BAB III
PELAKSANAAN KERJA PROFESI

3.1 Bidang Kerja

Dalam masa Kerja Praktik di PT Inti Corpora Teknologi, Praktikan diberi mandat untuk melaksanakan tugas dan tangggung jawab sebagai Frontend Developer di bawah naungan Technical Lead. Praktikan mendapatkan tugas dan tanggung jawab pada *Code Review* dan *white-box Testing* pada salah satu produk yaitu IAM. Praktikan menjalankan tugas sebagai Frontend Developer, adapun tugas dan tanggung jawab Frontend Developer secara umum di PT Inti Corpora Teknologi antara lain:

a. Melakukan analisis terhadap *Functional Spesification Diagram*.
b. Melakukan analisis desain antarmuka.
c. Melakukan *code review* produk.
d. Melakukan pengujian *white-box* pada *environment local developer*.
e. Melakukan pelaporan pekerjaan kepada *lead Frontend Developer*.

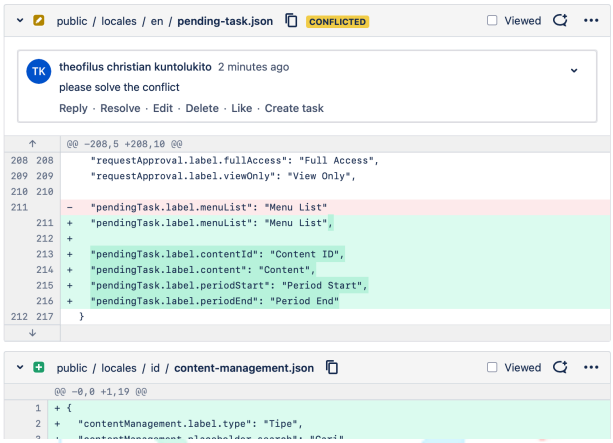
Selama masa Kerja Profesi, praktikan telah mendapatkan pengalaman dan penambahan wawasan dalam proses kerja Frontend Developer. Praktikan juga telah melaksanakan tahapan – tahapan dalam siklus hidup pengembangan sistem antara lain perencanaan, desain, implementasi, uji coba dan pemeliharaan.

3.2 Pelaksanaan Kerja

Praktikan bekerja sebagai karyawan professional selama 58 hari, terhitung mulai dari 1 Januari 2025 hingga 1 April 2025. Sebagai *Code Reviewer* dan *Code Tester* parktikan terlibat dalam proyek perusahaan untuk membantu mengembangkan sistem informasi manajemen akun berbasis *website.* Produk ini digunakan telah digunakan oleh beberapa *client* perusahaan. Pengembangan sistem informasi tersebut telah dimulai sebelum praktikan melaksanakan kerja praktik di PT Inti Corpora Teknologi.

Sistem informasi manajemen akun yang disebut juga dengan IAM merupakan sistem informasi yang digunakan untuk mengelola akun – akun sistem informasi yang saling terhubung. Sistem informasi ini bertujuan untuk meningkatkan pengawasan dan kontrol terhadap aktivitas akun – akun yang akan digunakan *client*.

Didalam pengembangan IAM praktikan diberikan tugas dan tanggung jawab untuk melakukan *review code* dan *white-box testing* pada *pull request* dari *frontend developer* yang menjadi *contributor* pada proyek IAM. Dalam melakukan *review code,* praktikan dibantu dengan senior *frontend developer* melakukan pengecheckan standar dan kebersihan *code.*



Gambar 3.1 Komentar Pada *Pull Request Review Code*

Pada Gambar 3.1 ditampilkan salah satu *pull request* dari developer yang telah diberikan komentar oleh praktikan serta ditandai untuk dilakukan permintaan perubahan kode (*code change request*). Komentar yang diberikan oleh praktikan pada *pull request* tersebut berisi permintaan perbaikan terhadap konflik kode (*code conflict*). Konflik kode ini terjadi karena adanya baris kode yang diubah secara bersamaan oleh dua atau lebih developer dalam versi yang berbeda, sehingga menyebabkan tabrakan (*conflict*) saat proses *merge*. Untuk menyelesaikan konflik tersebut, diperlukan proses pemilihan atau penggabungan bagian kode yang paling sesuai dari masing-masing *commit*, agar integrasi kode dapat berjalan dengan lancar dan tidak mengganggu fungsionalitas sistem

secara keseluruhan. Dalam melakukan *review code* pada *pull request*, terdapat alat – alat yang digunakan guna membantu proses *review code*. Adapun alat – alat yang digunakan pada proses *review code* pada *pull request* adalah sebagai berikut.

- Git Repository
  - Atlassian Bitbucket

Gambar 3.2 Logo Atlassian Bitbucket

(Sumber : https://www.atlassian.com/software/bitbucket)

Atlassian Bitbucket digunakan sebagai repository git produk – produk perusahaan. Bitbucket memungkinkan developer untuk berkolaborasi dan berkerja sama dalam suatu proyek. Salah satu fitur Bitbucket yang digunakan dalam *review code* adalah *comment*. Dengan menggunakan *comment* developer dimungkinkan untuk membuat saling berkomunikasi langsung pada masalah yang ditemukan dalam code pada proses *code review.*

- Quality Code
  - SonarQube dan SonarLint

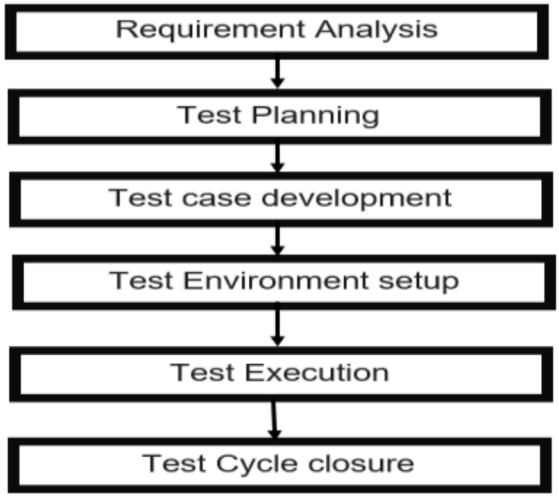
Gambar 3.3 Logo Sonar

(Sumber : https://www.sonarsource.com)

SonaQube adalah sebuah alat yang digunakan untuk memastikan kualitas kode tetap memenuhi standar yang telah ditentukan. Dengan menggunakan SonarQube, *code reviewer* akan terbantu untuk memvalidasi kualitas code. Untuk

pengechekan kualitas code dalam mode statis digunakan SonarLint. SonarQube dan SonarLint dapat dihubungkan untuk berbagi aturan dan standar kualitas kode.

Setelah dilakukan *code review,* praktikan melakukan pengujian terhadap kode tersebut. Pengujian dalam produk IAM menggunakan metode STLC *(Software Testing Life Cycle)*. STLC merupakan proses verifikasi sebuah software dalam memenuhi ekspektasi atau kebutuhan. STLC memiliki enam fase dalam proses nya, yaitu: analisis kebutuhan, perencanaan pengujian, pengembangan kasus uji, pengaturan lingkungan pengujian, pelaksanaan pengujian, dan penutupan pengujian. STLC pada sistem informasi IAM ini digambarkan dalam gambar diagram berikut:



Gambar 3.4 Diagram *Software Test Life Cycle*
(Sumber: Taley., 2020)

Fase pertama dari STLC dilaksanakan pada minggu – minggu terakhir suatu *sprint* yang sedang berjalan. Dengan demikan, dapat dipastikan bahwa modul atau fitur yang dikembangkan dalam satu *sprint* telah melalui tahap pengujian dan telah dipastikan memenuhi kebutuhan yang tercantum dalam dokumen fungsional sebelum dilakukan perilisan atau serah terima kepada *client* perusahaan.

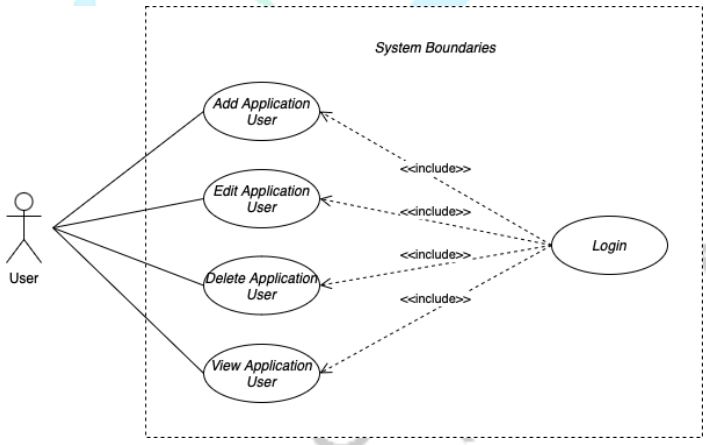
3.2.1 Analisis Kebutuhan (*Requirement Analysis*)

Pada fase analisis kebutuhan tim penguji dan tim QA (*Quality Assurance)* menganalisis dan mempelajari seluruh dokumen kebutuhan (Taley, 2020). Pada fase ini praktikan mempelajari dokumen *Functional Spesification Document (FSD)*. *Functional Spesification Document (FSD)* adalah dokumen yang mengdeskripsikan fungsi dan komponen didalam sistem (ISO/IEC/IEEE, 2010). Dokumen *Functional Spesification* ini membantu pengembang dalam memahami kebutuhan dan apa saja yang akan dikembangkan dalam sistem.

Berdasarkan analisis dokumen fungsional, terdapat beberapa elemen yang perlu diperhatikan sebelum melakukan perencanaan pengujian. Elemen – elemen tersebut adalah sebagai berikut.

3.2.1.1. *Use Case*

Use Case bertujuan untuk menggambarkan ringkasan mengenai kebutuhan pengguna didalam sistem. Diagram *Use Case* merupakan representasi visual yang digunakan untuk menunjukan interaksi pengguna atau aktor dengan sistem. Adapun *Use Case* pada *website* IAM adalah sebagai berikut.



Gambar 3.5 *Use Case* IAM Pada *Application User*

(Sumber : Dokumen perusahaan)

Pada Gambar 3.5 ditunjukkan empat *use case* utama, yaitu *Add Application User*, *Edit Application User*, *Delete Application User* dan *View Application User*, yang dibutuhkan oleh seorang pengguna (*user*). Untuk dapat menjalankan *use case* tersebut, pengguna harus terlebih dahulu melakukan *Login*. Dalam diagram use case ini, hubungan antara *Add Application User*, *Edit Application User*, *Delete Application User, View Application User* dan *Login* merupakan hubungan *include*, yang berarti bahwa proses *Login* secara wajib harus dijalankan sebagai bagian dari eksekusi *Add Application User*, *Edit Application User*, *Delete Application User* dan *View Application User*. Dengan kata lain, *Login* bukanlah opsi tambahan, melainkan langkah yang selalu dilakukan setiap kali pengguna ingin menambahkan akun aplikasi baru. Hubungan *include* ini menggambarkan keterkaitan fungsional dimana *Login* menyediakan layanan otentikasi yang menjadi prasyarat penting untuk menjaga keamanan sistem sebelum akses ke fungsi penambahan pengguna diberikan. Hal ini memastikan bahwa hanya pengguna yang telah terverifikasi yang dapat melakukan penambahan aplikasi user, sehingga integritas dan keamanan data tetap terjaga. Pada deskirpsi *use case,* dapat mencakup semua detail yang diperlukan untuk pembuatan diagram *use case.* Dengan membuat deskripsi *use case.* Pengguna dapat mengetahu secara rinci tentang setiap *use case* yang diperlukan. Adapun deskripsi *use case* dari gambar 3.5 adalah sebagai berikut.

Tabel 3.1 ***Use Case Description Add Application User***

Use Case Name	Add Application User	**ID**	UC1	**Importance Level**	High
Primary Actor	User			**Use Case Type**	Essential
Stakeholder and Interest :					
User	Dapat menambah pengguna aplikasi yang terhubung				

Brief Description :	
Dalam use case ini dijabarkan bagaimana user dapat menambahkan user aplikasi yang terhubung dengan sistem IAM	
Trigger :	
Saat user klik tombol Add Application User	
Preconditions :	
1) User sudah login 2) Memiliki akses terhadap add application user	
Normal Flow :	
Actor Actions	System Responses
1) User klik tombol Add Application User.	1) Sistem menampilkan popup pengisian form Add Application user.
2) User melakukan pengisian form Add Application User.	2) Sistem mevalidasi inputan user pada form Add Application User.
3) User klik tombol Submit pada form	3) Sistem memproses data yang telah diinputkan dan menampilkan konfirmasi bahwa data telah berhasil ditambahkan.
4) User menerima konfirmasi sukses	4) Sistem menutup popup dan memperbarui tampilan daftar pengguna yang ada dengan data yang baru.
Alternative Flow :	
Actor Actions	System Responses
Exceptional Flow :	
1E) Jika terjadi kegagalan dalam menyimpan user aplikasi, maka sistem akan menampilkan pesan error dan pesan kegagalannya. 2E) Jika terjadi kegagalan dalam menampilkan pilihan aplikasi, maka sistem akan menampilkan pesan error dan pesan kegagalannya.	

Tabel 3.2 ***Use Case Description Edit Application User***

Use Case Name	Edit Application User	**ID**	UC1	**Importance Level**	High
Primary Actor	User			**Use Case Type**	Essential
Stakeholder and Interest :					
User	Dapat merubah pengguna aplikasi yang terhubung				
Brief Description :					
Dalam use case ini dijabarkan bagaimana user dapat merubah user aplikasi yang terhubung dengan sistem IAM					
Trigger :					
Saat user klik tombol Edit Application User					
Preconditions :					
1) User sudah login 2) Memiliki akses terhadap edit application user					
Normal Flow :					
Actor Actions			System Responses		
1) User klik tombol Edit Application User.			1) Sistem menampilkan popup pengisian form Edit Application user.		
2) User melakukan pengisian form Edit Application User.			2) Sistem mevalidasi inputan user pada form Edit Application User.		
3) User klik tombol Submit pada form			3) Sistem memproses data yang telah diinputkan dan menampilkan konfirmasi bahwa data telah berhasil dirubah.		
4) User menerima konfirmasi sukses			4) Sistem menutup popup dan memperbarui tampilan daftar pengguna yang ada dengan data yang baru.		
Alternative Flow :					
Actor Actions			System Responses		

Exceptional Flow :
1E) Jika terjadi kegagalan dalam merubah user aplikasi, maka sistem akan menampilkan pesan error dan pesan kegagalannya. 2E) Jika terjadi kegagalan dalam menampilkan pilihan aplikasi, maka sistem akan menampilkan pesan error dan pesan kegagalannya.

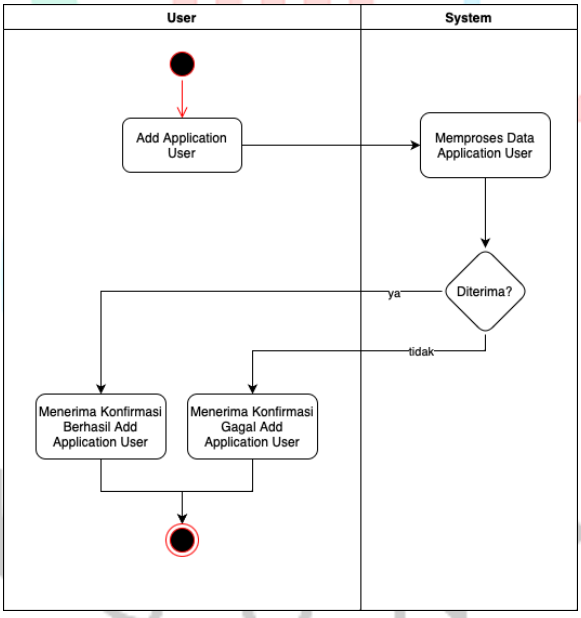
Tabel 3.3 ***Use Case Description Delete Application User***

Use Case Name	delete Application User	**ID**	UC1	**Importance Level**	High
Primary Actor	User			**Use Case Type**	Essential
Stakeholder and Interest :					
User	Dapat menghapus pengguna aplikasi yang terhubung				
Brief Description :					
Dalam use case ini dijabarkan bagaimana user dapat menghapus user aplikasi yang terhubung dengan sistem IAM					
Trigger :					
Saat user klik tombol Delete Application User					
Preconditions :					
1) User sudah login 2) Memiliki akses terhadap delete application user					
Normal Flow :					
Actor Actions			System Responses		
1) User klik tombol Delete Application User.			1) Sistem menampilkan popup konfirmasi data detail user yang akan dihapus.		
2) User klik tombol confirm pada popup delete.			2) Sistem mevalidasi request delete user. 3) Sistem memproses data yang telah dihapus dan menampilkan konfirmasi bahwa data telah berhasil dihapus.		

3) User menerima konfirmasi sukses hapus user.	4) Sistem menutup popup dan memperbarui tampilan daftar pengguna yang ada dengan data yang baru.
Alternative Flow :	
Actor Actions	System Responses
Exceptional Flow :	
1E) Jika terjadi kegagalan dalam menghapus user aplikasi, maka sistem akan menampilkan pesan error dan pesan kegagalannya. 2E) Jika terjadi kegagalan dalam menampilkan pilihan aplikasi, maka sistem akan menampilkan pesan error dan pesan kegagalannya.	

3.2.1.2. *Activity Diagram*

Activity diagram digunakan untuk merepresentasikan proses bisnis didalam sistem. *Activity diagram* adalah penggambaran alur bisnis pada sistem informasi. *Activity diagram* ini dibuat berdasarkan *use case* diagram dan deskripsi *use case* yang telah diuraikan sebelumnya.

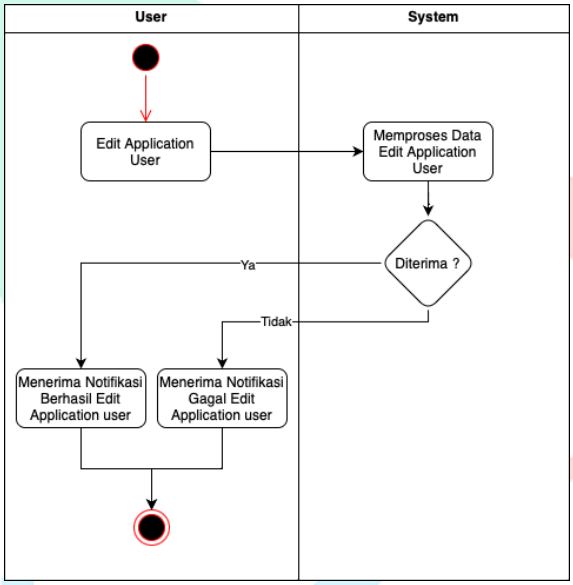


Gambar 3.6 *Activity Diagram Add Application User*

(Sumber : Dokumen perusahaan)

Berikut adalah penjelasain dari *Activity Diagram* pada gambar 3.6.

1. User menambah user aplikasi
2. Sistem memproses data aplikasi user
3. Sistem melakukan penerimaan data user
    - Jika sistem menerima data user maka user akan menerima konfirmasi berhasil menambahkan data user aplikasi.
    - Jika sistem tidak menerima data user atau gagal karena suatu kesalahan maka user akan menerima konfirmasi kegagalan menambahkan data user aplikasi
4. Seluruh proses selesai dan aktivitas selesai dilakukan.



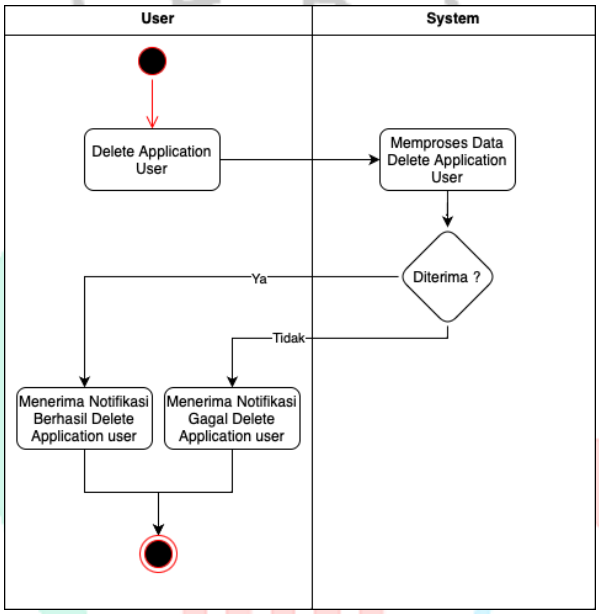
Gambar 3.7 *Activity Diagram Edit Application User*
(Sumber : Dokumen perusahaan)

Berikut adalah penjelasain dari *Activity Diagram* pada gambar 3.7.

1. User merubah user aplikasi
2. Sistem memproses data aplikasi user
3. Sistem melakukan penerimaan data user
    - Jika sistem menerima data user maka user akan menerima konfirmasi berhasil merubah data user aplikasi.

- Jika sistem tidak menerima data user atau gagal karena suatu kesalahan maka user akan menerima konfirmasi kegagalan merubah data user aplikasi

4. Seluruh proses selesai dan aktivitas selesai dilakukan.



Gambar 3.8 *Activity Diagram Delete Application User*
(Sumber : Dokumen perusahaan)

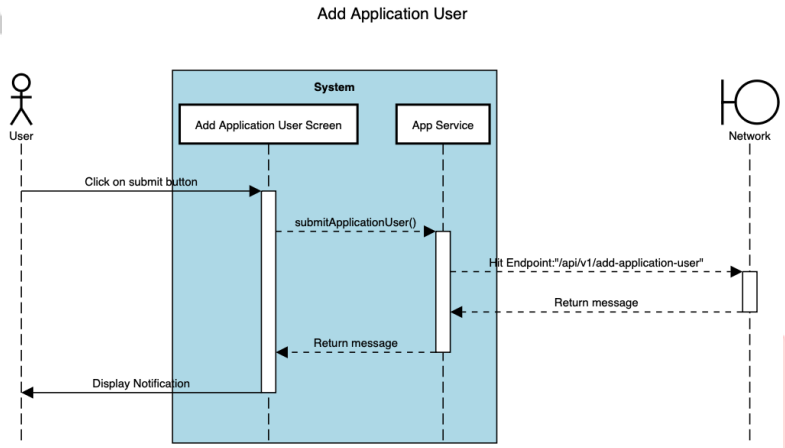
Berikut adalah penjelasain dari *Activity Diagram* pada gambar 3.8.

1. User menghapus user aplikasi
2. Sistem memproses data hapus user aplikasi
3. Sistem melakukan penerimaan data hapus user
   - Jika sistem menerima data hapus user maka user akan menerima konfirmasi berhasil menghapus data user aplikasi.
   - Jika sistem tidak menerima data user atau gagal karena suatu kesalahan maka user akan menerima konfirmasi kegagalan menghapus data user aplikasi
4. Seluruh proses selesai dan aktivitas selesai dilakukan.

3.2.1.3. *Sequence Diagram*

Sequence Diagram adalah salah satu jenis diagram interaksi. Diagram ini menggambarkan urutan pesan yang dikirim kan antar objek – objek selama interaksi. Fokus utama dari *sequence diagram* adalah untuk mengilustrasikan urutan aktivitas pada objek. Hal ini sangat bermanfaat untuk memahami spesifikasi sistem dan alur sistem. Contoh *sequnce diagram* pada sistem informasi IAM adalah sebagai berikut.

- Alur *Add Application User*



Gambar 3.9 *Sequence Diagram Add Application User*

Gambar 3.9 menggambarkan interaksi antara aktor User, komponen-komponen sistem seperti Add Application User Screen dan App Service, serta entitas eksternal Network selama proses penambahan pengguna aplikasi (*Add Application User*).

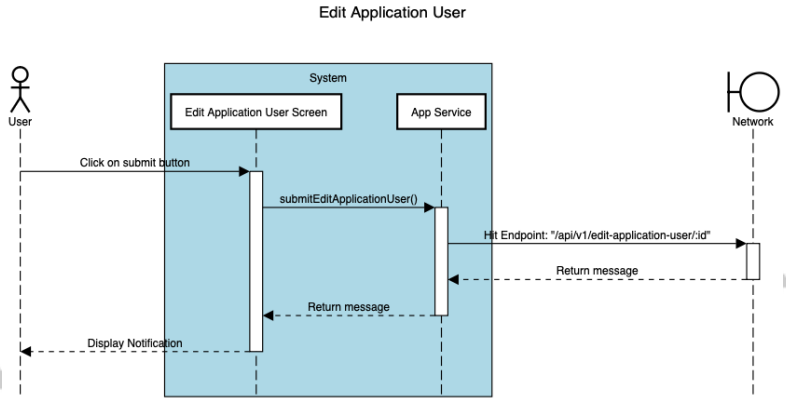
1. User memulai proses dengan mengklik tombol submit pada tampilan Add Application User Screen. Tindakan ini ditandai dengan panah pesan "Click on submit button" yang dikirim dari User ke Add Application User Screen.
2. Setelah menerima perintah dari pengguna, Add Application User Screen memanggil metode submitApplicationUser() pada App Service untuk memproses permintaan penambahan pengguna aplikasi. langkah ini merupakan pesan sinkron yang

ditandai dengan panah dari Add Application User Screen ke App Service.

3. App Service kemudian melakukan permintaan ke endpoint API eksternal melalui Network dengan mengirimkan pesan ke "api/v1/add-application-user". langkah Ini merupakan pemanggilan layanan backend yang melibatkan komunikasi jaringan, ditunjukkan dengan panah dari App Service ke Network.
4. Setelah server di jaringan memproses permintaan tersebut, Network mengirimkan kembali pesan balasan (*return message*) ke App Service. langkah Ini menandakan bahwa proses penambahan pengguna telah selesai dan hasilnya dikirimkan kembali ke sistem.
5. App Service meneruskan pesan balasan tersebut ke Add Application User Screen sebagai konfirmasi hasil operasi. Pesan ini ditandai dengan panah kembali dari App Service ke Add Application User Screen.
6. Terakhir, Add Application User Screen menampilkan notifikasi kepada User berupa hasil proses penambahan pengguna, baik berupa pesan sukses ataupun kegagalan. Ditandai dengan panah dari Add Application User Screen ke User dengan pesan "Display Notification".

*D*iagram Add Application User ini menjelaskan alur komunikasi antara pengguna dengan antarmuka aplikasi, bagaimana aplikasi mengirim permintaan tambah user aplikasi ke layanan backend melalui jaringan, dan bagaimana respons dikembalikan ke pengguna. Proses ini memastikan bahwa pengguna mendapatkan umpan balik secara real-time mengenai status penambahan pengguna aplikasi.

- Alur *Edit Application User*



Gambar 3.10 *Sequence Diagram Edit Application User*

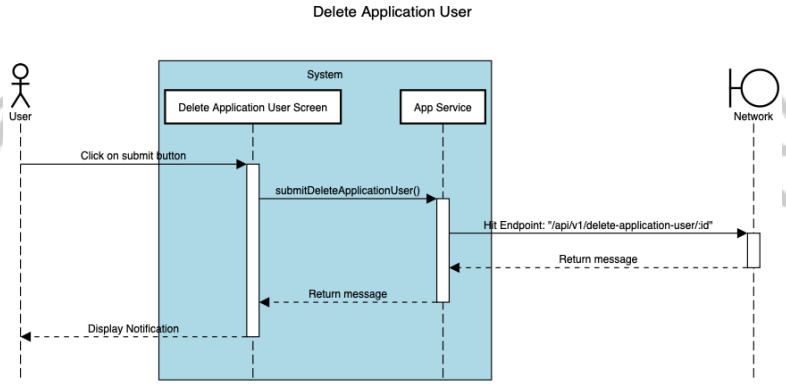
Gambar 3.10 menggambarkan interaksi antara aktor User, komponen-komponen sistem seperti Edit Application User Screen dan App Service, serta entitas eksternal Network selama proses perubahan pengguna aplikasi (*Edit Application User*).

1. User memulai proses dengan mengklik tombol *Submit* pada tampilan *Edit Application User Screen*. Tindakan ini ditandai dengan panah pesan "Click on submit button" yang dikirim dari *User* ke *Edit Application User Screen*.
2. Setelah menerima perintah dari pengguna, *Edit Application User Screen* memanggil metode submitEditApplicationUser() pada *App Service* untuk memproses permintaan peruabahan pengguna aplikasi. langkah ini merupakan pesan sinkron yang ditandai dengan panah dari *Edit Application User Screen* ke *App Service*.
3. *App Service* kemudian melakukan permintaan ke *endpoint API eksternal* melalui *Network* dengan mengirimkan pesan ke "api/v1/edit-application-user/:id". langkah Ini merupakan pemanggilan layanan *backend* yang melibatkan komunikasi jaringan, ditunjukkan dengan panah dari *App Service* ke *Network*.

4. Setelah server di jaringan memproses permintaan tersebut, Network mengirimkan kembali pesan balasan (*return message*) ke *App Service*. langkah Ini menandakan bahwa proses penambahan pengguna telah selesai dan hasilnya dikirimkan kembali ke sistem.
5. App Service meneruskan pesan balasan tersebut ke *Edit Application User Screen* sebagai konfirmasi hasil operasi. Pesan ini ditandai dengan panah kembali dari *App Service* ke *Edit Application User Screen*.
6. Terakhir, *Edit Application User Screen* menampilkan notifikasi kepada *User* berupa hasil proses perubahaan pengguna, baik berupa pesan sukses ataupun kegagalan. Ditandai dengan panah dari *Edit Application User Screen* ke *User* dengan pesan "Display Notification".

*D*iagram *Edit Application User* ini menjelaskan alur komunikasi antara pengguna dengan antarmuka aplikasi, bagaimana aplikasi mengirim permintaan merubah pengguna aplikasi ke layanan *backend* melalui jaringan, dan bagaimana respons dikembalikan ke pengguna. Proses ini memastikan bahwa pengguna mendapatkan umpan balik secara real-time mengenai status perubahan pengguna aplikasi.

- Alur *Delete Application User*



Gambar 3.11 *Sequence Diagram Delete Application User*

Gambar 3.11 menggambarkan interaksi antara aktor *User*, komponen-komponen sistem seperti *Delete Application User Screen* dan *App Service*, serta entitas *eksternal Network* selama proses penghapusan pengguna aplikasi (*Delete Application User*).

1. User memulai proses dengan mengklik tombol submit pada tampilan *Delete Application User Screen*. Tindakan ini ditandai dengan panah pesan "Click on submit button" yang dikirim dari *User* ke *Delete Application User Screen*.
2. Setelah menerima perintah dari pengguna, *Delete Application User Screen* memanggil metode submitDeleteApplicationUser() pada *App Service* untuk memproses permintaan penghapusan pengguna aplikasi. langkah ini merupakan pesan sinkron yang ditandai dengan panah dari *Delete Application User Screen* ke *App Service*.
3. *App Service* kemudian melakukan permintaan ke *endpoint API eksternal* melalui *Network* dengan mengirimkan pesan ke "api/v1/delete-application-user/:id". langkah Ini merupakan pemanggilan layanan backend yang melibatkan komunikasi jaringan, ditunjukkan dengan panah dari *App Service* ke *Network*.
4. Setelah server di jaringan memproses permintaan tersebut, *Network* mengirimkan kembali pesan balasan (*return message*) ke *App Service*. langkah Ini menandakan bahwa proses penghapusan pengguna telah selesai dan hasilnya dikirimkan kembali ke sistem.
5. *App Service* meneruskan pesan balasan tersebut ke *Delete Application User Screen* sebagai konfirmasi hasil operasi. Pesan ini ditandai dengan panah kembali dari *App Service* ke *Delete Application User Screen*.
6. Terakhir, *Delete Application User Screen* menampilkan notifikasi kepada *User* berupa hasil proses penghapusan pengguna, baik berupa pesan sukses

ataupun kegagalan. Ditandai dengan panah dari *Delete Application User Screen* ke *User* dengan pesan "Display Notification".

*D*iagram *Delete Application User* ini menjelaskan alur komunikasi antara pengguna dengan antarmuka aplikasi, bagaimana aplikasi mengirim permintaan penghapusan user aplikasi ke layanan backend melalui jaringan, dan bagaimana respons dikembalikan ke pengguna. Proses ini memastikan bahwa pengguna mendapatkan umpan balik secara real-time mengenai status penghapusan pengguna aplikasi.

3.2.1.4. *Requirement Traceability Matrix*

Requirement Traceability Matrix (RTM) membantu pengembang dalam memanage dan mengkontrol proses pengembangan dan evaluasi dari sistem informasi (Cleland-huang, 2012). RTM memungkinkan penelusuran setiap persyaratan dan kebutuhan yang ada baik bisnis maupun pengujian. Format RTM dapat disesuaikan dengan kebutuhan proyek. Tujuan utama dari RTM adalah agar anggota proyek dapat mengetahui apa saja persayaratan yang harus dipenuhi. Adapun matriks penelusuran persyaratan yang dihasilkan dari hasil analisis dokumen yang ada adalah sebagai berikut.

Tabel 3.4 *Requirement Traceability Matrix Application User*

REQUIREMENT TRACEABILITY MATRIX				
Project Name	IAM			
Created By	Theofilus C.K.			
Feature	Application User			
Requirement ID	Requirement Description	Test Case ID	Test Case Description	Expected Results
0001	Add Application User	0001/TC/0001	Login sebagai user IAM Pilih menu application user	User dapat menambahkan application user

			Pilih tombol add application user Muncul popup form add application user Isi form setelah itu pilih application Klik Submit Muncul notifikasi error atau berhasil dan popup add application user tertutup otomatis	
0002	Edit Application User	0002/TC/0001	Login sebagai user IAM Pilih menu application user Pilih tombol edeit application user Muncul popup form edit application user Klik Submit Muncul notifikasi error atau berhasil dan popup edit application user tertutup otomatis	User dapat merubah application user
0003	Delete Application User	0003/TC/0001	Login sebagai user IAM	User dapat menghapus

			application user
		Pilih menu application user Pilih tombol delete application user Muncul popup form delete application user Klik Submit	
		Muncul notifikasi error atau berhasil dan popup delete application user tertutup otomatis	

3.2.2 Perencanaan Pengujian (*Test Planning*)

Perencanaan pengujian adalah fase dimana *test plan* dan *test strategy* ditentukan (Taley, 2020). Aktivitas yang berlangsung pada fase ini adalah sebagai berikut.

a. Mengidentifikasikan tujuan dan ruang linkup pengujian

Ruang lingkup pengujian difokuskan pada fitur *Add Application User* yang terdapat dalam sistem informasi IAM. Fitur ini memungkinkan pengguna untuk menambahkan *user* baru ke dalam sistem melalui antarmuka aplikasi. Pengujian dilakukan secara terbatas pada level *unit testing* pada sisi antarmuka pengguna (UI). Pengujian difokuskan untuk memastikan bahwa setiap komponen UI seperti tombol, dan elemen interaktif lainnya, telah muncul dan berfungsi sesuai dengan ekspektasi. Dengan cakupan ini, praktikan dapat memastikan bahwa struktur dan perilaku antarmuka untuk fitur *Add Application User* telah sesuai dengan dokumen fungsional yang telah ditentukan.

b. Mengembangkan strategi pengujian

Dalam mengembangkan *website IAM,* praktikan mendapatkan tugas untuk melakukan pengujian *White-box. White-box testing* sendiri adalah pengujian yang menggunakan kode sumber atau kode biner dari progam yang diuji untuk menghasilkan data uji dan mengamati hasilnya (Honfi & Micskei, 2020). *White-box testing* sendiri dapat diterapkan pada beberapa tingkat pengujian seperti unit, integrasi, dan sistem. Dalam tahap ini, praktikan diberi tanggung jawab untuk menerapkan pengujian pada tingkat unit. Pengujian pada tingkat unit memungkinkan praktikan untuk menemukan kesalahan dan ketidaksesuaian pada kode sumber yang sedang diuji, sehingga memungkinkan untuk dilakukan perbaikan kesalahan lebih efisien.

White-box testing memiliki beberapa metode yang dapat digunakan pada pengujian perangkat lunak. Adapun metode – metode *white-box testing* adalah sebagai berikut (Nidhra & Dondeti, 2012).

1. *Control Flow Testing*

   Control Flow Testing adalah metode pengujian yang menggunakan alur kontrol program sebagai modelnya.

2. *Branch Testing*

   Branch Testing adalah pengujian yang memastikan setiap jalur atau cabang kode diuji setidaknya satu kali.

3. *Basis Path Testing*

   Basis Path Testing adalah pengujian yang bertujuan untuk mengukur kompleksitas kode program dan mengdefinisikan alur yang dieksekusi.

4. *Data Flow Testing*

   Data Flow Testing adalah pengujian yang bertujuan untuk melacak aliran variable pada progam. *Data Flow Testing* juga memastikan variabel dideklarasikan dengan benar dan digunakan di tempat yang tepat.

5. *Loop Testing*

Loop Testing adalah pengujian yang bertujuan untuk menguji dan memastikan bahwa perulangan pada program dapat berjalan dengan benar,sehingga tidak mengakibatkan kesalahan pada program.

Praktikan diberikan tugas untuk menggunakan metode *branch testing* dan *control flow testing*. Kedua metode ini mencakup pengujian terhadap kondisi logika serta alur program secara keseluruhan. *Control flow testing* bertujuan untuk memastikan bahwa setiap alur eksekusi dalam kode telah diuji, sehingga meningkatkan kemampuan dalam mendeteksi kesalahan logika. Sementara itu, *branch testing* memastikan bahwa seluruh kondisi percabangan dalam pernyataan kode telah diuji, sehingga membantu mengidentifikasi potensi kesalahan dalam pengambilan keputusan logis program.

Kedua teknik tersebut, *control flow testing* dan *branch testing,* memberikan keseimbangan yang baik antara cakupan pengujian dan tingkat kompleksitas, sehingga sangat sesuai untuk diterapkan dalam tahap pengembangan dan pengujian pada sistem informasi IAM. Selain itu, banyak alat bantu pengujian otomatis yang mendukung kedua metode ini, sehingga memudahkan praktikan dalam proses implementasi pengujian. Oleh karena itu, penggunaan *control flow testing* dan *branch testing* menjadi pilihan yang tepat dalam pelaksanaan *unit testing* pada sistem informasi IAM.

Adapun alat yang digunakan pada proses implementasi *white-box testing* pada sistem informasi IAM adalah sebagai berikut.

6. *Testing Framework* dan *Code Coverage*

Website sistem informasi IAM menggunakan Jest sebagai *testing framework* dan *code coverage*. Jest adalah *framework* pengujian javascript yang dirancang khusus untuk aplikasi berbasis React. Didalam Jest juga terdapat *code coverage* yang

berguna untuk memastikan bahwa seluruh *function, branch ,* dan *statement* telah diuji.

Gambar 3.12 Logo Jest

(Sumber: https://iconduck.com/icons/27545/jest)

c. Mengidentifikasikan lingkungan pengujian dan sumber daya yang dibutuhkan

Mengidentifikasikan lingkungan dan sumber daya pengujian pada *website* IAM digunakan untuk memastikan pengujian berjalan lancar. Berikut adalah lingkungan pengujian dan sumber daya yang dibutuhkan.

1. Perangkat Keras *(Hardware)*

   Perangkat keras yang dibutuhkan meliputi komputer personal yang digunakan untuk membuat dan menjalankan pengujian *website.*

2. Perangkat Lunak *(Software)*

   Perangkat lunak yang dibutuhkan adalah Webstrom sebagai *Integrated Development Environtment (IDE)* untuk pengembangan *unit test website* IAM.

3.2.3 Pengembangan Kasus Pengujian (*Test Case Development*)

Fase pengembangan kasus uji dalam STLC adalah fase pengembangan skenario pengujian yang digunakan untuk memverifikasi bahwa setiap fungsi dalam sistem berjalan sesuai kebutuhan yang telah ditentukan. Setiap kasus uji mencakup juga *unit test* yang dirancang untuk mengevaluasi kode. Unit test akan memeriksa setiap komponen untuk memastikan bahwa setiap bagian berfungis dengan baik dan sesuai harapan.

Penguji juga harus menentukan data pengujian yang akan digunakan selama proses pengujian. Data pengujian harus mencakup berbagai skenario pengguna yang mungkin akan terjadi nantinya. Penggunaan data pengujian yang sesuai disetiap skenarionya dapat meningkatkan kemungkinan penemuan bug dan anomali sejak dini. Setelah proses pengembangan kasus pengujian selesai, penguji harus mendokumentasikan kasus uji yang telah dibuat dalam bentuk *Requirement Traceability Matrix(RTM)*. RTM ini berfungsi untuk melacak kebutuhan – kebutuhan yang diuji dan memastikan bahwa semua kebutuhan telah diuji, sehingga dapat menjadi jaminan sistem telah memenuhi standar kualitas yang ditentukan. Hasil test case yang telah dikembangkan adalah sebagai berikut.

a. *Data Layer*

Data Layer adalah bagian *website* yang dimana digunakan untuk mengelola data dari API. Pengujian pada layer ini bertujuan untuk memastikan operasi dan interaksi dengan sumber eksternal sesuai dan berfungsi dengan benar. Adapun *Requirement Traceability Matrix* pada *layer* terkait Add Application User sebagai berikut.

Tabel 3.5 Tabel *Requirement Traceable Matrix Add Application User*

REQUIREMENT TRACEABILITY MATRIX			
Project Name	IAM		
Created By	Theofilus C.K.		
Feature	Add Application User		
Layer	Data		
Sub Layer	Store / Repository		
Requirement ID	Requirement Description	Test Case ID	Test Case Description

0001/UNT/001	Add Application User	TC/STR/0001	When setFormPayload() should store form data
		TC/STR/0002	When resetFormPayload() should remove form data

Requirement traceability matrix akan dijadikan sebagai tolak ukur tercapainya suatu *unit test*. Adapun *unit test* terkait adalah sebagai berikut.

```
it('should set formPayload', () : void => {
  const { result } = renderHook(() : ApplicationUserManagementSt.... =>
    useApplicationUserManagementStore());
  const formPayload: TApplicationUserManagementResponse = {
    userId: '1',
    id: '1',
    version: 1,
    createdDate: new Date(),
    createdBy: 'user1',
    updatedDate: new Date(),
    updatedBy: 'user2',
    status: 'active',
    details: null,
    name: 'John Doe',
    email: 'john.doe@example.com',
    nik: '1234567890',
    mobilePhone: '123-456-7890',
    application: [],
    passwordExpirationDate: new Date(),
    delete: false,
  };

  act(() : void => {
    result.current.setFormPayload(formPayload);
  });

  expect(result.current.formPayload).toEqual(formPayload);
});
```

Gambar 3.13 *Unit Test setFormPayload*

Gambar 3.13 menunjukkan sebuah unit test pada data layer yang terkait dengan TC/STR/0001 pada Tabel 3.3. Unit test ini menguji fungsi setFormPayload, yang bertugas menyimpan data yang akan dikirim ke Backend.

Unit test tersebut memiliki ekspektasi tertentu yang menjadi acuan untuk menentukan apakah pengujian berhasil atau gagal. Jika ekspektasi tersebut tidak terpenuhi, maka unit test dianggap gagal, dan kode perlu diperbaiki agar sesuai dengan ekspektasi yang telah ditentukan.

b. Domain Layer

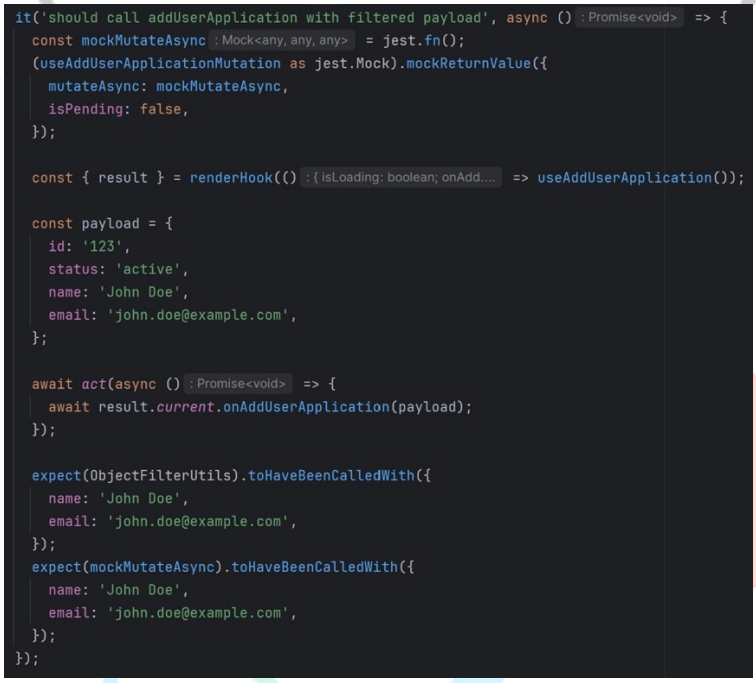
Domain Layer adalah bagian sistem yang mengkontrol proses – proses yang terjadi didalam sistem. Pada *Website* IAM, *Domain Layer* adalah *function action* atau *controller* yang berisi fungsi – fungsi untuk melakukan aksi ataupun pengelolaan data sebelum di simpan pada *Data Layer*. Adapun *Requirement Traceability Matrix* pada *Domain Layer* terkait *Add Application User* adalah sebagai berikut.

Tabel 3.6 Tabel *Requirement Traceable Matrix Add Application User*

REQUIREMENT TRACEABILITY MATRIX			
Project Name	IAM		
Created By	Theofilus C.K.		
Feature	Add Application User		
Layer	Domain / Controller		
Requirement ID	Requirement Description	Test Case ID	Test Case Description
0001/UNT/001	Add Application User	TC/CO/0001	When onFinished() should Fetch API add application user with expected payload
		TC/CO/0002	When onCancel() should execute resetFormPayload and

close popup add
application user

Requirement traceability matrix akan dijadikan sebagai tolak ukur tercapainya suatu *unit test*. Adapun *unit test* terkait adalah sebagai berikut.



```
it('should call addUserApplication with filtered payload', async () : Promise<void> => {
  const mockMutateAsync : Mock<any, any, any> = jest.fn();
  (useAddUserApplicationMutation as jest.Mock).mockReturnValue({
    mutateAsync: mockMutateAsync,
    isPending: false,
  });

  const { result } = renderHook(() : { isLoading: boolean; onAdd.... => useAddUserApplication());

  const payload = {
    id: '123',
    status: 'active',
    name: 'John Doe',
    email: 'john.doe@example.com',
  };

  await act(async () : Promise<void> => {
    await result.current.onAddUserApplication(payload);
  });

  expect(ObjectFilterUtils).toHaveBeenCalledWith({
    name: 'John Doe',
    email: 'john.doe@example.com',
  });
  expect(mockMutateAsync).toHaveBeenCalledWith({
    name: 'John Doe',
    email: 'john.doe@example.com',
  });
});
```

Gambar 3.14 *Unit Test Call API*

Gambar 3.14 menunjukan *unit test* memiliki 2 ekspektasi keberhasilan *Unit Test* yang mewakili TC/CO/001 pada Tabel 3.4. Fungsi ObjectFilterUtils diberikan syarat berhasil jika dipanggil dengan *payload* yang sesuai.

c. *Presentation layer*

Presentation layer adalah bagian sistem yang bertanggung jawab atas interaksi antarmuka dengan pengguna pada sistem informasi. Pada sistem informasi IAM ini, *Presentation layer* adalah *View model.* Pengujian

ini berguna untuk memastikan alur dan logika antarmuka benar dan sesuai kebutuhan. Adapun *Requirement Traceability Matrix* pada *Presentation Layer* terkait *Add Application User* adalah sebagai berikut.

Tabel 3.7 Tabel *Requirement Traceable Matrix Add Application User*

REQUIREMENT TRACEABILITY MATRIX			
Project Name	IAM		
Created By	Theofilus C.K.		
Feature	Add Application User		
Layer	Presentation		
Requirement ID	Requirement Description	Test Case ID	Test Case Description
0001/UNT/001	Add Application User	TC/VM/0001	When onClick() should open popup add application user
		TC/VM/0002	When button submit clicked should trigger onFinished()
		TC/VM/0003	When onMessage() should show notification element.

Requirement traceability matrix akan dijadikan sebagai tolak ukur tercapainya suatu unit test. Adapun unit test terkait adalah sebagai berikut.

1. Unit Test TC/VM/0001 – Popup Add Application User

   Memastikan bahwa saat fungsi onClick() sistem menampilkan popup untuk menambahkan pengguna aplikasi. Unit Test

2. TC/VM/0002 – Trigger Submit Action

   Memastikan bahwa saat tombol "Submit" pada popup ditekan, fungsi onFinished() berhasil dipanggil untuk memproses data input.

3. Unit Test TC/VM/0003 – Menampilkan Notifikasi

   Memastikan bahwa setelah proses onFinished() dijalankan, fungsi onMessage() memunculkan notifikasi kepada pengguna.

Dengan adanya unit test ini, pengujian pada layer presentasi menjadi lebih terstruktur dan terdokumentasi dengan baik. Pengujian ini juga menjamin bahwa komponen tampilan dapat merespons interaksi pengguna dengan benar, serta memastikan alur kerja antarmuka sesuai dengan kebutuhan fungsional yang telah ditentukan dalam requirement.

```
it('renders the form', () : void => {
  render(
    <Router>
      <ApplicationUserManagementForm
        isEdit={false}
        isLoading={false}
        onSubmit={mockOnSubmit}
      />
    </Router>,
  );

  expect(
    screen.getByLabelText('applicationUserManagement.label.fullName'),
  ).toBeInTheDocument();
  expect(
    screen.getByLabelText('applicationUserManagement.label.email'),
  ).toBeInTheDocument();
  expect(
    screen.getByLabelText('applicationUserManagement.label.employeeNo'),
  ).toBeInTheDocument();
  expect(
    screen.getByLabelText('applicationUserManagement.label.phoneNo'),
  ).toBeInTheDocument();
  expect(
    screen.getByLabelText('applicationUserManagement.label.userId'),
  ).toBeInTheDocument();
});
```

Gambar 3.15 *Unit Test Render Form Add Application User*

Gambar 3.15 menunjukan *unit test* yang disesuaikan untuk TC/VM/0001 pada Tabel 3.5, *unit test* menunjukan bahwa terdapat beberapa ekspektasi keberhasilan dimana *form* penambahan user muncul ketika popup tambah user muncul.

3.2.4 Pengaturan lingkungan pengujian (*Test Enviroment Setup*)

Dalam pengembangan *website* IAM, pengujian dilakukan dalam beberapa lingkungan. Dilakukan pengujian dalam beberapa lingkungan berguna untuk memastikan bahwa *website* IAM telah siap digunakan dalam lingkungan pengguna yang mungkin berbeda – beda. Adapun beberapa hal yang perlu diperhatikan dalam proses pengaturan lingkungan pengujian adalah sebagai berikut.

a. Kebutuhan lingkungan pengujian

Dalam pengembangan *website* IAM terdapat beberapa tools yang digunakan. Maka dari itu dibutuhkan environment pengujian yang sama dan memiliki tool serta versi sebagai berikut.

Tabel 3.8 Kebutuhan Lingkungan Pengujian

Tools	Version
ReactJs	18.2.0
NodeJs	18.20.0
Webstrom	2024.3.1.1
Vite	5.4.11

b. Konfigurasi .env

Vite menyediakan dukungan terhadap file konfigurasi *environment (environment variables)* melalui file .env. Fitur ini memungkinkan *frontend developer* untuk mengelola berbagai konfigurasi berdasarkan lingkungan pengembangan, seperti pengujian, *development* dan *production.* Dengan menggunakan file .env, *frontend*

developer dapat memisahkan nilai – nilai konfigurasi seperti URL API, token akses, dan beberapa *variabel* lainnya berdarsakan kebutuhan setiap lingkungan. Adapun file .env yang digunakan dalam pengembangan proyek ini antara lain :

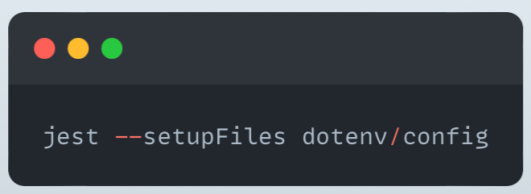
Tabel 3.9 File – File .env

File	Lingkungan
.env.development	*development*
.env.production	*production*
.env.stagging	*stagging*
.env.production.clientName	*specific client`s production environment*

File .env.development memungkinkan *frontend developer* dapat menjalankan dan *build website* dalam konfigurasi lingkungan *development.* File .env.production memungkinkan *frontend developer* dapat menjalankan dan *build website* dalam konfigurasi lingkungan *production* atau versi terilis. File .env.stagging digunakan *frontend developer* untuk menjalankan dan *build website* dalam konfigurasi test sehingga hasil *build* dapat digunakan juga oleh QA ataupun *tester* dalam pengujian lebih lanjut. Sedangkan *file* .env.production.clientName memungkinkan *frontend developer* untuk menjalankan dan *build website* dalam konfigurasi lingkungan *client* perusahaan.

3.2.5 Menjalankan Pengujian (*Test Execution*)

Fase menjalankan pengujian adalah fase dimana test yang telah direncakan dan diimplementaskan dijalankan guna memastikan sistem informasi berjalan sesuai kebutuhan dan test sesuai dengan *test case*. Untuk menjalankan unit test pada proyek React Vite dapat menggunakan perinah sperti dibawah.

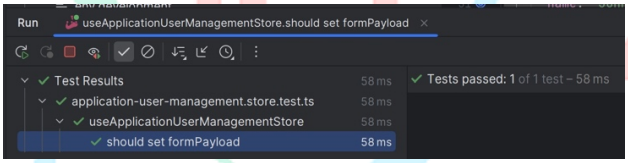


Gambar 3.16 Perintah Menjalankan *Unit Test*

Setelah menjalankan perintah tersebut akan terlihat berapa Unit test yang tereksekusi dan berapa uni test yang berhasil serta gagal. Laporan ini memudahkan penguji untuk mengetahui kecacatan kode dalam suatu *test.* Berikut laporan pengujian tiap layer yang ada.

a. *Data Layer*

Unit test yang telah diimplementasikan pada *data layer* dieksekusi dengan menjalankan perintah seperti gambar 3.16 . Berikut laporan hasil pengujian pada *data layer.*



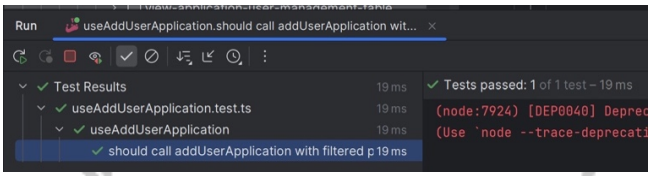
Gambar 3.17 Laporan Hasil *Unit Test Data Layer*

Gambar 3.17 adalah hasil laporan eksekusi *unit test application-user-management.store.test.ts*, dalam *unit test* tersebut terdapat satu unit test yang di eksekusi dengan hasil berhasil*. Unit test* yang berhasil menunjukan bahwa kode yang dibuat telah memenuhi kebutuhan dan *Test Case* yang telah ditentukan.

b. *Domain Layer*

Unit test yang telah diimplementasikan pada *domain layer* dieksekusi dengan menjalankan perintah seperti

gambar 3.18 . Berikut laporan hasil pengujian pada *domain layer.*

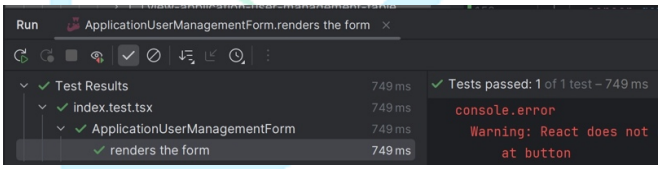


Gambar 3.18 Laporan Hasil *Unit Test Domain Layer*

Gambar 3.18 adalah hasil laporan eksekusi *unit test useAddUserApplication.test.ts*, dalam *unit test* tersebut terdapat satu unit test yang di eksekusi dengan hasil berhasil. *Unit test* yang berhasil menunjukan bahwa kode yang dibuat telah memenuhi kebutuhan dan *Test Case* yang telah ditentukan.

c. *Presentation Layer*

Unit test yang telah diimplementasikan pada *presentation layer* dieksekusi dengan menjalankan perintah seperti gambar 3.19 . Berikut laporan hasil pengujian pada *presentation layer.*



Gambar 3.19 Laporan Hasil *Unit Test Domain Layer*

Gambar 3.19 adalah hasil laporan eksekusi *unit test index.test.ts*, dalam *unit test* tersebut terdapat satu unit test yang di eksekusi dengan hasil berhasil. *Unit test* yang berhasil menunjukan bahwa kode yang dibuat telah memenuhi kebutuhan dan *Test Case* yang telah ditentukan.

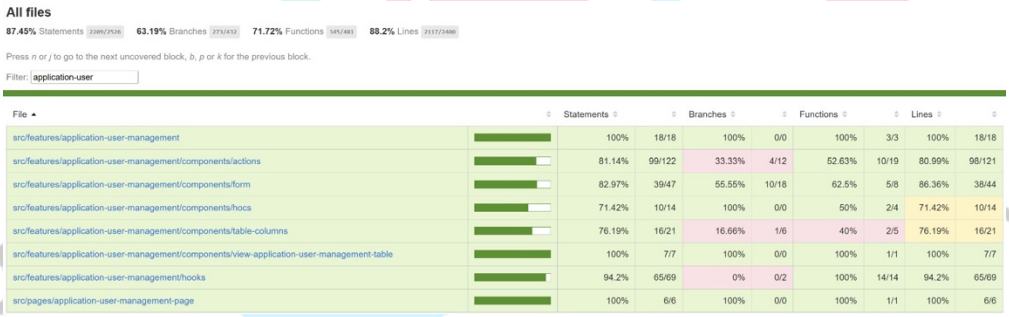
d. *Coverage Test*

Coverage test adalah metrik yang digunakan untuk mengukur sejauh mana kode telah diuji oleh unit test. *Coverage Test* juga menggambarkan berapa persen kode yang telah diuji dari total keseluruhan kode dalam proyek. *Coverage Test* dapat dijalankan dengan perintah dibawah.

```
jest --coverage --setupFiles dotenv/config
```

Gambar 3.20 Kode Eksekusi *Coverage Test Jest*

Setelah menjalankan perintah pada gambar 3.20 , maka Jest akan menghasilkan laporan *coverage test* dengan format yang sudah di tentukan. Pada laporan akan terdapat berapa persen kode yang sudah tercover oleh *unit test*. Berikut adalah laporan dari keseluruhan *coverage test*.



Gambar 3.21 Laporan hasil *coverage test*

Dari laporan hasil *coverage test* menunjukan *data layer, domain layer* dan *presentation layer* seluruhnya masih belum tercakup *unit test,* sehingga ini dapat menjadi catatan oleh *project manager* untuk dapat ditambahkan pada pekerjaan tambahan di *sprint* dan siklus pengujian selanjutnya.

3.2.6 Penutupan Pengujian (*Test Closure)*

Penutupan pengujian (*Test Closure*) merupakan fase terakhir dari fase - fase STLC. Fase ini bertujuan untuk memastikan semua kegiatan pengujian telah selesai. Fase ini juga memastikan semua *bug* dan *defect* telah ditangani dengan tepat dan dokumen – dokumen terkait testing telah diselesaikan. Dalam fase ini terdapat juga evaluasi hasil pengujian terhadap kesesuaian persyaratan proyek.

Dari hasil evaluasi pengujian, menunjukan bahwa terdapat beberapa *code* yang belum tercoverage dan belum memenuhi target. Pada hasil pengembangan pengujian proyek ini masih terdapat *code* yang belum memenuhi standar *threshold percentage coverage code,* ini membuktikan bahwa pengujian yang telah dilakukan belum mencakup seluruh function dan branch yang ada. Hal tersebut dikarenakan beberapa function dan statement serta branch belum tercatat dalam Requirement Traceability Matrix. Dalam siklus pengujian selanjutnya, perlu dilakukan pencatatan function, statement dan branch dari siklus sebelumnya yang belum tercatat, sehingga dapat meningkatkan coverage code dan memenuhi standar *threshold percentage coverage code.*

3.3 Kendala Yang Dihadapi

Selama menjalani kerja praktik, praktikan menghadapi sejumlah tantangan yang cukup kompleks, khususnya dalam proses penyesuaian unit test dengan dokumen fungsional aplikasi. Tantangan ini muncul karena setiap perubahan atau penambahan kode harus dikonfirmasi kesesuaiannya dengan spesifikasi fungsional yang telah dirancang sebelumnya. Proses ini sangat krusial untuk memastikan bahwa seluruh aspek dari kode program, termasuk setiap fungsi, percabangan logika *(branch)*, dan baris kode *(line of code)*, telah diuji secara menyeluruh dan memberikan hasil yang sesuai dengan ekspektasi sistem. Praktikan ditugaskan untuk melakukan *white-box* testing secara menyeluruh, dengan fokus utama pada pengujian antarmuka aplikasi IAM *(Identity and Access Management)*.

Salah satu hambatan signifikan yang dihadapi adalah keterlambatan dalam penyelesaian modul atau fitur oleh tim *developer*, khususnya menjelang fase pengujian. Kondisi ini menyebabkan praktikan kesulitan dalam menyusun dan menjalankan *unit test* secara tepat waktu, karena ketergantungan terhadap fitur yang belum selesai dikerjakan. Dengan waktu kerja praktik yang terbatas dan tekanan dari durasi *sprint* yang singkat, praktikan dituntut untuk mampu bekerja secara efisien dan adaptif. Siklus *sprint* yang cepat tidak memberikan banyak ruang untuk penundaan, sehingga keterlambatan pada satu bagian dapat berdampak pada seluruh rangkaian proses pengujian.

Untuk mengatasi hal ini, komunikasi yang intensif dan kolaboratif dengan *senior frontend developer* menjadi faktor yang sangat penting. Melalui koordinasi yang rutin, praktikan dapat memperoleh informasi lebih awal mengenai estimasi penyelesaian modul, potensi perubahan logika kode, serta klarifikasi terhadap ambiguitas dalam dokumen fungsional. Komunikasi yang baik juga memungkinkan dilakukannya penyesuaian strategi pengujian secara dinamis, sehingga pengujian *white-box* tetap dapat dilakukan secara tepat, efisien, dan sesuai dengan jadwal *sprint*. Pengalaman ini memberikan pelajaran penting bagi praktikan mengenai pentingnya kolaborasi lintas fungsi dan manajemen waktu dalam pengembangan perangkat lunak berbasis *agile*.

3.4 Cara Mengatasi Kendala

Dalam menghadapi berbagai kendala yang muncul selama masa kerja praktik, praktikan mengambil inisiatif untuk secara aktif membangun komunikasi dan kolaborasi dengan berbagai pihak yang terlibat dalam proyek, khususnya senior *frontend developer* dan *technical consultant*. Komunikasi ini dilakukan melalui berbagai saluran, baik secara langsung maupun melalui platform komunikasi internal tim, seperti *stand-up meeting* harian, diskusi kelompok melalui grup daring *(kanban tools)*, serta sesi *pair programming* saat dibutuhkan. Pendekatan ini memungkinkan praktikan untuk tidak hanya menyampaikan hambatan teknis yang dihadapi secara cepat, tetapi juga untuk mendapatkan masukan teknis yang konkret, klarifikasi terhadap perubahan atau spesifikasi proyek, serta bimbingan

langkah demi langkah dalam menyelesaikan masalah teknis sesuai dengan standar dan *best practice* yang berlaku di perusahaan.

Lebih dari sekadar menyelesaikan tugas teknis, keterlibatan langsung dalam proses *problem solving* tim memberikan praktikan wawasan yang berharga mengenai dinamika kerja tim pengembang perangkat lunak profesional, khususnya dalam lingkungan kerja yang menerapkan metodologi *agile*. Praktikan belajar bagaimana keputusan teknis diambil berdasarkan diskusi terbuka, pentingnya dokumentasi dan transparansi dalam komunikasi tim, serta bagaimana setiap peran berkontribusi terhadap keberhasilan proyek secara keseluruhan. Pengalaman ini tidak hanya memperkuat keterampilan teknis, tetapi juga menumbuhkan kompetensi kolaboratif, pemikiran kritis, dan kematangan profesionalyang sangat penting dalam dunia kerja industri teknologi informasi.

3.5 Pembelajaran Yang Diperoleh dari Kerja Profesi

Dalam masa kerja praktik, praktikan mendapatkan berbagai pengalaman dalam mengembangkan sistem informasi serta produk – produk solusi layanan digital, terutama dalam mengembangkan *unit test website*. Adapun beberapa pembelajaran yang praktikan dapatkan selama masa kerja praktik adalah sebagai berikut:

1. Praktikan dapat menganalisis dokumen fungsional serta membuat *test case* serta dokumen *requirement traceablity matrix* berdasarkan dokumen fungsional.
2. Praktikan mendapatkan kesempatan untuk memahami bagaimana sebuah produk dibangun.
3. Praktikan dapat memahami cara pembuatan *test case* dan *unit test* pada sebuah produk.
4. Praktikan dapat memahami cara *code review* terhadap kode produk.
5. Praktikan lebih memahami penting nya *unit test* untuk mendeteksi bug lebih awal.

Secara umum, pembelajaran dan pengalaman selama masa kerja praktik telah memberikan pemahaman terhadap tahapan – tahapan

menjaga kualitas produk dan kode produk, dari mulai analisis kebutuhan hingga penutupan pengujian. Pembelajaran tidak hanya meningkatkan keterampilan teknis dalam pengujian kode, tetapi juga memberikan wawasan tentang memelihara kualitas produk dan kode produk secara umum.