapisan yang berfungsi sebagai lapisan terkait dengan proses bisnis. Lapisan ini tidak boleh bergantung dengan code diluar layer ini atau dari *framework* apapun.

- *Presentation/View Layer*

  Merupakan lapisan yang berfungsi untuk mengatur tampilan sesuai dengan data dari *domain* > *model*. Lapisan ini berisikan kode yang berhubungan dengan UI seperti *View Model*, *Activity/Fragment*, *Adapter* dan lain - lain.

Alur dimulai dari *view layer*, dimana *view layer* bagian dari *layer* yang berinteraksi langsung dengan pengguna. *View layer* akan mengakses *core* bisnis atau *business logic* aplikasi pada *domain layer*. Kemudian dari *domain layer*, mengakses data ke *data layer* untuk mengakses data yang diperlukan.

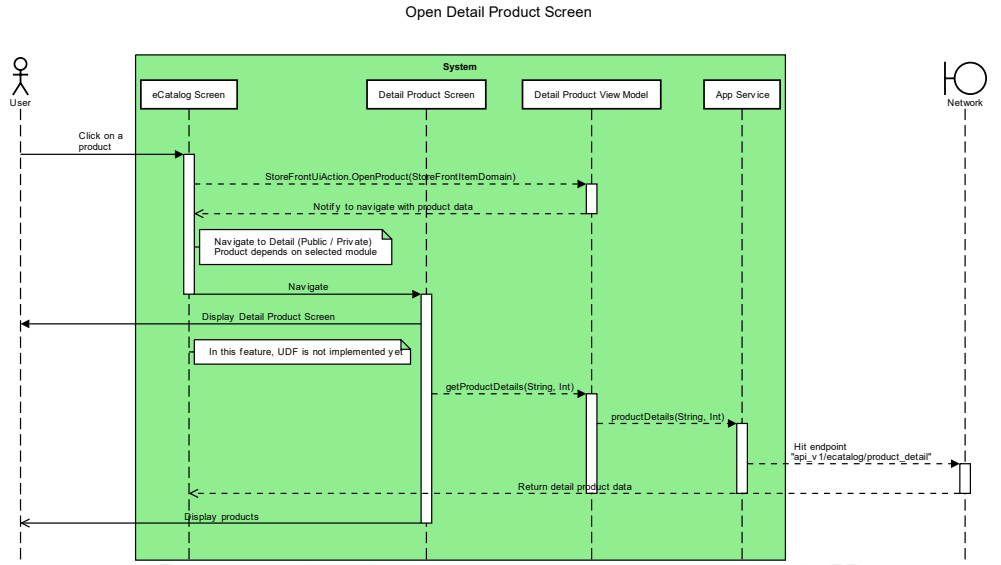
3.2.1.4. *Sequence Diagram*

Sequence diagram merupakan salah satu dari dua jenis diagram interaksi. Diagram ini merupakan model dinamis yang menggambarkan urutan pesan yang dikirimkan antara objek-objek selama interaksi yang sudah ditentukan. Fokus utama dari *sequence diagram* adalah mengilustrasikan urutan aktivitas berdasarkan waktu di antara sekelompok objek. Hal ini sangat bermanfaat untuk memahami spesifikasi yang berkaitan dengan waktu nyata dan use case yang kompleks. Contoh *sequence diagram* yang disajikan dalam dokumen panduan pengembangan untuk modul *eCatalog* adalah sebagai berikut:.

a. Alur membuka daftar katalog (termasuk memilih tipe katalog, pencarian dan penyortiran)

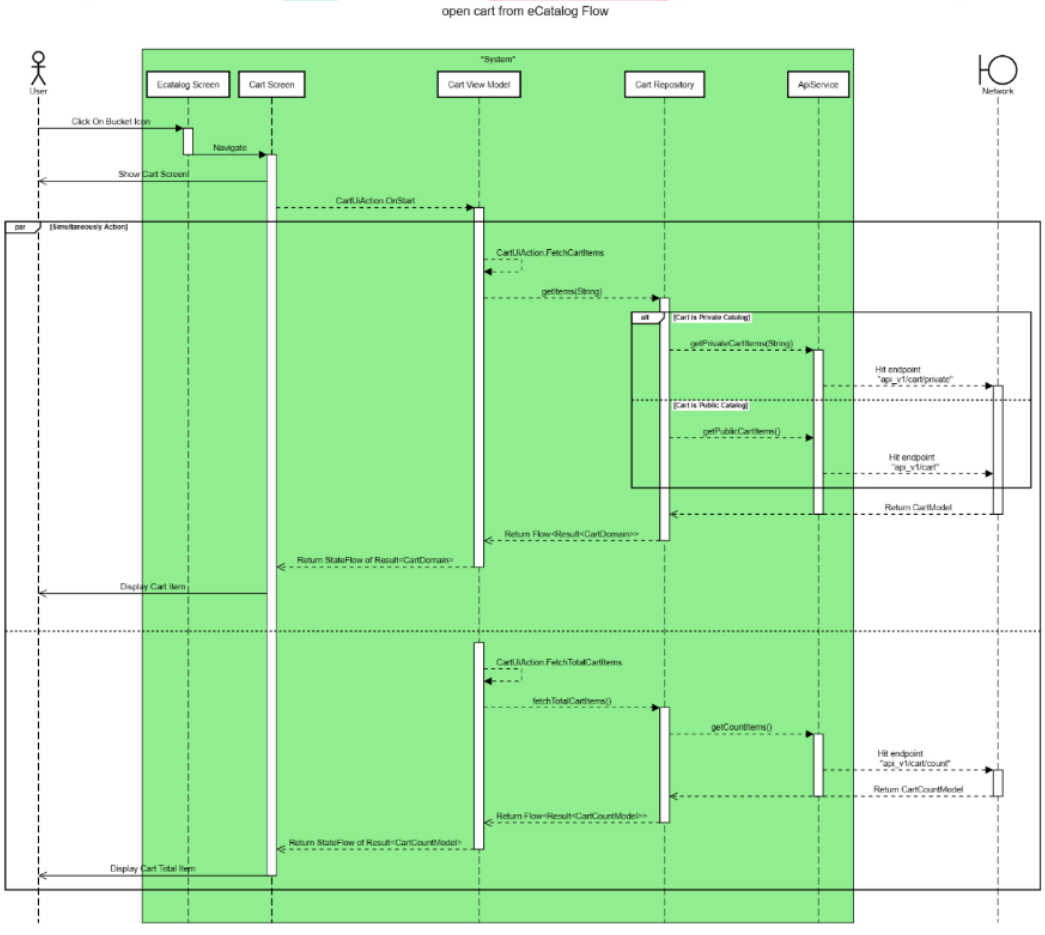
Gambar 3.6 Sequence Diagram alur membuka daftar katalog

b. Alur membuka detil produk



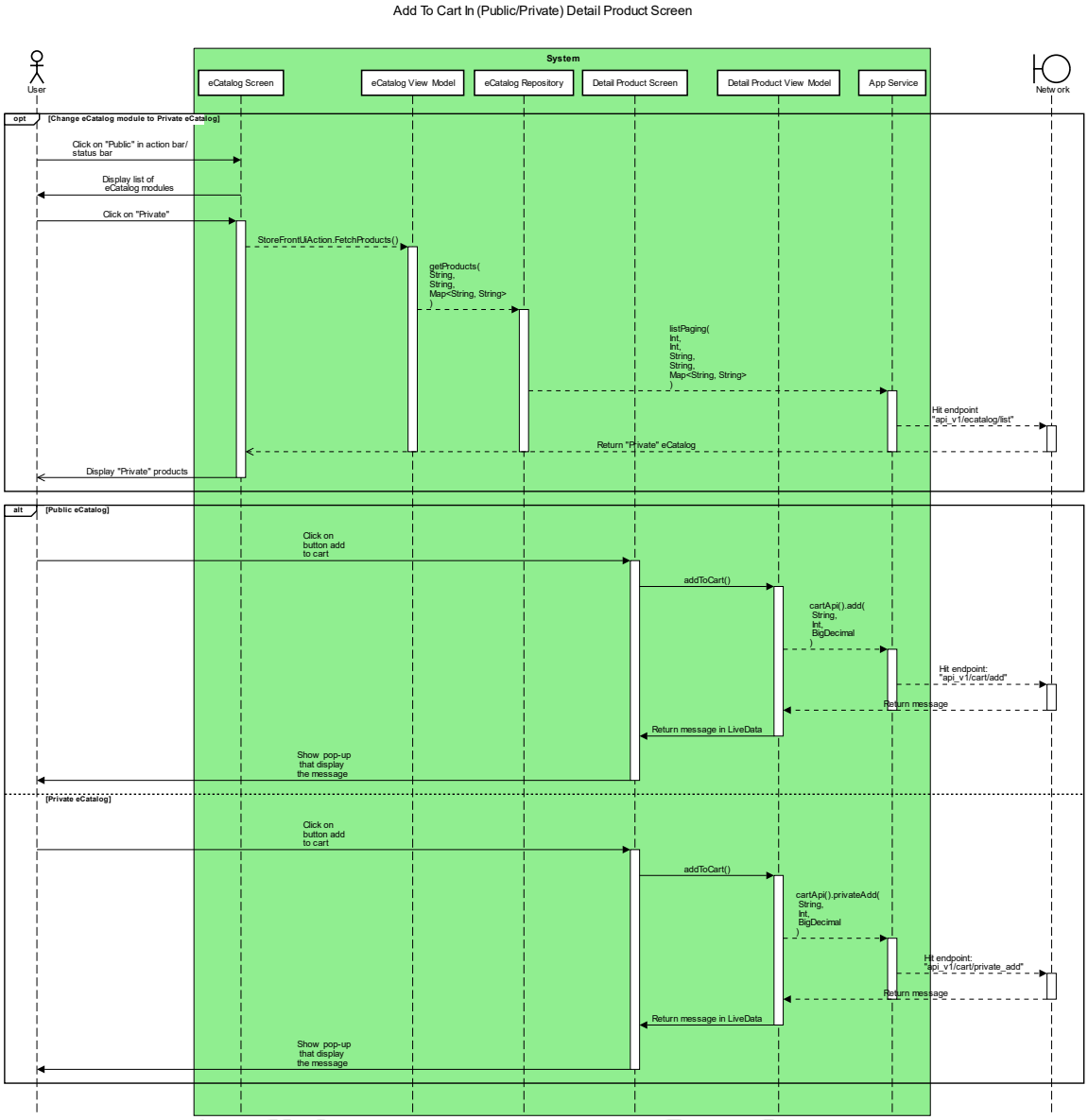
Gambar 3.7 Sequence Diagram alur membuka detil produk

c. Alur membuka halaman keranjang belanja



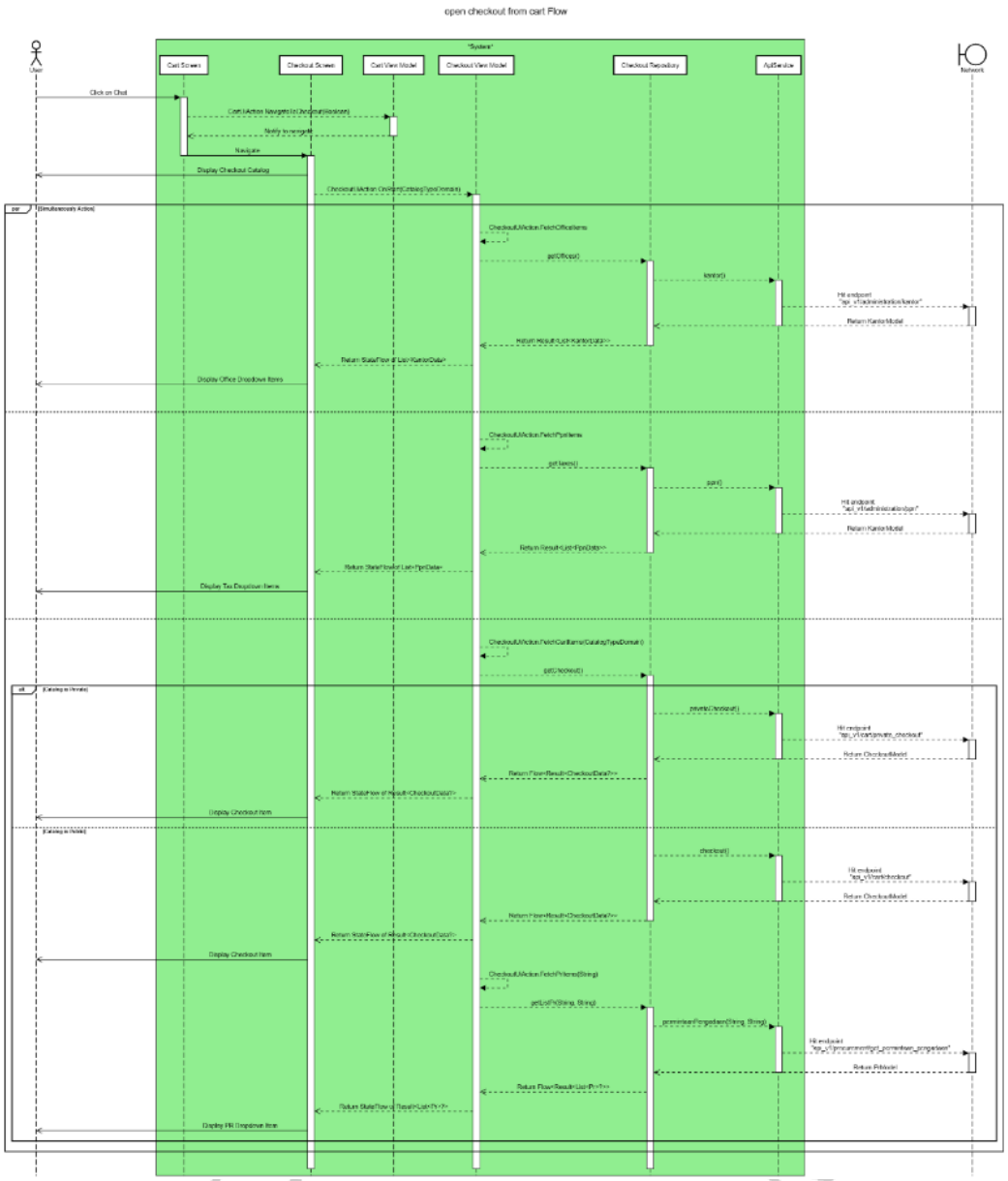
Gambar 3.8 Sequence Diagram alur membuka halaman keranjang belanja

d. Alur menambahkan produk ke dalam keranjang



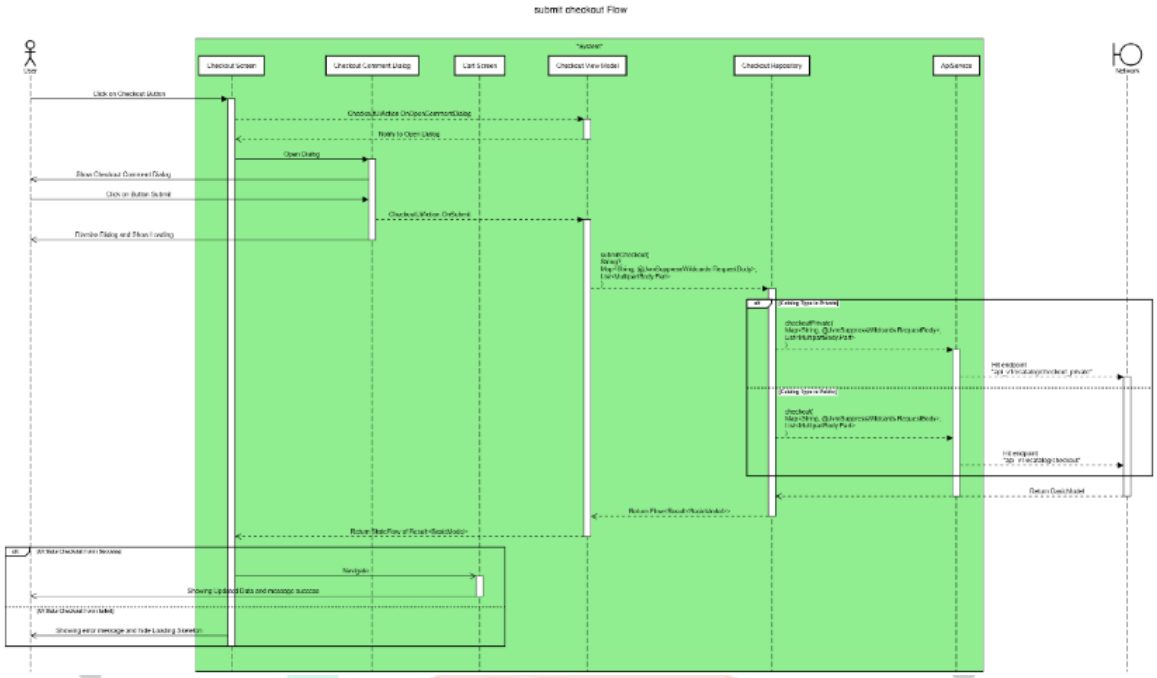
Gambar 3.9 Sequence Diagram alur menambahkan produk ke dalam keranjang

e. Alur membuka halaman *checkout*



Gambar 3.10 Sequence Diagram alur membuka halaman checkout

f. Alur memilih kurir



Gambar 3.11 Sequence Diagram alur memilih kurir

Berdasarkan hasil analisa komponen – komponen yang terdapat pada beberapa dokumen, maka akan menghasilkan *requirement traceability matrix* (RTM).

3.2.1.4. *Requirement Traceability Matrix*

Requirement traceability matrix (RTM) adalah tabel yang mencantumkan daftar persyaratan, berbagai atribut yang terkait dengan setiap persyaratan, dan status dari persyaratan tersebut untuk memastikan semua persyaratan terpenuhi. RTM memungkinkan penelusuran setiap persyaratan yang ada, mulai dari persyaratan bisnis hingga pengujian. Format RTM dapat disesuaikan sesuai dengan kebutuhan masing-masing proyek. Tujuan utama dari RTM adalah agar semua anggota proyek mengetahui persyaratan apa saja yang harus dipenuhi. Adapun matriks penelusuran persayaratan yang dihasilkan dari hasil analisa dokumen yang ada adalah sebeagai berikut.

Table 3.5 Requirement Traceability Matrix Order Catalog

REQUIREMENT TRACEABILITY MATRIX				
Project Name		Iproc 2GO		
Created By		Muhammad Syaifullah		
Feature		Order eCatalog		
Requirement ID	Requirement Description	Test Case ID	Test Case Description	Expected Results
001	Order eCatalog public	001/TC/01	Login sebagai PIC User di aplikasi iProc Mobile Intranet	User dapat menambahkan produk ke keranjang belanja
			Pilih menu eCatalog	
			Search item yang sedang dicari, gunakan Filter jika ingin mendapatkan hasil search yang akurat.	
			Jika sudah pilih produk, di halaman Detail Produk. User dapat menambahkan produk ke keranjang	
			Setelah itu klik icon Keranjang, pilih Public. Maka akan diarahkan ke halaman Cart. Lalu klik Checkout dan akan diarahkan ke halaman Checkout.	
		001/TC/02	Di halaman Checkout user memilih PR Catalog (sistem secara otomatis hanya menampilkan PR Catalog berdasarkan kategori barang jasa dan total nilai belanja yang ada di halaman checkout).	User dapat memilih PR yang sesuai dan memilih kantor tujuan.

			Lalu user memilih kantor tujuan (alamat kantor diambil dari master kantor).	
		001/TC/03	Lalu pilih Pengiriman / Kurir beserta biayanya.	User dapat memilih jasa pengiriman.
		001/TC/04	Setelah itu klik tombol Checkout, pilih Aksi Lanjutkan, isikan Komentar, dan tambahkan Lampiran. Lalu klik Simpan	User dapat melakukan checkout.
002	Pengiriman Barang / Jasa	001/TC/05	Login sebagai vendor di iProc Mobile Extranet	Vendor berhasil melakukan Pengiriman Barang/Jasa
			Pilih menu To Do List, lalu pilih data eCatalog berstatus 'Pengiriman'	
			Masukan Nomor Resi Pengiriman, akan tervalidasi otomatis jika nomor resi valid.	
			Pada tombol Action Required, pilih aksi Lanjut. Lalu isi Komentar, tambahkan Lampiran, dan klik Simpan	
003	Penerimaan Barang / Jasa	001/TC/06	Login sebagai PIC User di aplikasi iProc Mobile Intranet	PIC User berhasil melakukan Penerimaan Barang
			Pilih Menu To Do List dan pilih data Penerimaan Barang / Jasa	
			Konfirmasi barang yang diterima. Pada tombol Action Required, Pilih Aksi lalu Isikan Komentar dan klik Simpan	

004	Penilaian Barang	001/TC/07	Login sebagai PIC User di aplikasi iProc Mobile Intranet	PIC User berhasil melakukan Penilaian Barang / Jasa
			Pilih Menu To Do List dan pilih data Penilaian Barang / Jasa	
			Berikan Penilaian / Rating pada Bintang, maka akan muncul pemberian rating	
			Pada tombol Action Required, Pilih Aksi lalu Isikan Komentar dan klik Simpan	

3.2.2. Perencanaan Pengujian (Test Planning)

Perencanaan Pengujian adalah fase paling efisien dari siklus hidup pengujian perangkat lunak di mana semua rencana pengujian ditentukan. Dalam fase manajer pengujian ini, tim menghitung perkiraan upaya dan biaya untuk pekerjaan pengujian. Fase ini dimulai setelah fase pengumpulan kebutuhan selesai. Aktivitas yang berlangsung pada tahap analisa kebutuhan antara lain:

a. Mengidentifikasi tujuan dan ruang lingkup pengujian

Mengidentifikasi tujuan dan cakupan pengujian merupakan langkah krusial dalam memastikan bahwa fitur tersebut beroperasi dengan baik dan memenuhi kebutuhan pengguna. Tujuan utama dari pengujian ini adalah untuk memverifikasi bahwa fungsi dasar dalam aplikasi berjalan dengan baik. Pada laporan kali ini, ruang likup pengujian juga dibatasi. Pengujian pada aplikasi hanya terbatas pada **sub fitur ecatalog list**. Pengujian juga terbatas pada pengujian unit (*unit test)*. *Unit testing* adalah suatu proses pengujian otomatis yang bertujuan untuk memverifikasi fungsi terkecil dari sebuah kode. Proses ini dilakukan dengan cepat dan terisolasi, memfokuskan pada unit-unit kecil kode secara terpisah. Setiap *unit test* memiliki struktur

yang mengarahkannya, meskipun mungkin ada variasi dalam penerapannya tergantung pada bahasa pemrograman yang digunakan.

b. Mengembangkan strategi pengujian

Dalam dunia pengujian perangkat lunak, terdapat dua metode yang umum digunakan: *white box testing* dan *black box testing*. Dalam *black box testing*, perangkat lunak diperlakukan seperti sebuah "Kotak Hitam", yang berarti pengujian dilakukan tanpa pengetahuan tentang struktur internalnya. Pengujian ini berfokus pada fitur fundamental sistem dan tidak memerlukan akses ke kode sumber, meskipun pengujian harus memahami arsitektur umum dari sistem yang diuji.

White box testing merupakan teknik pengujian yang menggunakan struktur kontrol dari desain prosedural untuk merancang kasus uji. Teknik ini bertujuan untuk menemukan kesalahan implementasi dengan menganalisis struktur dan logika internal dari perangkat lunak. *White box testing* dapat diterapkan pada berbagai tingkat pengujian seperti unit, integrasi, dan sistem. Dalam proses ini, penguji memeriksa kode sumber untuk mengidentifikasi unit kode yang mungkin tidak berperilaku sesuai dengan yang diharapkan, sehingga memungkinkan untuk penemuan dan perbaikan kesalahan lebih efisien.

White box testing memiliki beberapa metode atau teknik yang digunakan untuk menguji perangkat lunak, di antaranya adalah sebagai berikut:

- *Control Flow Testing*

  Pengujian ini adalah metode pengujian struktural yang menggunakan alur kontrol program sebagai modelnya, dengan lebih mengutamakan jalur yang lebih banyak namun lebih sederhana daripada jalur yang lebih sedikit namun lebih kompleks.

- *Branch Testing*

  Pengujian ini memastikan setiap pernyataan dalam kode dieksekusi setidaknya sekali. Tujuannya adalah

untuk mengungkap kesalahan dengan menjalankan setiap baris kode, meskipun ini adalah kriteria yang lemah karena kurang sensitif terhadap beberapa struktur kontrol.

- *Basis Path Testing*

  Pengujian ini memungkinkan perancang kasus uji untuk menghasilkan metrik kompleksitas logis dari desain prosedural, dan kemudian menggunakan metrik tersebut sebagai cara untuk merangkum serangkaian jalur eksekusi dasar.

- *Data Flow Testing*

  Dalam konteks pengujian, grafik aliran kontrol menyertakan detail mengenai definisi dan penggunaan variabel dalam program.

- *Loop Testing*

  Pengujian ini Ini secara eksklusif fokus pada validitas konstruksi *loop*.

Pada pengujian ini, lebih sesuai menggunakan teknik control flow testing dan branch testing daripada teknik pengujian lainnya. Kedua teknik ini dinilai lebih cocok karena kesederhanaan, efektivitas, dan keseimbangan cakupan yang ditawarkannya. *Control flow testing* membantu memastikan bahwa semua jalur dalam unit kode telah diuji, memungkinkan deteksi dini terhadap potensi kesalahan dalam logika program. Di sisi lain, *branch testing* memastikan bahwa setiap kondisi cabang pada pernyataan kontrol telah diuji secara terpisah, sehingga memastikan bahwa semua kemungkinan jalur eksekusi telah diuji.

Kedua teknik tersebut menawarkan kompromi yang baik antara cakupan dan kompleksitas, membuatnya ideal untuk digunakan pada tahap pengembangan dan pengujian aplikasi Iproc 2GO. Selain itu, banyak alat pengujian otomatis mendukung kedua teknik ini, memudahkan implementasinya dalam proses pengujian sehari-hari. Oleh karena itu, *control flow testing* dan *branch testing* merupakan pilihan yang optimal pada proses implementasi *unit testing*.

Dalam melakukan *whitebox testing* pada aplikasi, terdapat beberapa alat yang dapat digunakan untuk mempermudah proses pengerjaan dan validasi terkait kualitas implementasi *test case*. Adapun alat dan teknologi yang digunakan pada proses implementasi *whitebox testing* adalah sebagai berikut.

- *Testing Framework*
  - Junit

Gambar 3.12 Junit (Sumber junit.org/junit4)

JUnit memungkinkan pengembang untuk menulis dan menjalankan tes untuk kode mereka dengan mudah. Fitur-fiturnya termasuk anotasi untuk menandai metode tes, assert statement untuk memverifikasi hasil, dan integrasi dengan berbagai alat build dan IDE

  - Mockito

Gambar 3.13 Mockito (Sumber site.mockito.org)

Mockito adalah framework mocking yang populer di komunitas Java yang juga dapat digunakan dalam proyek Kotlin. Mockito memungkinkan pengembang untuk membuat mock, stub, dan verify interaksi antara objek dalam tes unit dan integrasi. Meskipun tidak dirancang khusus untuk Kotlin

- Code Coverage Tools

  Code coverage tools adalah perangkat yang digunakan untuk mengukur sejauh mana kode sumber telah diuji oleh *test case* yang ada. Alat ini memberikan metrik yang menunjukkan persentase kode yang dijalankan selama pengujian. Adapun alat yang digunakan untuk mengukur *coverage code* adalah sebagai berikut

  - Kover

    Kover adalah alat code coverage khusus untuk Kotlin, yang dirancang untuk mengukur cakupan kode dalam proyek Kotlin. Kover menyediakan laporan visual dan statistik cakupan yang membantu pengembang memahami bagian kode mana yang sudah diuji

c. Mengidentifikasi lingkungan pengujian dan sumber daya yang dibutuhkan

   Mengidentifikasi lingkungan pengujian dan sumber daya yang dibutuhkan untuk aplikasi mobile Iproc 2Go adalah langkah penting untuk memastikan bahwa pengujian berjalan lancar dan komprehensif. Berikut adalah rincian mengenai lingkungan pengujian dan sumber daya yang dibutuhkan:

   - Perangkat Keras (*Hardware*)

     Perangkat keras ini meliputi komputer sebagai alat pengembang pengujian dan untuk menjalankan pengujian diatas JVM

   - Perangkat Lunak (*Software*)

     Android Studio untuk pengembangan aplikasi Android

3.2.3. Pengembangan Kasus Uji (*Test Case Development*)

Tahap Pengembangan Kasus Uji dalam Siklus Hidup Pengujian Perangkat Lunak (STLC) dimulai setelah tahap perencanaan pengujian selesai. Kasus uji (*test case*) adalah skenario pengujian yang dirancang untuk memverifikasi bahwa

setiap fungsi dalam aplikasi berjalan sesuai dengan yang diharapkan. Pada tahap ini, tim penguji mulai merinci dan mendokumentasikan berbagai kasus uji berdasarkan persyaratan dan skenario yang telah ditetapkan. Setiap kasus uji ini mencakup *unit test* yang dirancang untuk mengevaluasi berbagai aspek fungsionalitas dan kinerja aplikasi. *Unit test* akan memeriksa setiap komponen individual aplikasi seperti *presentation, domain*, dan *data layer* untuk memastikan bahwa setiap bagian berfungsi sesuai dengan yang diharapkan.

Penguji juga menentukan data pengujian yang akan digunakan dalam proses tersebut. Data pengujian harus mencakup berbagai skenario penggunaan yang mungkin terjadi, termasuk kasus normal, ekstrem, dan skenario kesalahan. Penggunaan data pengujian yang tepat memungkinkan deteksi *bug* dan anomali sejak dini. Setelah proses pengembangan kasus uji selesai, penguji harus mendokumentasi dengan baik kasus uji yang telah dibuat dalam bentuk *requirement traceability matrix* (RTM). RTM ini berfungsi untuk melacak setiap kebutuhan yang diuji, memastikan bahwa semua kebutuhan telah diuji dengan benar dan hasilnya sesuai dengan yang diharapkan, sehingga memberikan jaminan bahwa aplikasi telah memenuhi standar kualitas yang ditetapkan. Adapun hasil *test case* yang berhasil dikembangkan adalah sebagai berikut

a. Data Layer

Data layer bertanggung jawab untuk mengelola akses data dari sumber eksternal seperti *database*, layanan web, atau API eksternal. Ini mencakup implementasi *repository* dan *data source*. Pengujian pada *layer* ini bertujuan untuk memastikan operasi data dan interaksi dengan sumber data eksternal berfungsi dengan benar. Adapun *requirement traceability matrix* pada *data layer* terkait fitur *catalog* sebagai berikut.

Table 3.6 Requirements Tracebility Matrix Catalog Repository layer

REQUIREMENT TRACEABILITY MATRIX	
Project Name	Iproc 2GO

Created By	Muhammad Syaifullah		
Feature	Ecatalog		
Layer	Data		
Sub Layer	Repository		
Requirement ID	Requirement Description	Test Case ID	Test Case Description
001/UNT/001	Menampilkan daftar modul catalog	TC/RP/001	When getModules() Should return Loading Then Success
		TC/RP/002	Given modules in storage When getModules() Should return Loading Then Error
		TC/RP/003	Given empty modules in storage When getModules() Should return Loading Then Error
001/UNT/002	Menampilkan daftar filter	TC/RP/004	When getFilters() Should return Loading Then Success
		TC/RP/005	Given task filter list response When getFilters() Should return Loading Then Success with correct data
		TC/RP/006	Given empty task filter list When getFilters() Should return Loading Then Success with correct data
001/UNT/003	Menampilkan daftar detail filter	TC/RP/007	Given list detail filter When getSeeAllFilters() Should return Loading Then Success
		TC/RP/008	Given failure response When getSeeAllFilters() Should return Loading Then Error
001/UNT/004	Menampilkan total katalog produk berdasarkan filter yang diterapkan	TC/RP/009	Given selected module with empty input filter parameter When countFilteredProducts() Should return Loading Then Success
		TC/RP/010	Given selected module with not empty input filter parameter When countFilteredProducts() Should return Loading Then Success
		TC/RP/011	Given failure response When countFilteredProducts() Should return Loading Then Error

001/UNT/008	Menampilkan daftar katalog menggunakan metode pagination berdasarkan filter dan tipe yang diterapkan	TC/RP/012	Given selected module with empty input filter parameter When getProducts() Should return Loading Then Success
		TC/RP/013	Given selected module with not empty input filter parameter When getProducts() Should return Loading Then Success
		TC/RP/014	When getProducts() Should return Loading Then Error

Table 3.7 Requirements Tracebility Matrix Data Source layer

REQUIREMENT TRACEABILITY MATRIX			
Project Name	Iproc 2GO		
Created By	Muhammad Syaifullah		
Feature	Ecatalog		
Layer	Data		
Sub Layer	Data Source		
Requirement ID	Requirement Description	Test Case ID	Test Case Description
001/UNT/001	Menampilkan daftar modul catalog	TC/DS/001	Given service_getCatalogTypes return Success Response When fetch module list Should return result success
		TC/DS/002	Given service_getCatalogTypes return HttpException When fetch module list Should return result failure with HttpException
		TC/DS/003	Given service_getCatalogTypes return UnknownHostException When fetch module list Should return result failure with UnknownHostException
		TC/DS/004	Given service_getCatalogTypes return ConnectException When fetch module list Should return result failure with ConnectException

		TC/DS/005	Given service_getCatalogTypes return JsonEncodingException When fetch module list Should return result failure with JsonEncodingException
		TC/DS/006	Given service_getCatalogTypes return SocketTimeoutException When fetch module list Should return result failure with SocketTimeoutException
		TC/DS/007	Given service_getCatalogTypes return SSLHandshakeException When fetch module list Should return result failure with SSLHandshakeException
		TC/DS/008	Given service_getCatalogTypes return Exception When fetch module list Should return result failure with Exception
001/UNT/002	Menampilkan daftar filter	TC/DS/009	Given service_getFilters return Success Response When fetch catalog filter list Should return result success
		TC/DS/010	Given service_getFilters return HttpException When fetch catalog filter list Should return result failure with HttpException
		TC/DS/011	Given service_getFilters return UnknownHostException When fetch catalog filter list Should return result failure with UnknownHostException
		TC/DS/012	Given service_getFilters return ConnectException When fetch catalog filter list Should return result failure with ConnectException

		TC/DS/013	Given service_getFilters return JsonEncodingException When fetch catalog filter list Should return result failure with JsonEncodingException
		TC/DS/014	Given service_getFilters return SocketTimeoutException When fetch catalog filter list Should return result failure with SocketTimeoutException
		TC/DS/015	Given service_getFilters return SSLHandshakeException When fetch catalog filter list Should return result failure with SSLHandshakeException
		TC/DS/016	Given service_getFilters return Exception When fetch catalog filter list Should return result failure with Exception
001/UNT/003	Menampilkan daftar detail filter	TC/DS/017	Given service_getFilterDetail return Success Response When fetch catalog detail filter list Should return result success
		TC/DS/018	Given service_getFilterDetail return HttpException When fetch catalog detail filter list Should return result failure with HttpException
		TC/DS/019	Given service_getFilterDetail return UnknownHostException When fetch catalog detail filter list Should return result failure with UnknownHostException
		TC/DS/020	Given service_getFilterDetail return ConnectException When fetch catalog detail filter list Should return result failure with ConnectException

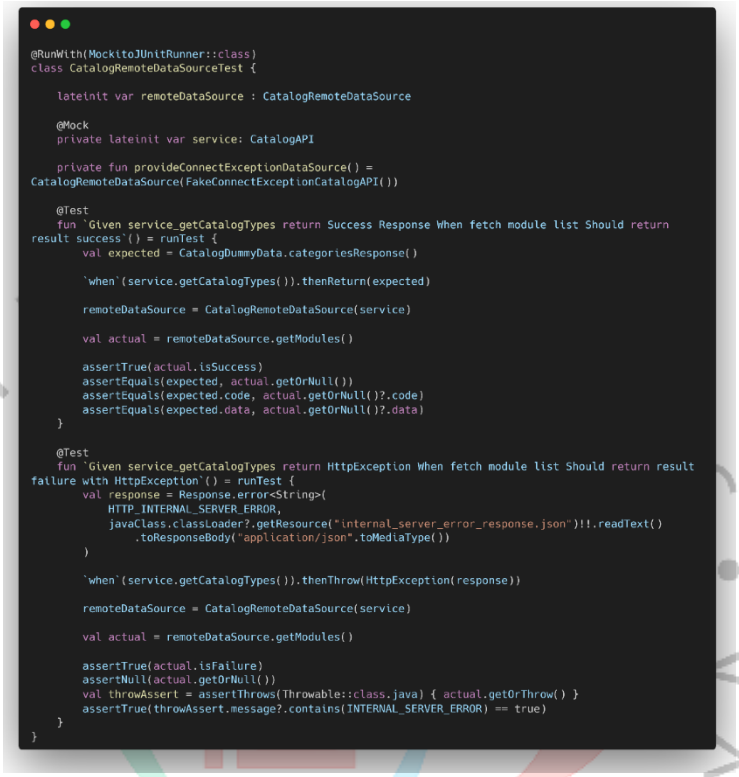
		TC/DS/021	Given service_getFilterDetail return JsonEncodingException When fetch catalog detail filter list Should return result failure with JsonEncodingException
		TC/DS/022	Given service_getFilterDetail return SocketTimeoutException When fetch catalog detail filter list Should return result failure with SocketTimeoutException
		TC/DS/023	Given service_getFilterDetail return SSLHandshakeException When fetch catalog detail filter list Should return result failure with SSLHandshakeException
		TC/DS/024	Given service_getFilterDetail return Exception When fetch catalog detail filter list Should return result failure with Exception
001/UNT/004	Menampilkan total katalog produk berdasarkan filter yang diterapkan	TC/DS/025	Given service_getTotalFilteredProducts return Success Response When get total filtered product Should return result success
		TC/DS/026	Given service_getTotalFilteredProducts return HttpException When get total filtered product Should return result failure with HttpException
		TC/DS/027	Given service_getTotalFilteredProducts return UnknownHostException When get total filtered product Should return result failure with UnknownHostException

		TC/DS/028	Given service_getTotalFilteredProducts return ConnectException When get total filtered product Should return result failure with ConnectException
		TC/DS/029	Given service_getTotalFilteredProducts return JsonEncodingException When get total filtered product Should return result failure with JsonEncodingException
		TC/DS/030	Given service_getTotalFilteredProducts return SocketTimeoutException When get total filtered product Should return result failure with SocketTimeoutException
		TC/DS/031	Given service_getTotalFilteredProducts return SSLHandshakeException When get total filtered product Should return result failure with SSLHandshakeException
		TC/DS/032	Given service_getTotalFilteredProducts return Exception When get total filtered product Should return result failure with Exception
001/UNT/008	Menampilkan daftar katalog menggunakan metode pagination berdasarkan filter dan tipe yang diterapkan	TC/DS/033	Given service_listPaging return Success Response When fetch catalog list paging Should return result success
		TC/DS/034	Given service_listPaging return HttpException When fetch catalog list paging Should return result failure with HttpException

		TC/DS/035	Given service_listPaging return UnknownHostException When fetch catalog list paging Should return result failure with UnknownHostException
		TC/DS/036	Given service_listPaging return ConnectException When fetch catalog list paging Should return result failure with ConnectException
		TC/DS/037	Given service_listPaging return JsonEncodingException When fetch catalog list paging Should return result failure with JsonEncodingException
		TC/DS/038	Given service_listPaging return SocketTimeoutException When fetch catalog list paging Should return result failure with SocketTimeoutException
		TC/DS/039	Given service_listPaging return SSLHandshakeException When fetch catalog list paging Should return result failure with SSLHandshakeException
		TC/DS/040	Given service_listPaging return Exception When fetch catalog list paging Should return result failure with Exception
		TC/DS/041	Given response data not empty When getCatalogPagingSource_load() Should return Page
		TC/DS/042	Given first page data When getCatalogPagingSource_getRefreshKey() Should return null refresh key

Unit test akan dibuat berdsarkan *test case* yang telah dijabarkan pada *requirement traceability matrix*. *Requirement*

traceability matrix akan dijadikan sebagai indikator tercapai atau tidaknya pembuatan *unit test*. Adapun *unit test* sebagai berikut.



```
@RunWith(MockitoJUnitRunner::class)
class CatalogRemoteDataSourceTest {

    lateinit var remoteDataSource : CatalogRemoteDataSource

    @Mock
    private lateinit var service: CatalogAPI

    private fun provideConnectExceptionDataSource() =
CatalogRemoteDataSource(FakeConnectExceptionCatalogAPI())

    @Test
    fun `Given service_getCatalogTypes return Success Response When fetch module list Should return
result success`() = runTest {
        val expected = CatalogDummyData.categoriesResponse()

        `when`(service.getCatalogTypes()).thenReturn(expected)

        remoteDataSource = CatalogRemoteDataSource(service)

        val actual = remoteDataSource.getModules()

        assertTrue(actual.isSuccess)
        assertEquals(expected, actual.getOrNull())
        assertEquals(expected.code, actual.getOrNull()?.code)
        assertEquals(expected.data, actual.getOrNull()?.data)
    }

    @Test
    fun `Given service_getCatalogTypes return HttpException When fetch module list Should return result
failure with HttpException`() = runTest {
        val response = Response.error<String>(
            HTTP_INTERNAL_SERVER_ERROR,
            javaClass.classLoader?.getResource("internal_server_error_response.json")!!.readText()
                .toResponseBody("application/json".toMediaType())
        )

        `when`(service.getCatalogTypes()).thenThrow(HttpException(response))

        remoteDataSource = CatalogRemoteDataSource(service)

        val actual = remoteDataSource.getModules()

        assertTrue(actual.isFailure)
        assertNull(actual.getOrNull())
        val throwAssert = assertThrows(Throwable::class.java) { actual.getOrThrow() }
        assertTrue(throwAssert.message?.contains(INTERNAL_SERVER_ERROR) == true)
    }
}
```

Gambar 3.14 Kode unit test catalog remote data source

```
@RunWith(MockitoJUnitRunner::class)
class CatalogRepositoryTest : BaseRepositoryTest<CatalogRepository>() {

    @Mock
    private lateinit var remoteDataSource: CatalogRemoteDataSource

    @Mock
    private lateinit var moduleLocalDataSource: ModuleDataSource

    override lateinit var repository: CatalogRepository

    private fun provideRepository() = CatalogRepository(
        remoteDataSource = remoteDataSource,
        moduleLocalDataSource = moduleLocalDataSource,
        dispatcher = Dispatchers.IO
    )

    @Test
    fun `When getModules() Should return Loading Then Success`() = runTest
{       val expected = ModuleDummyData.catalogModuleModelList()

        `when`(remoteDataSource.getModules()).thenReturn(
            Result.success(ModuleDummyData.catalogModuleListResponse())
        )

        `when`(moduleLocalDataSource.getModules(ECATALOG)).thenReturn(
            flow {
                emit(ModuleDummyData.catalogModuleEntities())
            }
        )

        repository = provideRepository()

        repository.getModules().test {
            val emissionLoading = awaitItem()
            assertTrue(emissionLoading is Resource.Loading)
            assertEquals(expected, emissionLoading.data)

            verify(remoteDataSource).getModules()
            verify(moduleLocalDataSource, times(2)).getModules(ECATALOG)

            val emissionSuccess = awaitItem()
            assertTrue(emissionSuccess is Resource.Success)
            assertEquals(expected, emissionSuccess.data)
            assertEquals(expected.data, emissionSuccess.data?.data)

            cancelAndIgnoreRemainingEvents()
        }
    }
}
```

Gambar 3.15 Kode unit test catalog repository

b. *Domain Layer*

Domain layer adalah inti dari aplikasi yang berisi logika bisnis utama dan aturan *domain*. Ini mencakup entitas dan *use case* yang mendefinisikan bagaimana data harus diproses. Pengujian pada *layer* ini bertujuan untuk bahwa logika bisnis dan aturan *domain* berfungsi dengan benar. Adapun

requirement traceability matrix pada *domain layer* terkait fitur katalog sebagai berikut.

Table 3.8 Requirements Tracebility Matrix Domain layer

REQUIREMENT TRACEABILITY MATRIX			
Project Name	Iproc 2GO		
Created By	Muhammad Syaifullah		
Feature	Ecatalog		
Layer	Use Case		
Requirement ID	Requirement Description	Test Case ID	Test Case Description
001/UNT/001	Menampilkan daftar modul catalog	TC/UC/001	When getModules() Should return Loading Then Success
		TC/UC/002	Given modules in storage When getModules() Should return Loading Then Error
		TC/UC/003	Given empty modules in storage When getModules() Should return Loading Then Error
001/UNT/002	Menampilkan daftar filter	TC/UC/004	When getFilters() Should return Loading Then Success
		TC/UC/005	Given filter catalog list response When getFilters() Should return Loading Then Success with correct data
		TC/UC/006	Given error response When getFilters() Should return Loading Then Error
001/UNT/003	Menampilkan daftar detail filter	TC/UC/007	When getSeeAllFilter() Should return Loading Then Success
		TC/UC/008	Given list detail filter When getSeeAllFilter() Should return Loading Then Success with correct data
		TC/UC/009	Given error response When getSeeAllFilter() Should return Loading Then Error
001/UNT/004	Menampilkan total katalog produk berdasarkan filter yang diterapkan	TC/UC/010	Given module and filter parameter with total product response When countFilteredProducts() Should return Loading Then Success

		TC/UC/011	Given module and filter parameter with error response When countFilteredProducts() Should return Loading Then Error
001/UNT/008	Menampilkan daftar katalog menggunakan metode pagination berdasarkan filter dan tipe yang diterapkan	TC/UC/012	When getProducts() Should return Loading Then Success
		TC/UC/013	When getProducts() Should return Loading Then Error

Requirement traceability matrix akan dijadikan sebagai indikator tercapai atau tidaknya pembuatan *unit test*. Adapun *unit test* terkait sebagai berikut.

```
@RunWith(MockitoJUnitRunner::class)
class CatalogUseCaseTest : BaseUseCaseTest<CatalogUseCase>() {

    @Mock
    lateinit var repository: ICatalogRepository

    override lateinit var useCase: CatalogUseCase

    private fun provideUseCase() = CatalogInteractor(repository, Dispatchers.Default)

    @Test
    fun `When getModules() Should return Loading Then Success`() = runTest {
        `when`(repository.getModules()).thenReturn(
            flow {
                emit(Resource.Loading())
                emit(Resource.Success(ModuleDummyData.catalogModuleListModel()))
            }
        )

        useCase = provideUseCase()

        useCase.getModules().test {
            val emissionLoading = awaitItem()
            assertTrue(emissionLoading is Resource.Loading)

            verify(repository).getModules()

            val emissionSuccess = awaitItem()
            assertTrue(emissionSuccess is Resource.Success)
            assertEquals(ModuleDummyData.catalogModuleListModel(), emissionSuccess.data)

            cancelAndIgnoreRemainingEvents()
        }
    }
}
```

Gambar 3.16 Kode unit test catalog use case

c. *Presentation Layer*

Presentation layer bertanggung jawab untuk interaksi dengan pengguna akhir dan menampilkan data. Pada kasus ini yang bertindak sebagai *presentation layer* adalah view model. Pengujian pada *layer* ini bertujuan untuk bahwa logika presentasi berfungsi dengan benar dan data ditampilkan sesuai yang diharapkan. Adapun *requirement traceability matrix* pada *presentation layer* terkait fitur katalog sebagai berikut.

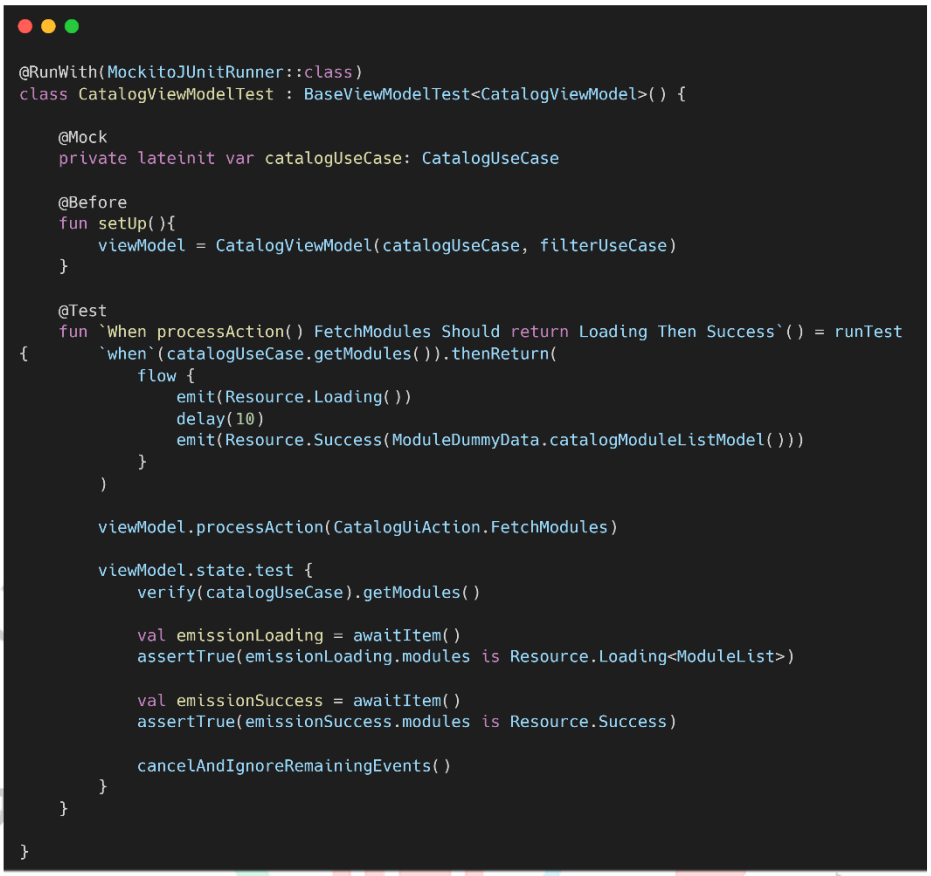
Table 3.9 Requirement Traceability Matrix Presentation layer

REQUIREMENT TRACEABILITY MATRIX			
Project Name	Iproc 2GO		
Created By	Muhammad Syaifullah		
Feature	Ecatalog		
Layer	ViewModel		
Requirement ID	Requirement Description	Test Case ID	Test Case Description
001/UNT/001	Menampilkan daftar modul catalog	TC/VM/001	Given empty modules in local storage When Fetch Modules Should return Loading Then Error
		TC/VM/002	Given empty modules cache When Fetch Modules Should return Loading Then Success with correct data
		TC/VM/003	Given not empty modules in local storage When FetchModules Should return Loading Then Error with old data
001/UNT/002	Menampilkan daftar filter	TC/VM/004	Given empty active filter When processAction() FetchFilters Should update filter state
		TC/VM/005	Given not empty active filter list When Fetch Filters Should update filter and reset active filter state
		TC/VM/006	Given error catalog filter list When Fetch Filters Should update filter state with error filter

		TC/VM/007	When Show Filter Dialog with show param is true Should showing dialog state
		TC/VM/008	When Show Filter Dialog with show param is false Should dismiss dialog state
001/UNT/003	Menampilkan daftar detail filter	TC/VM/009	When Fetch See All Filters Should return Loading Then Success
		TC/VM/010	When Fetch See All Filters Should return Loading Then Error
		TC/VM/011	When Show Filter Detail Should update detail dialog state
		TC/VM/012	When Hide Filter Detail Dialog Should update detail dialog state to null
001/UNT/004	Menampilkan total katalog produk berdasarkan filter yang diterapkan	TC/VM/013	When Count Filtered Products Should return Loading Then Success
		TC/VM/014	When Count Filtered Products Should return Loading Then Error
		TC/VM/015	Given not empty active filter state When ApplyFilters Should update filter chip and main chip state
		TC/VM/016	When After Apply Filters Should update triggerToReload state to false
001/UNT/005	Menghapus semua filter yang telah diterapkan	TC/VM/017	When Reset Filters Should set active filter state to empty and reset filter list and chip state
001/UNT/006	Menampilkan total katalog produk berdasarkan detail filter yang diterapkan	TC/VM/018	Given selected filter Item When On Click Filter Detail Item Should update tempActiveFilterDetails state
		TC/VM/019	Given not empty active filter detail state When Apply Filter Details Should update data dialog state

001/UNT/007	Menghapus semua filter yang telah diterapkan	TC/VM/020	Given not empty active filter and detail filter When Reset Filter Details Should reset active indicator on filter Details state
001/UNT/008	Menampilkan daftar katalog menggunakan metode pagination berdasarkan filter dan tipe yang diterapkan	TC/VM/021	Given selected module with empty query search and not empty active filter When processAction() FetchProducts Should update data paging with not empty page
		TC/VM/022	Given selected module with empty query search and not empty active filter When processAction() FetchProducts Should update data paging with empty page
		TC/VM/023	Given selected module is null When processAction() FetchProducts Should update data paging flow with empty flow
001/UNT/009	Membuka Halaman Kejanjang	TC/VM/024	When process Action NavigateToCart Should process Effect NavigateToCart
001/UNT/010	Membuka Detail Produk	TC/VM/025	Given not null task When process Action OpenProduct Should process Effect OpenProduct
		TC/VM/026	Given null task When process Action OpenProduct Should bot process Effect OpenProduct

Requirement Traceability Matrix akan dijadikan sebagai indikator tercapai atau tidaknya pembuatan *unit test*. Adapun *unit test* terkait sebagai berikut.



```
@RunWith(MockitoJUnitRunner::class)
class CatalogViewModelTest : BaseViewModelTest<CatalogViewModel>() {

    @Mock
    private lateinit var catalogUseCase: CatalogUseCase

    @Before
    fun setUp(){
        viewModel = CatalogViewModel(catalogUseCase, filterUseCase)
    }

    @Test
    fun `When processAction() FetchModules Should return Loading Then Success`() = runTest
{       `when`(catalogUseCase.getModules()).thenReturn(
            flow {
                emit(Resource.Loading())
                delay(10)
                emit(Resource.Success(ModuleDummyData.catalogModuleListModel()))
            }
        )

        viewModel.processAction(CatalogUiAction.FetchModules)

        viewModel.state.test {
            verify(catalogUseCase).getModules()

            val emissionLoading = awaitItem()
            assertTrue(emissionLoading.modules is Resource.Loading<ModuleList>)

            val emissionSuccess = awaitItem()
            assertTrue(emissionSuccess.modules is Resource.Success)

            cancelAndIgnoreRemainingEvents()
        }
    }

}
```

Gambar 3.17 Kode unit test catalog view model

3.2.4. Pengatuan Lingkungan Pengujian (*Test Environment Setup*)

Dalam pengembangan aplikasi Android, proses pengujian memegang peranan yang sangat penting untuk memastikan aplikasi berfungsi sesuai dengan harapan dan bebas dari bug sebelum dirilis kepada pengguna. Salah satu aspek kunci dalam pengujian aplikasi Android adalah pengaturan lingkungan pengujian. Lingkungan pengujian yang tepat memungkinkan pengembang untuk melakukan berbagai jenis pengujian secara efisien dan efektif. Adapun hal – hal yang perlu diperhatikan dalam proses pengaturan lingkungan pengujian sebagai berikut.

a. Kebutuhan lingkungan pengujian

Table 3.10 Kebutuhan lingkungan pengujian

Tools	Version

Kotlin	1.9.22
Java	18.0.2
Build Gradle	8.4.0
Android Studio	Iguana | 2023.2.1
SDK Build Tools	34.0.3

b. Konfigurasi *Build Variants*

Android Studio menyediakan fitur *build variants* yang memungkinkan pengembang untuk membuat berbagai varian build dari aplikasi yang sama. *Build variants* dapat digunakan untuk memisahkan kode dan sumber daya yang digunakan dalam lingkungan pengujian dari yang digunakan dalam produksi. Adapaun *variant* yang digunakan saat ini sebagai berikut

Table 3.11 Konfigurasi build variants

Nama Variant	Namespace	Internal / External
uat	com.adw.iproc2go	Internal
live	com.adw.iproc2go.uat	Internal
vendorUat	com.adw.iproc2go.vendor.uat	External
vendorLive	com.adw.iproc2go.vendor	External

Variant uat dan vendorUat digunakan untuk fase development sedangkan live dan vendorLive digunakan untuk *release* ke *production* / Playstore. Setiap variant akan menghasilkan 2 *build variant* yaitu debug dan release sebagai contoh variant uat akan menghasilkan uatDebug dan uatRelease. *Build variant* uatDebug digunakan untuk internal atau vendorUatDebug untuk external untuk keperluan development. *Build variant* liveRelease digunakan untuk internal dan vendorLiveRelease untuk external untuk keperluan release ke *production*.

3.2.5. Eksekusi Uji (*Test Execution*)

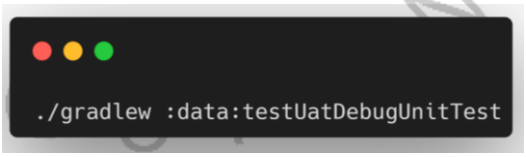
Fase eksekusi uji (*test execution*) adalah langkah penting dalam proses pengujian aplikasi Android. Pada fase ini, tes yang telah direncanakan dan diimplementasikan sebelumnya dijalankan untuk memverifikasi bahwa perangkat lunak berfungsi sebagaimana mestinya. menjalankan *unit test* pada android, dapat menggunakan perintah seperti dibawah

Gambar 3.18 perintah menjalankan unit test

Setelah menjalankan perintah diatas berdasarkan modul dan *build variant* masing masing, gradle akan langsung menjalankan semua *unit testing* yang ada. Setelah selesai menjalankan semua test case, *gradle* akan membuat dokumen terkait laporan hasil pengetesan. Hasil ini memuat data sebarapa banyak *test case* yang kita buat sukses atau gagal. Laporan ini akan memudahkan penguji untuk mengetahui kecatatan dalam *test case* yang telah dibuat dan pada titik mana kecatatan itu terjadi. Berikut hasil pengetesan pada tiap *layer* yang ada.

a. Data layer

Pengetesan pada *data layer* dapat dilakukan dengan menjalankan perintah berikut



Gambar 3.19 perintah unit test pada data layer

Gradle akan menjalankan semua *unit test* yang ada pada *data layer* dan membuat laporan terkait status sukses dan galat dari tiap *test case* yang diuji. Berikut untuk laporan hasil pengujian pada *data layer*.

Class com.adw.iproc2go.data.repository.CatalogRepositoryTest

all > com.adw.iproc2go.data.repository > CatalogRepositoryTest

14	0	0	1.553s	100%
tests	failures	ignored	duration	successful

Tests

Test	Duration	Result
Given empty modules in storage When getModules() Should return Loading Then Error	0.015s	passed
Given empty task filter list When getFilters() Should return Loading Then Success with correct data	0.009s	passed
Given error response When getFilters() Should return Loading Then Error	0.008s	passed
Given failure response When countFilteredProducts() Should return Loading Then Error	0.016s	passed
Given failure response When getSeeAllFilters() Should return Loading Then Error	0.009s	passed
Given list detail filter When getSeeAllFilters() Should return Loading Then Success	1.144s	passed
Given modules in storage When getModules() Should return Loading Then Error	0.032s	passed
Given selected module with empty input filter parameter When countFilteredProducts() Should return Loading Then Success	0.012s	passed
Given selected module with empty input filter parameter When getProducts() Should return Loading Then Success	0.222s	passed
Given selected module with not empty input filter parameter When countFilteredProducts() Should return Loading Then Success	0.007s	passed
Given selected module with not empty input filter parameter When getProducts() Should return Loading Then Success	0.033s	passed
Given task filter list response When getFilters() Should return Loading Then Success with correct data	0.014s	passed
When getModules() Should return Loading Then Success	0.016s	passed
When getProducts() Should return Loading Then Error	0.016s	passed

Gambar 3.20 laporan hasil unit test pada catalog repository test

Class com.adw.iproc2go.data.remote.datasource.CatalogRemoteDataSourceTest

all > com.adw.iproc2go.data.remote.datasource > CatalogRemoteDataSourceTest

42	0	0	1.708s	100%
tests	failures	ignored	duration	successful

Tests

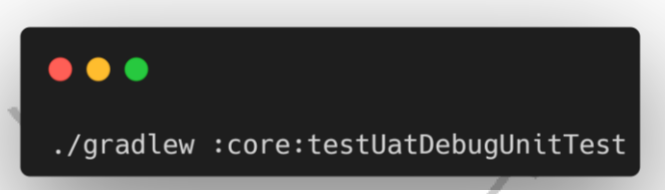
Test	Duration	Result
Given first page data When getCatalogPagingSource_getRefreshKey() Should return null refresh key	0.011s	passed
Given response data not empty When getCatalogPagingSource_load() Should return Page	0.047s	passed
Given service_getCatalogTypes return ConnectException When fetch module list Should return result failure with ConnectException	0.005s	passed
Given service_getCatalogTypes return Exception When fetch module list Should return result failure with Exception	0.005s	passed
Given service_getCatalogTypes return HttpException When fetch module list Should return result failure with HttpException	0.260s	passed
Given service_getCatalogTypes return JsonEncodingException When fetch module list Should return result failure with JsonEncodingException	0.007s	passed
Given service_getCatalogTypes return SSLHandshakeException When fetch module list Should return result failure with SSLHandshakeException	0.008s	passed
Given service_getCatalogTypes return SocketTimeoutException When fetch module list Should return result failure with SocketTimeoutException	0.004s	passed
Given service_getCatalogTypes return Success Response When fetch module list Should return result success	0.007s	passed
Given service_getCatalogTypes return UnknownHostException When fetch module list Should return result failure with UnknownHostException	0.006s	passed
Given service_getFilterDetail return ConnectException When fetch catalog detail filter list Should return result failure with ConnectException	0.008s	passed
Given service_getFilterDetail return Exception When fetch catalog detail filter list Should return result failure with Exception	0.005s	passed
Given service_getFilterDetail return HttpException When fetch catalog detail filter list Should return result failure with HttpException	0.009s	passed
Given service_getFilterDetail return JsonEncodingException When fetch catalog detail filter list Should return result failure with JsonEncodingException	0.006s	passed
Given service_getFilterDetail return SSLHandshakeException When fetch catalog detail filter list Should return result failure with SSLHandshakeException	0.005s	passed
Given service_getFilterDetail return SocketTimeoutException When fetch catalog detail filter list Should return result failure with SocketTimeoutException	0.008s	passed
Given service_getFilterDetail return Success Response When fetch catalog detail filter list Should return result success	0.006s	passed
Given service_getFilterDetail return UnknownHostException When fetch catalog detail filter list Should return result failure with UnknownHostException	0.005s	passed
Given service_getFilters return ConnectException When fetch catalog filter list Should return result failure with ConnectException	0.005s	passed
Given service_getFilters return Exception When fetch catalog filter list Should return result failure with Exception	1.126s	passed
Given service_getFilters return HttpException When fetch catalog filter list Should return result failure with HttpException	0.010s	passed
Given service_getFilters return JsonEncodingException When fetch catalog filter list Should return result failure with JsonEncodingException	0.005s	passed
Given service_getFilters return SSLHandshakeException When fetch catalog filter list Should return result failure with SSLHandshakeException	0.005s	passed
Given service_getFilters return SocketTimeoutException When fetch catalog filter list Should return result failure with SocketTimeoutException	0.005s	passed
Given service_getFilters return Success Response When fetch catalog filter list Should return result success	0.011s	passed
Given service_getFilters return UnknownHostException When fetch catalog filter list Should return result failure with UnknownHostException	0.004s	passed
Given service_getTotalFilteredProducts return ConnectException When get total filtered product Should return result failure with ConnectException	0.006s	passed
Given service_getTotalFilteredProducts return Exception When get total filtered product Should return result failure with Exception	0.007s	passed
Given service_getTotalFilteredProducts return HttpException When get total filtered product Should return result failure with HttpException	0.013s	passed
Given service_getTotalFilteredProducts return JsonEncodingException When get total filtered product Should return result failure with JsonEncodingException	0.005s	passed
Given service_getTotalFilteredProducts return SSLHandshakeException When get total filtered product Should return result failure with SSLHandshakeException	0.005s	passed
Given service_getTotalFilteredProducts return SocketTimeoutException When get total filtered product Should return result failure with SocketTimeoutException	0.012s	passed
Given service_getTotalFilteredProducts return Success Response When get total filtered product Should return result success	0.005s	passed
Given service_getTotalFilteredProducts return UnknownHostException When get total filtered product Should return result failure with UnknownHostException	0.007s	passed
Given service_listPaging return ConnectException When fetch catalog list paging Should return result failure with ConnectException	0.007s	passed
Given service_listPaging return Exception When fetch catalog list paging Should return result failure with Exception	0.006s	passed
Given service_listPaging return HttpException When fetch catalog list paging Should return result failure with HttpException	0.017s	passed
Given service_listPaging return JsonEncodingException When fetch catalog list paging Should return result failure with JsonEncodingException	0.006s	passed
Given service_listPaging return SSLHandshakeException When fetch catalog list paging Should return result failure with SSLHandshakeException	0.004s	passed
Given service_listPaging return SocketTimeoutException When fetch catalog list paging Should return result failure with SocketTimeoutException	0.008s	passed
Given service_listPaging return Success Response When fetch catalog list paging Should return result success	0.006s	passed
Given service_listPaging return UnknownHostException When fetch catalog list paging Should return result failure with UnknownHostException	0.011s	passed

Gambar 3.21 laporan hasil unit test pada catalog remote data source test

Gambar 3.20 dan 3.21 menunjukan seluruh *test case* yang telah dibuat berhasil diekseskusi dan tidak ditemukan kecatatan dalam penulisan *unit test*.

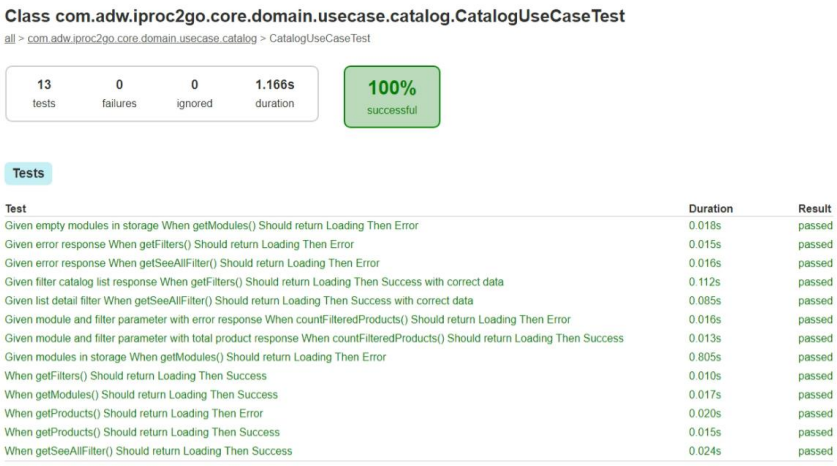
b. *Domain Layer*

Pengetesan pada *domain layer* dapat dilakukan dengan menjalankan perintah berikut



Gambar 3.22 perintah unit test pada domain layer

Gradle akan menjalankan semua *unit test* yang ada pada *domain layer* dan membuat laporan terkait status sukses dan galat dari tiap *test case* yang diuji. Berikut untuk laporan hasil pengujian pada *domain layer*.



Class com.adw.iproc2go.core.domain.usecase.catalog.CatalogUseCaseTest

all > com.adw.iproc2go.core.domain.usecase.catalog > CatalogUseCaseTest

13 tests	0 failures	0 ignored	1.166s duration	100% successful

Tests

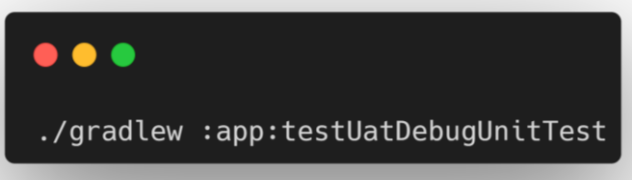
Test	Duration	Result
Given empty modules in storage When getModules() Should return Loading Then Error	0.018s	passed
Given error response When getFilters() Should return Loading Then Error	0.015s	passed
Given error response When getSeeAllFilter() Should return Loading Then Error	0.016s	passed
Given filter catalog list response When getFilters() Should return Loading Then Success with correct data	0.112s	passed
Given list detail filter When getSeeAllFilter() Should return Loading Then Success with correct data	0.085s	passed
Given module and filter parameter with error response When countFilteredProducts() Should return Loading Then Error	0.016s	passed
Given module and filter parameter with total product response When countFilteredProducts() Should return Loading Then Success	0.013s	passed
Given modules in storage When getModules() Should return Loading Then Error	0.805s	passed
When getFilters() Should return Loading Then Success	0.010s	passed
When getModules() Should return Loading Then Success	0.017s	passed
When getProducts() Should return Loading Then Error	0.020s	passed
When getProducts() Should return Loading Then Success	0.015s	passed
When getSeeAllFilter() Should return Loading Then Success	0.024s	passed

Gambar 3.23 laporan hasil unit test pada catalog use case test

Gambar 3.23 menunjukan seluruh *test case* yang telah dibuat berhasil diekseskusi dan tidak ditemukan kecatatan dalam penulisan *unit test*.

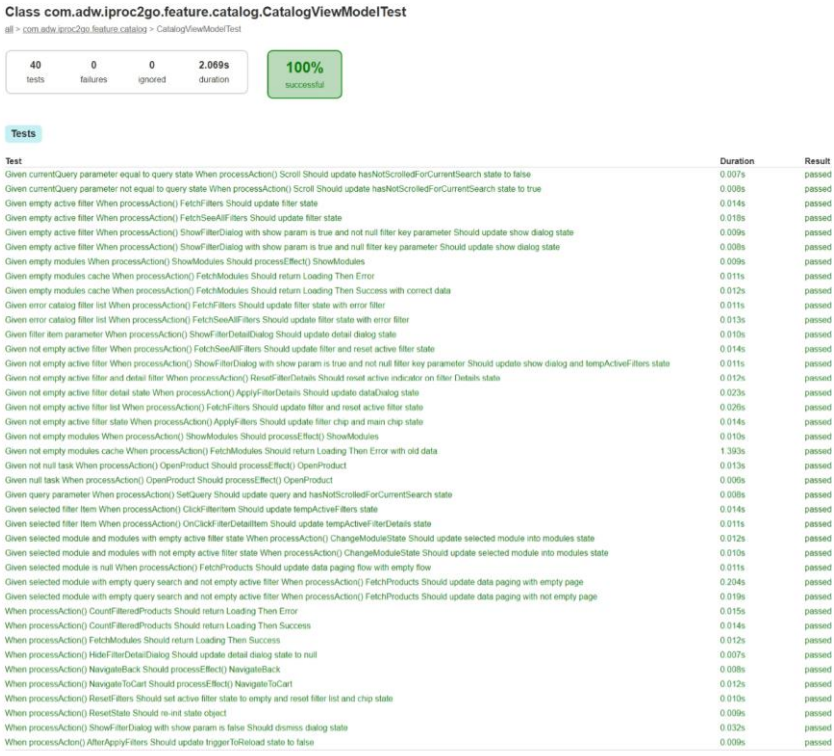
c. *Presentation Layer*

Pengetesan pada *presentation layer* dapat dilakukan dengan menjalankan perintah berikut



Gambar 3.24 perintah unit test pada presentation layer

Gradle akan menjalankan semua *unit test* yang ada pada *presentation layer* dan membuat laporan terkait status sukses dan galat dari tiap *test case* yang diuji. Berikut untuk laporan hasil pengujian pada *presentation layer*.



Class com.adw.iproc2go.feature.catalog.CatalogViewModelTest

all > com.adw.iproc2go.feature.catalog > CatalogViewModelTest

40 tests	0 failures	0 ignored	2.069s duration	100% successful

Tests

Test	Duration	Result
Given currentQuery parameter equal to query state When processAction() Scroll Should update hasNotScrolledForCurrentSearch state to false	0.007s	passed
Given currentQuery parameter not equal to query state When processAction() Scroll Should update hasNotScrolledForCurrentSearch state to true	0.008s	passed
Given empty active filter When processAction() FetchFilters Should update filter state	0.014s	passed
Given empty active filter When processAction() FetchSeeAllFilters Should update filter state	0.018s	passed
Given empty active filter When processAction() ShowFilterDialog with show param is true and not null filter key parameter Should update show dialog state	0.009s	passed
Given empty active filter When processAction() ShowFilterDialog with show param is true and null filter key parameter Should update show dialog state	0.008s	passed
Given empty modules When processAction() ShowModules Should processEffect() ShowModules	0.009s	passed
Given empty modules cache When processAction() FetchModules Should return Loading Then Error	0.011s	passed
Given empty modules cache When processAction() FetchModules Should return Loading Then Success with correct data	0.012s	passed
Given error catalog filter list When processAction() FetchFilters Should update filter state with error filter	0.011s	passed
Given error catalog filter list When processAction() FetchSeeAllFilters Should update filter state with error filter	0.013s	passed
Given filter item parameter When processAction() ShowFilterDetailDialog Should update detail dialog state	0.010s	passed
Given not empty active filter When processAction() FetchSeeAllFilters Should update filter and reset active filter state	0.014s	passed
Given not empty active filter When processAction() ShowFilterDialog with show param is true and not null filter key parameter Should update show dialog and tempActiveFilters state	0.011s	passed
Given not empty active filter and detail filter When processAction() ResetFilterDetails Should reset active indicator on filter Details state	0.012s	passed
Given not empty active filter detail state When processAction() ApplyFilterDetails Should update dataDialog state	0.023s	passed
Given not empty active filter list When processAction() FetchFilters Should update filter and reset active filter state	0.026s	passed
Given not empty active filter state When processAction() ApplyFilters Should update filter chip and main chip state	0.014s	passed
Given not empty modules When processAction() ShowModules Should processEffect() ShowModules	0.010s	passed
Given not empty modules cache When processAction() FetchModules Should return Loading Then Error with old data	1.393s	passed
Given not null task When processAction() OpenProduct Should processEffect() OpenProduct	0.013s	passed
Given null task When processAction() OpenProduct Should processEffect() OpenProduct	0.006s	passed
Given query parameter When processAction() SetQuery Should update query and hasNotScrolledForCurrentSearch state	0.008s	passed
Given selected filter Item When processAction() ClickFilterItem Should update tempActiveFilters state	0.014s	passed
Given selected filter Item When processAction() OnClickFilterDetailItem Should update tempActiveFilterDetails state	0.011s	passed
Given selected module and modules with empty active filter state When processAction() ChangeModuleState Should update selected module into modules state	0.012s	passed
Given selected module and modules with not empty active filter state When processAction() ChangeModuleState Should update selected module into modules state	0.010s	passed
Given selected module is null When processAction() FetchProducts Should update data paging flow with empty flow	0.011s	passed
Given selected module with empty query search and not empty active filter When processAction() FetchProducts Should update data paging with empty page	0.204s	passed
Given selected module with empty query search and not empty active filter When processAction() FetchProducts Should update data paging with not empty page	0.019s	passed
When processAction() CountFilteredProducts Should return Loading Then Error	0.015s	passed
When processAction() CountFilteredProducts Should return Loading Then Success	0.014s	passed
When processAction() FetchModules Should return Loading Then Success	0.012s	passed
When processAction() HideFilterDetailDialog Should update detail dialog state to null	0.007s	passed
When processAction() NavigateBack Should processEffect() NavigateBack	0.008s	passed
When processAction() NavigateToCart Should processEffect() NavigateToCart	0.012s	passed
When processAction() ResetFilters Should set active filter state to empty and reset filter list and chip state	0.010s	passed
When processAction() ResetState Should re-init state object	0.009s	passed
When processAction() ShowFilterDialog with show param is false Should dismiss dialog state	0.032s	passed
When processAction() AfterApplyFilters Should update triggerToReload state to false	0.009s	passed

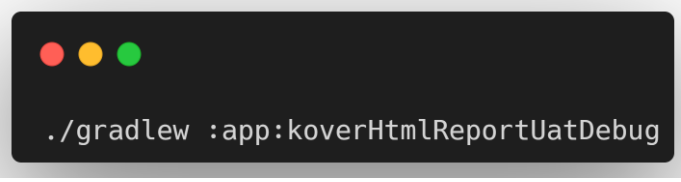
Gambar 3.25 laporan hasil unit test pada catalog view model test

Gambar 3.25 menunjukan seluruh test case yang telah dibuat berhasil diekseskusi dan tidak ditemukan kecatatan dalam penulisan *unit test*.

d. Coverage Test

Coverage test adalah metrik penting dalam pengujian perangkat lunak yang mengukur sejauh mana kode aplikasi diuji oleh tes unit. Ini memberikan gambaran tentang persentase kode yang dieksekusi selama pengujian,

membantu mengidentifikasi bagian mana dari aplikasi yang belum diuji dan mungkin rentan terhadap *bug*. *Coverage test* dapat dijalankan dengan perintah dibawah



Gambar 3.26 perintah untuk coverage test keseluruhan aplikasi

Setelah menjalankan perintah di atas, kover akan membuat laporan *coverage test* dengan format html yang akan memudahkan penguji mengetahui apakah ada kode yang belum tercakupi oleh pengetesan. Berikut *coverage report* dari pengetesan yang telah dilakukan.

Coverage Summary for Class: CatalogRemoteDataSource (com.adw.iproc2go.data.remote.datasource)

Class	Method, %	Branch, %	Line, %	Instruction, %
CatalogRemoteDataSource	53.8% (7/13)		66.7% (12/18)	47.8% (96/201)
CatalogRemoteDataSource$countFilteredProducts$1				
CatalogRemoteDataSource$countFilteredProducts$2	100% (1/1)		100% (3/3)	100% (16/16)
CatalogRemoteDataSource$getCourier$1				
CatalogRemoteDataSource$getCourier$2	0% (0/1)	0% (0/2)	0% (0/1)	0% (0/20)
CatalogRemoteDataSource$getCourierCost$1				
CatalogRemoteDataSource$getCourierCost$2	0% (0/1)		0% (0/1)	0% (0/24)
CatalogRemoteDataSource$getDeliveryHistory$1				
CatalogRemoteDataSource$getDeliveryHistory$2	0% (0/1)		0% (0/1)	0% (0/15)
CatalogRemoteDataSource$getFilters$1				
CatalogRemoteDataSource$getFilters$2	100% (1/1)		100% (1/1)	100% (12/12)
CatalogRemoteDataSource$getModules$1				
CatalogRemoteDataSource$getModules$2	100% (1/1)		100% (1/1)	100% (12/12)
CatalogRemoteDataSource$getProduct$1				
CatalogRemoteDataSource$getProduct$2	0% (0/1)		0% (0/1)	0% (0/16)
CatalogRemoteDataSource$getProductPrivate$1				
CatalogRemoteDataSource$getProductPrivate$2	0% (0/1)		0% (0/1)	0% (0/19)
CatalogRemoteDataSource$getProducts$1				
CatalogRemoteDataSource$getProducts$2	100% (1/1)		100% (6/6)	100% (25/25)
CatalogRemoteDataSource$getSeeAllFilters$1				
CatalogRemoteDataSource$getSeeAllFilters$2	100% (1/1)		100% (2/2)	100% (14/14)
CatalogRemoteDataSource$submitCheckout$1				
CatalogRemoteDataSource$submitCheckout$2	0% (0/1)	0% (0/4)	0% (0/2)	0% (0/47)
Total	50% (12/24)	0% (0/6)	65.8% (25/38)	41.6% (175/421)

Gambar 3.27 hasil coverage test pada catalog remote data source

Coverage Summary for Class: CatalogViewModel (com.adw.iproc2go.feature.catalog)

Class	Method, %	Branch, %	Line, %	Instruction, %
CatalogViewModel	100% (26/26)	83.3% (10/12)	100% (91/91)	99.2% (640/645)
CatalogViewModel$applyFilters$1	100% (1/1)	50% (4/8)	100% (4/4)	100% (72/72)
CatalogViewModel$Companion	100% (1/1)		100% (1/1)	100% (2/2)
CatalogViewModel$countFilteredProducts$1	100% (1/1)	100% (2/2)	100% (4/4)	100% (33/33)
CatalogViewModel$countFilteredProducts$1$1	100% (1/1)		100% (4/4)	100% (32/32)
CatalogViewModel$fetchFilters$2	100% (1/1)	50% (1/2)	100% (1/1)	100% (20/20)
CatalogViewModel$fetchFilters$2$1	100% (1/1)		100% (4/4)	100% (32/32)
CatalogViewModel$fetchModules$1	100% (1/1)	50% (1/2)	100% (1/1)	100% (19/19)
CatalogViewModel$fetchModules$1$1	100% (1/1)		100% (3/3)	100% (20/20)
CatalogViewModel$fetchSeeAllFilters$1	100% (1/1)	50% (2/4)	100% (1/1)	100% (40/40)
CatalogViewModel$fetchSeeAllFilters$1$1	100% (1/1)	62.5% (10/16)	100% (14/14)	100% (175/175)
CatalogViewModel$fetchSeeAllFilters$1$1$emit$1				
CatalogViewModel$onApplyFilterDetails$1	100% (1/1)	50% (4/8)	100% (4/4)	100% (84/84)
CatalogViewModel$onFilterItemClicks$1	100% (1/1)	50% (1/2)	100% (3/3)	100% (22/22)
CatalogViewModel$resetFilterDetail$1	100% (1/1)	50% (4/8)	100% (3/3)	100% (79/79)
CatalogViewModel$special$$inlined$filterIsInstance$1	0% (0/2)			
CatalogViewModel$special$$inlined$filterIsInstance$1$2	0% (0/1)			
CatalogViewModel$special$$inlined$filterIsInstance$1$2$1				
CatalogViewModel$special$$inlined$flatMapLatest$1	0% (0/1)			
CatalogViewModel$updateStates$1	100% (1/1)	100% (44/44)	100% (32/32)	100% (221/221)
Total	90.9% (40/44)	76.9% (83/108)	100% (170/170)	99.7% (1491/1496)

Gambar 3.28 hasil coverage test pada catalog view model

Coverage Summary for Class: CatalogInteractor (com.adw.iproc2go.core.domain.usecase.catalog)

Class	Method, %	Branch, %	Line, %	Instruction, %
CatalogInteractor	100% (6/6)		100% (8/8)	100% (39/39)
CatalogInteractor$getSeeAllFilter$2	100% (1/1)		100% (1/1)	100% (16/16)
CatalogInteractor$getSeeAllFilter$2$1	100% (1/1)	100% (8/8)	100% (9/9)	100% (71/71)
Total	100% (8/8)	100% (8/8)	100% (18/18)	100% (126/126)

Gambar 3.29 hasil coverage test pada catalog interactor

Coverage Summary for Class: CatalogRepository (com.adw.iproc2go.data.repository)

Class	Method, %	Branch, %	Line, %	Instruction, %
CatalogRepository	41.7% (5/12)		41.9% (13/31)	36.2% (54/149)
CatalogRepository$countFilteredProducts$1	100% (3/3)		100% (4/4)	100% (23/23)
CatalogRepository$countFilteredProducts$1$requestFromRemote$1				
CatalogRepository$getCourier$1	0% (0/3)		0% (0/3)	0% (0/18)
CatalogRepository$getCourier$1$requestFromRemote$1				
CatalogRepository$getCourierCost$1	0% (0/3)	0% (0/2)	0% (0/4)	0% (0/30)
CatalogRepository$getCourierCost$1$requestFromRemote$1				
CatalogRepository$getDeliveryHistory$1	0% (0/3)		0% (0/3)	0% (0/19)
CatalogRepository$getDeliveryHistory$1$requestFromRemote$1				
CatalogRepository$getFilters$1	100% (3/3)		100% (3/3)	100% (16/16)
CatalogRepository$getFilters$1$requestFromRemote$1				
CatalogRepository$getModules$1	100% (5/5)	50% (1/2)	100% (6/6)	96.9% (62/64)
CatalogRepository$getModules$1$loadCurrentLocalData$1				
CatalogRepository$getModules$1$loadResult$$inlinedmap1	0% (0/2)			
CatalogRepository$getModules$1$loadResult$$inlinedmap1$2	0% (0/1)			
CatalogRepository$getModules$1$loadResult$$inlinedmap1$2$1				
CatalogRepository$getModules$1$requestFromRemote$1				
CatalogRepository$getProduct$1	0% (0/3)		0% (0/3)	0% (0/24)
CatalogRepository$getProduct$1$requestFromRemote$1				
CatalogRepository$getProductPrivate$1	0% (0/3)		0% (0/3)	0% (0/30)
CatalogRepository$getProductPrivate$1$requestFromRemote$1				
CatalogRepository$getProducts$1	0% (0/1)		0% (0/4)	0% (0/13)
CatalogRepository$getSeeAllFilters$1	100% (3/3)		100% (4/4)	100% (21/21)
CatalogRepository$getSeeAllFilters$1$requestFromRemote$1				
CatalogRepository$submitCheckout$1	0% (0/3)		0% (0/4)	0% (0/22)
CatalogRepository$submitCheckout$1$requestFromRemote$1				
Total	39.6% (19/48)	25% (1/4)	41.7% (30/72)	41% (176/429)

Gambar 3.30 hasil coverage test pada catalog viewmodel

Berdsarkan Hasil laporan diatas menunjukan bahwa *coverage test* pada domain dan presentation layer telah mencapai 100%. Hal tersebut berarti semua *control flow* dan *branch* yang ada pada kedua *class* tersebut telah tercakupi oleh pengujian yang dibuat. Hal sebaliknya terjadi pada *domain layer*. Pengujian yang dibuat pada *layer* ini tidak dapat mencakupi seluruh *control flow* dan *branch* yang ada. Hal ini perlu akan dicatat oleh *project manager* sebagai pekerjaan tambahan pada siklus pegujian perangkat lunak selanjutnya.

3.2.6. Penutupan Pengujian (*Test Closure*)

Penutupan pengujian adalah fase terakhir dalam siklus hidup pengujian perangkat lunak. Tujuan dari fase ini adalah memastikan bahwa semua kegiatan pengujian telah selesai, semua temuan telah direspon dengan tepat, dan dokumentasi yang diperlukan telah diselesaikan dan disimpan. Penutupan pengujian melibatkan evaluasi hasil pengujian terhadap tujuan awal dan persyaratan proyek, serta penyusunan laporan akhir yang merangkum seluruh kegiatan pengujian.

Hasil dari evaluasi pada saat penutupan pengujian menunujukan bahwa terdapat beberapa pekerjaan yang belum memenuhi target. Pengujian yang telah dikembangkan tidak dapat mencakupi seluruh *control flow* dan *branch* yang ada pada semua *layer* yang telah ditentukan. Hal tersebut dikarenakan terdapat beberapa function dan statement yang belum tercatat pada *requirement traceability matrix*. Hal ini perlu dimasukkan ke dalam catatan pengujian dan dikerjakan pada siklus pengujian selanjutnya.

3.3 Kendala Yang Dihadapi

Selama masa kerja praktik, praktikan menghadapi beberapa tantangan, di antaranya adalah kesulitan yang cukup signifikan dalam mengembangkan aplikasi. Aplikasi yang ditangani oleh praktikan merupakan produk baru yang dibuat oleh perusahan untuk kebutuhan banyak. Hal ini menyebabkan sumber daya manusia yang diberikan perusahaan tidak terlalu banyak, sehingga menjadi tantangan bagi praktikan bekerja dalam tim kecil yang ditujukan sebagai tim riset pada perusahaan. Praktikan dituntut untuk memberikan invoasi dan solusi terbaik yang dapat memberikan keutungan bagi perusahaan. Oleh karena itu, praktikan perlu melakukan komunikasi dengan tim lainnya agar produk yang dibangun dapat langsung diimplementasikan pada klien – klien yang ada tanpa perlu banyak kustomisasi.

3.4 Cara Mengatasi Kendala

Untuk menghadapi tantangan tersebut, praktikan secara teratur berkomunikasi dengan senior dan anggota tim terkait secara mendalam. Komunikasi ini ditujukan sebagai media penyamaan visi antara tim teknis dan tim bisnis, sehingga produk yang dibangun dapat mengakomodir seluruh kebutuhan bisnis kedepannya. Pembuatan aplikasi dengan metode *server driven ui* menjadi solusi yang tepat diterapkan untuk menyelesaikan terkait perbedaan komponen inputan pada setiap proses di masing – masing klien. Hal ini membuat aplikasi dapat lebih adaptif terhadap perubahan yang dikirimkan oleh *server*.

3.5 Pembelajaran Yang Diperoleh dari Kerja Profesi

Selama menjalani karir profesionalnya, praktikan memperoleh berbagai pengalaman berharga, terutama dalam mengembangkan aplikasi Android dan membuat *unit test*. yaitu sebagai berikut:

1. Praktikan dapat menganalisa dokumen fungsional yang ada menjadi *test case,* kode program dan dokumen *requirement traceability matrix*.
2. Praktikan berkesempatan untuk ikut terlibat dalam proses perecenaaan dan pembuatan produk sehingga dapat memahami bagaimana sebuah produk dapat dibangun.
3. Praktikan lebih memahami terkait proses penulisan *test case* yang baik dan efektif, termasuk pembuatan *test case* untuk berbagai skenario.
4. Praktikan lebih memahami betapa pentingnya *unit testing* dalam pengembangan perangkat lunak untuk memastikan kualitas dan stabilitas aplikasi.
5. Praktikan lebih memahami pentingnya pengujian otomatis dapat mendeteksi *bug* lebih awal dan mengurangi biaya perbaikan.
6. Praktikan lebih memhamai pentingnya *refactoring* kode untuk meningkatkan keterbacaan dan pemeliharaan, serta bagaimana *unit test* dapat mendukung proses *refactoring* dengan memastikan bahwa perubahan tidak merusak fungsionalitas yang ada.

Secara umum, pengalaman dalam profesi ini memberikan pemahaman yang menyeluruh tentang semua tahapan dalam pengembangan perangkat lunak, dari perancangan dan implementasi hingga pengujian dan pemeliharaan. Pembelajaran ini tidak hanya meningkatkan keterampilan teknis dalam pengembangan aplikasi android dan pembuatan *unit test*, tetapi juga memperkuat pemahaman tentang praktik terbaik dalam pengembangan perangkat lunak secara umum.