

**BUKU TUGAS AKHIR
CAPSTONE DESIGN**



**SISTEM SIMULASI JARINGAN 5G BERBASIS
DASHBOARD TERPUSAT DALAM LINGKUNGAN
CLOUD NATIVE UNTUK MANAGED TELECOM
*LABORATORY***

Oleh :

Ari Erginta Ginting / 1101204178

Bagus Dwi Prasetyo / 1101204109

Ima Dewi Arofani / 1101204375

Mochamad Rafli Hadiana / 1101202426

**PRODI S1 TEKNIK TELEKOMUNIKASI
FAKULTAS TEKNIK ELEKTRO
UNIVERSITAS TELKOM
BANDUNG
2024**

LEMBAR PENGESAHAN

BUKU CAPSTONE DESIGN

**SISTEM SIMULASI JARINGAN 5G BERBASIS DASHBOARD TERPUSAT DALAM
LINGKUNGAN CLOUD NATIVE UNTUK MANAGED TELECOM LABORATORY**

*5G NETWORK SIMULATION SYSTEM BASED ON CENTRALIZED DASHBOARD IN
CLOUD NATIVE ENVIRONMENT FOR MANAGED TELECOM LABORATORY*

Telah disetujui dan disahkan sebagai bagian dari Capstone Design

Program S1 Teknik Telekomunikasi

Fakultas Teknik Elektro

Universitas Telkom

Bandung

Disusun oleh:

Ari Erginta Ginting Bagus Dwi Prasetyo Ima Dewi Arofani M. Rafli Hadiana
1101204178 1101204109 1101204375 1101202426

Bandung, 18 Juli 2024 Bandung, 18 Juli 2024 Bandung, 18 Juli 2024 Bandung, 18 Juli 2024

Menyetujui,

Pembimbing 1

Pembimbing 2

Ridha Muldhina Negara, S.T., M.T.
NIP. 99730017

Prananto Bayu Herlambang, S.T.
NIP. -

LEMBAR PERNYATAAN ORISINALITAS

Saya, yang bertanda tangan di bawah ini

Nama : Ari Erginta Ginting
NIM : 1101204178
Alamat : Medan, Sumatera Utara
No. Telepon : +62 823 6614 1311
Email : gintingari73@gmail.com

Menyatakan bahwa Buku Capstone Design ini merupakan karya orisinal saya sendiri bersama dengan kelompok Capstone Design saya, dengan judul:

SISTEM SIMULASI JARINGAN 5G BERBASIS DASHBOARD TERPUSAT DALAM LINGKUNGAN CLOUD NATIVE UNTUK MANAGED TELECOM LABORATORY

5G NETWORK SIMULATION SYSTEM BASED ON CENTRALIZED DASHBOARD IN CLOUD NATIVE ENVIRONMENT FOR MANAGED TELECOM LABORATORY

Atas pernyataan ini, saya siap menanggung resiko/sanksi yang dijatuhkan kepada saya apabila dikemudian ditemukan adanya pelanggaran terhadap kejujuran akademik atau etika keilmuan dalam karya ini, atau ditemukan bukti yang menunjukkan ketidak aslian karya ini.

Bandung, 9 Mei 2024

Ari Erginta Ginting

1101204178

LEMBAR PERNYATAAN ORISINALITAS

Saya, yang bertanda tangan di bawah ini

Nama : Bagus Dwi Prasetyo
NIM : 1101204109
Alamat : Banyuwangi, Jawa Timur
No. Telepon : +62 812 1655 1263
Email : bpras747@gmail.com

Menyatakan bahwa Buku Capstone Design ini merupakan karya orisinal saya sendiri bersama dengan kelompok Capstone Design saya, dengan judul:

SISTEM SIMULASI JARINGAN 5G BERBASIS DASHBOARD TERPUSAT DALAM LINGKUNGAN CLOUD NATIVE UNTUK MANAGED TELECOM LABORATORY

5G NETWORK SIMULATION SYSTEM BASED ON CENTRALIZED DASHBOARD IN CLOUD NATIVE ENVIRONMENT FOR MANAGED TELECOM LABORATORY

Atas pernyataan ini, saya siap menanggung resiko/sanksi yang dijatuhkan kepada saya apabila dikemudian ditemukan adanya pelanggaran terhadap kejujuran akademik atau etika keilmuan dalam karya ini, atau ditemukan bukti yang menunjukkan ketidak aslian karya ini.

Bandung, 9 Mei 2024

Bagus Dwi Prasetyo

1101204109

LEMBAR PERNYATAAN ORISINALITAS

Saya, yang bertanda tangan di bawah ini

Nama : Ima Dewi Arofani
NIM : 1101204375
Alamat : Jl. Malahayu Raya 67, Brebes, Jawa Tengah
No. Telepon : +62 852 2980 8907
Email : dewiarofani@student.telkomuniversity.ac.id

Menyatakan bahwa Buku Capstone Design ini merupakan karya orisinal saya sendiri bersama dengan kelompok Capstone Design saya, dengan judul:

SISTEM SIMULASI JARINGAN 5G BERBASIS DASHBOARD TERPUSAT DALAM LINGKUNGAN CLOUD NATIVE UNTUK MANAGED TELECOM LABORATORY

5G NETWORK SIMULATION SYSTEM BASED ON CENTRALIZED DASHBOARD IN CLOUD NATIVE ENVIRONMENT FOR MANAGED TELECOM LABORATORY

Atas pernyataan ini, saya siap menanggung resiko/sanksi yang dijatuhkan kepada saya apabila dikemudian ditemukan adanya pelanggaran terhadap kejujuran akademik atau etika keilmuan dalam karya ini, atau ditemukan bukti yang menunjukkan ketidak aslian karya ini.

Bandung, 9 Mei 2024

Ima Dewi Arofani
1101204375

LEMBAR PERNYATAAN ORISINALITAS

Saya, yang bertanda tangan di bawah ini

Nama : Mochamad Rafli Hadiana
NIM : 1101202426
Alamat : Jl. Mega Mekar No. 9A Bandung, Jawa Barat
No. Telepon : +62 812 2390 1690
Email : raflihadiana@student.telkomuniversity.ac.id

Menyatakan bahwa Buku Capstone Design ini merupakan karya orisinal saya sendiri bersama dengan kelompok Capstone Design saya, dengan judul:

SISTEM SIMULASI JARINGAN 5G BERBASIS DASHBOARD TERPUSAT DALAM LINGKUNGAN CLOUD NATIVE UNTUK MANAGED TELECOM LABORATORY

5G NETWORK SIMULATION SYSTEM BASED ON CENTRALIZED DASHBOARD IN CLOUD NATIVE ENVIRONMENT FOR MANAGED TELECOM LABORATORY

Atas pernyataan ini, saya siap menanggung resiko/sanksi yang dijatuhankan kepada saya apabila dikemudian ditemukan adanya pelanggaran terhadap kejujuran akademik atau etika keilmuan dalam karya ini, atau ditemukan bukti yang menunjukkan ketidak aslian karya ini.

Bandung, 9 Mei 2024

M. Rafli Hadiana

1101202426

ABSTRAK

Teknologi seluler telah berkembang pesat terutama dengan hadirnya teknologi 5G yang merupakan generasi kelima dari teknologi seluler dan menawarkan kecepatan yang lebih tinggi serta latensi yang lebih rendah. Tugas akhir ini bertujuan untuk membuat dashboard pelatihan berupa sistem simulasi jaringan 5G RAN dalam bentuk dashboard web. Dashboard sistem simulasi ini menggunakan React.js untuk frontend, Django untuk backend, dan Kubernetes sebagai orchestrator container. Sistem 5G yang dikembangkan baik di bagian core maupun RAN menggunakan aplikasi open-source 5G dari EURECOM yaitu OpenAirInterface (OAI). Tujuan utama dari pengembangan dashboard ini adalah untuk menciptakan simulasi 5G end-to-end yang dibangun dengan konsep Cloud-native Network Function dan divisualisasikan pada dashboard. Pengguna yang mengakses dashboard ini diharuskan untuk mengkonfigurasi komponen-komponen RAN seperti CU, DU, dan UE sehingga target akhir dari simulasi adalah terciptanya koneksi 5G end-to-end. Hasil yang diperoleh menunjukkan bahwa sistem simulasi ini mampu mendukung hingga 10 pengguna, yang berarti 10 koneksi end-to-end yang berbeda. Selain itu, hasil pengujian frontend dan backend menunjukkan bahwa semua fungsi dan fitur berjalan dengan baik, dan berdasarkan pengujian UAT diperoleh skor 65.4 dengan indikator cukup.

Kata kunci : 5G, Simulasi, Dasbor, OAI, CNF

ABSTRACT

Cellular technology has advanced rapidly, particularly with the advent of 5G technology, which represents the fifth generation of cellular technology, offering higher speeds and lower latency. This final project aims to create a training dashboard in the form of a 5G RAN network simulation system presented as a web dashboard. The simulation system dashboard uses React.js for the front end, Django for the back end, and Kubernetes as the container orchestrator. The 5G system developed, both in the core and RAN sections, utilizes the open-source 5G application from EURECOM, OpenAirInterface (OAI). The primary goal of this dashboard development is to create an end-to-end 5G simulation built using the Cloud-native Network Function concept and visualized on the dashboard. Users accessing this dashboard are required to configure RAN components such as CU, DU, and UE, with the final target of the simulation being the establishment of an end-to-end 5G connection. The results show that this simulation system can support up to 10 users, meaning 10 different end-to-end connections. Additionally, frontend and backend testing results indicate that all functions and features operate well, and based on UAT testing, a score of 65.4 was obtained with an adequate indicator.

Keywords: 5G, Simulation, Dashboard, OAI, CNF

KATA PENGANTAR

Puji Syukur kami panjatkan kehadirat Tuhan yang Maha Esa yang telah melimpatkan segala berkat dan rahmatNya, sehingga atas izin-Nya laporan Capstone Design dengan judul **“SISTEM SIMULASI JARINGAN 5G BERBASIS DASHBOARD TERPUSAT DALAM LINGKUNGAN CLOUD NATIVE UNTUK MANAGED TELECOM LABORATORY”** dapat terselesaikan dengan baik.

Tugas akhir ini disusun sebagai salah satu syarat kelulusan untuk memperoleh gelar Sarjana Teknik pada program studi Teknik Telekomunikasi, Fakultas Teknik Elektro, Universitas Telkom Bandung. Penulis menyadari adanya berbagai keterbatasan dalam penyusunan tugas akhir ini sehingga masih jauh dari kesempurnaan. Oleh karena itu, penulis sangat mengharapkan kritik dan saran yang membangun agar dapat membuat produk ini menjadi lebih baik kedepannya.

Penulis juga berharap tugas akhir ini dapat dikembangkan lebih lanjut untuk memperluas pengetahuan dan wawasan, serta memberikan manfaat yang lebih besar bagi industri dan masyarakat. Semoga tugas akhir ini juga dapat menjadi referensi yang berguna untuk penelitian-penelitian berikutnya, sehingga kontribusinya terhadap bidang Teknik Telekomunikasi dapat semakin meningkat.

UCAPAN TERIMAKASIH

Dalam penyusunan Capstone Design ini, Penulis ingin memberikan ucapan terima kasih yang sebesar-besarnya kepada seluruh pihak yang terlibat dalam pembuatan Tugas Akhir. Penulis ucapkan banyak terima kasih kepada:

1. Tuhan Yang Maha Esa, atas segala berkat dan rahmat-Nya telah memberikan kesehatan hingga penulis dapat menyelesaikan Capstone Design ini
2. **Orang Tua dan Keluarga** yang tak pernah henti memberikan semangat, dukungan, dan mendoakan yang terbaik untuk Penulis
3. Ibu **Ridha Muldina Negara, S.T., M.T.**, selaku pembimbing I. Terima kasih atas bimbingan, saran, dan kritikannya selama Capstone Design. Semoga Ibu selalu sehat dan sukses dalam karir
4. Bapak **Prananto Bayu Herlambang, S.T.**, selaku pembimbing II. Terima kasih atas bimbingan, saran, dan kritikannya selama Capstone Design. Semoga Bapak selalu sehat dan sukses dalam karir
5. **Yosafat Marselino Agus**, selaku staff Telecom Infra Project. Terimakasih atas bimbingan, saran, serta kritikan teknis selama Penulis mengerjakan Capstone Design hingga selesai
6. Telecom Infra Project dan para staf yang telah membimbing dan menemani serta menyediakan fasilitas dalam penggerjaan Capstone Design ini
7. Terima kasih kepada Bapak **Bagus Aditya, S.T., M.T.**, Bapak **Harfan Hian Ryanu, S.T., M.E**, Ibu **Ir. Rita Magdalena, M.T.**, dan Ibu **Sri Astuti, S.T., M.T.** selaku dosen wali yang tak pernah henti memberikan arahan dan bimbingan semasa perkuliahan Penulis
8. Adaptive Network Laboratory dan para asisten yang telah memberikan fasilitas serta memberikan pengetahuan, pengalaman riset dan dukungan selama perkuliahan
9. Teman-teman capstone seperjuangan yang telah berjuang bersama, semoga dilancarkan segalanya dan sukses dalam karir kedepannya
10. Penulis tidak mungkin menyebut secara individu setiap pihak yang telah membantu dalam kehidupan penulis. Terima kasih dan semoga selalu dalam lindungan Tuhan yang Maha Esa

DAFTAR ISI

LEMBAR PENGESAHAN	i
BUKU CAPSTONE DESIGN	i
LEMBAR PERNYATAAN ORISINALITAS	ii
LEMBAR PERNYATAAN ORISINALITAS	iii
LEMBAR PERNYATAAN ORISINALITAS	iv
LEMBAR PERNYATAAN ORISINALITAS	v
ABSTRAK.....	vi
ABSTRACT.....	vii
KATA PENGANTAR	viii
UCAPAN TERIMA KASIH.....	ix
DAFTAR ISI.....	x
DAFTAR GAMBAR	xiv
DAFTAR TABEL.....	xviii
DAFTAR SINGKATAN	xx
BAB 1 USULAN GAGASAN	1
1.1 Latar Belakang Masalah	1
1.2 Informasi Pendukung Masalah	1
1.2.1 Aspek Manajemen	1
1.2.2 Aspek Kontrol.....	2
1.2.3 Aspek Kebutuhan Pengguna	2
1.3 Tujuan Capstone	2
1.4 Analisa Solusi yang Ada.....	2
BAB 2 DESAIN KONSEP SOLUSI.....	4
2.1 Dasar Penentuan Spesifikasi	4
2.2 Batasan dan Spesifikasi.....	5

2.2.1	Batasan Fungsionalitas.....	5
2.2.2	Spesifikasi Platform	5
2.2.2.2	Spesifikasi <i>Tech Stack</i>	6
2.3	Pengukuran/Verifikasi Spesifikasi	8
2.3.1	Spesifikasi Fungsi	8
2.3.2	Spesifikasi <i>Tech Stack</i>	9
BAB 3	DESAIN RANCANGAN SOLUSI.....	12
3.1	Alternatif Usulan Solusi.....	12
3.1.1	Physical Network Functions (PNF)	12
3.1.2	Virtual Network Function (VNF)	12
3.1.3	Cloud Native Network Function (CNF)	12
3.2	Analisis dan Pemilihan Solusi	13
3.2.1	Parameter Analisis Solusi	13
3.2.2	Mekanisme Pemilihan Solusi.....	14
3.3	Desain Solusi Terpilih.....	16
3.3.1	Desain Alur Kerja Aplikasi.....	16
3.3.2	Desain <i>Backend</i> Aplikasi	20
3.3.3	Desain <i>Frontend</i> Aplikasi	23
3.3.4	Desain Infrastruktur 5G <i>Network</i>	26
3.3.5	Desain Infrastruktur Aplikasi.....	27
3.3.6	Desain Automasi Pengembangan Perangkat Lunak	28
3.3.7	Desain Monitoring Klaster.....	29
3.3.8	Desain Keseluruhan Aplikasi pada Kubernetes	30
3.4	Jadwal Penggeraan.....	30
3.4.1	Jadwal Capstone.....	30
3.4.2	Anggaran.....	31
BAB 4	IMPLEMENTASI	32

4.1	Deskripsi Umum Implementasi	32
4.2	Detil Impelementasi	33
4.2.1	Implementasi Infrastruktur.....	33
4.2.2	Implementasi <i>5G Cloud</i>	40
4.2.3	Implementasi <i>Backend</i>	50
4.2.4	Implementasi <i>Frontend</i>	67
4.2.5	 Implementasi <i>Deployment</i>	83
4.3	Prosedur Pegoperasian.....	85
4.3.1	Fitur Otentikasi <i>User</i>	85
4.3.2	Fitur <i>User Management</i>	86
4.3.3	Fitur Grafik Topologi.....	88
4.3.4	Fitur <i>Component Management</i>	89
4.3.5	Fitur <i>Sniff Function</i>	92
4.3.6	Fitur Monitoring.....	94
BAB 5	PENGUJIAN SISTEM.....	95
5.1	Skenario Umum Pengujian	95
5.2	Detil Pengujian.....	95
5.2.1	API <i>Testing</i>	95
5.2.2	<i>End-to-End (E2E) Testing</i>	114
5.2.3	<i>Performance Testing</i>	127
5.2.4	<i>User Acceptance Testing</i>	133
5.3	Analisa Hasil Pengujian.....	149
5.3.1	Hasil API <i>Testing</i>	149
5.3.2	Hasil <i>End-to-End (E2E) Testing</i>	154
5.3.3	Hasil <i>Performance Testing</i>	160
5.3.4	Hasil <i>User Acceptance Testing</i>	163
5.4	Kesimpulan	165

DAFTAR PUSTAKA	168
LAMPIRAN CD-1.....	173
LAMPIRAN CD-2.....	174
LAMPIRAN CD-3.....	175
LAMPIRAN CD-4.....	176
LAMPIRAN CD-5.....	210

DAFTAR GAMBAR

Gambar 2.1 Implementasi Testbed Infrastruktur 5G Barcelona.....	4
Gambar 3.1 Alur Pembuatan Akun User.....	17
Gambar 3.2 Alur Konfigurasi Komponen RAN.....	18
Gambar 3.3 Alur Monitoring UE.....	19
Gambar 3.4 Alur Sniffing Wireshark	20
Gambar 3.5 Django MTV Pattern	21
Gambar 3.6 Desain Arsitektur Backend-REST API.....	22
Gambar 3.7 Desain Arsitektur Backend-Websocket.....	22
Gambar 3.8 Diagram Alir Inisialisasi Peran	23
Gambar 3.9 <i>Use case</i> Diagram Actor Admin	24
Gambar 3.10 <i>Use case</i> Diagram Actor User	24
Gambar 3.11 Desain Infrastruktur E2E 5G Network	26
Gambar 3.12 Arsitektur 5G Koneksi Multi gNB.....	27
Gambar 3.13 Desain Infrastruktur Aplikasi.....	27
Gambar 3.14 Desain Automasi Pengembangan Perangkat Lunak	28
Gambar 3.15 Desain Monitoring Klaster.....	29
Gambar 3.16 Desain Keseluruhan Aplikasi pada Kubernetes.....	30
Gambar 4.1 Arsitektur Kubernetes	33
Gambar 4.2 Arsitektur VM Kluster	34
Gambar 4.3 Arsitektur Monitoring	36
Gambar 4.4 Arsitektur Distributed Block Storage	38
Gambar 4.5 Arsitektur Automation	39
Gambar 4.6 Validasi Koneksi Port CU	46
Gambar 4.7 Validasi Koneksi Wireshark CU.....	46
Gambar 4.8 Validasi Koneksi Port DU	47
Gambar 4.9 Validasi Koneksi Wireshark DU	47
Gambar 4.10 Validasi Koneksi Port RFSim.....	47
Gambar 4.11 Routing Table UPF	48
Gambar 4.12 Routing Table UE	48
Gambar 4.13 Skema Komunikasi Multi Node	49
Gambar 4.14 Instalasi Django Sukses	52
Gambar 4.15 Flowchart Otentikasi Pengguna	61

Gambar 4.16 Struktur Folder.....	69
Gambar 4.17 File CSS	79
Gambar 4.18 Tampilan Login Page.....	85
Gambar 4.19 Tampilan Logout User	86
Gambar 4.20 Tampilan Create Users.....	86
Gambar 4.21 Tampilan Read User	87
Gambar 4.22 Tampilan Update User	87
Gambar 4.23 Tampilan Delete User	88
Gambar 4.24 Tampilan Components Lifecycle Management.....	88
Gambar 4.25 Tampilan Component Logging and Testing	89
Gambar 4.26 Tampilan Shell PING & CURL UE	89
Gambar 4.27 Tampilan Konfigurasi CU	90
Gambar 4.28 Tampilan Konfigurasi CU	91
Gambar 4.29 Tampilan Konfigurasi UE.....	91
Gambar 4.30 Tampilan Protocol Filter.....	92
Gambar 4.31 Tampilan File Capture Download.....	93
Gambar 4.32 Tampilan Grafik Monitoring	94
Gambar 4.33 Tampilan Table Value Monitoring	94
Gambar 5.1 Response API <i>Create user</i>	97
Gambar 5.2 Response API <i>Read All User</i>	98
Gambar 5.3 Response API <i>Update Password User</i>	98
Gambar 5.4 Response API <i>Delete User</i>	99
Gambar 5.5 Response API <i>Get User Info</i>	99
Gambar 5.6 Response API Login	100
Gambar 5.7 Response API Logout	101
Gambar 5.8 Response API Refresh Token	101
Gambar 5.9 Response API Verify Token	102
Gambar 5.10 API Response List Pod	103
Gambar 5.11 API Response List Deployment.....	104
Gambar 5.12 API Response Get Log Pod	104
Gambar 5.13 API Response Restart Deployment.....	105
Gambar 5.14 Proses Testing API Websocket User PING	105
Gambar 5.15 Proses Testing API Websocket User CURL.....	106
Gambar 5.16 Proses Testing API Websocket User KPI.....	106

Gambar 5.17 Proses Testing API Websocket SCTP	107
Gambar 5.18 Response API AMF Log.....	107
Gambar 5.19 Response API UPF Log	108
Gambar 5.20 Response API AMF Deployment	108
Gambar 5.21 Response API UPF Deployment.....	109
Gambar 5.22 Response API <i>Get Configuration Value</i>	110
Gambar 5.23 Response API <i>Post Configuration Value</i>	110
Gambar 5.24 Response API <i>Start Component</i>	111
Gambar 5.25 Response API <i>Stop Component</i>	111
Gambar 5.26 Proses Testing API Websocket Sniffing Pod	112
Gambar 5.27 Response API List PCAP File	113
Gambar 5.28 Response API Dowload PCAP File.....	113
Gambar 5.29 Response API Delete PCAP File	114
Gambar 5.30 Tampilan Admin Membuat User	115
Gambar 5.31 Tampilan Admin Melihat User	116
Gambar 5.32 Tampilan Admin Mengubah Password User	116
Gambar 5.33 Tampilan Admin Menghapus User	116
Gambar 5.34 Verifikasi Informasi User dari Browser.....	117
Gambar 5.35 Verifikasi Informasi User Berhasil Login.....	117
Gambar 5.36 Verifikasi Informasi Admin Berhasil Login	117
Gambar 5.37 Verifikasi User dan Admin Berhasil Logout	118
Gambar 5.38 Koneksi Topologi Terhubung	118
Gambar 5.39 Response API Pod Saat Topologi Terhubung	118
Gambar 5.40 Response API Deployment Saat Topologi Terhubung.....	119
Gambar 5.41 UE Berhasil Menampilkan Log	119
Gambar 5.42 UE Berhasil Di Restart	119
Gambar 5.43 AMF Berhasil Menampilkan Log	120
Gambar 5.44 UPF Berhasil Menampilkan Log	120
Gambar 5.45 AMF Berhasil Menampilkan State Pod	121
Gambar 5.46 UPF Berhasil Menampilkan State Pod	121
Gambar 5.47 UE Berhasil Menampilkan Hasil Ping.....	122
Gambar 5.48 UE Berhasil Menampilkan Hasil CURL	122
Gambar 5.49 User Berhasil Mendapatkan Hasil Monitoring	123
Gambar 5.50 User Berhasil Mendapatkan Key Table Value	123

Gambar 5.51 User Berhasil Mendapatkan Hasil Protocol SCTP	123
Gambar 5.52 User Berhasil Mendapatkan Hasil Value Config.....	124
Gambar 5.53 User Berhasil Mengubah Hasil Value Config	124
Gambar 5.54 User Berhasil Menghubah State UE	125
Gambar 5.55 User Berhasil Menghentikan UE	125
Gambar 5.56 User Berhasil Mendapatkan Hasil Sniffing UE.....	126
Gambar 5.57 User Berhasil Mendapatkan Hasil File Sniffing	126
Gambar 5.58 User Berhasil Mengunduh Hasil File Sniffing	127
Gambar 5.59 User Berhasil Menghapus Hasil File Sniffing	127
Gambar 5.60 Diagram Alir Pengujian <i>System Load Capacity</i>	128
Gambar 5.61 Daftar Pod dan Status Pod per <i>User</i>	129
Gambar 5.62 Monitoring Resource pada User	130
Gambar 5.63 Diagram Alir Pengujian E2E	131
Gambar 5.64 Proses UE Mencapai PDU <i>Establishment</i>	131
Gambar 5.65 Hasil Wireshark Saat Proses PDU <i>Establishment</i>	132
Gambar 5.66 Pod UE Mendapatkan Interface oaitun_ue1	132
Gambar 5.67 Index Parameter Penilaian System Usability Scale	146
Gambar 5.68 Grafik System Load Capacity	160
Gambar 5.69 Grafik End-to-End Connection	161

DAFTAR TABEL

Tabel 2.1 Rincian <i>Tech Stack</i>	6
Tabel 2.2 Verifikasi Spesifikasi Fungsi 1	8
Tabel 2.3 Verifikasi Spesifikasi Fungsi 2	8
Tabel 2.4 Verifikasi Spesifikasi Fungsi 3	9
Tabel 2.5 Verifikasi Spesifikasi Fungsi 4	9
Tabel 2.6 Verifikasi Spesifikasi <i>Tech Stack</i> 1	9
Tabel 2.7 Verifikasi Spesifikasi <i>Tech Stack</i> 2	10
Tabel 2.8 Verifikasi Spesifikasi <i>Tech Stack</i> 3	11
Tabel 2.9 Verifikasi Spesifikasi <i>Tech Stack</i> 4	11
Tabel 3.1 Penilaian Kualitatif dalam Pemilihan <i>Network Functions</i>	14
Tabel 3.2 Pengskalaan Rating pada Setiap Kriteria.....	15
Tabel 3.3 Prosedur Integrasi pada Setiap <i>Network Function</i>	15
Tabel 3.4 Penilaian Kuantitatif dalam Pemilihan <i>Network Functions</i>	16
Tabel 3.5 Tabel Daftar Pelaku	23
Tabel 3.6 Pendefinisian Usecase Diagram Actor Admin	24
Tabel 3.7 Pendefinisian Usecase Diagram Actor User.....	25
Tabel 3.8 Jadwal Capstone	31
Tabel 4.1 Spesifikasi VM Kluster Kubernetes	32
Tabel 4.2 Daftar baris kode fungsi manajemen komponen 5G cloud	57
Tabel 4.3 Penyesuaian Properties CSS	79
Tabel 5.1 Langkah API Testing.....	96
Tabel 5.2 List API User Management	96
Tabel 5.3 List API Token	99
Tabel 5.4 List Kubernetes API	102
Tabel 5.5 List Kubernetes Websocket API.....	103
Tabel 5.6 List Helm Chart API.....	109
Tabel 5.7 List Wireshark API.....	112
Tabel 5.8 List Wireshark Websocket API	112
Tabel 5.9 Langkah <i>E2E Testing</i>	114
Tabel 5.10 Langkah <i>Performance Testing</i>	128
Tabel 5.11 Pengujian <i>System Load Capacity</i>	129
Tabel 5.12 Alokasi CPU dan RAM Satu User	130

Tabel 5.13 Hasil Pengujian <i>Success Rate</i>	132
Tabel 5.14 Langkah Pengujian UAT	134
Tabel 5.15 Hasil <i>Functional Testing</i> pada Halaman Admin	135
Tabel 5.16 Hasil Functional Testing pada Halaman Pengguna	135
Tabel 5.17 Hasil Perhitungan SUS dari UAT 1	147
Tabel 5.18 Hasil Perhitungan SUS dari UAT 2	148
Tabel 5.19 Hasil User Management APIs	149
Tabel 5.20 Hasil Token APIs.....	150
Tabel 5.21 Kubernetes APIs	150
Tabel 5.22 Hasil Helm Chart APIs	152
Tabel 5.23 Hasil Wireshark APIs	153
Tabel 5.24 Hasil User Management Functions.....	154
Tabel 5.25 Hasil Token Functions.....	154
Tabel 5.26 Hasil Kubernetes Functions.....	155
Tabel 5.27 Hasil Helm Chart Function.....	157
Tabel 5.28 Hasil Wireshark Functions	159
Tabel 5.29 Total Kebutuhan Per User	162
Tabel 5.30 Nilai Rata-Rata Tiap Pertanyaan	164

DAFTAR SINGKATAN

AMF	: Access and Mobility Management Function
AMP	: Aether Management Platform
API	: Application Programming Interface
CI/CD	: Continuous Integration Continuous Delivery
CN	: Core Network
CNF	: Cloud Native Network Function
CNI	: Container Network Interface
COTS	: Commercial Off The Shelf
CPU	: Central Processing Unit
CRUD	: Create Read Update Delete
CSS	: Cascading Style Sheets
CU	: Central Unit
DNN	: Data Name Network
DU	: Distributed Unit
E2E	: End-to-End
gNB	: Next-Generation Node B
GTP	: GPRS Tunneling Protocol
GUAMI	: Globally Unique AMF Identifier
IPAM	: IP Address Management
KPI	: Key Performance Indicator
LCM	: Life Cycle Management
LTE	: Long Term Evolution
MCC	: Mobile Country Code
MCDM	: Multi-Criteria Decision Making
MiB	: Mebibyte

MNC	: Mobile Network Code
MTU	: Maximum Transmission Unit
MTV	: Model, Template, and View
MVC	: Model, View, and Controller
NF	: Network Function
NGAP	: Next Generation Application Protocol
NI-USRP	: National Instruments – Universal Software Radio Peripheral
NSSAI	: Network Slice Selection Assistance Information
OAI	: OpenAirInterface
O-Cloud	: Orchestrator and Cloud Platform
ONF	: Open Network Foundation
ORCA	: Open RAN Configuration Application
OS	: Operating System
PCC	: Policy and Charging Control
PDU	: Protocol Data Unit
PLMN	: Public Land Mobile Network
PNF	: Physical Network Function
PVC	: Persistent Volume Claim
RAM	: Random Access Memory
RAN	: Radio Access Network
REST	: Representational State Transfer
RFSIM	: Radio Frequency Simulator
ROC	: Runtime Operation Control
RU	: Radio Unit
SBA	: Service Based Architecture
SCTP	: Stream Control Transmission Protocol

SD	: Slice Differentiator
SDR	: Software Defined Network
SIM	: Subscriber Identity Module
SMF	: Session Management Function
SMO	: Service Management and Orchestration
SQL	: Structured Query Language
SST	: Slice/Service Type
SUS	: System Usability Scale
TAC	: Tracking Area Code
TCP	: Transmission Control Protocol
UAT	: User Acceptance Change
UDM	: Unified Data Management
UE	: User Equipment
UI	: User Interface
UPF	: User Plane Function
VM	: Virtual Machine
VNF	: Virtual Network Function
WSM	: Weighted Sum Model

BAB 1

USULAN GAGASAN

1.1 Latar Belakang Masalah

Seiring dengan adanya kebutuhan peningkatan kecepatan dan kapasitas layanan yang terus meningkat, menyebabkan pentingnya untuk merubah pola infrastruktur jaringan seluler agar dapat memenuhi kebutuhan pengguna dan menghindari ketidaksesuaian permintaan trafik [1]. Teknologi jaringan 5G saat ini memberikan fitur yang bisa mengatasi kebutuhan tersebut diantaranya menyediakan kecepatan hingga sepuluh kali lipat lebih besar dan latensi dua kali lipat lebih rendah dari generasi sebelumnya. Namun solusi implementasi untuk jaringan 5G saat ini memerlukan kemampuan spesialisasi khusus untuk diaplikasikan [2]. Terlebih lagi jika mengaplikasikan konfigurasi 5G menggunakan baris perintah[3]. Oleh karena itu, pelatihan dan edukasi yang mendalam tentang teknologi 5G menjadi sangat krusial untuk memastikan tenaga kerja memiliki keterampilan yang diperlukan. Dengan pemahaman yang komprehensif, para profesional dapat mengoptimalkan dan mengelola jaringan 5G secara lebih efektif, serta menghadirkan inovasi dan efisiensi yang lebih tinggi dalam berbagai sektor industri. Maka dari itu, diperlukan sebuah *dashboard* berbasis *cloud native* sebagai platform yang menjembatani proses konfigurasi dan mempermudah *troubleshooting* pengguna.

Pendekatan berbasis *cloud native* merupakan sebuah cara untuk mengembangkan, membangun, menjalankan, dan mengelola aplikasi yang sepenuhnya memanfaatkan model komputasi awan. Teknologi ini adalah faktor utama yang meningkatkan efisiensi, *fast recovery*, agilitas, elastisitas, dan akses dalam skala besar untuk layanan 5G [4]. Pendekatan *cloud native* didasarkan pada penggunaan kontainer dan penerapan Cloud Native Network Function (CNF), juga memanfaatkan metodologi Continues Integration Continues Delivery (CI/CD) untuk melakukan otomasi.

1.2 Informasi Pendukung Masalah

1.2.1 Aspek Manajemen

Manajemen perangkat pada infrastruktur teknologi jaringan 5G konvensional menimbulkan tantangan signifikan yang berhubungan dengan risiko *human error*, serta kurangnya fleksibilitas manajemen. Perangkat harus dikelola secara terpisah sehingga menciptakan kerentanan terhadap konfigurasi yang mungkin salah diimplementasikan [5]. Hal tersebut mengakibatkan kurangnya efisiensi, fleksibilitas, dan reliabilitas pada sistem.

1.2.2 Aspek Kontrol

Dalam teknologi jaringan 5G konvensional, seringkali terdapat kendala dalam aspek kontrol, seperti terbatasnya kemampuan administrator dalam mengumpulkan metrik dan melakukan monitoring anomali perangkat secara *real-time*. Pada dasarnya, teknologi 5G konvensional tidak memiliki fitur untuk monitoring secara *real-time*. Kendala ini dapat mengakibatkan kurangnya efisiensi dan reliabilitas sistem secara keseluruhan. Monitoring bertujuan memeriksa data performa dari infrastruktur, dan juga konektivitas pengguna [6].

1.2.3 Aspek Kebutuhan Pengguna

Baik perusahaan maupun institusi pendidikan dalam sektor telekomunikasi merasakan perlunya solusi yang memungkinkan untuk mengadopsi dan memahami teknologi 5G secara efektif dalam skala kecil. Tujuannya termasuk mengaplikasikan teknologi ini untuk keperluan riset yang mencakup eksperimen dan pengembangan konsep, serta pendidikan. Namun, kesulitan utama yang dihadapi adalah adanya keterbatasan akses terhadap sumber daya untuk mendukung upaya ini [7].

1.3 Tujuan Capstone

Dengan beragam permasalahan yang telah disebutkan diatas menghasilkan tujuan dari projek capstone ini yaitu membuat *dashboard* untuk pengelolaan jaringan 5G *cloud* terpusat dengan teknologi *cloud native* menggunakan orkestrator Kubernetes yang dapat digunakan untuk monitoring dan manajemen jaringan 5G secara *real-time*, serta simulasi jaringan 5G secara virtual untuk tujuan riset, eksperimen, dan pelatihan.

1.4 Analisa Solusi yang Ada

Saat ini terdapat Network Function (NF) 5G *open-source* yang dikembangkan oleh EURECOM sebagai platform untuk penelitian yaitu OpenAirInterface (OAI). *Software* ini adalah platform Software Defined Radio (SDR) Open5G untuk membangun teknologi virtualisasi jaringan Radio Access Network (RAN) dan Core Network (CN) [8]. Para peneliti disana telah membuat suatu produk implementasi yang menjadi solusi pengujian *end-to-end* (E2E) 5G bernama OAIBOX. Platform ini adalah perangkat keras siap pakai dan terintegrasi dengan National Instruments – Universal Software Radio Peripheral (NI-USRP), 5G User Equipment (UE) Quectel RM500Q-GL, dan programmable Subscriber Identity Module (SIM) card. OAIBOX terdiri dari single Next-Generation Node B (gNB) dan 5G CN yang diimplementasikan pada satu *host* secara monolitik. Setelah mendapatkan catu daya dan internet, OAIBOX akan terhubung dengan layanan *backend Dashboard* OAIBOX [9]. *Dashboard* ini dapat diakses secara publik dan digunakan untuk memvisualisasikan metrik

pemantauan *real-time* dengan Key Performance Indicator (KPI) yang ditentukan serta mengonfigurasi *testbed* gNB dan 5G Radio Access Network (RAN). Walaupun demikian produk tersebut tidak bisa dijangkau secara gratis dan hanya bisa diakses apabila *subscription* OAIBOX dibeli sesuai dengan paket lisensi, sehingga membatasi kebutuhan pelatihan dan penelitian skala kecil.

Di sisi lain Open Network Foundation (ONF) mengembangkan produk bernama Aether Management Platform (AMP) yang mempunyai sub-fungsi sebagai Aether Runtime Operation Control (ROC). Software ini dirancang dengan tujuan utama agar operator dapat mengonfigurasi *subscriber* dan mengimplementasikan *policy* konektivitas seluler *private* dan layanan *edge cloud* untuk platform 5G Connected Edge ONF [10]. Konfigurasi tersebut bersifat persisten dan dapat diobservasi secara manual maupun otomatis. Namun platform tersebut tidak secara langsung mengelola *lifecycle container* dan tidak secara langsung menyimpan informasi metrik atau *logging* [11], sehingga memerlukan aplikasi *third party* untuk berjalan secara maksimal.

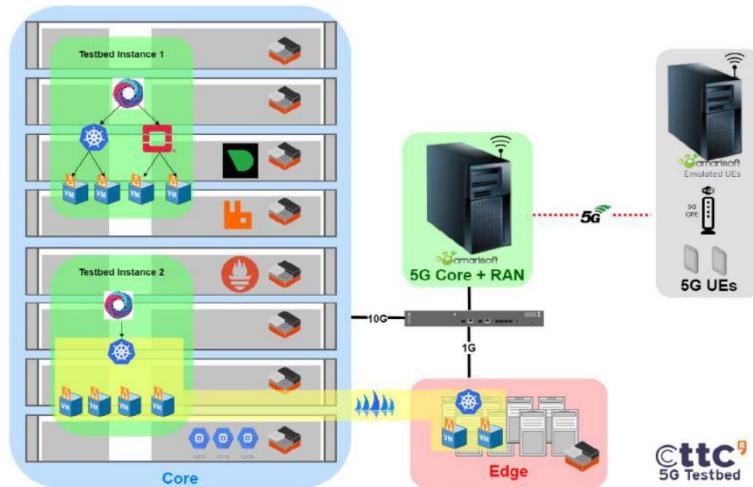
Selanjutnya terdapat hasil penelitian berupa produk dari platform Mosaic5G berupa 5G servis platform berbasis *cloud native* yaitu Kube5G-Operator. Kubernetes operator tersebut mengimplementasikan dan mengotomasi kebutuhan dasar dalam waktu yang singkat seperti instalasi, konfigurasi dan monitoring dan kebutuhan operasi kompleks seperti melakukan *update*, *backup* dan *failover* dari proses merancang CNF 4G/5G CN [12]. Masalahnya pada penelitian saat ini solusi operator yang ada tidak memiliki portal manajemen platform, dengan hanya menyajikan otomasi di backend jaringan 4G Long Term Evolution (LTE) saja. Sehingga diperlukan adanya penyesuaian aplikasi untuk melakukan konfigurasi pada 5G CN.

BAB 2

DESAIN KONSEP SOLUSI

2.1 Dasar Penentuan Spesifikasi

Spesifikasi menjadi krusial karena berasal dari standar O-RAN Alliance, dengan tambahan input dari kebutuhan klien operator dan kasus integrasi sistem di dalam Telecom Infra Project (TIP) Community Lab Telkom University [13]. Menurut wawancara yang bersumber dari teknisi sistem telekomunikasi disana, *Orchestrator and Cloud Platform* (O-Cloud) yang diterapkan tidak memiliki standarisasi khusus mengenai alat yang digunakan, sehingga memungkinkan adanya penggunaan berbagai bahasa pemrograman dan *framework* untuk membangun *user interface* aplikasi. Fleksibilitas juga diterapkan pada sistem basis data yang dapat berupa Structured Query Language (SQL) atau non-SQL. Meskipun demikian, *framework* teknologi pendukung aplikasi tetap dibangun menggunakan standarisasi yang teruji dari *developer community* [14]. Berdasarkan standarisasi O-RAN Allience fungsionalitas minimum O-Cloud melibatkan *interface* web untuk melakukan manajemen komponen CNF, serta adanya alat diagnostik seperti penampilan *log*, *traffic sniffer*, dan deteksi *failure* orkestrator pada platform [15].



Gambar 2.1 Implementasi Testbed Infrastruktur 5G Barcelona

Sebagai standar implementasi konkretnya, proyek 5G-EPICENTRE di Barcelona menunjukkan implementasi yang berhasil, di mana integrasi 5G CN dan RAN dilakukan melalui penggunaan klaster Kubernetes, dengan Prometheus berperan sebagai fungsi monitoring [16]. Dalam konteks ini, keberhasilan implementasi tersebut membuktikan peran Kubernetes dalam mendukung integrasi yang efisien dan monitoring performa yang akurat pada lingkungan 5G.

2.2 Batasan dan Spesifikasi

2.2.1 Batasan Fungsionalitas

Dalam lingkup penelitian tugas akhir ini, beberapa batasan fungsionalitas telah ditentukan untuk mengarahkan fokus penelitian, yaitu:

1. Implementasi monitoring dan pengelolaan jaringan 5G melalui sebuah *website dashboard* dengan memanfaatkan teknologi *cloud native*.
2. Simulasi jaringan 5G dilakukan dalam konteks lingkungan virtual, bukan pada jaringan fisik yang sebenarnya.
3. Konfigurasi komponen 5G yang melibatkan UE, Central Unit (CU), dan Distributed Unit (DU) telah diatur, mencakup aspek UE, RAN dan CN guna menciptakan representasi yang komprehensif dari infrastruktur 5G.

2.2.2 Spesifikasi Platform

Spesifikasi yang ditetapkan pada sistem simulasi 5G berbasis *dashboard* dibagi menjadi dua bagian yaitu spesifikasi fungsi dan spesifikasi *tech stack* dengan rincian sebagai berikut:

2.2.2.1 Spesifikasi Fungsi

1. VNF Life Cycle Management (LCM)

Simulasi jaringan 5G berbasis *dashboard* ini diimplementasikan di lingkungan berbasis *cloud* sehingga menerapkan Virtual Network Function (VNF) LCM yang bertujuan untuk instansiasi O-Cloud dan memberikan notifikasi pada Service Management and Orchestration (SMO) ketika melakukan instansiasi *resources Network Function* berhasil diimplementasikan.

2. Memodifikasi Konfigurasi VNF

SMO pada umumnya menginginkan konfigurasi serangkaian O-RAN VNF berbasis *cloud* yang dapat dimodifikasi agar dapat mengubah perilaku jaringan RAN seperti optimisasi ulang pola trafik jaringan atau memperbarui trafik *load balancing* di seluruh jaringan RAN. Oleh karena itu, pemodifikasi konfigurasi pada VNF diperlukan pada simulasi ini.

3. Monitoring OpenRAN NFs

Monitoring performa *management* dengan menampilkan grafik RAN *metric* yang berhubungan dengan sumber daya O-Cloud sehingga pengguna dapat melihat penggunaan metrik dan performanya pada RAN 5G.

4. CNF *Instance Management*

CNF memungkinkan fungsionalitas jaringan berjalan didalam suatu wadah yang biasa disebut dengan kontainer [17]. Manajemen CNF *instance* memiliki pengaruh besar dalam keberlangsungan siklus hidup NF. Mirip halnya dengan manajemen pada VNF, manajemen CNF diimplementasikan di lingkungan berbasis *cloud*. Segala proses pembuatan dan penghapusan *instance* akan memberikan notifikasi pada SMO.

2.2.2.2 Spesifikasi *Tech Stack*

Tabel 2.1 Rincian *Tech Stack*

Bagian	Tech Stack	Keterangan
<i>Frontend</i>	React.Js	<i>Library JavaScript front-end</i> bersifat <i>open source</i> untuk membuat desain UI
	Patternfly	<i>Framework</i> dengan bahasa Javascript dan Typescript untuk membangun aplikasi berbasis <i>User Interface</i> (UI) yang interaktif dan dinamis
	Material UI	Material UI adalah <i>open-source</i> React library yang menerapkan Google Material Design untuk membuat grafis <i>frontend</i>
<i>Backend</i>	Django Python	<i>Framework</i> Representational State Transfer (REST) Application Programming Interface (API) berbasis bahasa <i>python</i> sebagai penghubung antara sebuah aplikasi dan aplikasi lainnya
	PostgreSQL	Database relasional sebagai tempat penyimpanan kumpulan item data yang saling berhubungan

	Redis Server	<i>Message broker</i> yang berfungsi untuk mengirimkan data dari sisi backend hingga ke frontend melalui koneksi websocket.
CNF	OpenAirInterface	Platform <i>open-source</i> yang menerapkan teknologi 5G pada <i>cloud</i> dengan integrasi antar komponen menggunakan teknologi 3GPP USRP
Infrastruktur	Kubernetes	Orkestrator kontainer untuk manajemen aplikasi yang dikontainerisasi dan sebagai salah satu syarat penerapan CNF
	Multus	Sebagai Container Network Interface (CNI) tambahan untuk memudahkan pengelolaan jaringan pada Kubernetes
	Metal LB	Sebagai <i>load balancer</i> untuk mengelola lalu lintas aplikasi pada Kubernetes
	Nginx Ingress	Mengelola trafik yang masuk ke klaster dengan menerapkan domain berdasarkan pengalamatan Kubernetes <i>service</i>
	Longhorn	<i>Distributed block storage</i> yang digunakan pada Kubernetes untuk mengelola penyimpanan kontainer
	Jenkins	<i>Automation tool open-source</i> untuk membantu mengotomasi bagian-bagian dari proses pengembangan aplikasi
	Prometheus	Sistem monitoring berbasis metrik yang bekerja dengan mengolah metrik dari <i>node exporter</i> untuk keperluan pengolahan data

	Grafana	Mengelola data dari Prometheus grafik yang dapat digunakan untuk analisis dan monitoring
--	---------	--

2.3 Pengukuran/Verifikasi Spesifikasi

Sesuai dengan batasan dan spesifikasi yang dijelaskan pada bagian sebelumnya, Tabel 2.2 hingga 2.9 memberikan rincian mengenai alat beserta mekanisme verifikasi untuk setiap poin spesifikasi fungsi beserta spesifikasi *tech stack*.

2.3.1 Spesifikasi Fungsi

2.3.1.1 Spesifikasi Fungsi 1

Tabel 2.2 Verifikasi Spesifikasi Fungsi 1

Hal	VNF LCM
Rincian	Siklus hidup VNF dapat terkelola dengan baik, dimulai dari penjalanan, penjalanan ulang, dan penghentian.
Alat Verifikasi	<i>Dashboard</i> yang sedang dikembangkan
Mekanisme Verifikasi	Memproses penjalanan dan memastikan VNF berhasil dijalankan tanpa kesalahan. Kemudian diakhiri dengan menghentikan dan memastikan VNF berhasil dihentikan.

2.3.1.2 Spesifikasi Fungsi 2

Tabel 2.3 Verifikasi Spesifikasi Fungsi 2

Hal	Memodifikasi Konfigurasi VNF
Rincian	VNF yang telah berhasil dibuat dapat dikonfigurasi sebagaimana mestinya sesuai dengan kebutuhan dan hasil yang didapatkan selaras dengan konfigurasi yang dilakukan
Alat Verifikasi	<i>Dashboard</i> yang sedang dikembangkan
Mekanisme Verifikasi	Memastikan VNF dapat dikonfigurasi, menjalankan konfigurasi VNF, dan berakhir dengan memastikan konfigurasi telah berhasil dilakukan dengan hasil yang sesuai tanpa kesalahan

2.3.1.3 Spesifikasi Fungsi 3

Tabel 2.4 Verifikasi Spesifikasi Fungsi 3

Hal	Monitoring OpenRAN CNFs
Rincian	Data KPI UE berhasil didapatkan dan divisualisasikan pada <i>dashboard</i> .
Alat Verifikasi	<i>Dashboard</i> yang sedang dikembangkan
Mekanisme Verifikasi	Mendapatkan data KPI UE menggunakan servis <i>backend</i> yang terintegrasi dengan komponen 5G, kemudian data ditampilkan pada laman monitoring pada <i>dashboard</i> secara real-time.

2.3.1.4 Spesifikasi Fungsi 4

Tabel 2.5 Verifikasi Spesifikasi Fungsi 4

Hal	OpenRAN CNFs <i>Instance Management</i>
Rincian	Manajemen CNF <i>instance</i> dapat dibuat tanpa kesalahan melalui <i>dashboard</i> . Manajemen CNF meliputi pembuatan dan penghapusan <i>instance</i> dalam klaster.
Alat Verifikasi	<i>Dashboard</i> yang sedang dikembangkan
Mekanisme Verifikasi	Memproses pembuatan dan memastikan CNF <i>instance</i> berhasil dibuat tanpa kesalahan. Setelahnya, memproses untuk penghapusan <i>instance</i> dan memastikan CNF <i>instance</i> berhasil dihapus dengan benar.

2.3.2 Spesifikasi *Tech Stack*

2.3.2.1 Spesifikasi *Tech Stack* 1

Tabel 2.6 Verifikasi Spesifikasi *Tech Stack* 1

Hal	Performa <i>Frontend Services</i>
Rincian	Memiliki desain <i>interface</i> yang bersih, responsif, dan mudah dinavigasi. Memastikan bahwa <i>website</i> dapat berfungsi dengan

	baik di berbagai <i>web browser</i> seperti Chrome, Firefox, Safari, dan Edge. Memastikan integrasi dengan <i>backend</i> bekerja dengan baik tanpa kesalahan.
Alat Verifikasi	<i>Web Browser</i>
Mekanisme Verifikasi	Menjalankan <i>dashboard</i> pada <i>web browser</i> untuk memastikan segala fungsionalitas yang dikembangkan dapat bekerja dengan baik secara alur, fungsi, keamanan, aksesibilitas, dan kompatibilitas.

2.3.2.2 Spesifikasi *Tech Stack* 2

Tabel 2.7 Verifikasi Spesifikasi *Tech Stack* 2

Hal	Performa <i>Backend Services</i>
Rincian	Membuat desain dan pengelolaan basis data dengan baik untuk menyimpan dan mengelola data. Menyediakan API yang jelas dan konsisten untuk berkomunikasi antara <i>frontend</i> dan <i>backend</i> . Menerapkan sistem otorisasi yang tepat untuk mengelola hak akses pengguna dengan baik. Menerapkan koneksi websocket untuk pengolahan data secara <i>real-time</i> .
Alat Verifikasi	<i>Web Browser</i> , Postman, dan Django libraries
Mekanisme Verifikasi	Memastikan sistem otentikasi dan otorisasi dengan fokus tiap <i>user</i> hanya dapat mengakses sumber daya nya masing-masing. Menjalankan tes API dan memastikan API <i>calls</i> secara keseluruhan berfungsi dengan benar. Memastikan <i>database</i> telah sepenuhnya aman dari ancaman yang dapat mengganggu kinerja <i>dashboard</i> .

2.3.2.3 Spesifikasi *Tech Stack* 3

Tabel 2.8 Verifikasi Spesifikasi *Tech Stack* 3

Hal	Performa <i>Infrastructure Services</i>
Rincian	Melakukan uji coba untuk memastikan bahwa aplikasi di dalam klaster Kubernetes dapat diakses dan berjalan sepanjang waktu, serta mengkonfigurasi alat pemantauan agar dapat memantau kesehatan dan kinerja aplikasi serta mampu mengotomatisasi siklus hidup pengembangan perangkat lunak.
Alat Verifikasi	Jenkins, Grafana UI, K9S
Mekanisme Verifikasi	Melakukan analisis dan pemantauan visual status <i>node</i> , <i>pod</i> , dan sumber daya lainnya di klaster menggunakan Lens Dashboard untuk memahami keadaan infrastruktur.

2.3.2.4 Spesifikasi *Tech Stack* 4

Tabel 2.9 Verifikasi Spesifikasi *Tech Stack* 4

Hal	Performa <i>CNF Services</i>
Rincian	Memverifikasi desain jaringan secara menyeluruh dengan memastikan bahwa setiap komponen jaringan 5G, seperti Radio Unit (RU), CU, dan DU, terhubung dengan baik satu sama lain. Serta memastikan sistem dapat mengelola dan memproses data pelanggan dengan baik.
Alat Verifikasi	Wireshark
Mekanisme Verifikasi	Melakukan analisis paket dapat dengan mengidentifikasi potensi masalah, seperti kegagalan koneksi atau anomali lalu lintas, serta penerapan yang berfokus pada protokol 5G seperti GTP-U dan sinyal RAN.

BAB 3

DESAIN RANCANGAN SOLUSI

3.1 Alternatif Usulan Solusi

Dalam penelitian ini terdapat beberapa alternatif usulan solusi tersebut merupakan pendekatan *network functions* yang dapat digunakan dalam aplikasi *dashboard* yang dibangun.

3.1.1 Physical Network Functions (PNF)

PNFs adalah konsep dalam jaringan telekomunikasi yang merujuk kepada fungsi jaringan yang diimplementasikan menggunakan perangkat keras fisik [4]. PNF mencangkup layanan seperti *physical access*, *backbone network*, dan *standalone virtual machine*, yang tidak bisa divirtualisasi. PNF mengimplementasikan interaksi antara *interface* dengan tindakan eksternal di luar *hardware*. Oleh karena itu, PNF merujuk pada perangkat fisik dan *network node* seperti *router* dan *switch* [18].

3.1.2 Virtual Network Function (VNF)

Layanan Jaringan 5G telah bertransformasi kedalam bentuk virtualisasi dimana layanan jaringan individual sekarang disediakan sebagai entitas virtual berbasis perangkat lunak yang dikenal sebagai VNFs, yang terlepas dari *middle-box* yang mahal dan khusus [19]. VNF menangani tugas-tugas jaringan, termasuk *routing*, *switching*, *firewall*, dan *load balancer*. Hal ini menghilangkan kebutuhan akan perangkat keras terpisah, khusus, dan mahal dari vendor, memungkinkan layanan jaringan dijalankan pada *generic hardware* atau *Commercial-Off-The-Shelf* (COTS) dengan tingkat komputasi, penyimpanan, memori, dan antarmuka jaringan yang bervariasi. VNF dapat dihosting menggunakan dua teknologi virtualisasi, yaitu Virtual Machine (VM) dan kontainerisasi [20].

3.1.3 Cloud Native Network Function (CNF)

CNF adalah evolusi dari VNF yang dimaksudkan dan dibangun untuk berjalan dalam lingkungan *cloud native* menggunakan kontainer [21]. Kontainerisasi komponen arsitektur jaringan ini bertujuan agar serangkaian layanan yang berjalan pada klaster yang sama dapat terintegrasi lebih mudah pada aplikasi yang sudah dirancang, sambil secara dinamis merutekan lalu lintas jaringan ke masing-masing pod yang sesuai [22]. CNF dapat mengatasi beberapa kendala utama VNF dengan mengalihkan banyak fungsi ke dalam kontainer.

3.2 Analisis dan Pemilihan Solusi

3.2.1 Parameter Analisis Solusi

Berikut ini adalah beberapa parameter yang digunakan untuk pemilihan konsep sistem:

3.2.1.1 Parameter Performasi

Performa *network function* merupakan parameter penting yang mencakup kecepatan, keandalan, dan kapasitas pengolahan data. Ini berkaitan dengan seberapa cepat dan efisien jaringan dapat memproses permintaan dan mengelola trafik data. Dalam konteks alternatif *network function* sebelumnya, performa bisa dipengaruhi oleh *software overhead*, penggunaan sumber daya, dan kemampuan untuk mengoptimalkan alokasi sumber daya komputasi sesuai dengan beban kerja [23]. Kinerja yang optimal diperlukan untuk memastikan bahwa aplikasi dan layanan yang bergantung pada jaringan berjalan dengan lancar dan efisien.

3.2.1.2 Parameter Availability

Availability merujuk pada kemampuan jaringan untuk tetap beroperasi dan dapat diakses kapan pun saat diperlukan. Hal ini sangat penting dalam aplikasi yang bersifat siap untuk produksi, di mana *downtime* jaringan dapat menyebabkan gangguan layanan dan kerugian finansial. *Availability* tinggi sering dicapai melalui redundansi dan mekanisme *failover*, yang memungkinkan jaringan untuk terus beroperasi bahkan ketika ada kegagalan komponen [24]. Dalam konteks alternatif *network function* sebelumnya, parameter *availability* juga mempertimbangkan hal lain seperti waktu *booting*, skalabilitas, dan kemampuan untuk pulih dengan cepat dari *down* komponen.

3.2.1.3 Parameter Kompleksitas

Kompleksitas dalam *network function* berkaitan dengan tingkat kesulitan dalam implementasi, manajemen, dan *maintenance* infrastruktur jaringan dan *software* terkait. Dalam hal alternatif *network function* sebelumnya, kompleksitas sering muncul karena tergantung pada fitur inti atau *kernel* sistem operasi yang menangani jaringan [25]. Selain itu kompleksitas dapat meningkatkan biaya operasional serta memperpanjang waktu untuk *update* dan *deployment*. Pengurangan kompleksitas sering menjadi tujuan utama, karena ini dapat menyederhanakan operasi, mengurangi kemungkinan kesalahan konfigurasi, dan mempercepat proses *update* dan *maintenance*.

3.2.1.4 Parameter Harga

Efisiensi biaya menjadi faktor kunci dalam memilih solusi jaringan di mana organisasi berusaha untuk mengoptimalkan investasi awal sambil meminimalkan biaya uang dikeluarkan. Pertama terdapat biaya kapital (CAPEX) yang berkaitan dengan pengeluaran untuk perangkat keras, perangkat lunak, dan infrastruktur yang diperlukan untuk mendukung fungsi jaringan jangka panjang. Kedua, terdapat biaya operasional (OPEX) mencakup biaya yang berkelanjutan untuk menjalankan dan memelihara infrastruktur tersebut, termasuk tenaga kerja, energi, dan biaya pemeliharaan [26].

3.2.2 Mekanisme Pemilihan Solusi

3.2.2.1 Penilaian Kualitatif

Tabel 3.1 Penilaian Kualitatif dalam Pemilihan Network Functions

Kriteria	PNF	VNF	CNF
Performansi	Dikhususkan untuk tugas-tugas jaringan, menawarkan performa terbaik dengan latensi rendah.	Virtualisasi menambah <i>overhead</i> yang bisa mempengaruhi performa penggunaan resource.	Dioptimalkan untuk lingkungan <i>cloud</i> , mengurangi <i>overhead</i> dan meningkatkan efisiensi,
Availability	Perangkat keras yang andal dan optimasi untuk jaringan menyediakan stabilitas dan ketersediaan yang baik.	Terpengaruh oleh stabilitas infrastruktur virtual. Redundansi dan pemulihan bisa lebih kompleks dibandingkan dengan PNF.	Kemampuan <i>cloud-native</i> seperti orkestrasi otomatis dan kontainerisasi meningkatkan ketersediaan dan pemulihan dari kegagalan.
Kompleksitas	Memerlukan pengetahuan spesifik tentang perangkat keras dan jaringan, serta manajemen fisik yang lebih intensif.	Memerlukan pemahaman tentang virtualisasi dan manajemen VM, namun kurang kompleks dari sisi perangkat keras.	Meskipun memerlukan pengetahuan tentang <i>cloud</i> dan kontainerisasi, pendekatan <i>cloud native</i> cenderung menyederhanakan manajemen dan operasional.
Harga	Memerlukan investasi awal yang besar untuk perangkat keras khusus. Serta biaya pemeliharaan dan <i>upgrade</i> perangkat keras serta energi bisa cukup signifikan.	Mengurangi kebutuhan perangkat keras khusus tapi memerlukan server yang mampu virtualisasi. Memiliki biaya operasi yang lebih rendah daripada PNF karena pengurangan biaya perangkat keras.	Dapat memanfaatkan infrastruktur <i>cloud</i> yang sudah ada, mengurangi kebutuhan akan perangkat keras khusus. Juga efisiensi dalam pengelolaan dan otomatisasi sumber daya <i>cloud</i> mengurangi biaya operasional.

3.2.2.2 Penilaian Kuantitatif

Metode perhitungan kualitatif menggunakan prinsip umum analisis Multi-Criteria Decision Making (MCDM), khususnya metode Weighted Sum Model (WSM). Prinsip dasarnya adalah memberikan bobot pada setiap kriteria untuk menunjukkan pentingnya relatif, dan kemudian mengalikan bobot ini dengan rating untuk setiap opsi atau alternatif. Rumus yang digunakan berdasarkan penelitian terkait [27].

Tabel 3.2 Pengskalaan Rating pada Setiap Kriteria

Kriteria	Parameter	Skala Rating				
		1	2	3	4	5
Performansi	Resource Overhead	> 50%	40% - 50%	30% - 40%	20% - 30%	< 20%
Availability	Uptime Server (Jam/Hari)	< 21,36	21,6 - 22,56	22,8 - 23,28	23,52 - 23,76	≥ 23,88
Kompleksitas	Prosedur Integrasi (Langkah)	≥ 10	7 - 9	5 - 6	3 - 4	≤ 2
Harga (Rp)	Lifetime Perangkat (5 Tahun)	> 1.400.000.000	1.050.000.000 - 1.400.000.000	700.000.000 - 1.050.000.000	350.000.000 - 700.000.000	< 350.000.000

Tabel 3.2 menjelaskan terkait pengskalaan rating yang akan digunakan pada penilaian kuantitatif dalam pemilihan *network functions*. Kriteria-kriteria yang telah ditentukan memiliki parameternya masing-masing sebagai tolak ukur dalam skala rating.

Performansi digambarkan dengan *resource overhead* yang dapat menimbulkan penurunan performa *resource* tersebut. *Availability* digambarkan dengan *uptime server* yang harus tersedia setiap saat. Serta, harga dalam rupiah yang digambarkan dengan *lifetime* perangkat selama 5 tahun rata-rata usia perangkat telekomunikasi.

Tabel 3.3 Prosedur Integrasi pada Setiap Network Function

Langkah Prosedur Integrasi	PNF	VNF	CNF
Pengadaan <i>hardware</i>	v	v	v
Perakitan <i>hardware</i>	v	v	v
Konfigurasi awal sistem operasi	v	v	v
Pemilihan platform virtualisasi		v	v
Integrasi VM dengan <i>hardware</i>		v	v
Pemilihan <i>container orchestrator</i>			v
<i>Auto scaling</i>			v

Kompleksitas menggambarkan seberapa banyak prosedur integrasi yang dilakukan pada setiap solusi. Ini mengacu pada tingkat kesulitan atau kerumitan, semakin banyak prosedur integrasi yang dilakukan, maka rating semakin kecil. Tabel 3.3 menunjukkan seberapa banyak prosedur integrasi yang dilakukan pada setiap solusi [28].

Tabel 3.4 Penilaian Kuantitatif dalam Pemilihan *Network Functions*

Kriteria Seleksi	Bobot	PNF		VNF		CNF	
		Rating	Nilai Bobot	Rating	Nilai Bobot	Rating	Nilai Bobot
Performance	20%	4	0.16	2	0.14	3	0.12
Availability	30%	4	0.24	3	0.18	4	0.24
Complexity	15%	4	0.12	3	0.09	2	0.06
Harga	35%	2	0.14	3	0.12	4	0.28
Total Nilai		0.66		0.56		0.7	
Peringkat		2		3		1	

Tabel 3.4 menunjukkan PNF unggul dalam performa tetapi mahal dan kompleks, cocok untuk tugas intensif. VNF adalah pilihan serbaguna dengan kinerja moderat di semua aspek. CNF menawarkan keseimbangan terbaik antara efisiensi biaya, ketersediaan, dan kompleksitas, menjadikannya ideal untuk lingkungan yang dinamis dan skalabel. Secara keseluruhan, CNF menawarkan solusi yang paling komprehensif, sedangkan PNF dan VNF memiliki keunggulan spesifik mereka sendiri.

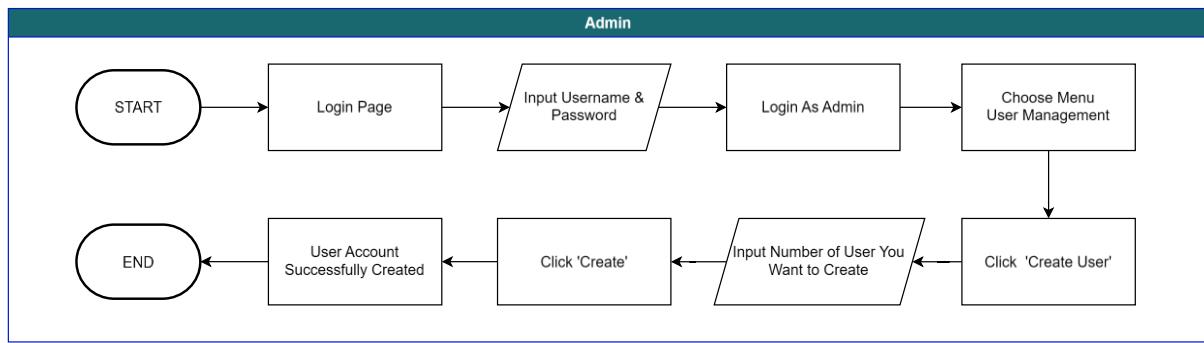
3.3 Desain Solusi Terpilih

Berdasarkan solusi yang telah dipilih yakni CNF, desain detail terkait akan dipaparkan kedalam dua bagian, yakni bagian deskripsi umum dan penjelasan detail. Penelitian ini akan dilakukan dengan pendekatan CNF pada teknologi *cloud native*, sehingga desain detail yang akan dipaparkan meliputi keseluruhan sistem arsitektur yang akan digunakan didalamnya termasuk desain *website*.

3.3.1 Desain Alur Kerja Aplikasi

Sistem aplikasi simulasi 5G dirancang dengan beberapa alur terkait dengan fungsi utama yang dikembangkan agar memiliki ketersediaan dan kinerja optimal, memungkinkan simulasi koneksi E2E secara efisien.

3.3.1.1 Alur Pembuatan Akun User

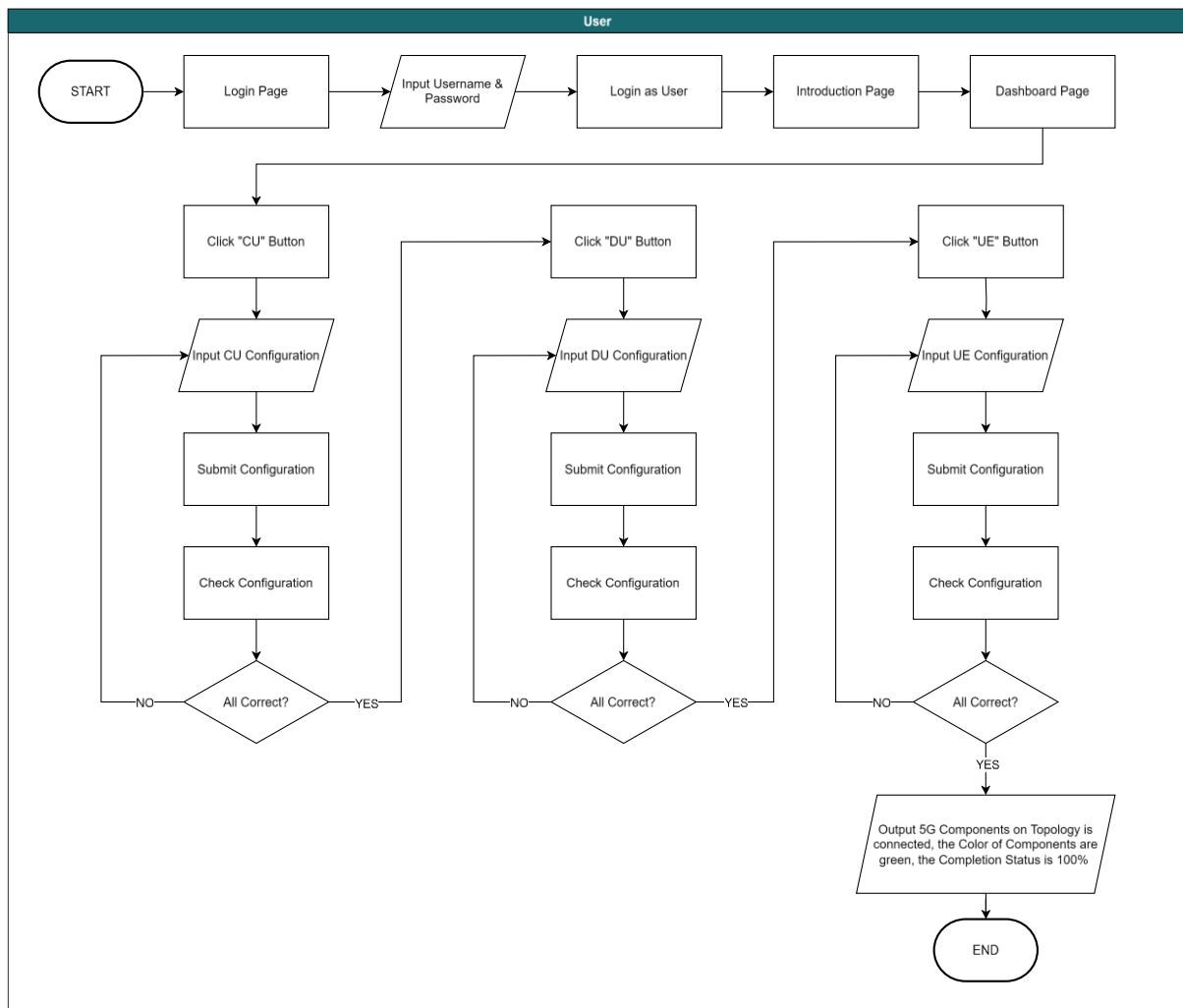


Gambar 3.1 Alur Pembuatan Akun User

Gambar 3.1 menjelaskan bagian konfigurasi akun *user* yang dilakukan oleh admin. Bermula dari halaman *login*, admin memasukkan *username* dan *password* kemudian *login* sebagai admin. Admin memilih menu ‘User Management’ dan dilanjutkan menekan tombol ‘Create User’. Pada menu ini, *admin* akan menginputkan jumlah akun *user* yang ingin dibuat. Setelah *submit*, akun *user* telah berhasil dibuat dan alur telah berakhir.

3.3.1.2 Alur Konfigurasi Komponen RAN

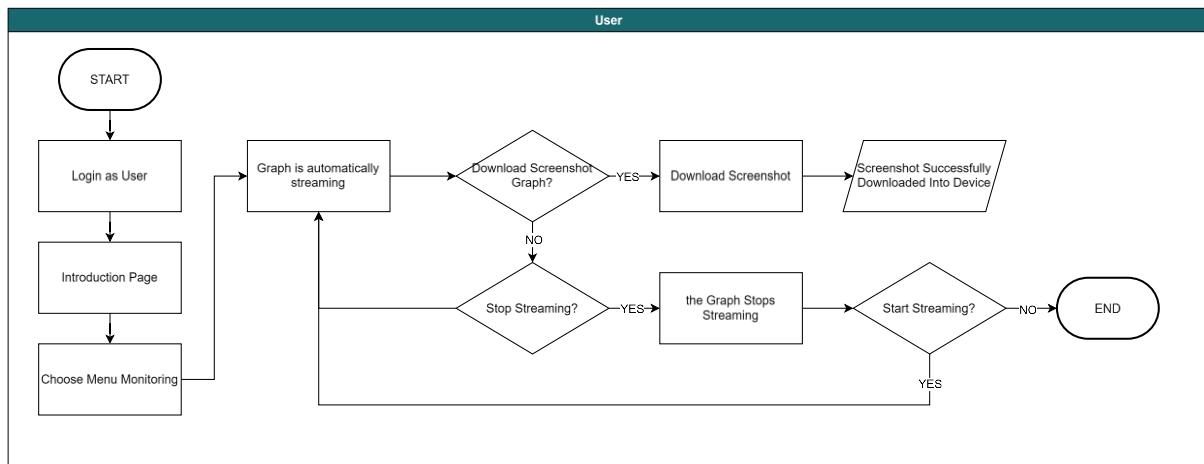
Gambar alur di bawah menjelaskan bagian konfigurasi komponen RAN yang dilakukan oleh *user*. Pada halaman Login, *user* memasukkan *username* dan *password* kemudian *login* sebagai *user*. Lalu user akan diarahkan ke halaman Introduction. Di halaman Introduction terdapat penjelasan materi mengenai jaringan 5G dan panduan untuk mengkonfigurasi tiap komponen RAN 5G. Selanjutnya *user* dapat memilih menu Dashboard untuk mengkonfigurasi komponen RAN. Pada halaman Dashboard ini terdapat topologi grafik jaringan 5G, *configuration panel* untuk mengkonfigurasi komponennya, dan fitur *sniffing Wireshark*.



Gambar 3.2 Alur Konfigurasi Komponen RAN

User menginputkan konfigurasi komponen RAN yang diperlukan pada *configuration panel* dan dilanjut dengan *submit configuration*. Setelah itu, akan muncul *checklist* pada tabel konfigurasi untuk mengecek value konfigurasi yang diinputkan sudah benar atau belum. Jika semua komponen sudah berhasil terkonfigurasi, maka komponen pada topologi akan berwarna hijau. Selain itu juga user dapat melihat log pada komponen. Log berguna untuk mengecek hasil konfigurasi apakah sudah berjalan sesuai ataupun tidak. Apabila kondisi konfigurasi tidak berjalan sesuai, maka user harus mengulangi konfigurasi hingga benar. Apabila konfigurasi telah berjalan sesuai, maka alur konfigurasi komponen RAN telah berakhir.

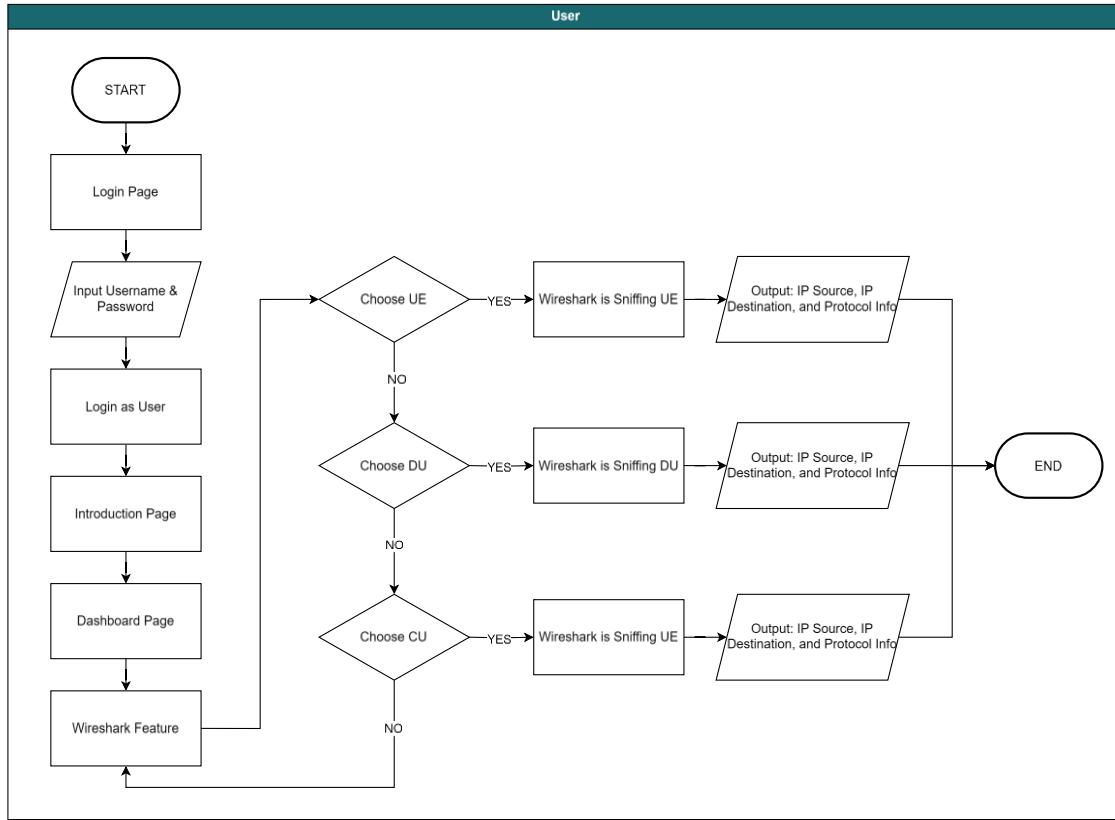
3.3.1.3 Alur Monitoring UE



Gambar 3.3 Alur Monitoring UE

Gambar alur diatas menjelaskan bagian monitoring komponen UE yang dilakukan oleh *user*. Bermula dari *login* sebagai *user*. Pada *navbar*, *user* memilih menu Monitoring untuk memunculkan grafik performansi komponen UE. Di menu Monitoring terdapat tiga grafik yang dapat *user* monitor. Selain itu juga, *user* dapat mengunduh data grafik tersebut dalam bentuk *file* PNG. Untuk mengunduhnya, *user* diharuskan untuk menekan tombol ‘Download Screenshot Graph’. File hasil *screenshot* grafik akan otomatis terunduh dan tersimpan pada *device user*. *User* dapat menekan tombol ‘Stop Stream’ jika ingin grafik berhenti berjalan. Apabila *user* ingin grafik kembali berjalan, maka *user* dapat menekan tombol ‘Start Stream’.

3.3.1.4 Alur Sniffing Wireshark



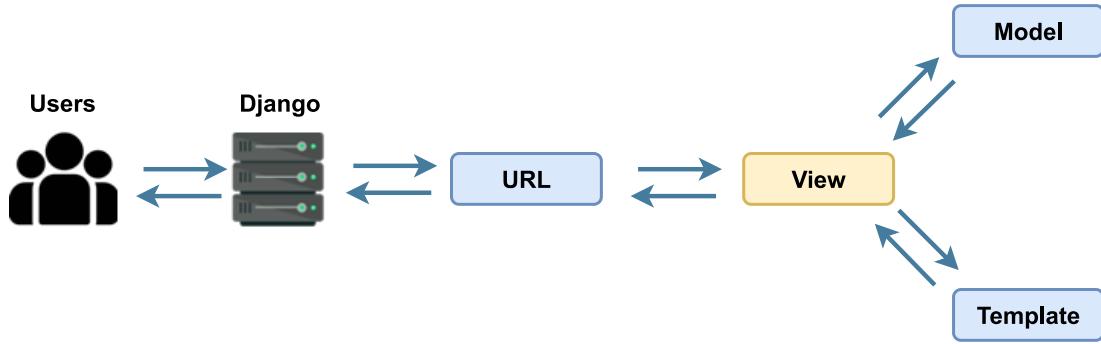
Gambar 3.4 Alur Sniffing Wireshark

Gambar alur di atas menjelaskan bagian *sniffing* Wireshark yang dilakukan oleh *user*. Fitur Wireshark dapat ditemukan pada halaman Dashboard. Untuk bisa mengakses fitur ini, tentunya *user* harus *login* terlebih dahulu. *User* dapat memilih komponen mana yang ingin di-*sniff*. Setelah memilih komponen, maka Wireshark akan melakukan proses *sniffing*. IP source, IP destination, dan protokol info akan ditampilkan. Selain dapat melihat protokol stack, *user* juga dapat memfilter protokol yang diinginkan dan mengunduh file hasil *sniffing* dalam bentuk pcap.

3.3.2 Desain *Backend* Aplikasi

Perancangan desain *backend* mencakup penentuan cara kerja *framework* yang akan digunakan dan merancang arsitektur layanan yang akan dikembangkan. Pengembangan *backend* berfokus pada sisi server sebuah aplikasi yang berkonsentrasi pada *scripting* dan komunikasi dengan *database* [29]. Tujuan dari desain ini adalah untuk menggambarkan sistem *backend* secara rinci, sehingga mempermudah proses pengembangan aplikasi.

3.3.2.1 Django Model, Template, and View (MTV) Pattern



Gambar 3.5 Django MTV Pattern

Penggambaran diatas merupakan cara kerja Django *framework* yang digunakan dalam pengembangan sistem *backend*. Django *framework* mengadopsi Model, View, and Controller (MVC) *pattern* dalam penggunaannya. Hasil adopsi tersebut menghasilkan *pattern* tersendiri yang digunakan pada Django, yakni Model, Template, and View (MTV) *pattern* [30]. Dimana *view* berperan untuk melakukan *request* dan menerima data berdasarkan *model* dan *template* yang digunakan untuk dapat ditampilkan melalui URL.

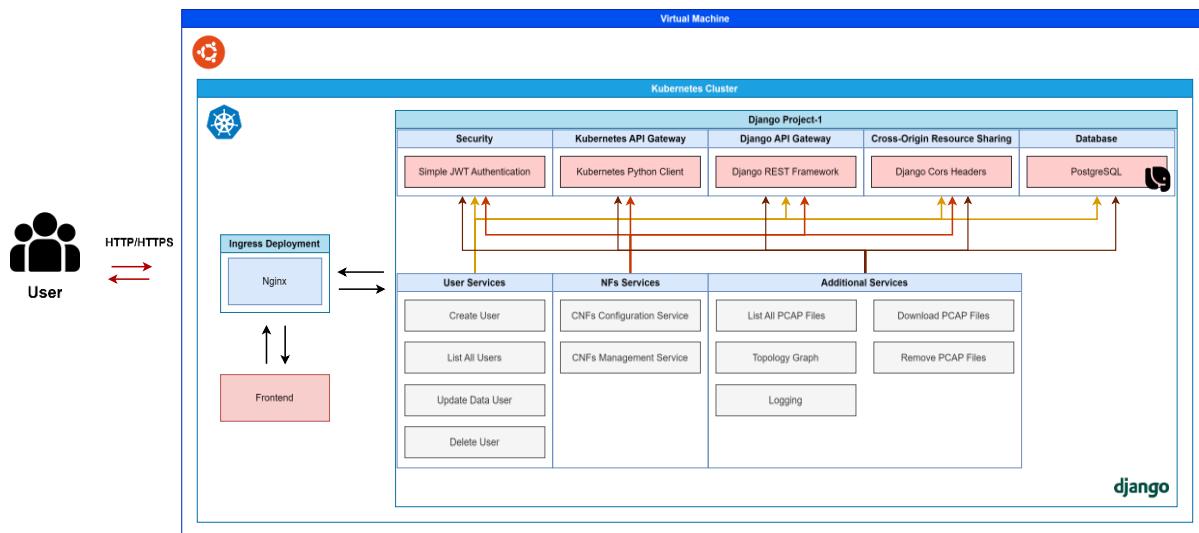
3.3.2.2 Desain Arsitektur Backend

Pengembangan *backend* ini akan menggunakan dua buah proyek Django yang berbeda. Yang pertama akan fokus digunakan untuk implementasi REST API menggunakan *libraries* Django REST Framework yang mengimplementasikan komunikasi *stateless* dan berfokus untuk membuat API *endpoint* yang terotentikasi. Sedangkan proyek kedua akan fokus digunakan untuk implementasi Websocket API menggunakan *libraries* Django Channels yang mengimplementasikan komunikasi *stateful* dan berfokus untuk membuat API *Endpoint* yang dapat mengirimkan dan mengolah data secara *real-time*.

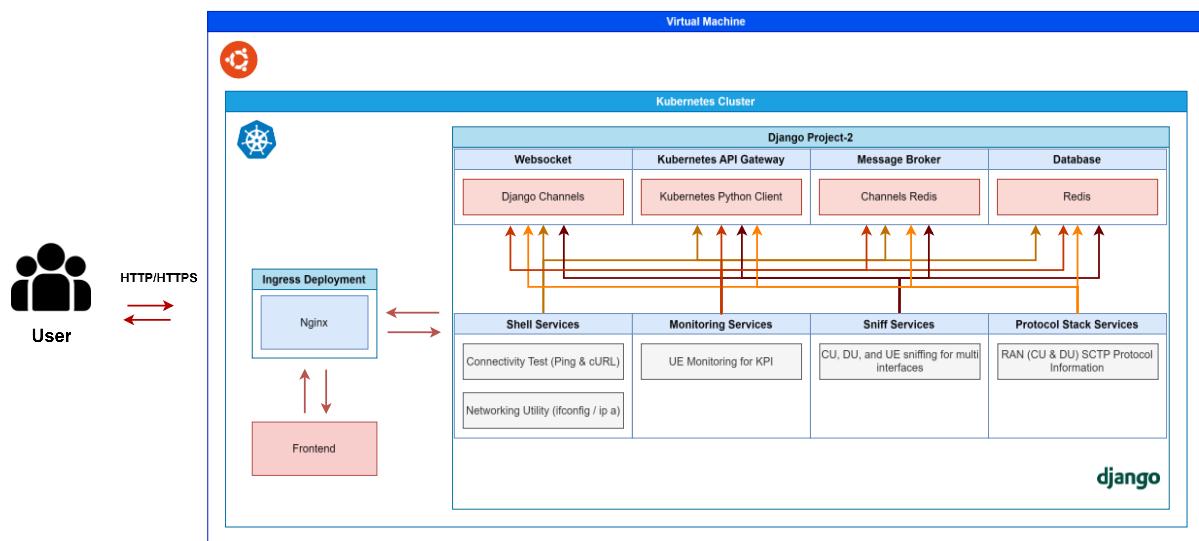
Alasan penggunaan dua proyek Django yang berbeda adalah untuk memudahkan implementasi di dalam klaster Kubernetes dan juga mengisolasi *environment* yang keduanya menggunakan komunikasi yang berbeda antara *stateful* dan *stateless*.

Arsitektur *backend* di bawah menggambarkan keseluruhan infrastruktur *backend* berjalan diatas *virtual machine* dan berada di dalam sebuah kluster Kubernetes. Ingress Nginx berfungsi sebagai jembatan agar infrastruktur *frontend* dapat terkoneksi dengan infrastruktur *backend* melalui alamat Ingress berdasarkan alamat *domain* yang ditetapkan. Selain itu, Ingress Nginx juga berfungsi agar *user* dapat mengakses keseluruhan sistem aplikasi yang dibangun.

Sehubungan dengan digunakannya dua proyek Django yang berbeda untuk mengimplementasikan REST API dan Websocket API, maka dari itu desain arsitektur *backend* akan dibagi menjadi dua bagian yang berbeda.



Gambar 3.6 Desain Arsitektur Backend-REST API



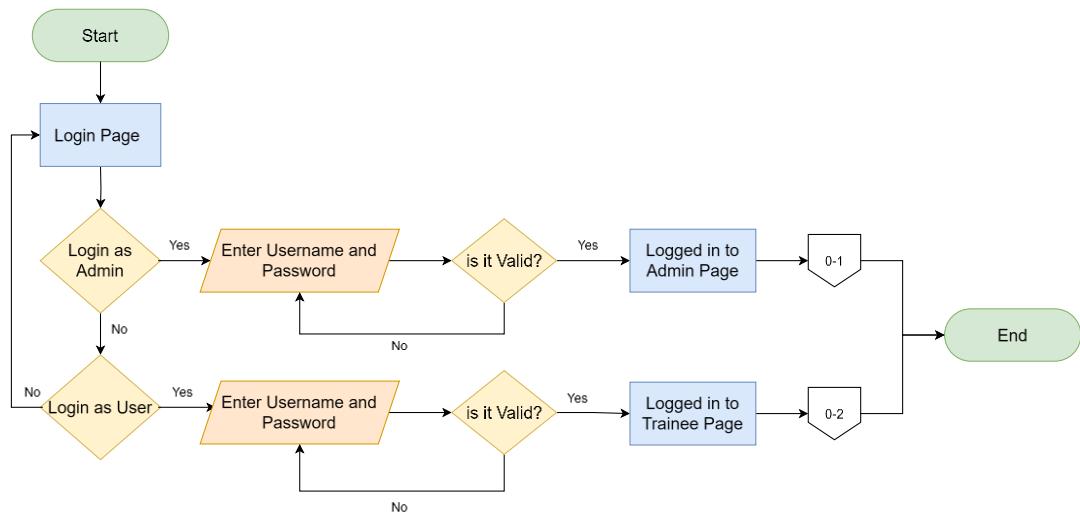
Gambar 3.7 Desain Arsitektur Backend-Websocket

Beragam servis yang dikembangkan memiliki keterkaitan antara satu dengan yang lain membentuk keseluruhan fungsi *website*. Sistem *backend* akan diintegrasikan dengan sistem *frontend* menggunakan Django Cors Headers, sehingga *user* dapat mengakses *website* dengan *User Interface* yang telah didesain pada sisi *frontend*.

3.3.3 Desain Frontend Aplikasi

3.3.3.1 Peran Pada Aplikasi

Desain sistem yang dirancang dan fitur-fitur yang dibuat berdasarkan pada jaringan 5G yang dibangun dan kebutuhan laboratorium TIP dan *user*.



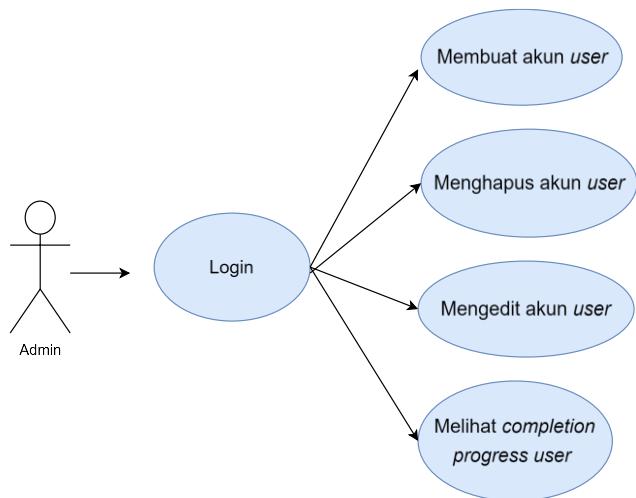
Gambar 3.8 Diagram Alir Inisialisasi Peran

Fungsionalitas peran *website* ini terbagi menjadi dua, yaitu untuk admin selaku *trainer* dan *user* selaku *trainee*. Halaman untuk admin dan *user* berbeda. Halaman untuk admin hanya ada satu yaitu ‘User Management’. Sedangkan *user* memiliki tiga halaman yaitu ‘Introduction’, ‘Dashboard’, dan ‘Monitoring’.

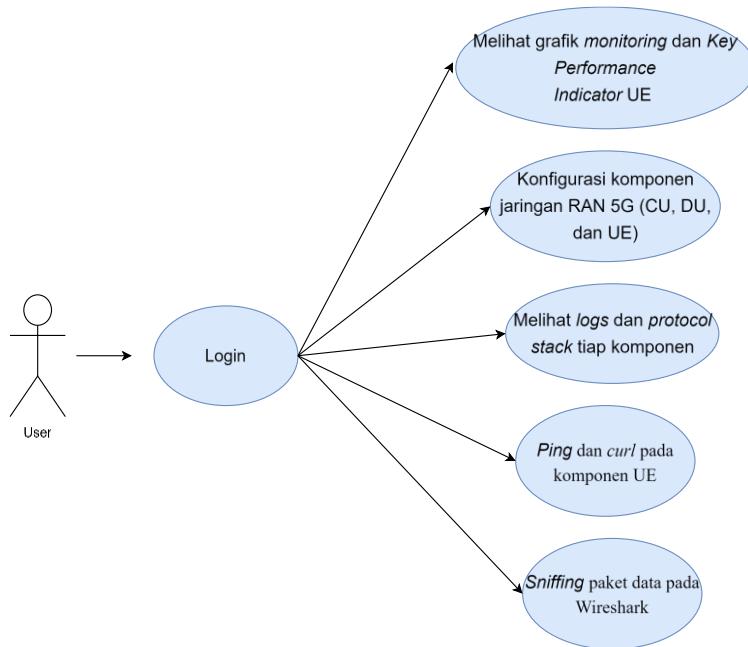
Tabel 3.5 Tabel Daftar Pelaku

No	Pelaku (Actor)	Deskripsi
1.	<i>Admin</i>	Orang yang memberikan pelatihan dan berperan sebagai <i>Trainer</i>
2.	<i>User</i>	Orang yang menjalani pelatihan dan berperan sebagai <i>Trainee</i>

Gambar 3.9 dan 3.10 dibawah merupakan *use case* diagram beserta desain dari tampilan aplikasi yang nantinya akan dibuat.



Gambar 3.9 Use case Diagram Actor Admin



Gambar 3.10 Use case Diagram Actor User

Berikut pada Tabel 3.6 dan Tabel 3.7 adalah deskripsi pendefinisian *use case* diagram berdasarkan diagram *use case* yang digambarkan pada gambar diatas seperti dibawah ini:

Tabel 3.6 Pendefinisian Usecase Diagram Actor Admin

No	Use case	Deskripsi
1.	Melihat completion progress user	Admin dapat melihat completion progress tiap user
2.	Membuat akun user	Admin dapat membuat akun user baru
3.	Menghapus akun user	Admin dapat menghapus akun user
4.	Mengedit akun user	Admin dapat mengedit informasi akun user seperti <i>username</i> dan <i>password</i>

Tabel 3.7 Pendefinisian Usecase Diagram Actor User

No	Usecase	Deskripsi
1.	Melihat grafik <i>monitoring</i> dan KPI UE	<i>User</i> dapat melihat grafik <i>monitoring</i> dan KPI UE
2.	Konfigurasi komponen jaringan RAN 5G (CU, DU, dan UE)	<i>User</i> dapat mengkonfigurasi komponen jaringan RAN 5G (CU, DU, dan UE)
3.	Melihat <i>logs</i> dan <i>protocol stack</i> tiap komponen	<i>User</i> dapat melihat <i>logs</i> dan <i>protocol stack</i> tiap komponen
4.	<i>Ping</i> dan <i>curl</i> pada komponen UE	<i>User</i> dapat <i>ping</i> dan <i>curl</i> pada komponen UE ke internet
5.	<i>Sniffing</i> paket data pada Wireshark	<i>User</i> dapat <i>sniffing</i> paket data komponen UE, DU, and CU pada Wireshark

3.3.3.2 Fitur Aplikasi

3.3.3.2.1 Login Page

Login page adalah halaman yang dirancang khusus untuk memungkinkan pengguna untuk masuk ke dalam sistem atau aplikasi dengan cara mengautentikasi identitas mereka. Namun, di *login page website* ini pengguna tidak perlu *register account* karena akun mereka sudah dibuat oleh admin sehingga pengguna cukup *login* akun saja.

3.3.3.2.2 Introduction Page

Halaman ini berisi penjelasan mengenai Open RAN Configuration Application (ORCA), *guideline* konfigurasi komponen 5G, dan materi mengenai 5G di mana *user* dapat mengeksplor tiap kontennya pada *sidebar*.

3.3.3.2.3 Dashboard Page

Fitur-fitur yang terdapat pada *dashboard page* yaitu grafik topologi jaringan 5G *end-to-end*, Wireshark *realtime*, dan *configuration panel*. Di fitur *configuration panel* dibagi lagi menjadi tiga yaitu konfigurasi *state* (CU, DU, UE). Fitur *configuration panel* berfungsi untuk mengkonfigurasi tiap komponen 5G. Lalu pengguna dapat melihat tiap komponen 5G sudah saling terhubung atau tidak di fitur grafik topologi jaringan 5G *end-to-end* (E2E). Terakhir ada fitur Wireshark untuk menganalisa transmisi paket data dalam jaringan.

3.3.3.2.4 Monitoring Page

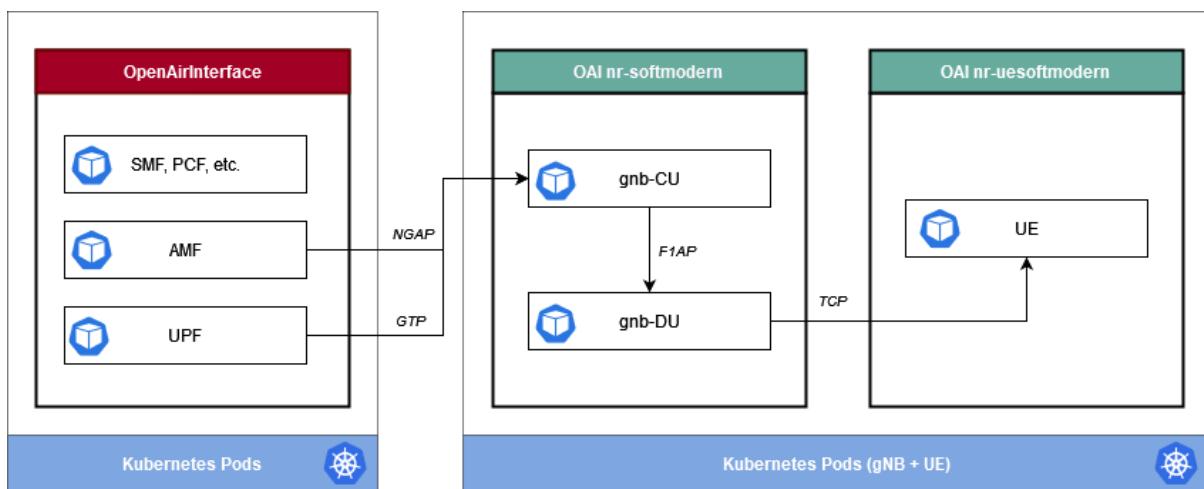
Pada monitoring *page* terdapat UE *metric* dan tabel *key performance indicator*. Di UE *metric page*, pengguna dapat melihat grafik performansi UE. Pengguna juga dapat mengunduh data grafik dalam bentuk file PNG.

3.3.3.2.5 User Management Page

User management page adalah *page* yang hanya bisa diakses oleh admin. *Page* ini dapat dibagi menjadi 4 fitur yaitu *create user*, *completion progress*, *edit user*, dan *remove user* sehingga admin dapat membuat akun baru untuk *user*, merubah *password user*, dan menghapus akun *user*.

3.3.4 Desain Infrastruktur 5G Network

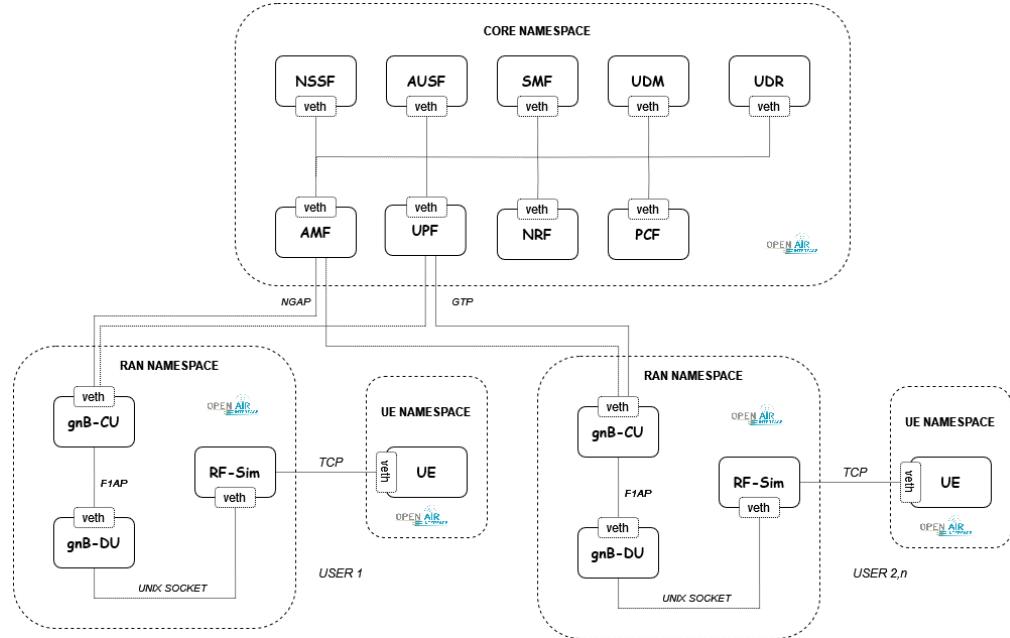
Arsitektur 5G *cloud-native* mengadopsi Service-Based Architecture (SBA) dari UE, sampai ke core berbentuk *container* dengan menggunakan Kubernetes sebagai *platform* utama orkestrasi. Diagram di bawah menggambarkan alur komunikasi E2E dalam arsitektur jaringan 5G, yang dimulai dari pengguna hingga ke internet. Prosesnya dimulai dengan UE, yang berkomunikasi melalui protokol Transmission Control Protocol (TCP) dengan gNodeB Distributed Unit (gNB-DU). gNB-DU ini merupakan komponen dari Open Air Interface (OAI) nr-softmodem yang bertugas untuk menerima dan mengirimkan sinyal radio [31].



Gambar 3.11 Desain Infrastruktur E2E 5G Network

Setelah itu, gNB-CU berkomunikasi dengan *core network* yang diimplementasikan sebagai OAI melalui Next Generation Application Protocol (NGAP). NGAP bertanggung jawab atas pembentukan sesi, manajemen mobilitas, dan signalisasi [32].

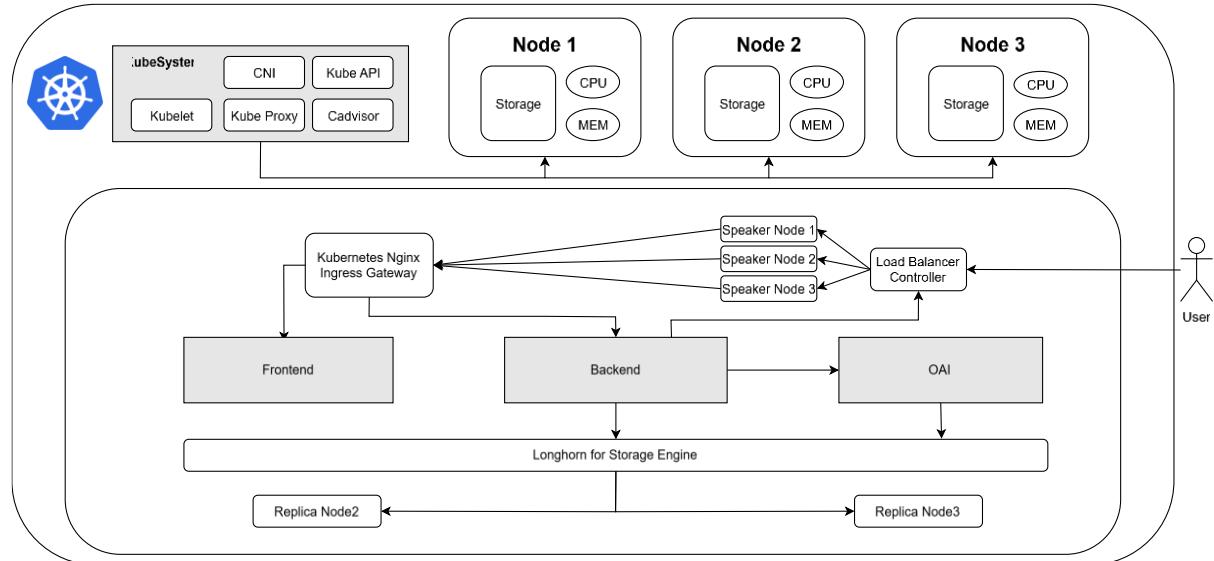
Sedangkan di dalam *core network*, Access and Mobility Management Function (AMF) berinteraksi dengan User Plane Function (UPF) menggunakan GPRS Tunneling Protocol (GTP) untuk mengatur dan mengirimkan lalu lintas data pengguna menuju internet [33].



Gambar 3.12 Arsitektur 5G Koneksi Multi gNB

Dalam penelitian ini, UE memiliki koneksi ke masing-masing gNB-DU melalui sesi yang berbeda. Setiap gNB-DU ini berkomunikasi dengan gNB-CU yang kemudian terhubung ke *core network*. *Functional split* yang digunakan antara gNB-CU dan gNB-DU adalah Option 2 split, yang merupakan bagian dari standar 3GPP untuk jaringan 5G [34]. Dalam split ini, fungsi kontrol dan user plane dipisahkan antara CU dan DU.

3.3.5 Desain Infrastruktur Aplikasi

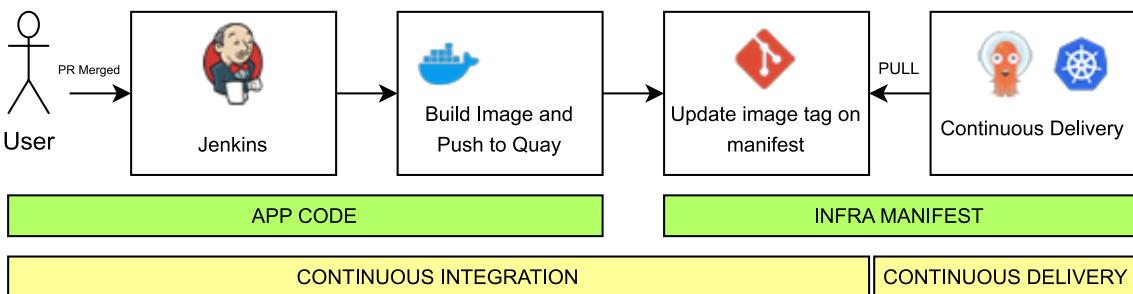


Gambar 3.13 Desain Infrastruktur Aplikasi

Seperi yang tertera pada Gambar 3.11, semua layanan yang tersedia seperti *backend*, *frontend*, dan juga OAI akan berjalan di atas kontainer dan *pod* yang berjalan diatas platform Kubernetes. Pada gambar yang ditampilkan adalah komponen *core* dari sebuah aplikasi, yakni layanan yang akan berjalan, seperti *storage* dan juga *networking*. Dimana layanan adalah klasifikasi dari setiap fungsi dilanjut dengan *storage* yang menggunakan *platform* pendukung bernama *Longhorn* kemudian untuk *networking* ditunjang dengan platform Nginx Ingress yaitu suatu servis yang berjalan untuk menghubungkan antar servis melalui *public endpoint* dengan hasil akhir adalah semua *domain endpoint* [35].

3.3.6 Desain Automasi Pengembangan Perangkat Lunak

Dalam melakukan pengembangan yang cepat, sistem automasi sangat dibutuhkan untuk membantu dalam men-*deploy* aplikasi menggunakan aplikasi *tracking* kode yaitu Git. Aplikasi automasi akan mengantikan peran operator yang menjalankan *command* secara satu persatu yang membutuhkan lebih banyak waktu. Penerapan otomasi menggunakan konsep CI/CD, dimana ini adalah sebuah proses pengembangan perangkat lunak yang menekankan pada integrasi kode secara terus-menerus dan peluncuran fitur baru secara terus-menerus. CI/CD membantu mengurangi waktu yang dibutuhkan untuk mengeluarkan fitur baru dan memastikan bahwa kode yang diintegrasikan ke sistem secara terus-menerus teruji dengan baik sebelum diluncurkan [36].

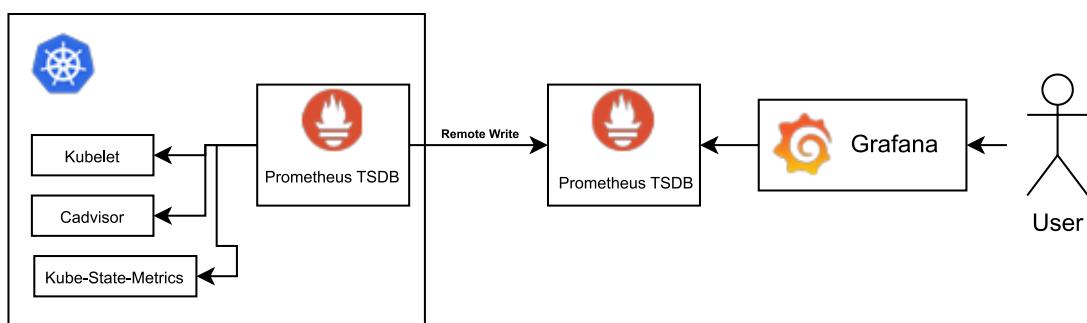


Gambar 3.14 Desain Automasi Pengembangan Perangkat Lunak

Terlihat pada Gambar 3.12 bahwa proses CI/CD terdiri dari beberapa tahap diantaranya *Continuous integration*. Cara kerja *Continuous Integration* adalah setiap kali developer mengupdate kode di repositori, sistem secara otomatis akan melakukan serangkaian test untuk memastikan update tersebut tidak merusak apa pun. Kemudian, terdapat *Continuous Delivery*, dimana setelah semua tahapan dari *Continuous Integration* berhasil, system akan memproses untuk men-*deploy* ke lingkungan produksi. Namun, *deploy* ini tidak langsung otomatis, melainkan butuh ijin dari *developer* atau *team management* dengan memastikan bahwa setiap update kode yang diintegrasikan sudah diuji dengan baik sebelum benar-benar dijalankan [37].

3.3.7 Desain Monitoring Klaster

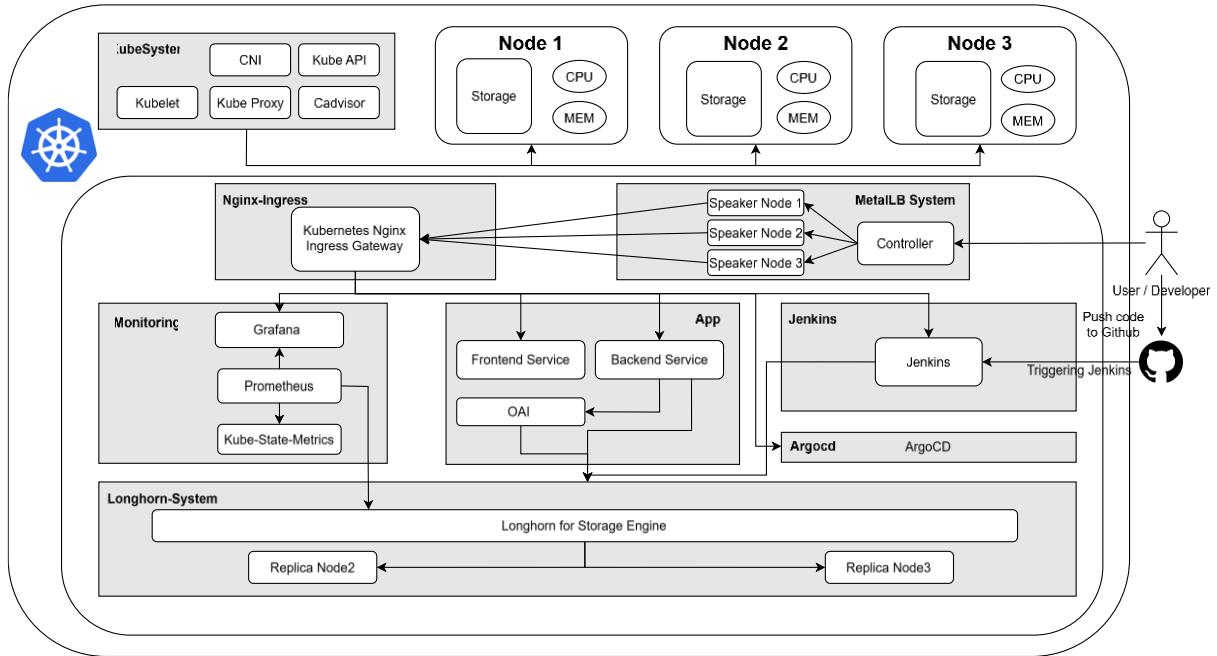
Setelah aplikasi dan infrastrukturnya aktif, tahap berikutnya adalah memastikan semuanya beroperasi dengan baik dan tetap tersedia melalui proses monitoring. Monitoring merupakan kegiatan untuk mengumpulkan dan menganalisis data untuk memahami performa, kapasitas, dan keamanan dari infrastruktur *cloud* yang sedang dioperasikan. Ini dilakukan untuk menjaga sistem *cloud* agar berjalan lancar, mengoptimalkan kapasitas dan biaya, dan juga untuk mengidentifikasi serta menyelesaikan masalah secepat mungkin. Beberapa aspek yang umumnya dipantau mencakup performa sistem, seperti kecepatan dan kapasitas, serta penggunaan sumber daya termasuk *CPU*, *memory*, dan *storage*.



Gambar 3.15 Desain Monitoring Klaster

Data yang diperoleh dari proses monitoring dapat digunakan untuk mengelola kapasitas dan biaya sistem, serta untuk memprediksi dan mencegah masalah yang mungkin terjadi di masa depan. Sistem monitoring yang dirancang menggunakan konsep aplikasi basis data urutan waktu yang dapat menyimpan metrik *resource* aplikasi maupun *resource* infrastruktur seperti yang ditunjukkan pada gambar aplikasi tersebut berjalan di lingkungan kubernetes dan melakukan agregasi terhadap metrik infrastruktur dan aplikasi dari program produser metrik. Data yang disimpan dalam aplikasi tersebut selanjutkan akan divisualisasikan menggunakan *dashboard* terpusat yang berisikan panel-panel berkaitan dengan data metrik seperti CPU, Memori, Disk dan *Network Connection*.

3.3.8 Desain Keseluruhan Aplikasi pada Kubernetes



Gambar 3.16 Desain Keseluruhan Aplikasi pada Kubernetes

Pada Gambar 3.14 digambarkan keseluruhan arsitektur aplikasi beserta servis penunjang jalannya aplikasi seperti *storage engine*, *logging*, monitoring, klaster kubernetes, dan CI/CD. Ketika pengembang melakukan perubahan pada code dan push ke Git, maka Git akan men-trigger Jenkins untuk melakukan serangkaian proses membuat image berdasarkan code yang terbaru pada Git. Kemudian image tersebut segera di-deploy pada klaster Kubernetes, sehingga servis yang dijalankan selalu memiliki versi yang terbaru berdasarkan update kode terakhir.

3.4 Jadwal Pengerjaan

3.4.1 Jadwal Capstone

Rencana jadwal kegiatan dalam penelitian ini telah disusun berdasarkan penelitian yang akan dilakukan. Jadwal kegiatan meliputi tahap implementasi rancangan, tahap *testing*, dan tahap evaluasi yang dimulai dari bulan Januari 2024 dan diakhiri pada bulan Juni 2024. Pada Tabel 3.8 akan dijelaskan terkait detail rencana jadwal kegiatan pada penelitian.

Tabel 3.8 Jadwal Capstone

Rencana Kegiatan		Januari 2024				Februari 2024				Maret 2024				April 2024				Mei 2024				Juni 2024				
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	
		Tahap Implementasi Rancangan																								
Backend	Inisiasi Django Project																									
	Implementasi User Service																									
	Implementasi JWT Authentication																									
	Implementasi Kubernetes Python Client																									
	Integrasi Kubernetes Python Client kedalam Django Views																									
	Integrasi Komponen OAI																									
	Integrasi data API kedalam infrastruktur																									
	Integrasi Backend service dengan Frontend service																									
Frontend	Fixing Bug and Deployment (Development)																									
	Create User Interface Design																									
	Slicing UI to Framework																									
5G	Consume API																									
	Core Network Setup																									
	Radio Access Network Setup																									
	Integrasi VNF to VNF																									
	Integrasi VNF to Backend																									
DevOps	Configuration Management																									
	Preparing Cluster																									
	Configuring CI/CD																									
	Configuring Logging																									
Tahap Testing																										
All Team	Testing Aplikasi																									
Tahap Evaluation																										
All Team	Melakukan Evaluasi Hasil Testing																									
	Fixing Bug dan Evaluasi Ulang																									

3.4.2 Anggaran

Penelitian kali ini memanfaatkan infrastruktur virtual, *open-source*, dan fasilitas dari TIP dapat dijalankan tanpa anggaran tambahan karena mengurangi kebutuhan akan perangkat keras melalui virtualisasi, mengeliminasi biaya lisensi dengan menggunakan perangkat lunak *open-source*, dan menekan biaya infrastruktur.

BAB 4

IMPLEMENTASI

4.1 Deskripsi Umum Implementasi

Implementasi aplikasi *software* pada penelitian ini berfungsi sebagai *controller testbed 5G* yang membutuhkan persiapan menyeluruh dari berbagai aspek. Pengembangan pada sisi *backend* dilakukan menggunakan Django yang merupakan salah satu *framework* dari Bahasa pemrograman Python. Selain itu, didukung dengan *database* PostgreSQL yang digunakan untuk menyimpan keseluruhan data yang berkaitan dengan aplikasi. Implementasi pada sisi *backend* akan meliputi beberapa aspek yang saling terhubung langsung dengan 5G *cloud* maupun *frontend*. Implementasi *backend* akan dipecah menjadi beberapa poin yang terstruktur dengan upaya menjelaskan keseluruhan proses implementasi. Seluruh baris perintah yang terkait dengan *source codes* akan diimplementasi menggunakan Operating System (OS) Linux.

Dari sisi 5G aplikasi ini didasarkan pada solusi OpenAirInterface dan dirancang untuk memfasilitasi berbagai tahapan konfigurasi dan operasi dalam lingkungan 5G. Tahapan pertama melibatkan konfigurasi E2E yang memungkinkan pengguna untuk melakukan monitoring, *sniffing*, dan konfigurasi *real-time* melalui *user interface* yang terintegrasi dengan komponen UE, DU, CU, hingga ke CN, memungkinkan pengujian jaringan 5G yang sederhana.

Tabel 4.1 Spesifikasi VM Kluster Kubernetes

Node	vCPU	RAM	Storage	OS	IP Address
node1 (master)	8	8 GiB	18 GiB	Ubuntu Server 22.04	10.30.1.212
node2 (worker)	16	23 GiB	85 GiB		10.30.1.213
node3 (worker)	16	23 GiB	85 GiB		10.30.1.214

Dari sisi infrastruktur, kami membangun kluster Kubernetes dan aplikasi-aplikasi pendukung diatas VM. Pada Tabel 4.1 dijelaskan spesifikasi VM yang digunakan untuk kluster Kubernetes dan aplikasi-aplikasi pendukungnya seperti Jenkins sebagai alat *automation* yang mendukung proses CI, Longhorn menyediakan backup block storage, Grafana untuk melakukan visualisasi data *monitoring*, dan ArgoCD mengotomatiskan *deployment* aplikasi di Kubernetes dengan sinkronisasi dari Git.

4.2 Detil Implementasi

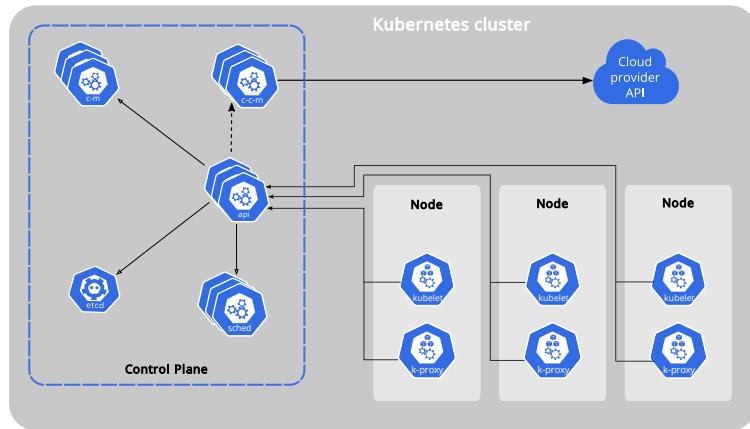
4.2.1 Implementasi Infrastruktur

Infrastruktur berperan menyediakan wadah untuk menjalankan aplikasi pada sumber daya yang tersedia. Aplikasi harus dipastikan dapat berjalan dengan baik dan mampu berkomunikasi satu sama lain didalam kluster Kubernetes. Selain itu, terdapat fitur pendukung dalam infrastruktur seperti *automation*, *monitoring*, dan *storage management* yang meningkatkan efisiensi waktu, akurasi, dan keandalan. Dalam infrastruktur, fitur-fitur tersebut diimplementasi dengan berbagai teknologi seperti Kubernetes, Grafana, Prometheus, Longhorn, Jenkins, dan ArgoCD.

4.2.1.1 Kluster Kubernetes

4.2.1.1.1 Arsitektur Kluster Kubernetes

Sebagai platform orkestrasi kontainer, Kubernetes terdiri dari sejumlah komponen penting untuk membentuk kluster. Pada gambar 4.1, setiap komponen bekerja bersama untuk menjalankan dan mengelola aplikasi yang berjalan dalam kontainer [38].



Gambar 4.1 Arsitektur Kubernetes

Berikut adalah beberapa penjelasan implementasi komponen yang digunakan dalam kluster Kubernetes diantaranya:

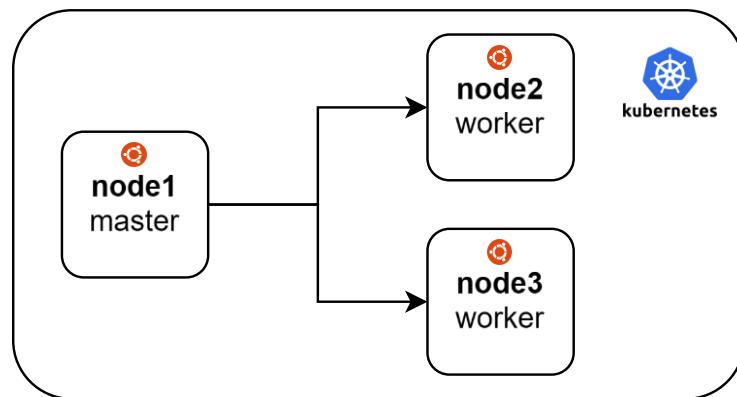
- **Kubernetes API server** yang menjadi komponen utama Kubernetes yang bertindak sebagai *gateway* untuk semua API *request*, memproses dan menyimpan data di etcd, serta mengelola *resource* dan objek Kubernetes.
- **Kube-scheduler** merupakan komponen yang bertugas untuk menentukan VM mana yang paling cocok untuk menjalankan *pod* berdasarkan kapasitas *node*.
- **Kube-controller-manager** menjadi komponen yang menjalankan fungsi kontrol untuk mengelola objek Kubernetes yang terkait dengan mengelola replikasi *pod*.

- **Kube-proxy** bertanggung jawab untuk mengatur jaringan di dalam kluster, seperti memastikan setiap *request* diarahkan ke *pod* yang tepat.
- **Kubelet** merupakan agen yang berjalan pada setiap VM dalam kluster dan bertanggung jawab untuk memastikan bahwa *container* dalam *pod* berjalan dengan baik dan sesuai dengan spesifikasi yang telah ditetapkan.
- **ETCD** merupakan *database* yang berfungsi menyimpan konfigurasi dan keadaan semua objek yang ada pada kluster, termasuk *pod*, *service*, *deployment*, dan lainnya.

4.2.1.1.2 Instalasi Kluster Kubernetes

Dapat dilihat pada gambar 4.1, kami membangun kluster Kubernetes menggunakan 3 VM, dengan 1 VM sebagai *master* dan 2 VM lainnya sebagai *worker*. Dalam Kubernetes, *master* bertanggung jawab untuk mengelola kluster dengan menjalankan komponen-komponen seperti *API server*, *scheduler*, dan *controller manager* yang mengatur penjadwalan, pengendalian, dan pengaturan kluster.

Sedangkan *worker* berperan menjalankan *pod* yang berisi aplikasi dan layanan yang dikelola oleh kluster, serta berkomunikasi dengan *master* melalui kubelet untuk menjalankan tugas yang telah dijadwalkan. *Master* mengatur keseluruhan infrastruktur dan mengawasi kinerja kluster, sementara *worker* menjalankan beban kerja aplikasi.



Gambar 4.2 Arsitektur VM Kluster

Penggunaan dua *worker node* dalam sistem Kubernetes didasari oleh batasan yang mengizinkan maksimal 110 *pod* setiap *worker node*. Berdasarkan estimasi, kluster kami menjalankan antara 120 hingga 150 *pod*, dengan jumlah yang meningkat seiring bertambahnya jumlah pengguna. Oleh karena itu, memiliki dua *worker nodes* meningkatkan redundansi dan keandalan infrastruktur, sehingga mampu menangani kegagalan yang tidak terduga, seperti kegagalan salah satu *node*.

Kluster Kubernetes dibangun menggunakan Kubespray sebagai *deployer*. Kubespray menggunakan *automation tool* bernama Ansible yang akan menginstall *dependencies* yang diperlukan untuk setiap *node*. Implementasi Kluster Kubernetes menggunakan Kubespray diringkas pada Lampiran 4.1 sesuai spesifikasi yang ada pada Tabel 4.1. Setelah membangun kluster Kubernetes menggunakan Kubespray, verifikasi konfigurasi dan fungsi sistem dapat dilakukan dengan kubectl. Seperti pada Lampiran 4.2, kubectl digunakan untuk memeriksa status semua VM dan memastikan mereka siap menjalankan *workload*.

Selain itu, verifikasi juga dilakukan dengan men-*deploy* sebuah ingress nginx sebagai *proxy* atau *gateway* awal dari setiap *client* yang mengakses ke *pod*. Hasilnya terlihat pada Lampiran 4.2, semua objek nginx ingress sudah di-*deploy* dan berjalan dengan baik dilihat dari status *pod* yang sudah *running*.

4.2.1.1.3 Mengelola Kluster Kubernetes

Dalam mengelola aplikasi pada kluster Kubernetes, terdapat dua metode yang sering digunakan yaitu melalui *file manifest* dan juga menggunakan Helm. *File manifest* dalam Kubernetes adalah *file YAML* yang mendefinisikan sumber daya yang diperlukan untuk aplikasi, seperti *pod*, *deployment*, *service*, dan lain-lain. Administrator mendefinisikan konfigurasi dari setiap komponen yang diinginkan, termasuk jumlah replika, *image* kontainer yang digunakan, *port* yang dibuka, dan *environment variable*. Untuk menerapkan *manifest*, digunakan perintah `kubectl apply`, yang memerintahkan Kubernetes untuk membuat atau memperbarui *resource* yang sesuai dengan definisi di *file manifest*.

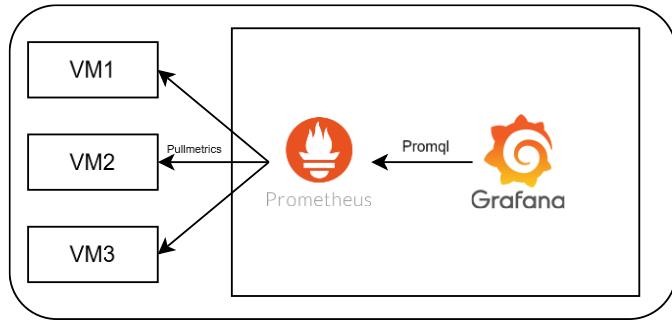
Sedangkan Helm merupakan alat manajemen paket yang disebut *charts*, yang merupakan koleksi *file* yang mendeskripsikan set aplikasi yang terkait. *Chart* Helm memungkinkan definisi, instalasi, dan *upgrade* bahkan aplikasi Kubernetes yang paling kompleks sekalipun secara otomatis. Setiap *chart* dapat mengandung metadata yang mendeskripsikan versi *chart* dan aplikasi yang diinstal, serta *template YAML* untuk mendefinisikan *resource* Kubernetes.

4.2.1.2 Monitoring

4.2.1.2.1 Arsitektur *Monitoring*

Proses *monitoring* memantau tiga VM yang dibangun menjadi Kluster Kubernetes. Tiga VM tersebut diamati berdasarkan berbagai parameter seperti CPU, memori, penyimpanan, dan lain-lain. Nilai dari setiap parameter tersebut menentukan apakah diperlukan tindakan lebih lanjut atau tidak.

Sistem kami menggunakan Prometheus dan Grafana untuk *monitoring* kinerja kluster dan aplikasi. Prometheus adalah sebuah platform *open-source* yang digunakan untuk mengawasi kinerja sistem dan aplikasi, sementara Grafana juga platform *open-source* yang berperan memvisualisasikan data dari berbagai sumber seperti Prometheus [39]. Kombinasi penggunaan kedua platform ini sangat umum untuk efektifitas dalam pemantauan dan analisis kondisi sistem serta aplikasi.



Gambar 4.3 Arsitektur Monitoring

Seperti yang ditunjukkan pada Gambar 4.3, Prometheus dan Grafana bekerja sama dalam memonitoring sumber daya. Prosesnya dimulai dengan menginstal *node exporter* di VM untuk mengumpulkan metrik. Prometheus kemudian mengambil metrik ini dari *node exporter* setiap VM dan menyimpannya dalam *internal database* dengan tipe *time series*. Grafana menggunakan data metrik dari Prometheus untuk dikelola dan divisualisasikan dalam bentuk *dashboard*. *User* dapat menyesuaikan *dashboard* sesuai kebutuhan visual mereka. Selain itu, Grafana dapat mengirimkan notifikasi jika ada metrik yang melebihi batas *threshold* yang telah ditetapkan.

4.2.1.2.2 Instalasi *Monitoring*

Untuk mengaplikasikan sistem *monitoring*, langkah pertama adalah memastikan setiap VM sudah memiliki *node exporter*. Fungsinya adalah untuk memberikan data terkait kondisi VM secara *real-time*. Data yang diberikan adalah berupa metrik dan *value* yang berubah-ubah setiap saat. Pada Lampiran 4.3 adalah verifikasi jika *node exporter* pada VM sudah bekerja dengan baik.

Kemudian selanjutnya menginstall Prometheus yang akan digunakan untuk mengambil metrik dan *value* dari *node exporter* pada setiap VM serta mengumpulkannya. Kami menggunakan helm untuk instalasi Prometheus dan Grafana. Verifikasi Prometheus sudah berjalan dengan baik terdapat pada Lampiran 4.4. Terakhir adalah menginstall Grafana sebagai visualisasi metrik dari setiap VM. Karena Grafana juga sudah diinstall menggunakan Helm, verifikasi instalasi grafana terdapat pada Lampiran 4.5.

4.2.1.2.3 Mengelola *Monitoring*

Langkah pertama dalam manajemen *monitoring* adalah dengan menambahkan alamat setiap VM yang sudah terinstall *node exporter* pada Prometheus. Ini dilakukan agar Prometheus dapat mengenali dan dapat mengambil data pada masing-masing *node exporter* dan menyimpannya pada *database*.

```
10.30.1.212:9100/metrics  
10.30.1.212:9100/metrics  
10.30.1.212:9100/metrics
```

Selanjutnya adalah manajemen Grafana yang berperan sebagai platform yang akan digunakan oleh *user* untuk melihat kondisi terbaru dari VM melalui visualisasi. Agar dapat menampilkan data *metric* dalam bentuk visualisasi, dapat dilakukan dengan menambahkan *endpoint* Prometheus di Grafana.

```
10.30.1.214:9000
```

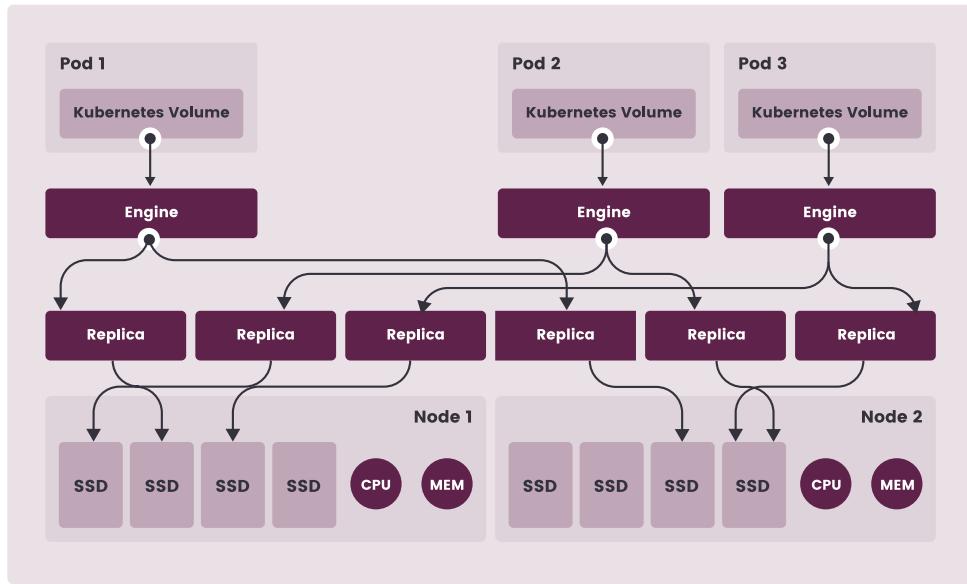
Terakhir adalah memodifikasi setiap *dashboard* sesuai dengan metrik yang diinginkan. Grafana menggunakan bahasa PromQL untuk mem-*parsing* setiap data dari *data source* yaitu prometheus. Penggunaan format PromQL dapat disesuaikan dengan tampilan metrik yang diinginkan.

4.2.1.3 *Distributed Block Storage*

4.2.1.3.1 Arsitektur *Distributed Block Storage*

Longhorn adalah solusi *block storage* terdistribusi yang dirancang khusus untuk Kubernetes dengan pendekatan *cloud native*. Ini memecah data menjadi blok-blok kecil, menyediakan efisiensi dan kecepatan tinggi dalam operasi data. Longhorn mendukung *data replication*, *snapshot*, dan *backup*, yang memastikan ketersediaan dan keandalan data yang tinggi. Integrasinya dengan Kubernetes memungkinkan manajemen penyimpanan yang otomatis dan skalabel.

Penggunaan Longhorn dalam sistem kami adalah untuk manajemen penyimpanan yang digunakan oleh beberapa aplikasi seperti Jenkins, *backend*, dan OAI. Berdasarkan arsitektur longhorn pada Gambar 4.4, Longhorn akan mereplika volume yang digunakan *pod* ke setiap VM *worker*.



Gambar 4.4 Arsitektur Distributed Block Storage

Replika merupakan hal yang krusial dalam manajemen penyimpanan untuk meningkatkan reliabilitas data. Pada kluster Kubernetes, jika terdapat satu *node* yang *down* maka data masih tetap aman karena masih tersedia di VM yang lain sehingga *pod* tetap dapat berjalan.

4.2.1.3.2 Instalasi *Distributed Block Storage*

Instalasi Longhorn pada kluster Kubernetes adalah dengan menggunakan metode Helm. Penggunaan metode Helm disebabkan karena Longhorn memiliki banyak objek yang di instal seperti *Deployment*, *ReplicaSet*, *Service*, *dll*.

Sehingga dengan menggunakan Helm, dapat lebih mudah menginstal seluruh objek tersebut terutama dalam manajemen parameter yang ingin disesuaikan sebelum melakukan penginstalan. Ketika Longhorn sudah terinstal dengan baik pada kluster Kubernetes, maka *dashboard* Longhorn sudah dapat diakses sesuai dengan yang tertera pada **Lampiran 4.6**. Longhorn juga dapat mengklasifikasi volume menjadi beberapa *state* seperti *Healthy*, *Degraded*, *dll*. Sehingga menjadi lebih mudah bagi *Infrastructure Administrator* untuk memanajemen volume yang ada.

4.2.1.3.3 Manajemen *Distributed Block Storage*

Ketika men-deploy sebuah aplikasi yang membutuhkan volume pada kluster, dibutuhkan sebuah objek Kubernetes bernama Persistent Volume Claim (PVC) yang mendefinisikan kebutuhan kapasitas *storage*, *access mode*, dan *storage class*. Longhorn tersedia pada Kubernetes sebagai sebuah *Storage Class*, sehingga ketika membuat sebuah PVC, perlu ditambahkan parameter *Storage Class* dengan *value* Longhorn agar Longhorn dapat memberikan volume yang di *request* serta dapat memantau volume tersebut.

```

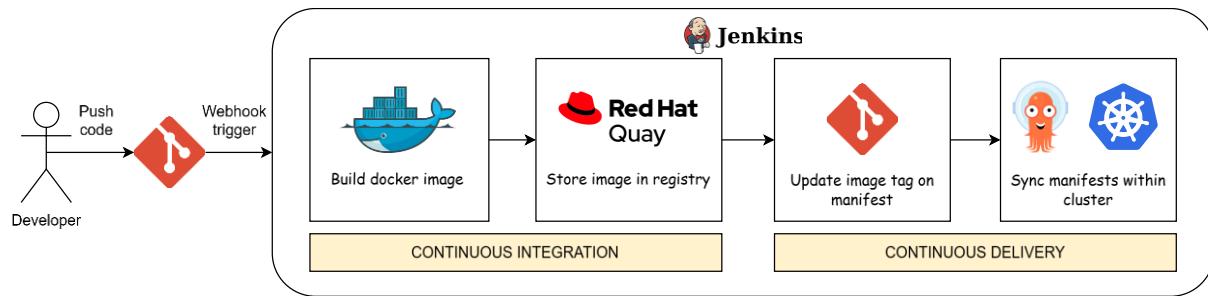
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example
  namespace: example
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: longhorn
  resources:
    requests:
      storage: 2Gi

```

4.2.1.4 Automation

4.2.1.4.1 Arsitektur Automation

Jenkins adalah platform open-source yang mengotomatiskan proses pembangunan, pengujian, dan pengiriman aplikasi perangkat lunak. Jenkins menyederhanakan siklus hidup pengembangan perangkat lunak, termasuk kompilasi kode, pengujian otomatis, dan deployment ke lingkungan kluster. Dalam sistem kami, Jenkins memainkan peran penting dalam mengoptimalkan lifecycle setiap aplikasi dengan menerapkan proses CI/CD.



Gambar 4.5 Arsitektur Automation

Kami menggunakan *automation* untuk mengotomasi setiap perubahan yang dilakukan oleh pengembang dari *build* aplikasi hingga *deploy* aplikasi ke lingkungan kluster. Pada Gambar 4.5 adalah *flow* otomasi yang akan dilakukan, dimulai dari pengembang yang membuat perubahan kode dan *push* ke Github, selanjutnya Github akan mengirimkan *trigger* berupa *webhook* kepada Jenkins. Dalam hal ini, Github memberi tahu Jenkins bahwa ada perubahan terbaru pada kode.

Jenkins menerima *trigger* tersebut dan kemudian menjalankan *pipeline* yang berisi serangkaian proses untuk membuat aplikasi berdasarkan kode terbaru. Terakhir, setelah aplikasi berhasil di-*build*, Jenkins men-*deploy* aplikasi terbaru tersebut kedalam kluster Kubernetes melalui proses *Continous Delivery* dengan bantuan ArgoCD agar aplikasi dapat segera digunakan.

4.2.1.4.2 Instalasi *Automation*

Jenkins diinstall didalam kluster Kubernetes menggunakan metode *file manifest*. Lampiran 4.7 menunjukkan Jenkins sudah berhasil diinstall dan menampilkan *dashboard* awal dari Jenkins. Selanjutnya, Argo CD juga perlu diinstall untuk mendukung proses CD yang lebih mudah dan efektif. Instalasi Argo CD menggunakan metode Helm karena Argo CD memiliki banyak objek yang perlu diinstall pada Kubernetes sehingga lebih mudah dalam proses instalasi. Lampiran 4.8 menunjukkan tampilan *dashboard* ketika Argo CD sudah berjalan.

4.2.1.4.3 Manajemen *Automation*

Otomasi pada Jenkins dapat didefinisikan sebagai *pipeline* yang berupa rangkaian otomatis dari langkah-langkah atau proses yang dijalankan untuk membangun, menguji, dan mengirimkan aplikasi perangkat lunak. *Pipeline* ini dirancang menggunakan sintaks khusus dalam bentuk *file Jenkinsfile*, yang memungkinkan pengguna mendefinisikan setiap tahap dalam proses pembangunan dan penerapan aplikasi secara detail.

Sistem kami menggunakan serangkaian proses pada *pipeline* yaitu *Setup Environment*, *Build Image*, *Push Image*, *Change Image Tag*, dan juga ArgoCD Sync sesuai dengan Lampiran 4.9. Adapun potongan kode *pipeline* berupa Jenkinsfile dapat dilihat pada Lampiran 4.10.

4.2.2 Implementasi 5G *Cloud*

Dalam implementasi jaringan 5G E2E pada subbab ini menekankan penjelasan detil setiap konfigurasi *core network* dan perangkat RAN untuk memfasilitasi interaksi antar komponen jaringan. Berbagai skenario antara gNB dan UE diterapkan sehingga *user* harus melakukan konfigurasi yang beragam disesuaikan dengan kondisi lapangan. Dalam konteks keterbatasan *resource* komputasi, pendekatan rekayasa yang efisien dieksplorasi untuk memastikan operasi jaringan tetap optimal.

4.2.2.1 *Core Network*

Konfigurasi *core network* OAI (`config.yaml`) menjadi kunci dalam menentukan bagaimana masing-masing NF beroperasi dan berinteraksi dalam jaringan SBA. *File* ini terbagi ke dalam beberapa bagian, di mana setiap bagian menargetkan konfigurasi spesifik dari NF

yang berbeda, meskipun beberapa pengaturan bersifat global dan mempengaruhi semua NF. Berikut adalah konfigurasi umum dari bagian-bagian yang ada dalam *file* konfigurasi dan NF yang terkait dengan masing-masing bagian:

1. **DNNs (dnns):** Bagian Data Network Names (DNNs) dalam konfigurasi **YAML** digunakan oleh Session Management Function (SMF) dan UPF. DNN mengidentifikasi jaringan data khusus yang UE terhubung, yang sangat penting dalam penyediaan dalam akses ke internet.

```
dnns:  
  - dnn: "oai"  
    pdu_session_type: "IPV4"  
    ipv4_subnet: "12.1.1.0/24"
```

- Nilai “*oai*” pada konfigurasi ini menentukan nama DNN yang akan digunakan, yang secara langsung berkorelasi dengan *service network* yang diakses oleh *user-end*.
- PDU *session type* menentukan jenis sesi Protokol Data Unit (PDU) yang didukung, yang mengatur jenis konektivitas IPv4 yang akan disediakan ke UE.

2. **Konfigurasi AMF:** Bagian AMF dalam *file* konfigurasi **YAML** memainkan peran penting dalam menentukan bagaimana fungsi AMF berperilaku dalam jaringan 5G. AMF merupakan komponen yang mengelola konektivitas dan otentikasi pengguna.

```
served_guami_list:  
  - mcc: 208  
    mnc: 99  
    amf_region_id: "FF"  
    amf_set_id: "2F1"  
    amf_pointer: "03"
```

Globally Unique AMF Identifier (GUAMI) dalam konfigurasi *file* adalah elemen identifikasi yang memungkinkan adanya identifikasi unik AMF secara global. GUAMI terdiri dari beberapa komponen yang di antaranya:

- Mobile Country Code (MCC) 208 adalah kode tiga digit yang mengidentifikasi secara unik di Prancis tempat AMF berada secara global.

- Mobile Network Code (MNC) bekerja bersama dengan MCC untuk menunjukkan jaringan khusus di dalam negara. Konfigurasi 99 ini digunakan untuk tujuan testing oleh regulator Prancis yaitu *Postes et de la Distribution de la Presse* (ARCEP).
- AMF Region ID adalah identifikasi *hexadecimal binary* yang menentukan wilayah spesifik di dalam network operator dimana AMF beroperasi.
- AMF Set ID menetukan set spesifik AMF dalam wilayah tertentu. Ini membantu dalam manajemen *workload* di dalam wilayah tersebut.
- AMF Pointer adalah komponen tambahan yang memberikan tingkat pengidentifikasi lebih AMF, memungkinkan alokasi dan identifikasi yang lebih dinamis.

Dengan mengkonfigurasi GUAMI dengan benar, operator jaringan memastikan bahwa setiap AMF dapat diidentifikasi dan diakses secara unik, memudahkan pengelolaan dan integrasi dengan komponen jaringan lain.

Konfigurasi PLMN:

```

plmn_support_list:
  - mcc: 208
    mnc: 99
    tac: 0xa001
    nssai:
      - *embb_slice1
  
```

Public Land Mobile Network (PLMN) menunjukkan jaringan seluler yang AMF layani. Konfigurasi PLMN dalam AMF mencakup:

- Tracking Area Code (TAC) adalah kode yang menentukan area geografis tertentu dalam jaringan. TAC *0xa001* pada konfigurasi ini menunjukkan untuk manajemen mobilitas dan pemilihan sel *custom*.
- Network Slice Selection Assistance Information (NSSAI) adalah konfigurasi yang berhubungan dengan *network slice* dengan fungsi isolasi jaringan. Setiap kombinasi SST dan SD menentukan jenis layanan dan karakteristik *slice* yang AMF dukung.

Dengan mengkonfigurasi PLMN dengan tepat, AMF dapat secara efisien mengelola konektivitas dan mobilitas UE dalam jaringan, memastikan layanan yang tepat dialokasikan dan irisan jaringan yang relevan digunakan.

3. **Konfigurasi SMF:** Komponen ini merupakan komponen krusial dalam arsitektur jaringan 5G *core* yang mengelola sesi *user* dan berinteraksi dengan berbagai elemen lainnya. Konfigurasi SMF mencakup beberapa parameter utama:

```
smf:
  ue_mtu: 1500
  support_features:
    use_local_subscription_info: no
    use_local_pcc_rules: yes
```

- UE Maximum Transmission Unit (MTU) menentukan ukuran maksimum paket data yang dapat diproses oleh UE. Nilai ini sangat penting untuk optimasi transmisi data, memastikan bahwa paket data dikirimkan dengan efisien tanpa perlu fragmentasi yang berlebihan.
- Informasi *Local Subscription* SMF bisa dikonfigurasi untuk menggunakan informasi langganan yang disimpan secara lokal atau mengambilnya dari Unified Data Management (UDM). Tidak memilih opsi ini memberikan fleksibilitas dalam manajemen profil pengguna, memungkinkan penggunaan data langganan yang lebih dinamis pada *dashboard database* berdasarkan kebutuhan *user*.
- Aturan Policy and Charging Control (PCC) Lokal menentukan apakah SMF akan menggunakan aturan PCC yang ditetapkan secara lokal atau PCF.

4. **UPF:** UPF berperan untuk mengelola lalu lintas data dan melakukan pemrosesan paket pengguna. Bagian UPF dalam *file* konfigurasi digunakan untuk menyesuaikan perilaku dan fitur dari UPF, memungkinkan penyesuaian yang fleksibel sesuai dengan kebutuhan jaringan.

```
upf_info:
  sNssaiUpfInfoList:
    - sNssai: *embb_slice1
      dnnUpfInfoList:
        - dnn: oai
```

Bagian ini mengikuti definisi UpfInfo dari 3GPP TS 29.510 [40] dan menyediakan informasi tentang dukungan UPF untuk berbagai NSSAI (*Network Slice Selection Assistance Information*) dan DNN (*Data Network Name*), yang kritis untuk pengalokasian dan manajemen irisan jaringan serta jaringan data pengguna.

5. **Database Manager:** Mengelola data *subscriber* memerlukan alat yang efisien dan mudah diakses. Untuk itu, penggunaan *database* MySQL dipilih karena kemampuannya dalam mengelola data besar melalui *script* SQL. Namun, karena *database* MySQL secara *default* tidak memiliki GUI yang ramah pengguna, penggunaan PhpMyAdmin digunakan untuk memudahkan administrator dalam melihat dan mengelola data pelanggan secara visual.

```
kubectl apply -f phypmyadmin/
```

4.2.2.2 Radio Access Network

4.2.2.2.1 Konfigurasi CU dan DU

Instalasi dan *deployment* dari OpenAirInterface (OAI) *open-source* RAN melibatkan beberapa langkah untuk menyiapkan dan mengonfigurasi komponen-komponen RAN, yang terdiri dari:

1. **Konfigurasi Umum CU DU:** Konfigurasi CU meliputi berbagai aspek, mulai dari pengaturan umum, identifikasi, manajemen area, konfigurasi *interface* jaringan, parameter keamanan, hingga pengaturan *logging*. Berikut ini adalah panduan langkah demi langkah untuk mengonfigurasi CU, mencakup berbagai parameter yang perlu disesuaikan dengan kebutuhan jaringan dan spesifikasi teknis.

a. Konfigurasi Identifikasi Parameter:

```
gNBs = ({
    gNB_ID = {{ .Values.config.cuId}};
    gNB_name = "{{ .Values.config.cuName}}";
```

Bagian ini mengidentifikasi CU secara spesifik dengan ID dan nama. `gNB_ID` memberikan pengenal unik untuk CU, dan `gNB_name` menentukan namanya. Ini penting untuk identifikasi dan manajemen CU dalam jaringan.

b. Konfigurasi TAC dan PLMN:

```
tracking_area_code = {{ .Values.config.tac}};
plmn_list = ({{ mcc = {{ .Values.config.mcc}}; mnc =
{{ .Values.config.mnc}}; mnc_length = 2; snssaiList = ({{ sst =
{{ .Values.config.sst}}}; }) }});
```

Konfigurasi ini menetapkan TAC untuk CU, yang digunakan dalam manajemen mobilitas untuk mengidentifikasi area tertentu dalam jaringan. Selain itu, `plmn_list` menentukan daftar PLMN yang CU layani, termasuk MCC, MNC dan daftar SNSSAI yang didukung.

c. **Konfigurasi Koneksi AMF CU:**

```
amf_ip_address      = ( { ipv4      =
    "{{ .Values.config.amfhost }}";
    ipv6      = "192:168:30::17";
    active    = "yes";
    preference = "ipv4";
}
);
;
```

Ini menetapkan parameter untuk koneksi ke AMF, termasuk alamat IP AMF dan preferensi untuk penggunaan IPv4 atau IPv6. Konfigurasi ini penting untuk memastikan CU dapat berkomunikasi dengan AMF untuk fungsi manajemen akses dan mobilitas.

d. **Konfigurasi RFSim DU:**

```
rfsimulator: {
    serveraddr = "server";
    serverport = 4043;
    options = (); #("saviq"); or/and "chanmod"
    modelname = "AWGN";
    IQfile = "/tmp/rfsimulator.iqs"
}
```

Ini adalah konfigurasi untuk simulator RF, yang menentukan alamat server, *port*, opsi, model *noise* yang digunakan, dan lokasi *file IQ*. Ini memungkinkan simulasi *air interface* radio tanpa memerlukan perangkat keras RF nyata.

2. **Konfigurasi UE:** Konfigurasi UE ini menentukan parameter identifikasi dan keamanan untuk perangkat pengguna dalam jaringan. Berikut ini adalah panduan untuk mengkonfigurasi UE, mencakup berbagai parameter yang perlu disesuaikan sesuai dengan kebutuhan jaringan dan spesifikasi teknis. Semua konfigurasi 5G E2E didokumentasikan di Lampiran 2.27.

```

uicc0 = {
    imsi = "{{ .Values.config.fullImsi }}";
    key = "{{ .Values.config.fullKey }}";
    opc= "{{ .Values.config.opc }}";
    dnn= "{{ .Values.config.dnn }}";
    nssai_sst="{{ .Values.config.sst }}";
    nssai_sd="{{ .Values.config.sd }}";
}

```

4.2.2.2.2 Validasi Protokol

Setelah instalasi komponen RAN dan UE selesai, langkah selanjutnya adalah melakukan validasi untuk menjamin setiap protokol komponen pada sisi *control plane* dan *data plane* berfungsi secara optimal. Proses validasi *control plane* mencakup verifikasi Wireshark koneksi *backhaul*, *fronthaul* dan *midhaul* serta *signalling* antara komponen RAN – UE dan *core network*.

- 1. Backhaul (NGAP):** Pengujian dilakukan melalui soket SCTP dan pemeriksaan *port* pada *pod* CU menggunakan netstat -Snp. Kemudian pastikan *address local* dengan *address foreign* berstatus *established* dengan *port* yang sesuai dengan konfigurasi.

```

root@oai-cu-user-n-54b8b446ff-zhpdc:/opt/oai-gnb# netstat -Snp
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
sctp     0      0 192.168.1.32:38472        192.168.1.33:46950    LISTEN    1/nr-softmodem
sctp     0      0 192.168.1.32:38472        172.21.6.94:38412    ESTABLISHED 1/nr-softmodem
sctp     0      0 172.21.6.90:52464        172.21.6.94:38412    ESTABLISHED 1/nr-softmodem

```

Gambar 4.6 Validasi Koneksi Port CU

Wireshark digunakan untuk menganalisis trafik jaringan dan memverifikasi keberhasilan komunikasi *backhaul*. Protokol NGAP beroperasi pada *interface N2*, yang menghubungkan gNB dan AMF dengan memastikan adanya *handovers* antar *cell* dan alokasi *resource* yang efisien.

No.	Time	Source	Destination	Protocol	Length	Info
3770	121.291569	172.21.6.90	172.21.6.94	NGAP/NAS-5GS	149	InitialUEMessage, Registration request
3771	121.323934	172.21.6.94	172.21.6.90	NGAP/NAS-5GS	148	SACK (Ack=0, Arvnd=106496) , DownlinkNASTransport, Authentication request
3774	121.332536	172.21.6.90	172.21.6.94	NGAP/NAS-5GS	144	SACK (Ack=0, Arvnd=106496) , UplinkNASTransport, Authentication response
3778	121.351729	172.21.6.94	172.21.6.90	NGAP/NAS-5GS	128	SACK (Ack=1, Arvnd=106496) , DownlinkNASTransport, Security mode command
3781	121.360497	172.21.6.90	172.21.6.94	NGAP/NAS-5GS	188	SACK (Ack=1, Arvnd=106496) , UplinkNASTransport
3782	121.365652	172.21.6.94	172.21.6.90	NGAP/NAS-5GS	252	SACK (Ack=2, Arvnd=106496) , InitialContextSetupRequest
3796	121.589501	172.21.6.90	172.21.6.94	NGAP	194	UERadioCapabilityInfoIndication
3808	121.797153	172.21.6.90	172.21.6.94	NGAP	84	InitialContextSetupResponse
3831	122.609622	172.21.6.90	172.21.6.94	NGAP/NAS-5GS	120	UplinkNASTransport
3841	122.801445	172.21.6.90	172.21.6.94	NGAP/NAS-5GS	140	UplinkNASTransport
3842	122.812412	172.21.6.94	172.21.6.90	NGAP/NAS-5GS	272	SACK (Ack=0, Arvnd=106496) , PDUSESSIONResourceSetupRequest
3848	122.826091	172.21.6.90	172.21.6.94	NGAP	120	SACK (Ack=3, Arvnd=106496) , PDUSESSIONResourceSetupResponse

Gambar 4.7 Validasi Koneksi Wireshark CU

- 2. *Midhaul (F1AP)*:** Komunikasi *midhaul* diverifikasi menggunakan soket SCTP pada pod DU menggunakan `netstat -Spn` dan analisis trafik dengan Wireshark, memastikan efisiensi transmisi data.

```
root@oai-du-user-n-dfd648778-6nsrn:/opt/oai-gnb# netstat -Spn
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
sctp      0      0 192.168.1.33:46950      192.168.1.32:38472  ESTABLISHED 1/nr-softmodem
```

Gambar 4.8 Validasi Koneksi Port DU

Protokol F1AP beroperasi pada *interface* F1, yang menyediakan layanan *signalling* antara gNB-CU dan gNB-DU dari gNB dalam NG-RAN.

No.	Time	Source	Destination	Protocol	Length: Info
3649	117.825161	192.168.1.33	192.168.1.32	F1AP	272 F1SetupRequest, MIB, SIB1
3651	117.826058	192.168.1.32	192.168.1.33	F1AP	112 F1SetupResponse
3764	121.271664	192.168.1.33	192.168.1.32	F1AP/NR RRC	256 InitialULRRCMessageTransfer,
3765	121.272354	192.168.1.32	192.168.1.33	F1AP/NR RRC	252 SACK (Ack=1, Arwnd=106496) ,
3769	121.291049	192.168.1.33	192.168.1.32	F1AP/NR RRC/NAS-5GS	156 SACK (Ack=1, Arwnd=106496) ,

Gambar 4.9 Validasi Koneksi Wireshark DU

- 3. *Fronthaul*:** Dalam simulasi, koneksi *fronthaul* diperiksa dengan memastikan bahwa port 4043 aktif mendengarkan pod DU menggunakan `netstat -apn4`. Konfigurasi ini diuji untuk menjamin bahwa UE dapat terkoneksi ke Rfsim tanpa hambatan.

```
root@oai-du-user-n-dfd648778-6nsrn:/opt/oai-gnb# netstat -apn4
Active Internet connections (servers and established)
Proto Recv-0 Send-0 Local Address          Foreign Address        State      PID/Program name
tcp      0      0 0.0.0.0:4043            0.0.0.0:*              LISTEN    1/nr-softmodem
tcp      57720  81184 192.168.1.33:4043   192.168.1.35:34288  ESTABLISHED 1/nr-softmodem
sctp     0      0 192.168.1.33:46950     192.168.1.32:38472  ESTABLISHED 1/nr-softmodem
udp      0      0 192.168.1.33:2152     0.0.0.0:*              1/nr-softmodem
```

Gambar 4.10 Validasi Koneksi Port RFSim

Setelah validasi *control plane* selesai, langkah selanjutnya adalah melakukan validasi dari sisi *data plane*. Proses validasi ini mencakup verifikasi Wireshark Rfsim dan GTP Tunnel serta PDU Session antara komponen RAN – UE dan *core network*.

- 1. Verifikasi Keberhasilan PDU Session UE:** Untuk memverifikasi keberhasilan koneksi *User Equipment* (UE) ke jaringan *core* 5G, langkah pertama adalah memastikan bahwa UE telah terdaftar dengan sukses.

Hal ini dapat dikonfirmasi dengan memeriksa apakah *interface* jaringan virtual `oaitun_ue1` sudah terbentuk dan memiliki alamat IP yang benar. Perintah ini harus menampilkan alamat IP yang dialokasikan, yang menunjukkan bahwa UE telah terintegrasi dengan benar.

```
kubectl exec -it -n user-n -c nr-ue $(kubectl get pods -n user-n | grep oai-nr-ue | awk '{print $1}') -- ifconfig oaitun_ue1 | grep -E '^(\^|\s)inet($|\s)' | awk '{print $2}'
```

2. **Konfigurasi routing gateway UPF:** Menetapkan *data path* yang harus diikuti oleh paket data untuk mencapai berbagai destinasi melalui UPF.

```
<<K9s-Shell>> Pod: core-network/oai-upf-d66f7b659-5vsmd | Container: upf
root@oai-upf-d66f7b659-5vsmd:/openair-upf# ip r
default via 10.102.0.1 dev n6
10.16.0.0/16 dev eth0 proto kernel scope link src 10.16.1.245
10.102.0.0/24 dev n6 proto kernel scope link src 10.102.0.248
12.1.1.0/24 dev tun0 proto kernel scope link src 12.1.1.1
14.1.1.0/24 via 12.1.1.1 dev tun0
172.21.8.0/24 dev n3 proto kernel scope link src 172.21.8.95
192.168.7.0/27 dev n4 proto kernel scope link src 192.168.7.2
root@oai-upf-d66f7b659-5vsmd:/openair-upf# |
```

Gambar 4.11 Routing Table UPF

3. **Konfigurasi routing UE:** Menetapkan aturan untuk arah data di UE, memungkinkan UE untuk mengirim dan menerima data ke dan dari jaringan melalui rute yang ditentukan.

```
<<K9s-Shell>> Pod: user-n/oai-nr-ue-user-n-665cc5778f-p6rxq | Container: nr-ue
root@oai-nr-ue-user-n-665cc5778f-p6rxq:/opt/oai-nr-ue# ip r
default via 12.1.1.1 dev oaitun_ue1
10.16.0.0/16 dev eth0 proto kernel scope link src 10.16.1.248
12.1.1.0/24 dev oaitun_ue1 proto kernel scope link src 12.1.1.100
192.168.1.0/28 dev net1 proto kernel scope link src 192.168.1.3
root@oai-nr-ue-user-n-665cc5778f-p6rxq:/opt/oai-nr-ue#
```

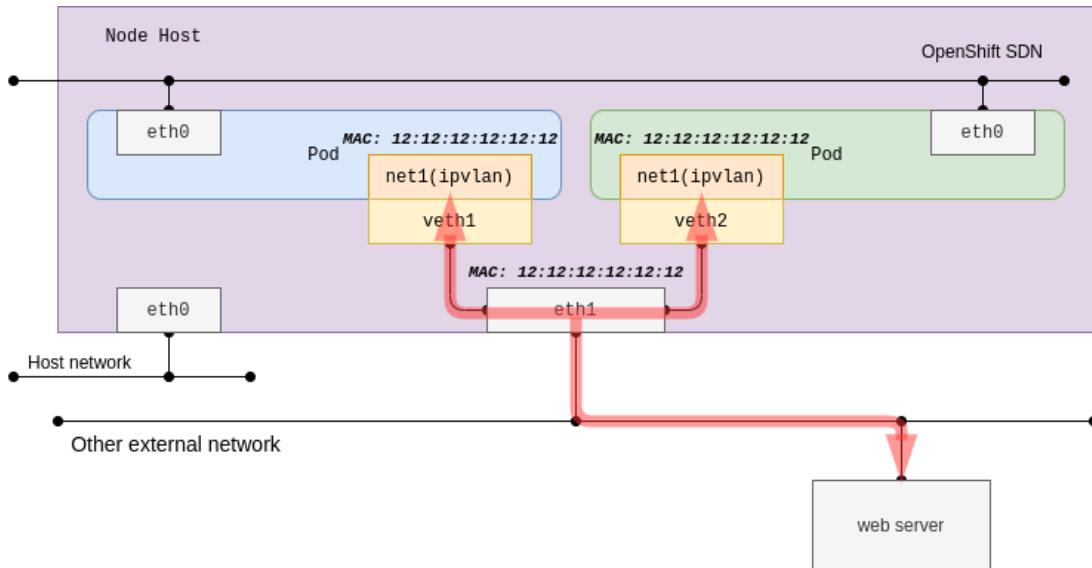
Gambar 4.12 Routing Table UE

4. **Verifikasi Koneksi GTP Tunnel:** Suksesnya ping menunjukkan bahwa UE tidak hanya terhubung ke jaringan *core 5G*, tetapi juga mampu berkomunikasi dengan entitas di luar jaringan lokalnya, memastikan fungsionalitas lengkap dari koneksi data UE.

```
kubectl exec -it -n user-n -c nr-ue $(kubectl get pods -n user-n | grep oai-nr-ue | awk '{print $1}') -- ping -I oaitun_ue1 -c4 google.com
```

4.2.2.3 Konfigurasi Multi Node

Dalam rangka meningkatkan efektivitas penggunaan *resource* dalam lingkungan Kubernetes, penting untuk mendistribusikan beban kerja secara merata di antara berbagai *node worker*. Pendekatan ini menghindari pemasaran *workload* pada satu *node* tertentu, yang dapat menyebabkan kegagalan dan menurunkan *performance* sistem.



Gambar 4.13 Skema Komunikasi Multi Node

Interaksi antar-*pod* pada *node* yang berbeda ini dapat direalisasikan melalui pemanfaatan KubeOVN CNI dan Multus CNI, yang mendukung konfigurasi tambahan *multi-interface* Multus dalam satu *pod*. Konfigurasi ini melibatkan pengaturan IP Address Management (IPAM) yang memfasilitasi alokasi dan manajemen alamat IP secara dinamis dan efisien.

```
{
    "cniVersion": "0.3.1",
    "type": "ipvlan",
    "master": "ens33",
    "mode": "l2",
    "ipam": {
        "type": "static",
        "addresses": [
            {
                "address": "192.168.1.1/29"
            }
        ]
    }
}
```

Dengan mengonfigurasikan Multus untuk menggunakan tipe IPVlan, berbeda dari MacVlan sebelumnya, serta mengubah modusnya menjadi L2 dari static, kita memungkinkan komunikasi antar-*pod* melalui *custom bridge interface* yang disediakan oleh CNI, memfasilitasi interaksi yang lebih lancar dan efisien antara *pod* yang berada di *node* yang berbeda.

4.2.3 Implementasi *Backend*

4.2.3.1 Inisialisasi Proyek Django

Proyek Django yang akan digunakan dibagi menjadi dua, yang pertama untuk difokuskan pada implementasi REST API dan yang kedua difokuskan pada implementasi Websocket API.

4.2.3.1.1 Persiapan Virtual Environment Untuk Proyek Django

Penggunaan *virtual environment* dalam pengembangan Python dianggap sebagai praktik terbaik dalam membantu mengelola *libraries* secara efisien, memastikan lingkungan yang terisolasi dan konsisten diseluruh tahap pengembangan dan *deployment*. Setiap kali akan menjalankan perintah dalam bentuk Bahasa Python pada *terminal prompt*, *virtual environment* harus dalam keadaan aktif agar perintah dapat terdeteksi oleh sistem.

Berikut adalah langkah-langkah pembuatan hingga verifikasi keberhasilan persiapan *virtual environment*.

1. Membuat dan menavigasi ke dalam direktori inti proyek dengan nama *ORCA*

```
mkdir ORCA && cd ORCA
```

2. Membuat dua buah *virtual environment* yang berbeda untuk setiap proyek Django dengan nama masing-masing **venv_orca_backend** untuk proyek Django REST API, dan **venv_orca_backend_ws** untuk proyek Django Websocket API.

```
python3 -m venv venv_orca_backend
```

```
python3 -m venv venv_orca_backend_ws
```

3. Aktivasi kedua *virtual environment* yang telah diinisialisasi

```
source venv_orca_backend/bin/activate
```

```
source venv_orca_backend_ws/bin/activate
```

4. Verifikasi kedua *virtual environment* telah berhasil diaktifasi

```
1. bagus@c5g-backend:~/ORCA$ source venv_orca_backend/bin/activate
2. (venv_orca_backend)bagus@c5g-backend:~/ORCA$
```

```
3. bagus@c5g-backend:~/ORCA$ source  
venv_orca_backend_ws/bin/activate  
4. (venv_orca_backend_ws)bagus@c5g-backend:~/ORCA$
```

Empat baris tulisan di atas menjelaskan keberhasilan aktivasi *virtual environment*. Baris pertama mengindikasikan direktori **ORCA** akan mengaktifkan *virtual environment* dengan baris perintah yang ditandai dengan warna kuning. Sedangkan pada baris kedua, bagian yang ditandai dengan warna hijau mengindikasikan *virtual environment* berhasil diaktifkan. Hal yang sama juga dilakukan untuk baris ketiga dan keempat untuk mengaktifasi *virtual environment* selanjutnya.

4.2.3.1.2 Menginstal Django dan Membuat Proyek Django Baru

Dengan mengaktifkan *virtual environment*, instalasi Django dapat dilakukan menggunakan tools *pip* sebagai *package-management system* bawaan dari Python untuk menginstal dan manajemen paket. Instalasi ini tidak akan memengaruhi *global environment* pada Python. Namun, instalasi *libraries* di dalam sebuah *virtual environment* akan mengisolasi pengimplementasian *libraries* tersebut di dalam proyek Django yang bersangkutan. Baris perintah di bawah berlaku untuk kedua proyek Django yang akan diinisialisasi dan dikembangkan.

1. Instalasi Django

```
Pip install django (untuk proyek REST API)  
Pip install django (untuk proyek Websocket API)
```

2. Verifikasi instalasi Django

```
Django-admin --version (untuk proyek REST API)  
Django-admin --version (untuk proyek Websocket API)
```

3. Membuat proyek Django baru

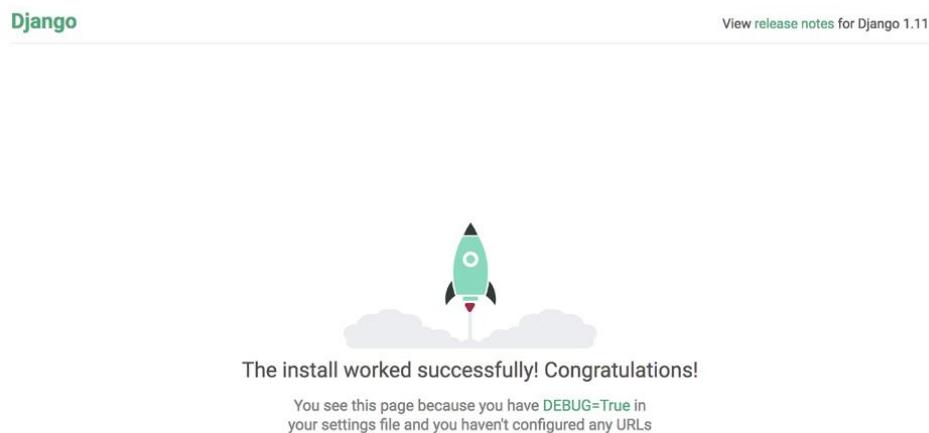
```
Django-admin startproject orca_backend (untuk proyek REST API)  
Django-admin startproject orca_backend_ws (untuk proyek Websocket API)
```

4. Verifikasi pembuatan proyek Django baru

```
cd orca_backend && python manage.py runserver  
cd orca_backend_ws && python manage.py runserver
```

Setelah baris perintah di atas telah dijalankan, tunggu beberapa saat untuk proses dan kemudian klik URL tertera pada *console terminal code editor* untuk mengakses proyek Django yang berhasil dijalankan pada *browser* [41].

5. Pembuatan proyek Django berhasil ditandai dengan munculnya laman seperti gambar dibawah pada *browser* ketika URL berhasil diakses.



Gambar 4.14 Instalasi Django Sukses

4.2.3.1.3 Mengkonfigurasi Pengaturan Proyek Django

Pengaturan pada proyek Django akan terus berubah seiring berjalananya waktu. Konfigurasi pengaturan pada tahap awal biasanya berfokus pada penambahan aplikasi, penyesuaian zona waktu maupun *database*. Pengaturan pada proyek Django berada didalam direktori proyek, dengan nama *file settings.py*. Semua baris kode yang ditambah dan diubah pada *file* tersebut berkaitan dengan keseluruhan pengembangan *backend*. Baris kode secara lengkap dapat dilihat pada Lampiran 4.11 dan Lampiran 4.12. Berikut adalah rangkuman poin-poin yang dikonfigurasi pada *file settings.py* untuk kedua proyek yang diinisialisasi.

- | | |
|-------------------------------------|---------------------------------|
| a. <i>Import Libraries</i> | f. Pengaturan <i>Middleware</i> |
| b. Pengaturan Akses Proyek | g. Pengaturan Corsheaders |
| c. Aplikasi Terinstall Pada Proyek | h. Pengaturan <i>Database</i> |
| d. Pengaturan Django Rest Framework | i. Pengaturan <i>Timezone</i> |
| e. Pengaturan JWT | j. Pengaturan Django Channels |

Verifikasi yang dapat dilakukan untuk memastikan tidak ada *error* pada pengaturan proyek adalah dengan cara menjalankan baris perintah berikut.

```
python manage.py runserver
```

Apabila terdapat suatu *error* pada pengaturan tersebut, maka *console terminal* pada *text editor* akan memunculkan notifikasi beserta kode ataupun petunjuk *error* yang sedang terjadi. Apabila proyek berhasil dijalankan dengan normal tanpa adanya notifikasi *error*, mengindikasikan pengaturan berhasil diimplementasikan.

4.2.3.2 Pengaturan Aplikasi

4.2.3.2.1 Inisialisasi Aplikasi Django Baru di Dalam Proyek

Setelah proyek Django berhasil diinisialisasi, langkah selanjutnya adalah inisialisasi aplikasi Django baru di dalam proyek tersebut. Aplikasi tersebut digunakan untuk merangkum keseluruhan fungsi *backend* yang akan dikembangkan. Berikut adalah baris perintah yang digunakan untuk inisialisasi aplikasi Django pada kedua proyek yang digunakan.

Django-admin startapp apps (untuk proyek REST API)
Django-admin startapp monitoring (untuk proyek WebSocket API)
Django-admin startapp protocolstack (untuk proyek WebSocket API)
Django-admin startapp shell (untuk proyek WebSocket API)
Django-admin startapp sniff (untuk proyek WebSocket API)

Verifikasi yang dapat dilakukan untuk memastikan aplikasi berhasil terinisialisasi adalah dengan cara melihat struktur *folder* pada proyek akan terdapat satu *folder* baru dengan nama dari setiap aplikasi Django yang diinisialisasi pada masing-masing proyek.

4.2.3.2.2 Mendefinisikan Struktur Aplikasi

Struktur aplikasi merupakan hal krusial yang berperan penting dalam proses pengembangan *backend* yang membuat keseluruhan fungsi yang dibuat dapat tertata dan terkelola dengan baik. Apabila struktur aplikasi tidak terdefinisi dengan baik, hal tersebut dapat berpengaruh terutama terhadap proses pengembangan berskala besar.

Pendefinisian struktur aplikasi dapat dilakukan dengan membagi setiap jenis fungsi terpisah ke dalam sebuah *subfolder*. Dengan begitu, pengembang dapat lebih mudah membaca

dan memahami keseluruhan kode pada aplikasi. Cara tambahan lainnya adalah dengan melakukan *versioning*. *Versioning* merupakan salah satu cara untuk membedakan versi aplikasi pada saat pengembangan berlangsung, sehingga pengembang dapat dengan mudah melakukan pengembangan tanpa harus melakukan perombakan secara menyeluruh. Struktur aplikasi yang digunakan pada pengembangan *backend* ini dapat dilihat pada Lampiran 4.13.

4.2.3.2.3 Instalasi *Libraries*

Libraries menjadi hal yang sangat penting dalam proses pengembangan *backend*, dikarenakan pada dasarnya proyek Django baru belum memiliki beragam *libraries* karena harus menyesuaikan dengan kebutuhan pengembangan. Dalam pengembangan *backend* ini, beragam *libraries* diperlukan agar keseluruhan fungsi dapat berjalan dengan baik. Instalasi *libraries* dapat dilakukan menggunakan pola baris perintah Python berikut.

```
pip install libraries-to-be-installed
```

Libraries yang telah digunakan selama proses pengembangan akan dibungkus dalam suatu file ber-ekstensi *txt* bernama *requirements.txt* yang dapat dibuat melalui baris perintah Python berikut.

```
pip freeze > requirements.txt
```

Keseluruhan baris kode yang terdapat pada file *requirements.txt* dapat dilihat pada Lampiran 4.14.

4.2.3.3 Pemodelan *Database*

4.2.3.3.1 Merancang dan Mendefinisikan Model Untuk Merepresentasikan Skema Database

Dalam perancangan dan pendefinisian model pada proyek Django untuk implementasi REST API, terdapat model yang telah disediakan oleh Django secara *default* yang digunakan yakni *User Model* yang akan dikustomisasi dengan tambahan model untuk *User Profile* yang mana setiap *user* akan memiliki profilnya masing-masing. Selain itu, terdapat model lain yang akan dirancang dan didefinisikan berkaitan dengan *user's completion progress*, dan berikut adalah daftar-daftarnya.

- a. *PCAP file*
- b. *CU config*
- c. *DU config*
- d. *UE config*

e. User configuration

Proses verifikasi keberhasilan pemodelan *database* menjadi sangat penting mengingat perannya yang sangat krusial terhadap jalannya proyek. Terlepas seperti apapun jenis model yang dibuat, proses verifikasinya memiliki cara yang sama. Proses verifikasi pemodelan *database* dapat dilakukan menggunakan baris perintah berikut.

```
python manage.py makemigrations
```

Apabila setelah baris perintah di atas dijalankan dan kemudian tidak terdapat notifikasi *error* yang muncul pada *console terminal*, maka dapat dipastikan proses pemodelan berhasil. Keseluruhan baris kode terkait model yang diperlukan dalam pengembangan *backend* ini dapat dilihat pada Lampiran 4.15.

4.2.3.3.2 Menggunakan Django's Object-Relational Mapping (ORM) Untuk Membuat Tabel Database Berdasarkan Model

Pada dasarnya, *Django's* ORM merupakan salah satu fitur yang disediakan oleh Django, yang berfungsi untuk mengubah model yang telah didefinisikan kedalam bentuk tabel *database*. Untuk mengaktifkan/menjalankan fitur tersebut, dapat dilakukan menggunakan baris perintah Python yang sama dengan poin sebelumnya terkait pemodelan *database*. Hal ini berlaku untuk kedua proyek Django yang digunakan.

```
python manage.py makemigrations
```

Baris perintah diatas mengubah kode untuk membuat model *database* yang berasal dari Bahasa Python menjadi Bahasa SQL. Dengan begitu, proses pembuatan tabel *database* berdasarkan model dapat dilakukan.

4.2.3.3.3 Menerapkan Migrasi Untuk Mengelola Perubahan Model Database

Penerapan baris perintah untuk migrasi tidak hanya diperuntukkan ketika akan membuat tabel *database* berdasarkan model yang telah didefinisikan. Namun, baris perintah migrasi juga diperlukan apabila terdapat perubahan yang bersifat penambahan ataupun pengurangan dalam model yang telah didefinisikan sebelumnya. Berikut merupakan baris perintah dalam Python untuk menerapkan migrasi sehingga setiap perubahan model *database* dapat langsung diterapkan pada tabel *database* yang telah ada. Hal ini berlaku untuk kedua proyek Django yang digunakan.

```
1. python manage.py migrate
```

```
2. python manage.py migrate --run-syncdb
```

Kedua baris perintah di atas pada dasarnya memiliki fungsi yang sama untuk melakukan migrasi terhadap perubahan model *database*. Namun, baris perintah nomor dua memiliki baris tambahan yang berfungsi untuk melakukan sinkronisasi *database* yang telah ada sehingga tanpa mereset data yang telah ada sebelumnya ketika terjadi perubahan pada tabel *database* tersebut.

4.2.3.4 Pembuatan API

4.2.3.4.1 Mendefinisikan RESTful Endpoints Menggunakan Django Rest Framework (DRF)

RESTful *endpoints* mengacu pada URL atau pola URI tertentu dalam layanan web yang mengikuti prinsip-prinsip arsitektur *Representational State Transfer* (REST). *Endpoint* ini menyediakan akses ke sumber daya (seperti data atau layanan) metode HTTP standar.

Pendefinisian RESTful *endpoint* ini membutuhkan *libraries* Python yang bernama “*djangorestframework*” dan mengaturnya pada *file settings.py*, untuk proyek Django REST-API berikut adalah langkah-langkahnya.

1. Instalasi *libraries djangorestframework*

```
pip install djangorestframework
```

2. Update *file settings.py*, tambahkan baris ‘rest_framework’ pada bagian INSTALLED_APPS seperti contoh dibawah.

```
INSTALLED_APPS = [  
    ...  
    'rest_framework', //Baris yang ditambahkan  
    ...
```

4.2.3.4.2 Membuat Serializer Untuk Mengonversi Model ke Format JSON dan Sebaliknya

Dalam pengembangan ini, banyak digunakan *serializer* terhadap data yang perlu dikelola dan kemudian diolah kembali oleh sisi *frontend*. Berikut adalah daftar pendefinisian *serializer* yang digunakan dalam pengembangan *backend* ini.

- a. Konfigurasi *class* untuk pembuatan akun *user*
- b. Konfigurasi *class* untuk pembuatan profil *user*
- c. Konfigurasi *class* untuk *list* akun *user*
- d. Konfigurasi *class* untuk *update* informasi *user*
- e. Konfigurasi *class* untuk *list* informasi *user*

f. Konfigurasi *class* untuk file PCAP

Keenam konfigurasi di atas merupakan daftar konfigurasi yang diperlukan untuk pembuatan *serializer* dalam pengembangan *backend* ini. Untuk keseluruhan baris kode yang terdapat di dalam *file serializer* dapat dilihat pada Lampiran 4.16.

4.2.3.4.3 Menerapkan View Untuk Menangani Logika API Endpoints

Dalam Django, *view* diimplementasikan dalam sebuah *file* bernama `views.py`. *View* biasanya berupa fungsi yang terdiri baris kode untuk melakukan sebuah perintah logika yang sedang dikembangkan. Pada pengembangan *backend* ini, terdapat lebih dari satu penerapan *view* yang masing-masing dibedakan berdasarkan fungsionalitasnya, seperti *view* yang berhubungan dengan manajemen *user*, manajemen komponen 5G *cloud*, manajemen kluster Kubernetes, dan manajemen token untuk otentikasi dan otorisasi.

Berikut adalah poin-poin yang akan menyajikan daftar baris kode dari *view* yang diterapkan pada pengembangan *backend* ini.

4.2.3.4.3.1 Manajemen User

View yang berkaitan dengan manajemen *user* diletakkan di dalam direktori bernama “**user**”, yang mana direktori ini dikhkususkan untuk membuat keseluruhan *view* yang berkaitan dengan manajemen *user*. Berikut adalah daftar baris kode yang didefinisikan pada *view* ini untuk memenuhi kebutuhan fungsionalitas manajemen *user*.

- | | | |
|-----------------------|-----------------------|---------------------------------|
| a. <i>Create user</i> | c. <i>Update user</i> | e. <i>Get users information</i> |
| b. <i>List users</i> | d. <i>Delete user</i> | |

Seluruh fungsi di atas akan dikonfigurasi kedalam bentuk API URL yang nantinya dapat diakses dan diverifikasi keberhasilan fungsinya menggunakan aplikasi untuk testing API. Keseluruhan baris kode untuk daftar fungsi di atas dapat dilihat pada **Lampiran 4.17**.

4.2.3.4.3.2 Manajemen Komponen 5G Cloud

View yang berkaitan dengan manajemen komponen 5G *cloud* diletakkan di dalam direktori bernama “**oai**”, yang mana direktori ini dikhkususkan untuk membuat keseluruhan *view* yang berkaitan dengan manajemen komponen 5G *cloud*. Berikut adalah daftar baris kode fungsi yang didefinisikan pada *view* ini untuk memenuhi kebutuhan fungsionalitas manajemen komponen 5G *cloud*.

Tabel 4.2 Daftar baris kode fungsi manajemen komponen 5G cloud

a. <i>Update chart name</i>	d. <i>Delete all 5G components</i>
-----------------------------	------------------------------------

<p>b. <i>Revert chart name</i></p> <p>c. <i>Create all 5G components</i></p> <p>Proses pembuatan seluruh komponen 5G <i>cloud</i> ini terdiri dari beberapa tahapan dan memiliki baris kode yang panjang. Berikut adalah rangkuman daftar baris kode untuk fungsi pembuatan komponen 5G <i>cloud</i>.</p> <ol style="list-style-type: none"> 1. Pendefinisian fungsi 2. Konfigurasi <i>update</i> nama <i>chart</i> dan direktori penyimpanan 3. Konfigurasi instalasi <i>helm chart</i> 4. Konfigurasi mengembalikan nama <i>chart</i> ke semula 	<p>e. <i>Get CU Component Values</i></p> <p>f. <i>Get DU Component Values</i></p> <p>g. <i>Get UE Component Values</i></p> <p>h. <i>Config CU Components</i></p> <p>i. <i>Config DU Components</i></p> <p>j. <i>Config UE Components</i></p> <p>k. <i>Start Components</i></p> <p>l. <i>Stop Components</i></p> <p>m. <i>Restart Componenets</i></p> <p>n. <i>Get User Helm Deployments List</i></p>
---	--

Seluruh fungsi di atas terintegrasi dengan implementasi 5G *cloud* pada kluster Kubernetes. Sehingga, tiap-tiap fungsi di atas mendefinisikan akses ke kluster Kubernetes untuk membuat, mengatur, ataupun mendapatkan *resources* yang berhubungan dengan komponen 5G *cloud* dibantu dengan *helm chart* untuk proses instalasi komponen 5G. Keseluruhan baris kode terkait fungsi di atas dapat dilihat pada Lampiran 4.18.

4.2.3.4.3.3 Manajemen Kluster Kubernetes

View yang berkaitan dengan manajemen komponen kluster Kubernetes diletakkan di dalam direktori bernama “**kube**”, yang mana direktori ini dikhususkan untuk membuat keseluruhan *view* yang berkaitan dengan manajemen komponen kluster Kubernetes.

- | | |
|--|--|
| <p>a. <i>Get User Pods</i></p> <p>b. <i>Get User Deployments</i></p> <p>c. <i>Restart 5G Component</i></p> | <p>d. <i>Get Pod Logs</i></p> <p>e. <i>Get AMF and UPF Logs</i></p> <p>f. <i>Get AMF and UPF Deployments</i></p> |
|--|--|

Seluruh baris kode yang berhubungan dengan integrasi terhadap kluster Kubernetes memerlukan akses kedalam kluster melalui sebuah *file kubeconfig*. *File* tersebut digunakan untuk mendapatkan akses ke sebuah kluster Kubernetes tertentu dengan bantuan

kubectl sebagai baris perintah yang berhubungan dengan Kubernetes. Keseluruhan baris kode di atas dapat dilihat pada Lampiran 4.19.

4.2.3.4.3.4 Manajemen Token

View yang berkaitan dengan manajemen token diletakkan di dalam direktori bernama “**token**”, yang mana direktori ini dikhususkan untuk membuat keseluruhan *view* yang berkaitan dengan manajemen token. Manajemen token akan memerlukan *libraries rest_framework_simplejwt* untuk implementasinya.

Penerapan *view* untuk manajemen token hanya diperlukan apabila terdapat kustomisasi pada *payload* token. Berhubung pada pengembangan *backend* ini memerlukan kustomisasi, maka penerapan *view* dilakukan agar kebutuhan terpenuhi. Keseluruhan baris kode terkait manajemen token dapat dilihat pada Lampiran 4.20.

4.2.3.4.4 Konfigurasi URL Untuk Memetakan View Endpoints

4.2.3.4.4.1 Manajemen User

URL yang berkaitan dengan manajemen *user* juga diletakkan di dalam direktori bernama “**user**” bersama dengan *view*-nya. Sehingga, seluruh fungsi yang telah dibuat pada *file views.py* sebelumnya akan dikonfigurasi sebagai API yang dapat diakses melalui URL. Keseluruhan baris kode terkait konfigurasi URL untuk fungsi manajemen *user* dapat dilihat pada Lampiran 4.21.

4.2.3.4.4.2 Manajemen Komponen 5G Cloud

URL yang berkaitan dengan manajemen komponen 5G *cloud* juga diletakkan di dalam direktori bernama “**oai**” bersama dengan *view*-nya. URL yang dikonfigurasi mencakup banyak hal seperti konfigurasi dan manajemen siklus hidup komponen 5G *cloud*. Secara umum, URL untuk konfigurasi komponen, menjalankan komponen, dan menghentikan komponen memiliki pola yang sama. Keseluruhan baris kode dapat dilihat pada Lampiran 4.22. Berikut adalah pola yang dimaksud.

```
urlpatterns = [
    path('values_component_name/',
views.values_component_name, name='values_component_name'),
    path('config_component_name/',
views.config_component_name, name='config_component_name'),
```

```

        path('start_component_name/', views.start_component_name,
name='start_component_name'),
        path('stop_component_name/', views.stop_component_name,
name='stop_component_name'),
        path('restart_component_name/',
views.restart_component_name, name='restart_component_name'),
    ]

```

4.2.3.4.4.3 Manajemen Kluster Kubernetes

URL yang berkaitan dengan manajemen komponen Kubernetes juga diletakkan di dalam direktori bernama “**kube**” bersama dengan *view*-nya. Baris kode fungsi yang berkaitan dengan manajemen kluster Kubernetes tidaklah banyak, sehingga URL yang dikonfigurasi juga disesuaikan dengan fungsi yang ada.

```

urlpatterns = [
    path('pods/', views.get_user_pods, name='get_user_pods'),
    path('deployments/', views.get_user_deployments,
name='get_user_deployments'),
    path('pods/<str:pod_name>/logs/', views.get_pod_logs,
name='get_pod_logs'),
]

```

4.2.3.4.4.4 Manajemen Token

URL yang berkaitan dengan manajemen token juga diletakkan di dalam direktori bernama “**token**” bersamaan dengan *view*-nya. Keseluruhan baris kode terdiri dari empat URL disesuaikan dengan fungsi yang telah didefinisikan.

```

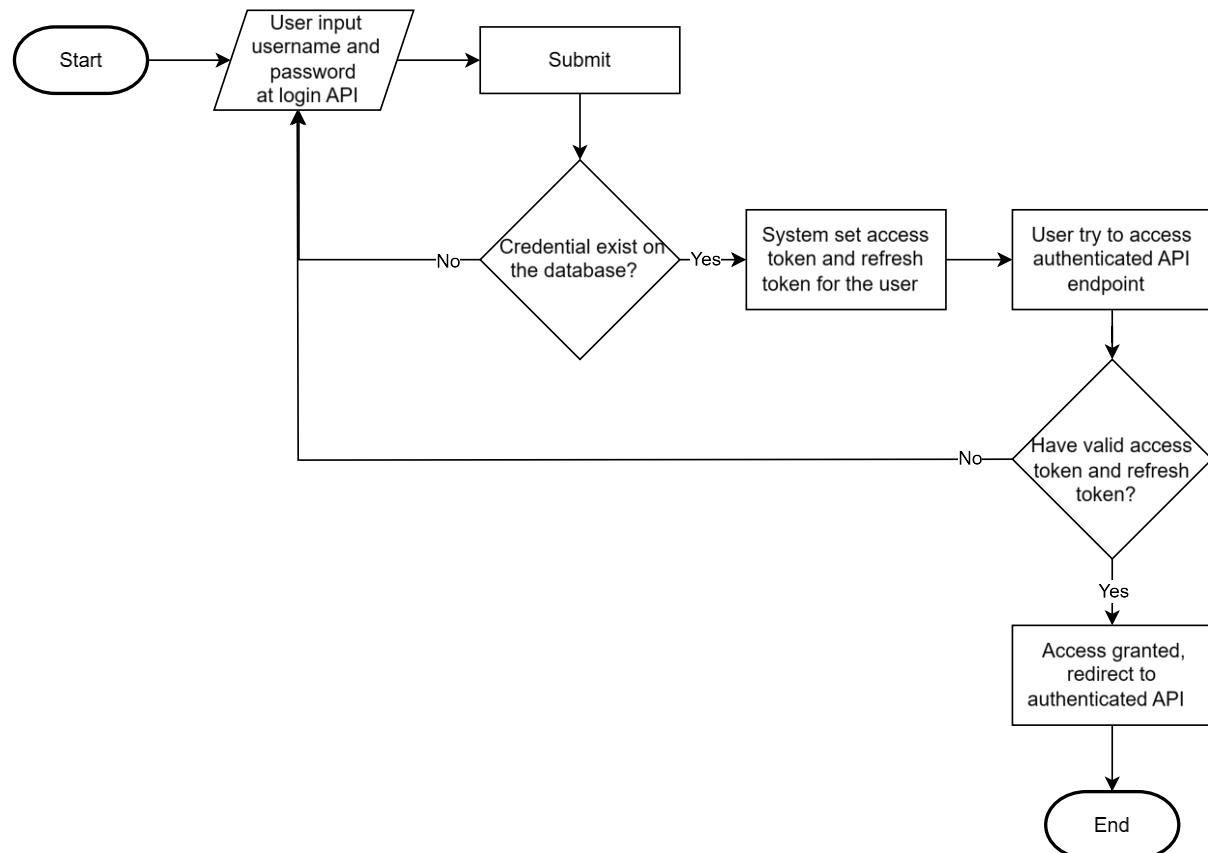
urlpatterns = [
    path('access/', MyTokenObtainPairView.as_view(),
name='token_obtain_pair'),
    path('refresh/', TokenRefreshView.as_view(),
name='token_refresh'),
    path('verify/', TokenVerifyView.as_view(),
name='token_verify'),
    path('blacklist/', TokenBlacklistView.as_view(),
name='token_blacklist'),
]

```

4.2.3.5 Otentikasi dan Otorisasi

4.2.3.5.1 Menerapkan Otentikasi Pengguna Untuk Mengamankan API Endpoints

Otentikasi pengguna diimplementasikan menggunakan JSON Web Token (JWT) yang dimulai dari fungsi *login* hingga tiap-tiap API yang akan diakses oleh *user*. Penerapan ini meliputi beberapa langkah, berikut adalah contoh *flowchart* untuk cara kerja otentikasi pengguna.



Gambar 4.15 Flowchart Otentikasi Pengguna

Flowchart diatas menggambarkan sebuah alur untuk mengakses *authenticated API*. *User* akan memerlukan token untuk dapat mengakses sebuah API yang terotentikasi, yang mana token tersebut diperoleh ketika *user* berhasil menginputkan *username* dan *password* yang sesuai pada *database* pada API *login*. Apabila *user* tidak memiliki token yang valid ketika akan mengakses API yang terotentikasi, maka *user* tidak akan dapat mengaksesnya. Apabila *user* memiliki token yang valid, maka *user* akan dapat mengakses API yang terotentikasi tersebut.

4.2.3.5.2 Mengkonfigurasi Permission Classes Untuk Mengelola Kontrol Akses

Permission classes merupakan sebuah fitur dari DRF yang berguna untuk mengelola kontrol akses API agar terlindungi sehingga tidak semua pengguna dapat mengaksesnya. Pengimplementasian *permission classes* biasa digunakan apabila terdapat API yang

dibutuhkan hanya dapat diakses apabila *user* telah terotentikasi untuk meningkatkan keamanan fungsionalitas.

Penerapan *permission classes* pada sebuah API hanya dengan menambahkan satu baris *source code* diatas sebuah *function* yang akan digunakan untuk membangun API.

- `@permission_classes([IsAuthenticated])`

Permission classes (`[IsAuthenticated]`) dapat digunakan apabila otentikasi ditujukan pada *user*.

- `@permission_classes([IsAdminUser])`

Sedangkan untuk *permission classes* (`[IsAdminUser]`) ini dapat digunakan apabila otentikasi ditujukan pada admin. Sehingga, hanya admin yang dapat mengakses API-nya.

4.2.3.5.3 Menyiapkan Otentikasi Berbasis Token dan Manajemen Sesi

Persiapan untuk implementasi otentikasi berbasis token dan manajemen sesi meliputi beberapa tahap yang dimulai dari instalasi *libraries* yang diperlukan hingga implementasi dan testing.

1) Persiapan *Libraries*

Terdapat dua *libraries* berikut yang perlu diinstal.

- a. `pip install rest_framework_simplejwt`
- b. `pip install rest_framework_simplejwt.token_blacklist`

Setelah instalasi berhasil dilakukan, terdapat perubahan yang perlu ditambahkan pada *file settings.py* agar kedua *libraries* tersebut terdeteksi dan dapat digunakan didalam proyek. Berikut adalah baris kode yang perlu ditambahkan pada bagian `INSTALLED_APPS`.

```
INSTALLED_APPS = [
    ...
    'rest_framework_simplejwt',
    'rest_framework_simplejwt.token_blacklist',
    ...
]
```

2) Kustomisasi Model *User Profile*

Model *User Profile* akan berkaitan dengan manajemen sesi, lebih tepatnya pada field *session_key*. Setiap kali *user* berhasil *login* menggunakan API, sistem akan secara otomatis membuat sebuah kunci sesi yang bersifat unik untuk dimasukkan kedalam token *payload* dan kemudian disimpan pada *database*, lebih tepatnya pada model *user profile* bagian *session_key*.

```
class UserProfile(models.Model):  
    ...  
    session_key = models.CharField(max_length=40,  
        blank=True, null=True)  
    ...
```

3) Kustomisasi Token *Serializer*

Token *serializer* berfungsi untuk mengatur apa saja yang ingin dimasukkan ke dalam token *payload*. Seperti halnya poin 2 di atas, kustomisasi token *serializer* ini bertujuan untuk membuat dan memasukkan kunci sesi ke dalam token *payload*.

Berikut merupakan rangkuman dari baris kode yang terkait dengan kustomisasi token *serializer*. Baris kode secara lengkap dapat dilihat pada Lampiran 4.23.

- a. *Import libraries*
- b. Pendefinisian *class* untuk *custom token serializer*
- c. Validasi sesi *user*
- d. Menambahkan *custom payload* pada token

4) Implementasi Pada *Views* dan URL

Libraries rest_framework_simplejwt memiliki *built-in* modul yang dapat digunakan untuk implementasi API otentifikasi berbasis token, yang diantaranya:

- a. *TokenObtainPairView*
- b. *TokenRefreshView*
- c. *TokenVerifyView*
- d. *TokenBlacklistView*

Karena pada kasus ini terdapat kustomisasi pada sisi token *serializer* untuk memasukkan kunci sesi ke dalam token *payload*, maka *built-in* modul *TokenObtainPairView* perlu diatur agar mengimplementasikan kustomisasi yang telah dilakukan.

Setelah pengaturan pada file `views.py` telah dilakukan, langkah selanjutnya adalah implementasi pada URL. Pada bagian ini, *class* yang telah dibuat pada `views.py` untuk kustomisasi *built-in* modul `TokenObtainPairView` akan digunakan.

4.2.3.6 Integrasi Aplikasi Pihak Ketiga

4.2.3.6.1 Menyertakan *Libraries* Untuk Memperluas Fungsionalitas

Untuk keperluan integrasi dengan kluster Kubernetes dan servis *frontend*, dua *libraries* terkait hal tersebut harus diinstal, diantaranya:

1) Corsheaders

```
pip install Django-corsheaders
```

2) Kubernetes Python Client

```
pip install kubernetes
```

Setelahnya, projek Django sudah siap diintegrasikan dengan kluster Kubernetes dan servis *frontend*.

4.2.3.6.2 Mengkonfigurasi dan Mengintegrasikan Layanan Pihak Ketiga Dengan Backend Django

Setelah *libraries* berhasil diinstal, akan diperlukan proses konfigurasi untuk memastikan keberhasilan integrasi.

1) Corsheaders

Untuk konfigurasi pada corsheaders cukup dilakukan sekali saja didalam file `settings.py` pada bagian `INSTALLED_APPS`.

```
INSTALLED_APPS = [
    ...
    'corsheaders',
    ...
]
```

2) Kubernetes Python Client

Konfigurasi yang diperlukan agar projek Django dapat terintegrasi dengan kluster Kubernetes adalah menambahkan beberapa baris kode pada file Python secara global yang akan digunakan untuk integrasi. Contohnya seperti berikut.

```

from kubernetes import client, config
# Load your Kubernetes configuration, either in-cluster or from a
local Kubeconfig file
config.load_kube_config()
# Initialize the Kubernetes API client
v1 = client.CoreV1Api()

```

4.2.3.7 Implementasi Koneksi Websocket

Implementasi Websocket API dilakukan pada proyek Django bernama **orca_backend_ws** yang telah diinisialisasi sebelumnya. Hal-hal yang diperlukan dalam pengembangan ini diantarnya adalah:

4.2.3.7.1 Instalasi *Libraries*

Libraries yang diperlukan dalam pengembangan Websocket API ini diantaranya adalah:

```
pip install django channels channels redis websockets
```

Libraries tersebut merupakan syarat agar Django dapat menerapkan koneksi websocket dan melakukan *script testing* untuk menguji konektivitas pada websocket tersebut.

4.2.3.7.2 Inisialisasi Setting Aplikasi dan Channel Layers

Langkah ini dilakukan pada file settings.py untuk membuat aplikasi yang telah dibuat dan *libraries* yang diinstall dapat terdeteksi oleh sistem.

```

INSTALLED_APPS = [
    . . .
    'channels', //Libraries untuk membuat koneksi websocket
    'monitoring',
    'shell',
    'sniff',
    'protocolstack',
]

CHANNEL_LAYERS = {
    'default': {
        'BACKEND': 'channels_redis.core.RedisChannelLayer',
        'CONFIG': {
            "hosts": [(env('REDIS_HOST')), env.int('REDIS_PORT'))],
        },
    },
}

```

4.2.3.7.3 Setting File ASGI

File ini berfungsi sebagai pusat routing terhadap setiap consumers yang terdapat pada setiap aplikasi yang telah dibuat. Apabila proses routing ini tidak dilakukan, maka consumers tidak akan berfungsi.

```
application = ProtocolTypeRouter({  
    "http": django_asgi_app,  
    "websocket": AuthMiddlewareStack(  
        URLRouter(  
            shell.routing.websocket_urlpatterns +  
            monitoring.routing.websocket_urlpatterns +  
            sniff.routing.websocket_urlpatterns +  
            protocolstack.routing.websocket_urlpatterns +  
            logs.routing.websocket_urlpatterns  
        )  
    ),  
})
```

4.2.3.7.4 Menerapkan Consumers Pada Setiap Django Application

Consumers berperan sebagai fungsi yang menghubungkan Django Channels dengan *database* Redis sebagai tempat caching untuk informasi yang dilewatkan. Setiap application yang telah diinisialisasi akan digunakan untuk menampung satu jenis fungsi koneksi websocket. Oleh karena itu, mengacu pada application yang telah diinisialisasi sebelumnya, akan terdapat empat consumers yang digunakan diantaranya:

1. **Consumers Monitoring**, berfungsi untuk mendapatkan Key Performance Indicator (KPI) dari komponen UE secara *real-time*.
2. **Consumers Protocol Stack**, berfungsi untuk mendapatkan informasi protokol SCTP dari komponen RAN 5G (CU dan DU) secara *real-time*.
3. **Consumers Shell**, berfungsi untuk melakukan tes konektivitas pada komponen UE dengan perintah ping maupun cURL yang menghasilkan output secara *real-time*.
4. **Consumers Sniff**, berfungsi untuk melakukan sniffing paket data secara *real-time* dari setiap komponen CU, DU, dan UE disertai dengan pemilihan interface yang tersedia pada setiap komponen tersebut.

Keseluruhan baris kode terkait consumers yang telah disebutkan di atas dapat dilihat pada Lampiran 4.24.

4.2.3.7.5 Menerapkan Routing Pada Setiap Django Application

Routing berfungsi untuk membuat consumers dapat diakses melalui sebuah URL endpoint layaknya seperti REST API endpoint. Untuk baris kode secara lengkap dapat dilihat pada Lampiran 4.25.

4.2.3.7.6 Membuat File Testing Untuk Koneksi Websocket

Setelah semua fungsi telah siap untuk melakukan koneksi websocket, hal yang dilakukan selanjutnya adalah mengetes koneksinya menggunakan sebuah file python untuk melihat hasil aktual dari koneksi websocket tersebut. Setiap koneksi websocket akan menggunakan satu file yang berbeda. Baris kode secara lengkap dapat dilihat pada Lampiran

4.2.4 Implementasi *Frontend*

4.2.4.1 Inisialisasi Proyek

4.2.4.1.1 React.Js

React.js adalah *library* JavaScript *front-end* bersifat gratis dan *open source* untuk membangun UI yang interaktif. React dapat digunakan tanpa *framework*. Namun, sebagian besar *framework* memiliki solusi untuk *fetching data*, *generate HTML*, *routing*, dan *split code*. Sehingga lebih baik menggunakan suatu *framework* yang sudah ada seperti Next.js, Gatsby, dan Remix supaya tidak perlu membuat *framework* sendiri nantinya.

Dalam pembuatan *dashboard website* ini, kami menggunakan dua *framework* yaitu Patternfly dan Material UI. Dua *framework* ini menggunakan React. Jadi untuk instalasi React sudah jadi satu pada instalasi *framework*-nya.

4.2.4.1.2 Instalasi Patternfly

Patternfly adalah *framework* yang dibutuhkan untuk mendesain dan membuat komponen-komponen pada *user interface*. Sebelum menginstal Patternfly, pastikan telah menginstal Node.js terlebih dahulu [42]. Berikut langkah-langkah menginstal Patternfly:

1. Instalasi dan Konfigurasi Patternfly React

Dengan menggunakan npm, jalankan perintah berikut di terminal untuk menginstal:

```
npm install @patternfly/react-core -save
```

2. HTML/CSS

HTML/CSS *library* menyediakan kumpulan contoh kode yang dapat digunakan untuk membuat *interface* dengan *style* Patternfly. Untuk menginstalnya jalankan perintah berikut:

```
npm install @patternfly/patternfly --save
```

3. Run the Development Server

Karena ada perubahan dalam *setting-an* CSS dan struktur folder dalam proyek ini, maka untuk menjalankan proyeknya lakukan perintah di bawah ini:

```
npx patch-package @patternfly/react-styles  
npm run postinstall  
//run development server  
npm run dev:clear
```

4.2.4.1.3 Instalasi Material UI

Material UI adalah *framework open source* untuk komponen React yang menerapkan Desain Material Google sehingga memudahkan untuk membuat grafis *front-end* yang diinginkan. Langkah-langkah instalasi Material UI yaitu:

```
npm install @mui/material @emotion/react @emotion/styled
```

4.2.4.2 Pengaturan Aplikasi

4.2.4.2.1 Instalasi Package

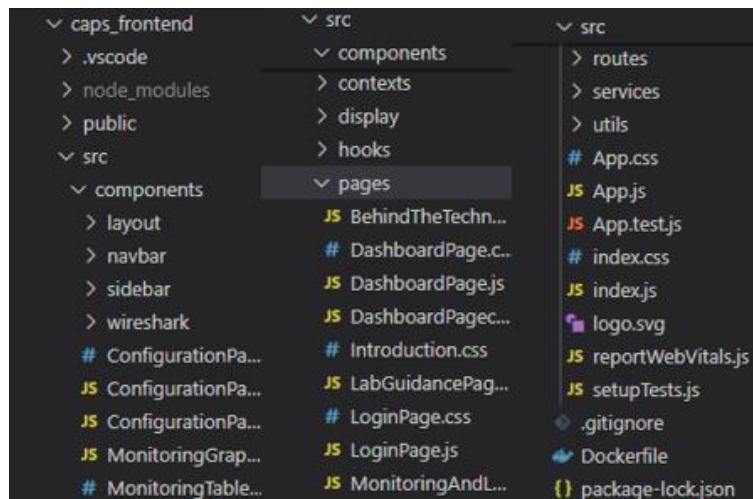
Dalam proyek ini, beberapa *package* akan diinstal untuk mendukung seluruh komponen serta fungsionalitas dari aplikasi yang sedang dibangun. Untuk menginstal beberapa package yang dibutuhkan dalam proyek, gunakan perintah berikut:

```
npm install @patternfly/react-charts @patternfly/react-icons  
@patternfly/react-styles @patternfly/react-table  
@patternfly/react-topology @patternfly/react-tokens @next/swc-  
win32-x64-msvc.
```

Paket-paket ini mencakup berbagai kebutuhan proyek, seperti: *@patternfly/react-charts* untuk komponen pembuatan chart, *@patternfly/react-icons* untuk ikon Patternfly sebagai komponen React, *@patternfly/react-styles* untuk kemampuan CSS-in-JS, *@patternfly/react-table* untuk komponen tabel Patternfly, *@patternfly/react-topology* untuk komponen topology Patternfly, *@patternfly/react-tokens* untuk semua token dan nilai CSS sebagai objek JavaScript, dan *@next/swc-*

win32-x64-msvc sebagai package Speedy Web Compiler (SWC) yang dibutuhkan oleh framework Next.js.

4.2.4.2.2 Struktur Folder



Gambar 4.16 Struktur Folder

Struktur *folder* untuk proyek *website* ini disusun sesuai dengan standar dan desain UI dari Figma, mencakup folder default dari Next.js dan Patternfly. Struktur ini terdiri dari `node_modules`, `public`, `src`, `package-lock.json`, dan `package.json`. Folder `public` berisi assets yang digunakan untuk membangun website, sedangkan folder `src` berisi source code untuk membuat UI *website*.

4.2.4.3 Pembuatan Halaman Website

4.2.4.3.1 Login Page

Login page adalah halaman di mana *user* harus memasukkan *username* dan *password* untuk mengakses *website*. *Login page* dibuat dengan menggunakan *framework* Material UI. Desain halaman ini *simple minimalist*. Form *username* dan *password* di sisi kanan dan di sisi kiri terdapat logo ORCA. Kodingan di bawah mengatur *user* untuk memasukan *username* dan *password* yang telah ditentukan.

```

const handleSubmit = async (event) => {
    event.preventDefault();
    const data = new FormData(event.currentTarget);
    const username = data.get('username');
    const password = data.get('password');
    let hasError = false;
    // Reset errors
    setFormError({ username: '', password: '' });
    setApiError('');
    if (!username) {
        setFormError(prev => ({ ...prev, username: 'Username must
be filled out' }));
        hasError = true;
    }
    if (!password) {
        setFormError(prev => ({ ...prev, password: 'Password
must be filled out' }));
        hasError = true;
    }
    if (hasError) return;
}

```

4.2.4.3.2 Introduction Page dan Navbar

Introduction page adalah halaman awal setelah *user* melakukan *login*. *Introduction page* berisi info mengenai ORCA dan langkah-langkah cara mengkonfigurasi. Pada halaman ini terdapat *sidebar* untuk memudahkan *user* dalam navigasi konten.

```

<PageSidebarBody>
    <Nav id="nav-primary-simple" theme="dark" onSelect={({ groupId
}) => setActiveGroup(groupId)}>
        <NavList id="nav-list-simple">
            <NavItem>
                <NavLink to="/introduction" exact activeClassName="pf-
m-current"> ORCA Introduction </NavLink>
            </NavItem>

```

Navbar adalah komponen website yang berisi menu navigasi yang memudahkan *user* mengakses berbagai halaman. Navbar di *website* ini terdapat logo ORCA di sisi kiri, menu di tengah, dan di sisi kanan ada level dan persentase *completion user* dan tombol *logout*. Menu navbar untuk admin berbeda dengan *user*. Menu admin hanya satu yaitu *user management*. Sedangkan pada halaman *user* terdapat tiga menu yaitu *introduction*, *dashboard*, dan *monitoring*.

```

<div className="navbar__logo">
  <Link to="/">
    
  </Link>
</div>
<nav className="navbar__nav">
  {user.isAdmin ? (
    <Link to="/user-management">User Management</Link>
  ) : (
    <>
      <Link to="/introduction">Introduction</Link>
      <Link to="/dashboard">Dashboard</Link>
      <Link to="/monitoring">Monitoring</Link>
    </>
  ) }
</nav>

```

4.2.4.3.3 Dashboard Page

Dashboard page adalah halaman yang berfokus untuk mengkonfigurasi jaringan 5G di mana kita bisa mengkonfigurasi komponennya yaitu UE, CU, dan DU. Selain itu juga di halaman ini terdapat fitur topologi jaringan dan juga fitur wireshark untuk *sniff function*. Jadi *dashboard page* dibagi ke dalam 3 *card* yang tiap *card*-nya memuat beberapa fitur yaitu *topology graph* dan *components management panel* (*card 1*), *configuration panel* (*card 2*), dan wireshark (*card 3*).

1. *Topology Graph*

Topology graph adalah fitur yang menampilkan topologi jaringan komponen 5G yang akan dikonfigurasi. Di fitur ini kami membuat 6 *nodes* yang terbagi ke dalam 3 grup. Grup pertama yaitu UE hanya berisi 1 *node* saja bernama UE. Grup kedua yaitu RAN berisi 3 *node* yaitu RRU, DU, dan CU. Lalu grup terakhir bernama *core* yang berisikan *node* AMF dan UPF. Kami menggunakan *framework* Patternfly untuk membuat topologi ini. Untuk membuat topologi yang diinginkan, hal utama yang perlu dimodifikasi adalah *custom nodes* dan *custom edges* pada kodingan berikut:

```

const NODES_INIT: NodeModel[] = [
  {
    id: 'UE',
    type: 'node',
    label: 'UE',
    labelPosition: LabelPosition.bottom,
    width: NODE_DIAMETER,
    height: NODE_DIAMETER,
    shape: NodeShape.rect,
    status: NodeStatus.danger,
    x: 70,
    y: 130
  },
  const EDGES: EdgeModel[] = [
  {
    id: `UE to RRU`,
    type: 'data-edge',
    source: 'UE',
    target: 'RRU',
    edgeStyle: EdgeStyle.dashedMd,
    animationSpeed: EdgeAnimationSpeed.mediumSlow
  },...];

```

2. Components Management Panel

Components managements panel adalah fitur yang bertujuan untuk *start*, *stop*, *restart* komponen 5G. Fitur ini masuk ke dalam *card* yang sama dengan *topology graph*, sehingga kodingan untuk fitur ini jadi satu dengan kodingan *topology graph*. Berbeda dengan fitur lainnya, fitur ini dibuat tersembunyi.

Fitur ini baru muncul Ketika *node* pada topologi di-*click*. Misal node UE pada topologi di-*click*, maka *component management panel* untuk UE akan muncul (*pop up*). Tiap *node* pada topologi terintegrasi langsung ke OAI melalui API. Pada diameter *node* terdapat warna yang memiliki artinya sendiri. Hijau berarti *node* atau komponen tersebut berhasil *running*. Kuning artinya komponen belum dikonfigurasi. Terakhir merah yang memiliki arti komponen tidak terhubung ke OAI. Hal ini bisa diatur dalam kode berikut:

```

const determineNodeStatus = (nodeName, podsData) => {
  let matchPart = nodeName.toLowerCase().substring(0, 2);
  const pod = podsData.find(p =>
    p.name.toLowerCase().includes(matchPart));
  if (!pod) return NodeStatus.danger;
  switch (pod.state) {
    case 'Running': return NodeStatus.success;
    case 'Pending': return NodeStatus.warning;
    default: return NodeStatus.danger;
  }
};

```

3. Configuration Panel

Configuration Panel adalah fitur untuk mengkonfigurasi tiap komponen 5G. Sama seperti fitur sebelumnya, fitur ini menggunakan *framework* Patternfly. Fitur ini berbentuk table dengan 3 kolom ‘*Keys*’, ‘*Current Value*’, dan ‘*Set Value*’. Kolom ‘*Keys*’ berisikan variable-variabel yang akan dikonfigurasi tiap komponennya. Kolom ‘*Set Value*’ bebentuk *text input* untuk mengisikan konfigurasinya. Sedangkan kolom ‘*Current Value*’ berisikan *value* yang telah dimasukkan sebelumnya. Pada fitur ini terdapat 3 *button* yaitu UE, DU, CU sehingga kita dapat mengkonfigurasi tiap komponen tersebut.

Untuk membuat tabel pada Patternfly, kita perlu menentukan *key* dan *value* seperti di bawah ini:

```

const componentData = {
  CU: { 'CU ID': '', 'Cell ID': '', 'F1 IP Address': '', ...},
  DU: { 'GNB ID': '', 'DU ID': '', 'Cell ID': '', ...},
  UE: { 'IP Address': '', 'RF Sim Server': '', 'Full Imsi': '',
        ...} };

```

Fitur ini terintegrasi ke *backend* agar *user* dapat mengubah *value* konfigurasi tiap komponennya.

```

const handleSubmit = async () => {
    const payload = transformDataForAPI(values);
    console.log("Payload to be sent:", payload);
    if (Object.keys(payload).length === 0) {
        alert('No changes to submit.');
        return;
    }
    const componentApiVariable = componentToApiMap[selectedComponent];
    const apiUrl = `http://10.30.1.221:8000/api/v1/oai/config_${componentApiVariable}/`;
    const authToken = localStorage.getItem('authToken');
    console.log("Sending payload to:", apiUrl);
    try {
        const response = await fetch(apiUrl, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${authToken}`
            },
            body: JSON.stringify(payload)
        });
        const responseData = await response.json();
        console.log('API Response:', responseData);
        if (response.ok) {
            alert('Configuration updated successfully!');
            const updatedValues = await
        ...} }

```

4. Wireshark

Wireshark adalah fitur yang berfungsi untuk meng-*capture* dan menampilkan detail *network traffic* secara *real-time* pada komponen 5G. Komponen yang akan di-*sniffing* yaitu UE, CU, dan DU. Komponen ini dapat dipilih pada *button dropdown*. Di fitur Wireshark ini, user juga dapat mengunduh file pcap hasil dari *sniffing* paket. Tiap komponen tentunya harus terintegrasi agar fitur *sniffing*-nya dapat berjalan ketika *user* menekan tombol ‘start’.

```

const handleStartClick = async () => {
    if (!isRunning && podsName) {
        console.log('Starting sniff packet for pod:', podsName);
        try {
            const response = await axios.get(`http://10.30.1.221:8000/api/v1/shark/testv1/${podsName}`);
            console.log('Sniff packet API response:', response.data); // Log the response
            setData(response.data); // Assuming response.data contains the Wireshark data
            setIsRunning(true);
        } catch (error) {
            console.error('Error fetching data:', error);
        }
    } else {
        console.log('Stopping sniff packet');
        setIsRunning(false);
    }
};

```

4.2.4.3.4 Monitoring Page

Monitoring page adalah halaman untuk melihat grafik performansi UE. Di halaman ini *user* dapat mengunduh *screenshot* grafik dan melihat value tiap variabel performansi UE. Jadi *monitoring page* terbagi ke dalam dua *card*. *Card* pertama yaitu *UE Simulator: L1 Graph* dan *card* kedua yaitu *Table Value*.

1. *UE Simulator: L1 Graph*

Pada fitur ini, terdapat tiga grafik performansi UE yang berjalan secara *real-time*. Grafik akan otomatis berjalan setelah pengguna berhasil mengkonfigurasi komponen RAN 5G. Pengguna dapat menekan tombol “Stop Stream” jika ingin menghentikan grafik dan dapat menekan tombol “Start Stream” jika ingin grafik berjalan kembali. Lalu pengguna juga dapat mengunduh *screenshot* masing-masing grafik dengan menekan tombol “Download Screenshot Graph”. Berikut adalah kodingan agar grafik dapat berjalan secara *real-time*.

```

const updateData = useCallback(() => {

  if (!isStreaming) return;

  const now = new Date();
  setTimestamp(now.toLocaleDateString('en-GB'));

  const findValueByName = (name) => {
    const entry = data.find(item => item.name === name);
    return entry && entry.value !== null ? parseFloat(entry.value)
    : 0;
  };

  const newDataPoint1 = { name: 'L1 RX Processing', x: now, y:
  ueStopped ? 0 : findValueByName('L1 RX processing') };

  const newDataPoint2 = { name: 'L1 TX Processing', x: now, y:
  ueStopped ? 0 : findValueByName('L1 TX processing') };

  const newDataPoint3 = { name: 'PDSCH Decoding', x: now, y:
  ueStopped ? 0 : findValueByName('PDSCH decoding') };

  ...

  setData1(prevData => [...prevData.filter(point => now - point.x
  <= 60000), newDataPoint1]);

  setData2(prevData => [...prevData.filter(point => now - point.x
  <= 60000), newDataPoint2]);

  setData3(prevData => [...prevData.filter(point => now - point.x
  <= 60000), newDataPoint3]);

  ...
}

```

2. Table Value

Pengguna dapat melihat value pasti dari masing-masing *Key Performance Indicator* (KPI). Value KPI diambil dari *fetching* data UE pada OAI.

```

const fetchData = async () => {

    while (isMounted) {

        setLoading(true);

        try { // Fetch the list of pods

            const podsResponse = await api.get('kube/pods/');

            const podsData = podsResponse.data.pods;

            // Check if podsData is an array

            if (!Array.isArray(podsData) || podsData.length === 0) {

                throw new Error('No pods data found.');

            } // Find the pod with 'oai-nr-ue' in its name

            const uePod = podsData.find(pod => pod.name.includes('oai-nr-
ue'));

            if (!uePod) {

                throw new Error('No UE pod found for the user.');

            } // Check if the pod is running

            if (uePod.state !== 'Running') {

                setAlertMessage(`UE still not configured correctly`);

                setAlertSeverity('warning');

                setSnackbarOpen(true);

                // Set repository values to zero

                const zeroRepositories = initialRepositories.map(repo =>
({ ...repo, value: '0', count: '0', totalTime: '0' }));

                setRepositories(zeroRepositories);

                // Prepare zero data for the graph

                const zeroGraphData = zeroRepositories.map(repo => ({
                    name: repo.content,
                    value: 0 }));

```

4.2.4.3.5 User Account Management Page

User account management page adalah halaman khusus untuk admin di mana terdapat 4 fitur untuk *create user*, *detail completion user*, *edit password user*, dan *delete user*. Berikut contoh kode untuk *create user*.

```

const handleCreateUsers = async () => {
    setCreatingUsers(true);

    const token = localStorage.getItem('authToken');

    const headers = {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
    };

    try {
        const response = await api.post('user/create/', { number: numUsers }, { headers });

        if (response || response.status === 200 || response.data || response.data.success) {
            await refreshUserList();
            showSuccessSnackbar("Users created successfully!");
        } else {
            throw new Error(response.data.message || "Failed to create users");
        }
    } catch (error) {
        console.error('Failed to create users:', error);
        showErrorSnackbar("Failed to create users! " + error.message);
    }
}

```

4.2.4.3.6 Styling User Interface

Styling User Interface (UI) diperlukan untuk mengubah UI agar sesuai dengan desain UI yang diinginkan. Di sini kami menggunakan *Cascading Style Sheets* (CSS) untuk *customization* UI. Ada dua cara yang digunakan untuk mengatur UI yaitu:

1. Membuat File CSS

Cara pertama untuk *styling* UI yaitu dengan membuat *file* CSS terpisah. Jadi kami menulis semua kode CSS yang kami inginkan dalam satu file, lalu file tersebut di-*import* ke file yang berisi kode utama. Foto di bawah adalah contoh *file* CSS yang terpisah.

```

NETRA > caps_frontend > src > components > # Topology.css
15  #reset-view {
16    position: absolute;
17    top: -80px; /* Y coordinate */
18    left: -20px; /* X coordinate */
19  }
20
21  #fit-to-screen {
22    position: absolute;
23    top: -40px; /* Y coordinate */
24    left: -20px; /* X coordinate */
25  }
26
27  .pf-v5-u-screen-reader {
28    position: absolute !important;
29    height: 1px;
30    width: 1px;
31    overflow: hidden;
32    clip: rect(1px, 1px, 1px, 1px);
33    clip-path: inset(50%);
34    white-space: nowrap; /* prevent content from wrapping */
35    border: 0;
36  }
37
38  .pf-topology__node__label__background {
39    filter: none;
40  }

```

Gambar 4.17 File CSS

2. Inline Styles

Cara kedua yaitu kami menuliskan *style* CSS pada kode utama. Tentunya perlu penyesuaian *properties* CSS ke dalam bahasa pemrograman utamanya yaitu JavaScript seperti tabel di bawah ini:

Tabel 4.3 Penyesuaian Properties CSS

CSS	JavaScript
background-color	backgroundColor
font-size	fontSize
margin-bottom	marginBottom
padding-left	paddingLeft
overflow	overflow

Di bawah ini adalah contoh penerapan *inline styles*:

```

<Card      ouiaId="BasicCard"      style={{height:      '700px',
borderRadius: '8px'}}>
  <CardTitle    style={{      backgroundColor:      '#2B7A7B',
color:'#fFffff',      padding:'10px',      marginTop:      '-24px',
marginLeft:      '-24px',      marginRight:      '-24px',
borderTopLeftRadius: '8px', borderTopRightRadius: '8px' }}>
    Wireshark
  </CardTitle>
  <CardBody>
    <Wireshark />
  </CardBody>
</Card>

```

4.2.4.4 Integrasi Backend

4.2.4.4.1 Token Services

Fungsi dari *token service* adalah untuk *token refreshing*. JSON Web Token (JWT) memiliki masa aktif yang dapat disesuaikan dengan kebutuhan aplikasi. Semakin pendek periode masa aktifnya maka semakin aman sistem yang dibangun. *Token refreshing* berfungsi untuk *auto generate token* sebelum *token expired*.

```

export const getNewToken = async () => {
  try {
    const refreshToken = localStorage.getItem('refreshToken');

    const response = await axios.post('http://10.30.1.221:8000/api/v1/token/refresh/', {
      refresh: refreshToken
    });

    const { access, refresh } = response.data;
    localStorage.setItem('authToken', access);
    if (refresh) {
      localStorage.setItem('refreshToken', refresh);
    }
    return access;
  } catch (error) {
    console.error('Error refreshing token:', error);
  }
}

```

Fungsi dari *token service* ini membantu pengembangan frontend untuk mengelola *auto generate token* tersebut dalam suatu fungsi untuk efektifitas kode yang ditujukan untuk melakukan pemanggilan API. Sehingga fitur *token refreshing* dapat terkelola secara efektif dan tersentralisasi.

4.2.4.4.2 API Services

API *service* memiliki fungsi agar pemanggilan API menggunakan *custom url* yang berfungsi untuk menggabungkan fungsi *token service* ke dalam setiap pemanggilan API. Sehingga setiap pemanggilan API akan men-trigger fungsi *token service* untuk melakukan cek terhadap masa aktif *token* dan apabila masa aktif *token* telah mendekati *expired*, maka *token* yang baru akan di-generate dan digunakan dalam pemanggilan API yang akan dilakukan.

```
const api = axios.create({
  baseURL: 'http://10.30.1.221:8000/api/v1/',
}) ;

api.interceptors.request.use(
  async (config) => {
    let authToken = localStorage.getItem('authToken');

    if (isTokenExpired(authToken)) {
      authToken = await getNewToken();
    }

    if (authToken) {
      config.headers['Authorization'] = `Bearer ${authToken}`;
    }
  }
);

api.interceptors.response.use(
  (response) => response,
  (error) => {
    return Promise.reject(error);
  }
);
```

4.2.4.4.3 Protected Routes

Protected route berfungsi untuk melindungi laman yang membutuhkan otentikasi dan otorisasi terhadap pengguna. Saat pengguna mengakses laman tersebut, *protected route* akan melakukan dua hal yaitu mengecek keberadaan *token* di *local storage browser*.

Apabila terdapat *token* di *local storage*, fungsi *protected route* akan memvalidasi *token*. Ketika *token* terbukti valid, pengguna akan diarahkan ke laman tersebut. Sebaliknya jika *token* tidak valid, maka pengguna akan diarahkan ke laman *login*.

```
const ProtectedRoute = ({ children }) => {

  const token = localStorage.getItem('authToken');

  if (!token) {

    return <Navigate to="/auth/login" replace />;

  } return children; };
```

4.2.4.4.4 Check Token

Check token berfungsi untuk memeriksa keberadaan token di *local storage* saat user mengakses root domain/url website. Jika token ditemukan, user akan diarahkan ke halaman yang sesuai berdasarkan peran mereka apabila admin akan diarahkan ke laman *user management*, dan *trainee* ke laman *home*, setelah validasi dilakukan. Jika token tidak ditemukan, akan diarahkan ke laman login.

```
const CheckToken = () => {

  const navigate = useNavigate();

  useEffect(() => {

    const token = localStorage.getItem('authToken');

    if (token) {

      try {

        const decoded = jwtDecode(token);

        if (decoded.is_staff) {

          navigate('/user-management');
```

4.2.4.4.5 API

API adalah penghubung *user interface* ke komponen lain. API digunakan pada beberapa fitur seperti *login*, *logout*, manajemen komponen (*start*, *stop*, *restart*), konfigurasi komponen, *display current config value*, wireshark (*sniff packet*), *logging* komponen, manajemen *user* (CRUD), *user level progression*.

4.2.5 Implementasi Deployment

4.2.5.1 Frontend Deployment

Aplikasi *frontend* dijalankan pada kluster Kubernetes dalam bentuk pod. Sehingga perlu bagi administrator untuk menyiapkan Image agar dapat dijalankan sebagai kontainer. *Frontend* dibungkus menjadi sebuah Image menggunakan Dockerfile. Dockerfile adalah file yang berisi dokumen teks berisi sebuah perintah untuk membangun sebuah Image. Berikut adalah Dockerfile yang digunakan untuk build image Frontend

```
FROM quay.tiplab.local/orca/node:18.17.0

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 3000

CMD ["npm", "start"]
```

Dockerfile tersebut mendefinisikan proses untuk membangun sebuah image Docker yang digunakan untuk aplikasi frontend dengan menggunakan basis image `node:18.17.0`. Langkah pertama adalah masuk ke direktori `/app` dan menyalin semua file frontend ke dalamnya. Kemudian menjalankan perintah `npm install` untuk menginstall semua dependencies yang didefinisikan pada file `package.json`. Image akhir ini kemudian mendefinisikan bahwa kontainer yang dibuat dari image tersebut akan mengekspor port 3000 dan menjalankan aplikasi dengan perintah `npm start`.

4.2.5.2 Backend Deployment

Setelah proses pengembangan yang dilakukan di VM *developer* diselesaikan, proses deployment dibutuhkan agar *source code* dapat digunakan di dalam klaster Kubernetes. Proses *deployment* tersebut memerlukan sebuah Dockerfile yang berisikan baris-baris perintah yang akan dieksekusi oleh sistem untuk membuat sebuah Docker *image* yang diperlukan untuk proses *deployment*. Berikut adalah isi dari Dockerfile tersebut.

```

FROM python:3.10-slim

LABEL maintainer="gintingari73@gmail.com"

# set and install requirement packages

COPY . /app

WORKDIR /install

ENV TZ=Asia/Jakarta

ENV KUBECONFIG=/app/deployment/kube-config

RUN apt-get update && apt-get install -y curl && \
    pip install --no-cache-dir -r /app/requirements.txt && \
    curl -LO https://storage.googleapis.com/kubernetes-
release/release/$(curl -s https://storage.googleapis.com/kubernetes-
release/release/stable.txt)/bin/linux/amd64/kubectl && \
    chmod +x ./kubectl && chmod 600 $KUBECONFIG && \
    mv ./kubectl /usr/local/bin/kubectl && \
    curl -fsSL -o get_helm.sh
    https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 && \
    \
    chmod +x get_helm.sh && ./get_helm.sh && \
    rm -rf /install && rm -rf /var/lib/apt/lists/*
WORKDIR /app

#running service

EXPOSE 8000

CMD [ "python3", "manage.py", "runserver", "0.0.0.0:8000" ]

```

Dockerfile Backend menggunakan image berbasis Python, yang dilengkapi dengan *tools* untuk manajemen Kubernetes, yaitu kubectl dan Helm. Langkah pertamanya diikuti dengan instalasi Python3, Pip, dan Curl. Setelah itu, file requirements.txt disalin ke dalam image, dan perintah pip digunakan untuk menginstal semua dependensi yang terdaftar. Zona waktu sistem diatur ke Asia/Jakarta. Selanjutnya, Dockerfile melakukan penginstalan kubectl dengan mengunduh binary dari Google, mengatur izin eksekusi, dan memindahkannya ke /usr/local/bin untuk dapat akses.

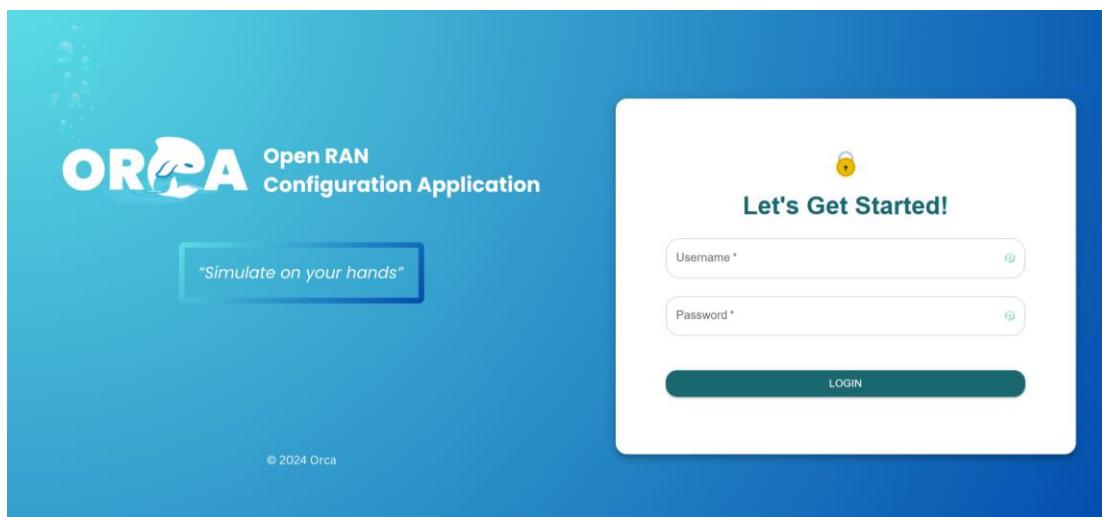
Helm juga diinstal menggunakan skrip dari repositori GitHub resminya, yang secara otomatis menginstal versi terbaru Helm 3. Terakhir, Dockerfile mengkonfigurasi kontainer untuk mengekspos port 8000 dan menginisialisasi server Django pada alamat ini, sehingga memungkinkan akses luar ke aplikasi backend yang dihosting di dalam kontainer.

4.3 Prosedur Pegoperasian

4.3.1 Fitur Otentikasi *User*

Fitur otentikasi *user* pada aplikasi ini terbagi kedalam dua bagian yaitu *login* dan *logout*. Fitur ini memiliki fungsi untuk mengisolasi *resource* yang akan digunakan oleh tiap-tiap *user* sehingga memiliki *resource* yang dependen. Selain itu, otentikasi *user* juga memiliki fungsi untuk meningkatkan keamanan terhadap aplikasi karena tidak semua orang dapat mengaksesnya.

4.3.1.1 *Login User*



Gambar 4.18 Tampilan Login Page

Fitur *login user* memiliki tampilan UI seperti gambar di atas, dimana *user* diharuskan untuk menginputkan *username* beserta *password* yang diberikan oleh admin aplikasi. Kemudian, tombol *login* berfungsi untuk melakukan otentikasi *user* terhadap server. Apabila *user* terotentikasi, maka *user* akan mendapatkan token unik sebagai otentikator dan akan diarahkan ke dalam aplikasi sesuai dengan *role* yang ditentukan. Apabila *role* sebagai admin, maka akan diarahkan ke *dashboard admin page*. Sebaliknya, apabila *role* sebagai *common user*, akan diarahkan ke *introduction page*.

4.3.1.2 Logout User



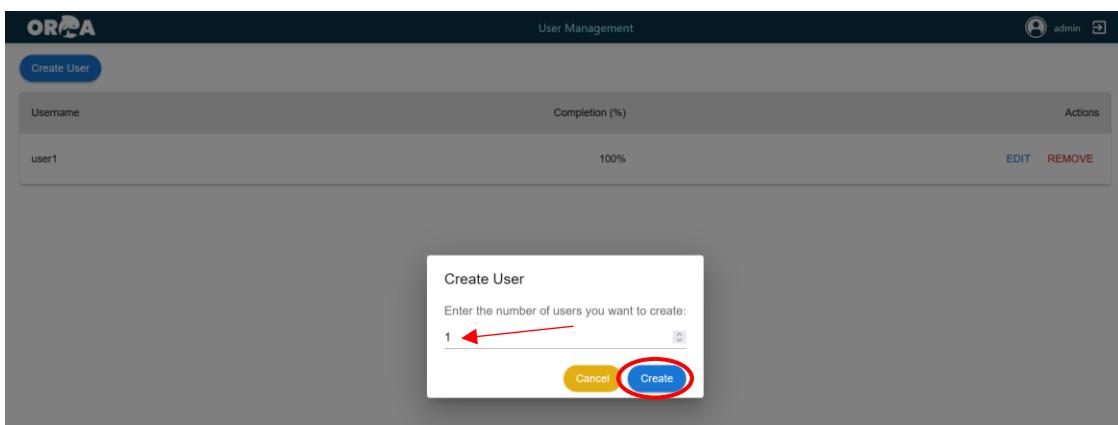
Gambar 4.19 Tampilan Logout User

Fitur *logout user* memiliki tampilan seperti gambar di atas yang diberi tanda kotak merah. Fitur ini terletak di ujung sebelah kanan *navigation bar* yang berupa sebuah *icon*. Apabila *user* akan melakukan *logout* dari aplikasi, *user* diharuskan untuk menekan *icon* tersebut dan kemudian *user* akan diarahkan ke *login page* serta token unik yang digunakan sebagai otentikator akan dihapuskan oleh sistem.

4.3.2 Fitur User Management

Fitur manajemen pengguna dalam aplikasi ini menyediakan kontrol administratif yang luas bagi admin untuk mengelola akun pengguna, termasuk kemampuan untuk membuat, membaca, memperbarui, dan menghapus (CRUD) data pengguna. Setiap pengguna yang terdaftar akan memiliki informasi detail tentang tingkat kemajuan dan level mereka. Secara otomatis, setiap kali pengguna baru dibuat, sistem akan langsung memulai proses instalasi komponen 5G dalam tiga tahap yang berbeda untuk mengonfigurasi lingkungan pengguna tersebut secara spesifik.

4.3.2.1 Create User



Gambar 4.20 Tampilan Create Users

Pengoperasian fitur *create user* memiliki beberapa langkah tahapan, yang pertama adalah menekan tombol untuk *create users* berwarna biru di sebelah kiri tampilan UI. Setelah tombol ditekan akan memunculkan sebuah modal untuk menginputkan jumlah *user* yang akan dibuat dengan jumlah minimal adalah satu. Setelah jumlah *user* ditetapkan,

tombol *create* perlu ditekan agar sistem memulai operasi untuk pembuatan akun *user* beserta *resources*-nya. Setelah *user* berhasil dibuat, akan muncul notifikasi dan tabel *user* akan secara otomatis ter-*update*.

4.3.2.2 Read User

The screenshot shows the 'User Management' section of the ORPA application. A single user entry is displayed in a table. The entry consists of a 'Username' column containing 'user1' and a 'Completion (%)' column showing '100%'. To the right of the table are 'Actions' buttons labeled 'EDIT' and 'REMOVE'. The entire row for 'user1' is highlighted with a red border.

Gambar 4.21 Tampilan Read User

Pengoperasian fitur *read user* hanya terdiri dari satu tahapan saja dimana ketika admin membuka lama *user management*, tabel *user* akan otomatis membaca data *user* berdasarkan *database* dan menampilkannya pada tampilan UI. Tabel *user* terdiri dari kolom *username*, *level*, *completion progress*, dan *actions* untuk meng-*update* dan menghapus akun *user*.

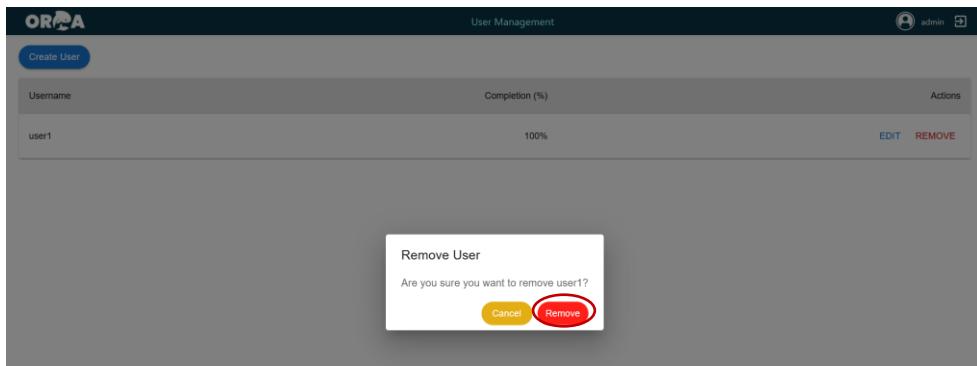
4.3.2.3 Update User

The screenshot shows the 'User Management' section of the ORPA application. A user entry for 'user1' is selected, and a modal dialog titled 'Edit User' has appeared. The modal contains two input fields: 'Username' (containing 'user1') and 'New Password'. Below the fields are 'Cancel' and 'Confirm' buttons. The 'Confirm' button is circled with a red marker. Arrows point from the text descriptions to the corresponding fields in the modal.

Gambar 4.22 Tampilan Update User

Pengoperasian fitur *update user* memiliki beberapa tahapan, dimulai dari menekan tombol edit yang terletak pada kolom *actions* di dalam tabel *user*. Setelah tombol ditekan, sebuah modal edit *user* akan muncul yang berisi kolom input untuk *username* baru dan *password* baru. Syarat untuk mengedit akun *user* adalah tidak boleh ada kolom yang kosong agar *update* dapat diproses oleh sistem. Setelah kolom *username* dan *password* telah terisi, pemrosesan untuk *update* akun *user* akan mulai berjalan tepat setelah tombol *confirm* ditekan. Setelah proses *update* selesai, akan muncul notifikasi dan informasi akun *user* telah ter-*update* dalam *database*.

4.3.2.4 Delete User



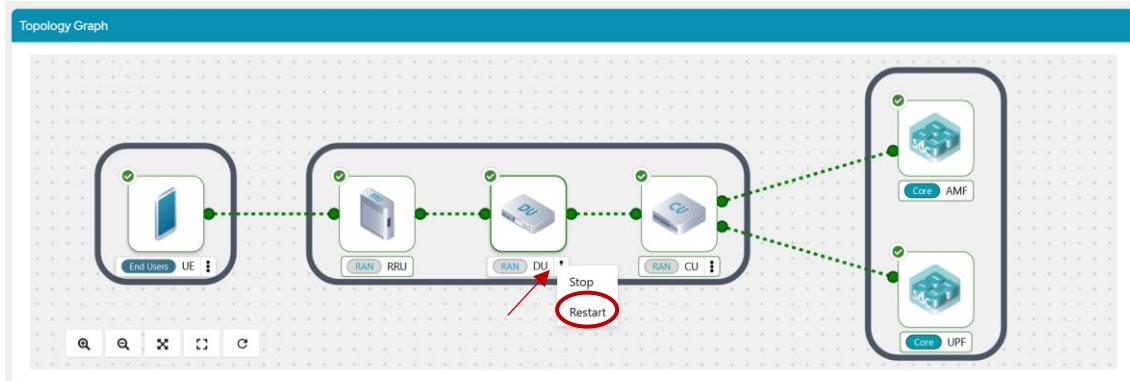
Gambar 4.23 Tampilan Delete User

Pengoperasian fitur *delete user* terdiri dari dua langkah tahapan. Dimulai dari menekan tombol *remove* yang terletak pada kolom *actions* dalam tabel *user*. Setelah tombol ditekan, sebuah modal akan muncul berisi pesan konfirmasi untuk penghapusan akun *user* disertai dengan tombol untuk menghapus akun *user*. Tepat setelah tombol ditekan, proses penghapusan akun *user* beserta *resource*-nya akan dimulai, dan ketika proses telah selesai akan muncul notifikasi dan secara otomatis tabel *user* akan ter-update.

4.3.3 Fitur Grafik Topologi

Fitur manajemen komponen dalam aplikasi ini menyediakan pengawasan komprehensif terhadap status N1, N2, dan N3 dari komponen jaringan fungsional (CNFs) yang diterapkan dalam arsitektur Open Air Interface (OAI). Aplikasi ini memungkinkan *user* untuk mengubah konfigurasi CU/DU dan melakukan *redeploy* pada *Pod* CU/DU secara *real-time*, menjamin kemampuan adaptasi dan responsivitas yang tinggi terhadap kebutuhan operasional.

4.3.3.1 Lifecycle Management



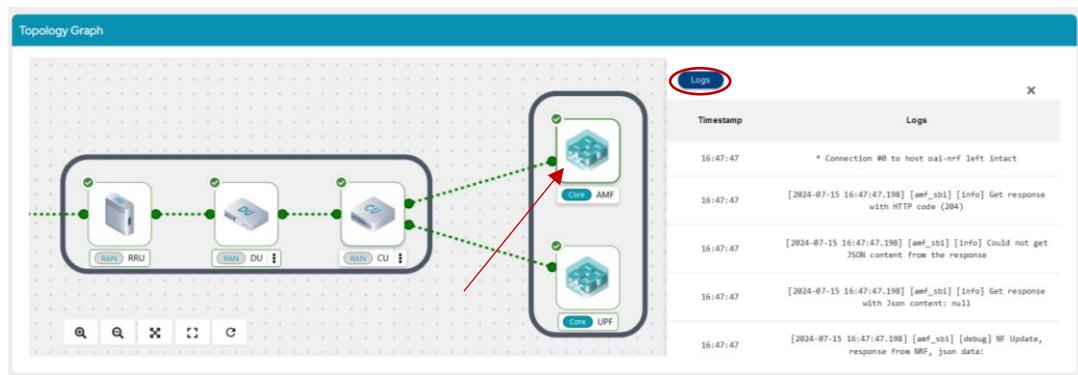
Gambar 4.24 Tampilan Components Lifecycle Management

Lifecycle management merupakan fitur yang terintegrasi dalam *topology graph* pada *dashboard page*. *Lifecycle management* terdapat pada tiap-tiap komponen yang tersedia

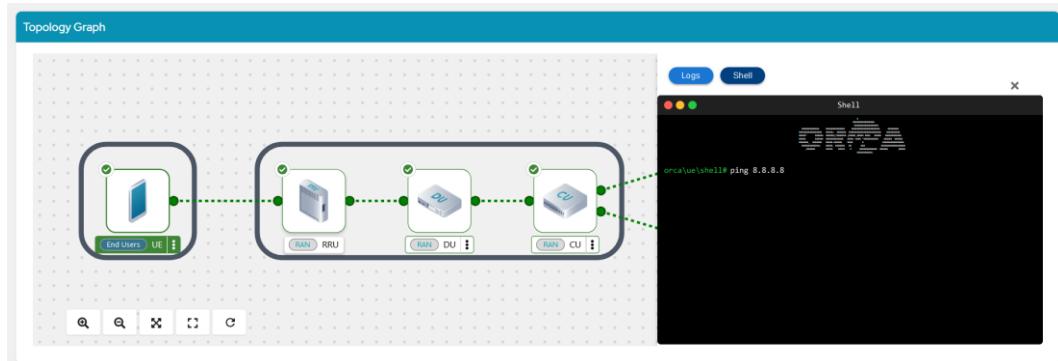
untuk dikonfigurasi. Tiap-tiap komponen memiliki status yang menandakan tersedia atau tidaknya *pod* yang mewakili komponen tersebut. Status komponen terbagi menjadi tiga, yaitu merah yang menandakan *pod* sedang tidak tersedia, kuning yang menandakan *pod* sedang dalam tahap pembuatan, dan hijau yang menandakan *pod* sudah tersedia.

Setiap tombol *start*, *stop*, ataupun *restart* akan memunculkan sebuah modal yang berisi pesan konfirmasi terhadap perintah yang akan dilakukan terhadap komponen. Setelah tombol konfirmasi ditekan, status notifikasi akan muncul berdasarkan hasil dari operasi yang dilakukan serta warna status pada komponen akan ter-*update* otomatis.

4.3.3.2 Component Logging & Testing



Gambar 4.25 Tampilan Component Logging and Testing



Gambar 4.26 Tampilan Shell PING & CURL UE

Fitur *component logging and testing* terdapat pada tiap-tiap komponen yang dapat diakses dengan menekan ikon komponen yang akan dipilih. Setelah ikon ditekan, sebuah sidebar akan muncul pada sisi kanan area *topology graph* yang berisi *log* komponen tersebut beserta *tab* untuk *testing*. Pada UE, terdapat dua *testing* yang dapat dilakukan yaitu tes ping dan tes cURL untuk memastikan UE dapat mengakses internet.

4.3.4 Fitur Component Management

Fitur manajemen komponen dalam aplikasi ini menyediakan pengawasan komprehensif terhadap status N1, N2, dan N3 dari komponen jaringan fungsional (CNFs) yang diterapkan

dalam arsitektur OAI. Aplikasi ini memungkinkan *user* untuk mengubah konfigurasi parameter komponen CU/DU/UE, menjamin kemampuan konfigurasi dengan responsivitas yang tinggi terhadap kebutuhan operasional.

4.3.4.1 Konfigurasi CU

The image shows two side-by-side screenshots. On the left is the 'Configuration Panel' with tabs for UE, DU, and CU (the latter is circled in red). It lists various configuration parameters with their set values and current values. A red arrow points from the 'F1 DU Port' row in the configuration panel to the same row in the Wireshark interface on the right. The Wireshark interface shows a list of network traffic with columns for No., Timestamp, IP src, IP dst, and Protocol Info. Both interfaces have a 'Submit' button at the bottom.

Key	Set Value	Current Value
CU ID	0xe01	✓
Cell ID	12345678L	✓
F1 IP Address	192.168.1.1	✓
F1 CU Port	2152	✓
F1 DU Port	2152	✓
N2 IP Address	172.21.6.1	✓
N3 IP Address	172.21.8.1	✓
MCC	208	✓
MNC	99	✓

Gambar 4.27 Tampilan Konfigurasi CU

Fitur konfigurasi CU terdapat pada *configuration panel* yang terletak di sebelah kiri bawah tampilan UI *dashboard*. Tombol untuk komponen CU terletak di sisi paling kanan, dan ketika tombol tersebut ditekan, *current value* pada komponen CU akan secara otomatis tertampilkan pada kolom sesuai dengan parameter-parameternya.

Apabila *user* ingin mengkonfigurasi *value* dari parameter tertentu, maka *user* diharuskan mengisi *value* yang baru pada kolom *set value* disesuaikan dengan parameter yang ingin dikonfigurasi. Untuk memproses perubahan konfigurasi tersebut, *user* harus menekan tombol *submit* dibagian bawah, setelah itu proses *update value* akan segera diproses. Kemudian, akan muncul notifikasi dari proses *update* berhasil atau tidak, dan kemudian *value* yang dikonfigurasi akan berubah sesuai dengan yang diinginkan.

4.3.4.2 Konfigurasi DU

The screenshot shows two panels side-by-side. On the left is the 'Configuration Panel' with tabs for UE, DU (circled in red), and CU. Below the tabs is a table with columns 'Key', 'Set Value', and 'Current Value'. The table contains the following data:

Key	Set Value	Current Value
GNB ID	0xe41	✓
DU ID	0xe81	✓
Cell ID	12345678L	✓
F1 IP Address	192.168.1.2	✓
F1 CU Port	2152	✓
F1 DU Port	2152	✓
MCC	208	✓
MNC	99	✓
TAC	0xa001	✓

A red arrow points from the 'MCC' row to the 'Submit' button at the bottom of the panel. The 'Submit' button is also circled in red.

On the right is the 'Wireshark' interface, showing a list of network traffic with columns: No., Timestamp, IP src, IP dst, and Protocol Info. A 'Start' button and a 'Select a Component' dropdown are also visible.

Gambar 4.28 Tampilan Konfigurasi CU

Fitur konfigurasi DU terdapat pada *configuration panel* yang terletak di sebelah kiri bawah tampilan UI *dashboard*. Tombol untuk komponen DU terletak di sisi tengah, dan ketika tombol tersebut ditekan, *current value* pada komponen DU akan secara otomatis tertampil pada kolom sesuai dengan parameter-parameternya.

Apabila *user* ingin mengkonfigurasi *value* dari parameter tertentu, maka *user* diharuskan mengisi *value* yang baru pada kolom *set value* disesuaikan dengan parameter yang ingin dikonfigurasi. Untuk memproses perubahan konfigurasi tersebut, *user* harus menekan tombol *submit* dibagian bawah, setelah itu proses *update value* akan segera diproses. Kemudian, akan muncul notifikasi dari proses *update* berhasil atau tidak, dan kemudian *value* yang dikonfigurasi akan berubah sesuai dengan yang diinginkan.

4.3.4.3 Konfigurasi UE

The screenshot shows two panels side-by-side. On the left is the 'Configuration Panel' with tabs for UE (circled in red), DU, and CU. Below the tabs is a table with columns 'Key', 'Set Value', and 'Current Value'. The table contains the following data:

Key	Set Value	Current Value
IP Address	192.168.1.3	✓
RF Sim Server	192.168.1.2	✓
Full IMSI	20890000000101	✓
Full Key	fec86ba6eb707ed08905757b...	✓
OPC	C42449363BAD02B66D16B...	✓
DNN	oai	✓
SST	1	✓
SD	FFFFFF	✓
USRP	rfsim	✓

A red arrow points from the 'SST' row to the 'Submit' button at the bottom of the panel. The 'Submit' button is also circled in red.

On the right is the 'Wireshark' interface, showing a list of network traffic with columns: No., Timestamp, IP src, IP dst, and Protocol Info. A 'Start' button and a 'Select a Component' dropdown are also visible.

Gambar 4.29 Tampilan Konfigurasi UE

Fitur konfigurasi UE terdapat pada *configuration panel* yang terletak di sebelah kiri bawah tampilan UI *dashboard*. Tombol untuk komponen UE terletak di sisi kiri, dan ketika tombol tersebut ditekan, *current value* pada komponen UE akan secara otomatis tertampilkan pada kolom sesuai dengan parameter-parameternya.

Apabila *user* ingin mengkonfigurasi *value* dari parameter tertentu, maka *user* diharuskan mengisi *value* yang baru pada kolom *set value* disesuaikan dengan parameter yang ingin dikonfigurasi. Untuk memproses perubahan konfigurasi tersebut, *user* harus menekan tombol *submit* dibagian bawah, setelah itu proses *update value* akan segera diproses. Kemudian, akan muncul notifikasi dari proses *update* berhasil atau tidak, dan kemudian *value* yang dikonfigurasi akan berubah sesuai dengan yang diinginkan.

4.3.5 Fitur Sniff Function

Fitur *sniffing* dalam aplikasi ini dirancang untuk memfasilitasi analisis trafik jaringan secara komprehensif pada komponen AMF, UPF, CU, dan DU. Aplikasi memungkinkan trafik yang tertangkap ditampilkan melalui *interface web* yang intuitif, memudahkan pengguna dalam melakukan analisis secara *real-time*. Untuk mendukung *troubleshooting* yang lebih mendalam, aplikasi ini juga terintegrasi dengan Wireshark, yang memperkaya kemampuan pengguna dalam mendeteksi dan mengatasi masalah jaringan. Fitur-fitur ini bersama-sama mendukung kebutuhan operasional dalam monitoring dan pengelolaan jaringan 5G, yang esensial untuk pengembangan dan pemeliharaan infrastruktur jaringan yang efisien dan efektif.

4.3.5.1 Protocol Filter

The image shows two side-by-side screenshots. On the left is the 'Configuration Panel' with tabs for UE, DU, and CU. The CU tab is selected, showing fields for CU ID (0xe01), Cell ID (12345678L), F1 IP Address (192.168.1.1), F1 CU Port (2152), F1 DU Port (2152), N2 IP Address (172.21.6.1), N3 IP Address (172.21.8.1), MCC (208), and MNC (99). A 'Submit' button is at the bottom. On the right is the 'Wireshark' interface. It has a 'Sniff' tab selected. A red arrow points from the 'CU' filter field in the top navigation bar to the 'Stop' button. Another red circle highlights the 'f1' filter field. Below the navigation bar is a table with columns: No., Timestamp, IP src, IP dst, and Protocol Info. One row is shown: No. 1, Timestamp 0.000000000, IP src 192.168.1.1, IP dst 192.168.1.2, Protocol Info SCTP 106 HEARTBEAT. A 'Filter' button is at the bottom of the Wireshark window.

Gambar 4.30 Tampilan Protocol Filter

Fitur *protocol filter* memiliki fungsi utama untuk menyaring *protocol* hasil dari proses *sniffing* yang dilakukan terhadap suatu komponen. Pada dasarnya, terdapat banyak macam *protocol* yang didapat dari proses *sniffing* komponen. Oleh karena itu, fitur ini dibuat dengan tujuan untuk memudahkan *user* dalam mencari *protocol* yang diinginkan untuk ditampilkan secara eksklusif.

Prosedur pengoperasian dari fitur *protocol filter* ini adalah dengan cara mengetikkan *protocol* yang ingin ditampilkan secara eksklusif pada kolom *filter* yang terletak di bawah data-data *protocol* hasil proses *sniffing*. Kemudian, secara otomatis *protocol* yang tertampil akan berubah dengan hanya menampilkan *protocol* yang telah di-*filter*.

4.3.5.2 File Capture Download

The screenshot displays two windows side-by-side. On the left is the 'Configuration Panel' with tabs for UE, DU, and CU. Under the CU tab, there is a table with columns 'Key', 'Set Value', and 'Current Value'. The table contains the following data:

Key	Set Value	Current Value
CU ID		0xe01
Cell ID		12345678L
F1 IP Address		192.168.1.1
F1 CU Port		2152
F1 DU Port		2152
N2 IP Address		172.21.6.1
N3 IP Address		172.21.8.1
MCC		208
MNC		99

A 'Submit' button is at the bottom. On the right is the 'Wireshark' interface. It has a 'Sniff' and 'Files' tab, with 'Files' being active. A red box highlights the 'Files' tab. Below it is a table with columns 'Timestamp', 'File Name', 'Size', and 'Actions'. The table shows four entries:

Timestamp	File Name	Size	Actions
7/15/2024 6:18:22 PM	user1_cu-level1_f1_20240715_171640.pcap	1.89 KB	
7/15/2024 3:02:25 AM	user1_du-level1_f1_20240715_020152.pcap	816 B	
7/15/2024 2:57:05 AM	user1_du-level1_f1_20240715_015652.pcap	816 B	
7/15/2024 2:53:08 AM	user1_cu-level1_f1_20240715_015211.pcap	816 B	

Gambar 4.31 Tampilan File Capture Download

Fitur *file capture download* merupakan bagian dari fitur *sniffing*. Fitur ini merupakan sebuah fitur pelengkap yang berguna untuk menyimpan hasil proses *sniffing* yang telah dilakukan. Hal ini dapat berguna apabila hasil proses *sniffing* ditujukan untuk keperluan lebih lanjut seperti keperluan riset atau lainnya yang memerlukan hasil *sniffing* terhadap suatu komponen.

Prosedur pengoperasian pada fitur ini memiliki tahapan yang sederhana. Apabila proses *sniffing* telah selesai dilakukan, secara otomatis sistem akan menyimpannya dalam bentuk **file pcap** pada *database* yang kemudian akan ditampilkan pada *tab files* yang terletak di dalam bagian Wireshark pada tampilan UI *dashboard*. *User* dapat men-*download* seluruh *file* yang tertampil pada *tab files* dalam bentuk format **pcap**.

4.3.6 Fitur Monitoring

Fitur monitoring pada aplikasi ini mencakup kemampuan *logging* dan monitoring yang luas untuk fungsi jaringan kontainer OpenAirInterface (OAI). Aplikasi ini menawarkan solusi log yang terpusat, memudahkan pengguna untuk mencari dan menganalisis log dari berbagai komponen dalam satu tempat.



Gambar 4.32 Tampilan Grafik Monitoring

Table Value			
Key Performance Indicator	Value (us)	Count	Total Time (us)
L1 TX processing	278.591	534786	1896.909
ULSCH encoding	50.514	8916	393.090
L1 RX processing	363.530	1416087	3885.333
UL Indication	89.215	534786	1626.870
PDSCH receiver	42.859	8917	601.514
PDSCH decoding	482.562	8917	13264.265
-> Deinterleave	5.124	8917	304.916
-> Rate Umatch	9.757	8917	193.135
-> LDPC Decode	297.504	8917	1513.700
PDSCH unscrambling	3.648	8917	84.928
PDCCH handling	85.754	1424322	1111.962

Gambar 4.33 Tampilan Table Value Monitoring

Selain itu, aplikasi ini menyediakan analisis log secara *real-time* dengan sistem *alert* yang dapat disesuaikan berdasarkan aturan tertentu. Ini memungkinkan user untuk cepat mengidentifikasi dan menanggapi kejadian tak terduga atau masalah operasional jaringan 5G.

BAB 5

PENGUJIAN SISTEM

5.1 Skenario Umum Pengujian

Dalam skenario umum pengujian aplikasi, dilakukan empat jenis pengujian utama. Pertama, pengujian API untuk memastikan bahwa setiap API berfungsi dengan baik dan benar. Kedua, pengujian *End-to-End* (E2E) yang bertujuan untuk memverifikasi seluruh alur kerja aplikasi dari perspektif pengguna. Ketiga, pengujian performa untuk mengukur batas kapasitas sistem aplikasi, dengan menilai seberapa banyak pengguna yang dapat menggunakan aplikasi hingga mencapai kapasitas maksimum. Terakhir, *User Acceptance Testing* (UAT) bertujuan menilai kepuasan pengguna terhadap aplikasi. UAT melibatkan pengguna akhir, admin, dan calon pembeli untuk memberikan *feedback* terhadap fungsionalitas dan pengalaman pengguna. Pengujian ini dilakukan di lingkungan pra-produksi setelah semua pengujian teknis selesai, guna memastikan aplikasi memenuhi ekspektasi pengguna sebelum diluncurkan ke produksi.

5.2 Detil Pengujian

Pengujian aplikasi kami melibatkan empat jenis utama pengujian yang dirancang untuk memastikan fungsionalitas, alur, kinerja, dan kepuasan pengguna terhadap aplikasi. Keempat jenis pengujian ini adalah *API Testing*, *End-to-End (E2E) Testing*, *Performance Testing*, dan *User Acceptance Testing* (UAT). Setiap jenis pengujian memiliki tujuan dan metodologi spesifik yang membantu memastikan bahwa aplikasi berfungsi sesuai dengan harapan pengguna dan standar kualitas yang telah ditetapkan. Berikut ini adalah penjelasan lebih rinci tentang skema masing-masing pengujian.

5.2.1 API Testing

API testing bertujuan untuk memastikan bahwa seluruh WebSocket API dan REST API berjalan sesuai dengan yang diharapkan menggunakan metode *Whitebox Testing*. Menggunakan metode ini, penguji berinteraksi dengan API yang telah tersedia untuk memastikan fungsionalitasnya bekerja dengan baik dan benar. Selain itu, penguji juga memiliki akses ke dalam *source code* yang apabila terdapat error selama testing, penguji dapat langsung memperbaiki kode yang error tersebut. Pada aplikasi Postman, penguji hanya memberikan input, dan memeriksa output terhadap spesifikasi yang diharapkan. Pengujian untuk setiap API akan dilakukan sebanyak 30x percobaan agar semakin meyakinkan fungsionalitasnya bekerja tanpa kendala. Berikut adalah tabel yang mendeskripsikan alur dari *API testing* yang akan dilakukan.

Tabel 5.1 Langkah API Testing

Langkah	Deskripsi
Membuat dan Menyimpan Request	Buka Postman, buat request baru, masukkan URL endpoint, pilih metode HTTP, dan tambahkan header serta body yang diperlukan. Simpan request dalam collection.
Mengirim Request dan Memeriksa Respons	Kirim request dengan mengklik tombol Send dan periksa status code serta body respons untuk memastikan sesuai dengan yang diharapkan.
Menganalisis Hasil Test	Analisis hasil pengujian yang ditampilkan, periksa test yang lulus atau gagal, dan identifikasi serta laporkan masalah jika ada.

Pengujian dilakukan menggunakan Postman dan Visual Studio Code dengan hasil yang diharapkan dari pengujian ini adalah semua API berfungsi sesuai spesifikasi yang diharapkan, interaksi antar *libraries* tidak menghasilkan kesalahan, dan sistem mampu menangani input yang tidak valid dengan tepat.

Pada tahap ini, *testing* untuk tipe REST API yang tersedia akan dilakukan menggunakan aplikasi Postman untuk melihat respons dari API tersebut beserta dengan HTTP *status code* yang dikeluarkan. Sedangkan *testing* untuk tipe WebSocket API yang tersedia akan dilakukan menggunakan bantuan aplikasi *code editor* untuk menjalankan *script* yang telah dibuat menggunakan Django *libraries* bernama Websockets untuk melakukan testing koneksi WebSocket secara lokal. Seluruh API akan dikelompokkan menjadi lima bagian berdasarkan jenis dan fungsionalitasnya untuk memudahkan pelaksanaan *testing* dan membuatnya lebih terstruktur.

5.2.1.1 User Management APIs

API terkait *user management* memiliki fungsi krusial dalam pembuatan, pengubahan, dan penghapusan akun *user* yang akan digunakan dalam aplikasi. API ini hanya dapat diakses oleh akun admin terkecuali satu, yang dapat digunakan oleh user untuk melihat informasi dari akun miliknya. Berikut adalah daftar API *endpoint* terkait *user management*.

Tabel 5.2 List API User Management

Category	Method	API Endpoint
----------	--------	--------------

<i>Create user</i>	<i>POST</i>	<i>{{base_url}}/api/v1/user/create/</i>
<i>List all users</i>	<i>GET</i>	<i>{{base_url}}/api/v1/user/list/</i>
<i>Update user's password</i>	<i>PATCH</i>	<i>{{base_url}}/api/v1/user/update/{{users_id}}/</i>
<i>Delete user</i>	<i>DELETE</i>	<i>{{base_url}}/api/v1/user/delete/{{users_id}}/</i>
<i>Get requested user information</i>	<i>GET</i>	<i>{{base_url}}/api/v1/user/information/</i>

Tabel di atas berisi daftar REST API *endpoint* yang tersedia terkait dengan *user management*. Setiap API di atas memiliki fungsionalitasnya masing-masing, berikut akan disajikan proses testing seluruh fungsionalitas untuk role admin dan user melalui User Management APIs yang dilakukan menggunakan aplikasi Postman.

1. Admin Berhasil Membuat Akun User Baru

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** {{base_url}}/api/v1/user/create/
- Headers:**
 - Authorization: Bearer eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ...
 - Key
- Body:** (Empty)
- Response Status:** 201 Created
- Response Body:**

```

1 {
2   "message": "Users created successfully: ['user1']"
3 }
```

Gambar 5.1 Response API *Create user*

2. Admin Berhasil Melihat Daftar Akun User

The screenshot shows the ORCA API testing interface. At the top, it displays the URL: `HTTP ORCA / APIs / user / List All User Account`. Below the URL, there are buttons for `Save` and `Share`. The main area shows a `GET` request with the URL `((base_url))/api/v1/user/list/`. The `Headers` tab is selected, showing a table with one row for `Authorization` containing the value `Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...`. Other tabs include `Params`, `Body`, `Scripts`, `Tests`, and `Settings`. A `Cookies` section is also present. Below the headers, the `Body` tab is selected, showing a JSON response with a single user account object:

```

1 [
2   {
3     "id": 49,
4     "username": "user1",
5     "profile": {
6       "level": 1,
7       "completion": 0.0
8     }
9   }
10 ]

```

At the bottom, status information is shown: `Status: 200 OK Time: 227 ms Size: 379 B`, along with `Save as example` and other icons.

Gambar 5.2 Response API Read All User

3. Admin Berhasil Mengubah Password Akun User

The screenshot shows the ORCA API testing interface. At the top, it displays the URL: `HTTP ORCA / APIs / user / Edit Username or Password`. Below the URL, there are buttons for `Save` and `Share`. The main area shows a `PATCH` request with the URL `((base_url))/api/v1/user/update/49/`. The `Body` tab is selected, showing a table with one row for `password` with the value `1user`. Other tabs include `Params`, `Authorization`, `Headers`, `Scripts`, `Tests`, and `Settings`. A `Cookies` section is also present. Below the body, the `Body` tab is selected, showing a JSON response with the updated username:

```

1 {
2   "username": "user1"
3 }

```

At the bottom, status information is shown: `Status: 200 OK Time: 1671 ms Size: 337 B`, along with `Save as example` and other icons.

Gambar 5.3 Response API Update Password User

4. Admin Berhasil Menghapus Akun User

The screenshot shows the ORCA API testing interface. At the top, it says "ORCA / APIs / user / Delete User Account". Below that, a "DELETE" method is selected with the URL "{{base_url}}/api/v1/user/delete/49/". The "Headers (7)" tab is active, showing a table with one row for "Authorization" containing the value "Bearer eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ...". The "Body" tab shows a JSON response with the key "message": "User and associated Kubernetes resources deleted successfully.". The status bar at the bottom indicates "Status: 204 No Content Time: 10.51 s Size: 397 B".

Gambar 5.4 Response API Delete User

5. User Berhasil Melihat Informasi Akunnya

The screenshot shows the ORCA API testing interface. At the top, it says "ORCA / APIs / user / Information of Requested User Account". Below that, a "GET" method is selected with the URL "{{base_url}}/api/v1/user/information/". The "Headers (7)" tab is active, showing a table with one row for "Authorization" containing the value "Bearer eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ...". The "Body" tab shows a JSON response with the key "username": "user1", "level": 1, "completion": 0.0. The status bar at the bottom indicates "Status: 200 OK Time: 370 ms Size: 357 B".

Gambar 5.5 Response API Get User Info

5.2.1.2 Token APIs

Terdiri dari API yang berkaitan dengan JSON Web Token (JWT) untuk layanan otentikasi dan otorisasi yang digunakan dalam pengembangan ini. Berikut adalah daftar API *endpoint* terkait manajemen token.

Tabel 5.3 List API Token

<i>Category</i>	<i>Method</i>	<i>API Endpoint</i>
<i>Login</i>	<i>POST</i>	<i>{{base_url}}/api/v1/token/access/</i>
<i>Logout</i>	<i>POST</i>	<i>{{base_url}}/api/v1/token/blacklist/</i>
<i>Refresh Token</i>	<i>POST</i>	<i>{{base_url}}/api/v1/token/refresh/</i>

<i>Verify Token</i>	<i>POST</i>	<i>{base_url}/api/v1/token/verify/</i>
---------------------	-------------	--

Tabel di atas berisi daftar REST API *endpoint* yang tersedia terkait dengan manajemen JSON Web Token (JWT). Setiap API di atas memiliki fungsionalitasnya masing-masing, berikut akan disajikan proses testing seluruh fungsionalitas Token APIs yang dilakukan menggunakan aplikasi Postman.

1. Admin/User berhasil mendapatkan token untuk *login* ke website dengan input data yang diperlukan dari sisi user adalah username dan password yang telah dibuat menggunakan User Management API.

The screenshot shows the Postman interface with the following details:

- Request URL:** {{base_url}}/api/v1/token/access/
- Method:** POST
- Body:** form-data (selected)
- Params:** none
- Headers:** (8)
- Body (form-data):**

Key	Value	Description	Bulk Edit
username	Text: admin		
password	Text: orca		
Key	Text: Value	Description	
- Response:**
 - Status: 200 OK
 - Time: 3.36 s
 - Size: 1.13 KB
 - Content-Type: application/json; charset=utf-8
 - Content-Length: 1024
 - Expires: -1
 - Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
 - Set-Cookie: session_id=b0321a80-cc05-4bb8-bcdd-24499563392d; expires=Thu, 01-Jan-1970 00:00:00 UTC; path=/; secure; HttpOnly

```

1 {
2   "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCj9.
3     eyJ0b2tlbl9oeXBlijoiiYWhjZXNzIwiZXhwIjoxNzE5MTTyMjQ1LCJpYXQiOjE3MTkxNTUwNDUsImp0aSI6Ijg0M2MzMmI0NzE4YTRmYzd1Mz
4     A4Zjg0YmQwMTJkODVjIiwidXNlc19pZC16MSwiaXNfc3RhZmY1OnRydWUsIm1zX3N1cGydXNlc16dHJ1ZSwic2Vzc2lvb19pZC16ImIwMzIx
5     YTgwLWNjMDUtNGJ10C1iy2RkLTi0NDk5NTyZMzkyZC1sImxldmVsIjoxfQ.PZGUAjGLWgtkzPG6emIx-hNeFFnU1o0P7xiccBh7s",
6   "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCj9.
7     eyJ0b2tlbl9oeXBlijoicmVmcmVzaC1sImV4cI6MTexOTI0MTQ0NSwiaWF0IjoxNzE5MTU1MDQ1LCJqdGkiOiIwYjVhMTA1ZTNjNWY0Yjk4Ym
8     Y0NWY0YTlkMjh1OTE3NSIsInVzZXJfaWQiOjEsIm1zX3N0YWZmIjp0cnVlLCJpc19zdX8lcnVzZXIiOnRydWUsInNlc3Npb25faWQiOjIiMDMy
9     MWE4MC1jYzA1LTr1YjgtYmNkZC0yNDQ50TU2Mz5MmQilCJsZXZlbCI6MX0.-XRff4zV45RTBsNdzNigiqY3jvAeeAZ7I6nq4Jt3gs",
10   "session_id": "b0321a80-cc05-4bb8-bcdd-24499563392d",

```

Gambar 5.6 Response API Login

2. Admin/User berhasil menghapus dan mem-*blacklist* token yang sedang digunakan ketika *logout* dari website dengan input data yang diperlukan dari sisi user adalah *refresh* token yang telah didapat sebelumnya.

HTTP ORCA / APIs / token / Logout

POST | {{base_url}}/api/v1/token/blacklist/

Params Authorization Headers (8) **Body** Scripts Tests Settings Cookies

None form-data x-www-form-urlencoded raw binary GraphQL

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> refresh	Text eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0...		
Key	Text Value	Description	

Body Cookies Headers (10) Test Results Status: 200 OK Time: 175 ms Size: 312 B Save as example

Pretty Raw Preview Visualize JSON

1 {}

Gambar 5.7 Response API Logout

3. Admin/User berhasil me-refresh access token untuk memperpanjang durasi dari validitas token yang diperlukan untuk mengakses *authenticated* API dengan input data yang diperlukan dari sisi user adalah *access token* yang sedang digunakan.

HTTP ORCA / APIs / token / Refresh Token

POST | {{base_url}}/api/v1/token/refresh/

Params Authorization Headers (9) **Body** Scripts Tests Settings Cookies

None form-data x-www-form-urlencoded raw binary GraphQL

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> refresh	Text eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0...		
Key	Text Value	Description	

Body Cookies Headers (10) Test Results Status: 200 OK Time: 121 ms Size: 688 B Save as example

Pretty Raw Preview Visualize JSON

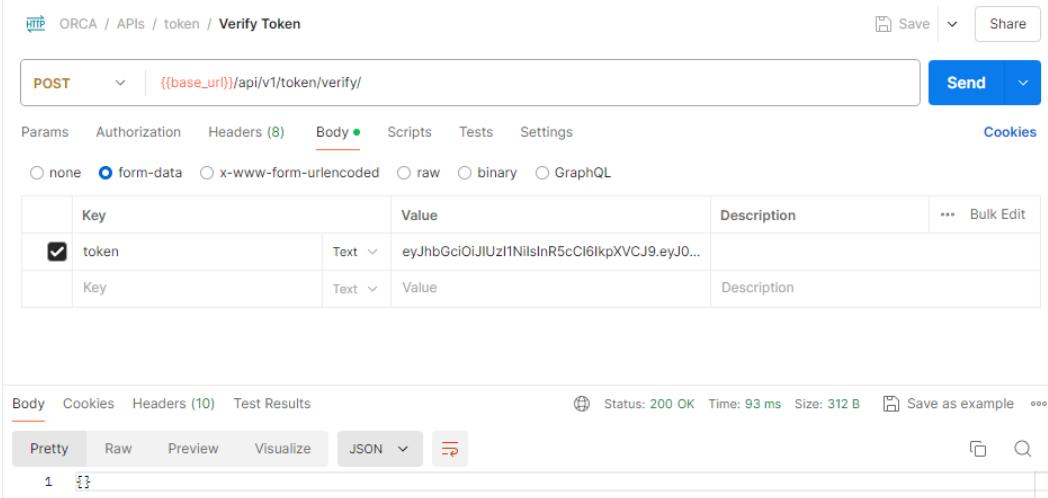
```

1 {
2   "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ0b2tlbl9oeXlibljojoiYmNjZXNzIwiZXhwIjoxNzE5MTY0NDYxLCJpYXQiOjE3MTkxNTcyMjcsImp0aSI6IjcwMGYwMGM1MTdjMTRmODFhO
MxYzJjNGYwMGZhZDjhIiwidXNlc19pZCI6NTAsImIzX3N0YWZmIjpmYWxzSwiaXNfc3VwZXJ1c2VyIjpmYWxzSwic2Vzc2lvb19pZCI6ImZm
NjhNmFkLTyWnjMtNGFlyi04ZWkLtk5ZTzmOTA5ZTY3MSIsImxldmVsIjoxfQ.MXrmsaYNC8W2oDucbpmplU2fhK-FJ3U04UmCXL3Rf0"
3 }

```

Gambar 5.8 Response API Refresh Token

4. Admin/User berhasil memverifikasi validitas token yang sedang digunakan untuk memutuskan apakah user masih memiliki otoritas mengakses *authenticated* API dengan input data yang diperlukan dari sisi user adalah *access token*.



Gambar 5.9 Response API Verify Token

5.2.1.3 Kubernetes APIs

API terkait kluster Kubernetes terdiri dari beberapa tipe *endpoint* yang memiliki pola dan fungsi yang mirip, sehingga daftar REST API akan dipersingkat hanya untuk tiap-tiap pola yang berbeda. Fungsi API ini meliputi manajemen *resources* di dalam kluster Kubernetes termasuk *deployment*, *pods*, dan *namespace*. Selain itu, terdapat Websocket API yang digunakan untuk mengintegrasikan data-data penting yang bersifat *real-time*. Berikut adalah daftar API-nya.

Tabel 5.4 List Kubernetes API

<i>Category</i>	<i>Method</i>	<i>API Endpoint</i>
<i>Get Requested Pod List</i>	<i>GET</i>	<i>{{base_url}}/api/v1/kube/pods/</i>
<i>Get Requested Deployment List</i>	<i>GET</i>	<i>{{base_url}}/api/v1/kube/deployments/</i>
<i>Get Requested Pod Log</i>	<i>GET</i>	<i>{{base_url}}/api/v1/kube/pods/{{pods_name}}/logs/</i>
<i>Restart 5G Components</i>	<i>POST</i>	<i>{{base_url}}/api/v1/kube/restart_{{component_name}}/</i>
<i>Get Core (AMF & UPF) Log</i>	<i>GET</i>	<i> {{base_url}}/api/v1/kube/get_amf_logs/ & {{base_url}}/api/v1/kube/get_upf_logs/</i>

<i>Get Core (AMF & UPF) Deployment Status</i>	<i>GET</i>	<i>{base_url}/api/v1/kube/get_amf_deployments/ & {base_url}/api/v1/kube/get_upf_deployments/</i>
---	------------	--

Tabel 5.5 List Kubernetes Websocket API

<i>Caterogy</i>	<i>Endpoint</i>
<i>Ping and cURL command</i>	<i>ws://{{base_url}}/ws/shell/</i>
<i>Get UE Monitoring (Key Performance Indicator)</i>	<i>ws://{{base_url}}/ws/monitoring/</i>
<i>Get SCTP Protocol Information</i>	<i>ws://{{base_url}}/ws/protocolstack/</i>

Kedua tabel di atas berisi daftar REST API dan Websocket API *endpoint* yang tersedia terkait dengan manajemen kluster Kubernetes. Setiap API di atas memiliki fungsionalitasnya masing-masing, berikut akan disajikan proses testing seluruh fungsionalitas Kubernetes APIs yang dilakukan menggunakan aplikasi Postman untuk tipe REST API dan juga *code editor* dengan Django *libraries* untuk tipe Websocket API.

1. User berhasil mendapatkan daftar informasi pod Kubernetes miliknya. Informasi ini akan digunakan untuk manajemen pod yang dilakukan oleh user.

The screenshot shows the ORCA API client interface. At the top, it says "ORCA / APIs / kube / Get Requested Pods". Below that is a search bar with "GET" selected and the URL "{{base_url}}/api/v1/kube/pods". To the right of the search bar are "Save" and "Share" buttons. Underneath the search bar are tabs for "Params", "Authorization", "Headers (7)", "Body", "Scripts", "Tests", "Settings", and "Cookies". The "Headers (7)" tab is currently active. It shows a table with two rows: one for "Authorization" with the value "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ..." and one for "Key" with the value "Value". Below the headers is a "Body" section with tabs for "Pretty", "Raw", "Preview", "Visualize", "JSON", and "CSV". The "Pretty" tab is selected. The response body is a JSON object:

```

1  {
2   "pods": [
3     {
4       "name": "oai-nr-ue-level1-user1-76b96bd65-dwdsh",
5       "ip": null,
6       "network_status": "Not available",
7       "state": "Pending",
8       "namespace": "user1",
9       "node": "node3"
10    }
]
  
```

At the bottom right of the response area, there are "Save as example" and "Copy" buttons. Above the response area, status information is displayed: "Status: 200 OK Time: 2.37 s Size: 479 B".

Gambar 5.10 API Response List Pod

2. User berhasil mendapatkan daftar informasi deployment Kubernetes miliknya. Informasi ini akan digunakan untuk manajemen siklus hidup dari komponen 5G.

The screenshot shows the ORCA API client interface. At the top, it says "HTTP ORCA / APIs / kube / Get Requested Deployments". Below that is a search bar with "GET" selected and the URL "127.0.0.1:8001/api/v1/kube/deployments/". To the right of the search bar are "Save" and "Share" buttons. Below the search bar are tabs for "Params", "Authorization", "Headers (7)", "Body", "Scripts", "Tests", and "Settings". The "Headers" tab is currently active, showing one header: "Authorization" with the value "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...". There are also "6 hidden" headers. To the right of the headers are "Cookies" and "Send" buttons. Below the headers table is a table with columns "Key", "Value", "Description", "Bulk Edit", and "Presets". The first row has a checked checkbox next to "Authorization". The second row is empty with "Key", "Value", and "Description" columns. At the bottom of the interface, there are tabs for "Body", "Cookies", "Headers (10)", and "Test Results". The "Test Results" tab is active, showing "Status: 200 OK Time: 531 ms Size: 942 B". Below this are buttons for "Pretty", "Raw", "Preview", "Visualize", "JSON", and "Copy". The main content area displays the JSON response:

```

1 [
2   {
3     "deployment_name": "oai-cu-level1-user1",
4     "namespace": "user1",
5     "replicas": 0,
6     "available_replicas": null,
7     "ready_replicas": null,
8     "updated_replicas": null,
9     "strategy": "Recreate",
10    "conditions": [
11      ...
12    ]
13  }
14 ]

```

Gambar 5.11 API Response List Deployment

3. User berhasil mendapatkan informasi log dari pod Kubernetes miliknya. Log ini akan berguna ketika user akan melakukan konfigurasi pada komponen 5G.

The screenshot shows the ORCA API client interface. At the top, it says "HTTP ORCA / APIs / kube / Get Requested Pod's Logs". Below that is a search bar with "GET" selected and the URL "({base_url})/api/v1/kube/pods/oai-nr-ue-level1-user1-ccd7864b7-gd2d2/logs/". To the right of the search bar are "Save" and "Share" buttons. Below the search bar are tabs for "Params", "Authorization", "Headers (7)", "Body", "Scripts", "Tests", and "Settings". The "Headers" tab is currently active, showing one header: "Authorization" with the value "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...". There are also "6 hidden" headers. To the right of the headers are "Cookies" and "Send" buttons. Below the headers table is a table with columns "Key", "Value", "Description", "Bulk Edit", and "Presets". The first row has a checked checkbox next to "Authorization". The second row is empty with "Key", "Value", and "Description" columns. At the bottom of the interface, there are tabs for "Body", "Cookies", "Headers (10)", and "Test Results". The "Test Results" tab is active, showing "Status: 200 OK Time: 588 ms Size: 1.31 KB". Below this are buttons for "Pretty", "Raw", "Preview", "Visualize", "JSON", and "Copy". The main content area displays the JSON response:

```

1 [
2   {
3     "timestamp": "00:04:20",
4     "log": "1687914.367127 [HW] I connect() to 192.168.1.2:4043 failed, errno(113)"
5   },
6   {
7     "timestamp": "00:04:21",
8     "log": "1687915.367814 [HW] I Trying to connect to 192.168.1.2:4043"
9   },
10  {
11    ...
12  }

```

Gambar 5.12 API Response Get Log Pod

4. User berhasil me-restart deployment Kubernetes (komponen 5G). Informasi terkait deployment didapatkan dari daftar informasi deployment pada API sebelumnya.

The screenshot shows the ORCA API interface. At the top, it says "HTTP ORCA / APIs / kube / Restart Single DU". Below that is a "POST" button and a URL field containing "{{base_url}}/api/v1/kube/restart_single_du/". To the right are "Save" and "Share" buttons. Underneath the URL, there are tabs for "Params", "Authorization", "Headers (8)", "Body", "Scripts", "Tests", and "Settings". The "Headers" tab is selected. It shows a table with two rows: one for "Authorization" with a checked checkbox and a value of "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...", and another for "Key" with a value of "Value". To the right of the table are "Description", "Bulk Edit", and "Presets" buttons. Below the headers, there are tabs for "Body", "Cookies", "Headers (10)", and "Test Results". The "Test Results" tab is selected, showing a status of "200 OK", time "666 ms", size "420 B", and a "Save as example" button. Below this is a JSON response pane with the following content:

```

1 {
2   "message": "Deployment restarted successfully.",
3   "details": "deployment.apps/oai-du-level1-user1 restarted\n"
4 }

```

Gambar 5.13 API Response Restart Deployment

5. User berhasil melakukan perintah ping dan melihat hasilnya. Perintah ini dijalankan menggunakan sebuah *script* pada Django untuk melakukan tes websocket.

The screenshot shows a terminal window titled "test_shell.py M". It contains the following Python code:

```

shell > 🐍 test_shell.py
1 import asyncio
2 import websockets
3 import json
4
5 async def test_websocket():
6     uri = "ws://10.30.1.221:8002/ws/shell/"
7     async with websockets.connect(uri) as websocket:
8         # Send initial data to start the command
9         message = json.dumps({
10             'pod_name': 'oai-nr-ue-level1-user1-59f86bc6db-gx5fx', # Replace with your actual pod name
11             'namespace': 'user1'. # Replace with your actual namespace
12         })
13         await websocket.send(message)
14
15     print(await websocket.recv())
16
17     # Read responses from the server
18     while True:
19         message = await websocket.recv()
20         print(message)
21
22     # Close the connection
23     await websocket.close()
24
25 if __name__ == "__main__":
26     test_websocket()

```

Below the code, the terminal shows the command being run and its output:

```

Received: {'command_output': 'PING google.com (172.253.118.101) 56(84) bytes of data.\n'}
Received: {'command_output': '64 bytes from sl-in-f101.1e100.net (172.253.118.101): icmp_seq=1 ttl=105 time=42.1 ms\n'}
Received: {'command_output': '64 bytes from sl-in-f101.1e100.net (172.253.118.101): icmp_seq=2 ttl=105 time=42.4 ms\n'}
Received: {'command_output': '64 bytes from sl-in-f101.1e100.net (172.253.118.101): icmp_seq=3 ttl=105 time=48.1 ms\n'}
Received: {'command_output': '64 bytes from sl-in-f101.1e100.net (172.253.118.101): icmp_seq=4 ttl=105 time=45.5 ms\n'}
Received: {'command_output': '\n'}
Received: {'command_output': '--- google.com ping statistics ---\n'}
Received: {'command_output': '4 packets transmitted, 4 received, 0% packet loss, time 3004ms\n'}
Received: {'command_output': 'rtt min/avg/max/mdev = 42.085/44.492/48.054/2.442 ms\n'}
^Z
[3]+ Stopped python test_shell.py

```

Gambar 5.14 Proses Testing API Websocket User PING

6. User berhasil melakukan perintah cURL dan melihat hasilnya. Perintah ini dijalankan menggunakan sebuah *script* pada Django untuk melakukan tes websocket.

```

shell > 🐍 test_shell.py M ×
shell > 🐍 test_shell.py
5   async def test_websocket():
7     async with websockets.connect(uri) as websocket:
9       message = json.dumps({
10         'pod_name': 'oai-nr-ue-level1-user1-59f86bc6db-gx5fx', # Replace with your actual pod name
11         'namespace': 'user1', # Replace with your actual namespace
12         'command': 'curl google.com' # User-defined command
13     })
14     await websocket.send(message)
15
16     # Listen for messages from the server
17     while True:
18
19
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS EXPOSED PORTS
(venv_orca_backend_ws) bagus@C5g-backend:~/ORCA/orca_backend_ws/shell$ python test_shell.py
Received: {'command_output': '<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">\n<TITLE>301 Moved</TITLE></HEAD><BODY>\n'}
Received: {'command_output': '<H1>301 Moved</H1>\n'}
Received: {'command_output': 'The document has moved\n'}
Received: {'command_output': '<A href="http://www.google.com/">here</A>.\r\n'}
Received: {'command_output': '</BODY></HTML>\r\n'}
^Z
[4]+  Stopped                  python test_shell.py
(venv_orca_backend_ws) bagus@C5g-backend:~/ORCA/orca_backend_ws/shell$ 

```

Gambar 5.15 Proses Testing API Websocket User CURL

7. User berhasil mendapatkan informasi Key Performance Indicator (KPI) dari komponen UE. Perintah ini dijalankan menggunakan sebuah *script* pada Django untuk melakukan tes websocket.

```

shell > 🐍 test_monitoring.py M ×
monitoring > 🐍 test_monitoring.py
4
5   async def test_websocket():
6     uri = "ws://10.30.1.221:8002/ws/monitoring/"
7     async with websockets.connect(uri) as websocket:
8       # Send initial data to start the monitoring command
9       message = json.dumps({
10         'pod_name': 'oai-nr-ue-level1-user1-59f86bc6db-gx5fx', # Replace with your actual pod name
11         'namespace': 'user1', # Replace with your actual namespace
12     })
13     await websocket.send(message)
14
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS EXPOSED PORTS
(venv_orca_backend_ws) bagus@C5g-backend:~/ORCA/orca_backend_ws/monitoring$ python test_monitoring.py
Received: {'monitoring_output': 'L1 TX processing: 283.365 us; 132087; 1496.548 us;\n'}
Received: {'monitoring_output': 'UL SCH encoding: 49.998 us; 2254; 313.902 us;\n'}
Received: {'monitoring_output': 'L1 RX processing: 370.994 us; 349849; 3960.349 us;\n'}
Received: {'monitoring_output': 'UL Indication: 102.741 us; 132087; 523.846 us;\n'}
Received: {'monitoring_output': 'PDSCH receiver: 42.069 us; 2242; 284.014 us;\n'}
Received: {'monitoring_output': 'PDSCH decoding: 490.710 us; 2242; 3116.306 us;\n'}
Received: {'monitoring_output': '>- Deinterleave: 5.040 us; 2242; 55.273 us;\n'}
Received: {'monitoring_output': '>- Rate Unmatch: 10.253 us; 2242; 217.152 us;\n'}
Received: {'monitoring_output': '>- LDPC Decode: 304.858 us; 2242; 1308.112 us;\n'}
Received: {'monitoring_output': 'PDSCH unscrambling: 3.831 us; 2242; 232.200 us;\n'}
Received: {'monitoring_output': 'PDCCH handling: 87.788 us; 350946; 921.633 us;\n'}
Received: {'monitoring_output': '\x00'}{'monitoring_output': 'L1 TX processing: 283.514 us; 132891; 1496.548 us;\n'}
Received: {'monitoring_output': 'UL SCH encoding: 50.007 us; 2268; 313.902 us;\n'}
Received: {'monitoring_output': 'L1 RX processing: 371.368 us; 351980; 6182.339 us;\n'}
Received: {'monitoring_output': 'UL Indication: 102.786 us; 132891; 523.846 us;\n'}
Received: {'monitoring_output': 'PDSCH receiver: 42.104 us; 2255; 284.014 us;\n'}
Received: {'monitoring_output': 'PDSCH decoding: 490.916 us; 2255; 3116.306 us;\n'}
Received: {'monitoring_output': '>- Deinterleave: 5.040 us; 2255; 55.273 us;\n'}
Received: {'monitoring_output': '>- Rate Unmatch: 10.252 us; 2255; 217.152 us;\n'}

```

Gambar 5.16 Proses Testing API Websocket User KPI

8. User berhasil mendapatkan informasi terkait protokol SCTP pada komponen 5G (CU dan DU).

Gambar 5.17 Proses Testing API Websocket SCTP

9. User berhasil mendapatkan informasi log dari komponen AMF dan UPF. Informasi ini digunakan untuk memantau aktivitas dari komponen Core.

Gambar 5.18 Response API AMF Log

The screenshot shows the ORCA API client interface. At the top, it says "ORCA / APIs / kube / Get UPF Log". Below that is a search bar with "GET" selected and the URL "{{base_url}}/api/v1/kube/get_upf_logs/". There are tabs for "Params", "Authorization", "Headers (7)", "Body", "Scripts", "Tests", and "Settings". The "Headers (7)" tab is active. A table below shows the headers: "Authorization" with value "Bearer eyJhbGciOiJIUzI1NlslnR5cCl6lkpXVCJ...", and an empty "Key" row. On the right, there are "Cookies" and "Send" buttons.

Below the header table, the status bar shows "Status: 200 OK Time: 604 ms Size: 1.35 KB" and "Save as example". The "Body" tab is selected, showing a JSON response with two log entries:

```

1 [
2   {
3     "timestamp": "13:40:53",
4     "log": "[2024-06-04 06:40:53.994] [upf_n4] [info] handle_receive(16 bytes)"
5   },
6   {
7     "timestamp": "13:40:53",
8     "log": "[2024-06-04 06:40:53.994] [upf_n4] [info] Received SX HEARTBEAT REQUEST"
9   },
10 ]

```

Gambar 5.19 Response API UPF Log

10. User Berhasil Mendapatkan Informasi Status Deployment AMF dan UPF. Informasi ini digunakan untuk mengindikasikan tersedia atau tidaknya komponen Core.

The screenshot shows the ORCA API client interface. At the top, it says "ORCA / APIs / kube / Get AMF Deployment". Below that is a search bar with "GET" selected and the URL "{{base_url}}/api/v1/kube/get_amf_deployments/". There are tabs for "Params", "Authorization", "Headers (7)", "Body", "Scripts", "Tests", and "Settings". The "Headers (7)" tab is active. A table below shows the headers: "Authorization" with value "Bearer eyJhbGciOiJIUzI1NlslnR5cCl6lkpXVCJ...", and an empty "Key" row. On the right, there are "Cookies" and "Send" buttons.

Below the header table, the status bar shows "Status: 200 OK Time: 543 ms Size: 346 B" and "Save as example". The "Body" tab is selected, showing a JSON response with one deployment entry:

```

1 [
2   {
3     "name": "oai-amf",
4     "replicas": 1
5   }
6 ]

```

Gambar 5.20 Response API AMF Deployment

The screenshot shows the ORCA API client interface. At the top, it says "ORCA / APIs / kube / Get UPF Deployment". Below that is a search bar with "GET" selected and the URL "{{base_url}}/api/v1/kube/get_upf_deployments/". There are tabs for "Params", "Authorization", "Headers (7)", "Body", "Scripts", "Tests", and "Settings". The "Headers" tab is active, showing a table with one row for "Authorization" and another for "Key". The "Body" tab is also active, showing the JSON response:

```

1 [ 
2   {
3     "name": "oai-upf",
4     "replicas": 1
5   }
6 ]

```

At the bottom, it says "Status: 200 OK Time: 368 ms Size: 346 B".

Gambar 5.21 Response API UPF Deployment

5.2.1.4 Helm Chart APIs

API yang berkaitan dengan Helm Chart memiliki peran utama untuk instalasi seluruh komponen 5G yang diperlukan oleh *user* ke dalam kluster Kubernetes berdasarkan *namespace* per masing-masing *user*. Selain daripada itu, fungsi API ini juga berkaitan dengan manajemen *values* yang terdapat pada setiap komponen 5G yang berhasil terinstal. Dan fungsi terakhirnya adalah termasuk untuk manajemen siklus hidup dari setiap komponen. Seperti halnya dengan Kubernetes APIs sebelumnya, API ini juga memiliki berbagai pola dan fungsi yang mirip. Berikut adalah daftar API terkait dengan Helm Chart.

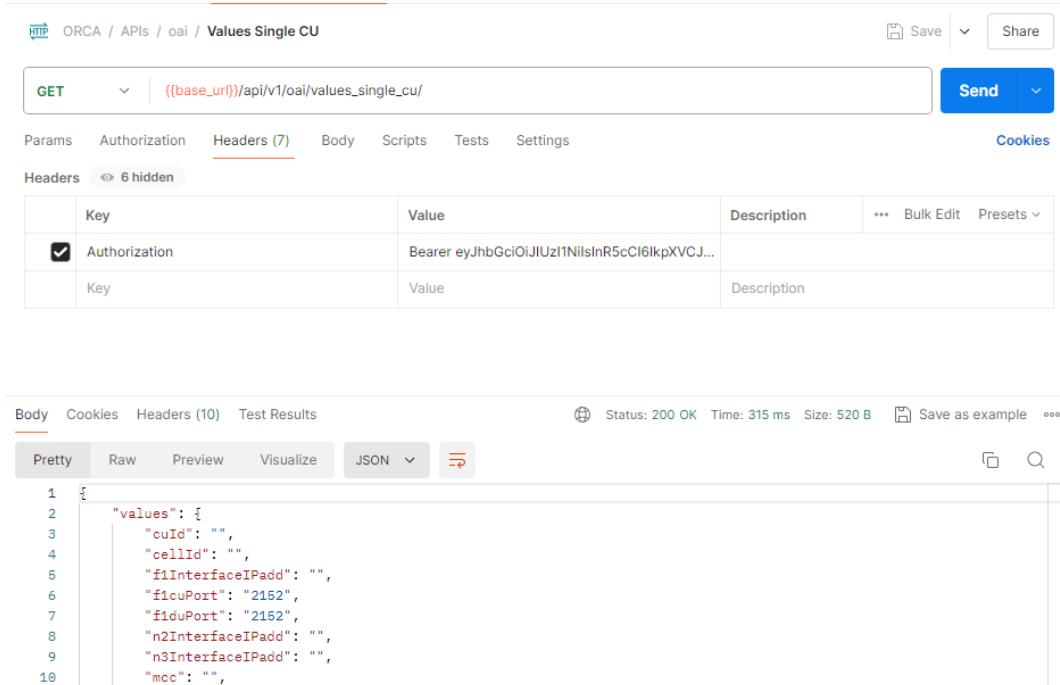
Tabel 5.6 List Helm Chart API

<i>Category</i>	<i>Method</i>	<i>API Endpoint</i>
<i>Get Configuration Values</i>	<i>GET</i>	<i>{{base_url}}/api/v1/oai/values_{{component_name}}/</i>
<i>Config Component Values</i>	<i>POST</i>	<i>{{base_url}}/api/v1/oai/config_{{component_name}}/</i>
<i>Start Component</i>	<i>POST</i>	<i>{{base_url}}/api/v1/oai/start_{{component_name}}/</i>
<i>Stop Component</i>	<i>POST</i>	<i>{{base_url}}/api/v1/oai/stop_{{component_name}}/</i>

API yang berkaitan dengan Helm Chart memiliki peran utama untuk instalasi seluruh komponen 5G yang diperlukan oleh *user* ke dalam kluster Kubernetes berdasarkan *namespace* per masing-masing *user*.

Selain daripada itu, fungsi API ini juga berkaitan dengan manajemen *values* yang terdapat pada setiap komponen 5G yang berhasil terinstal. Dan fungsi terakhirnya adalah termasuk untuk manajemen siklus hidup dari setiap komponen. Seperti halnya dengan Kubernetes APIs sebelumnya, API ini juga memiliki berbagai pola dan fungsi yang mirip. Berikut adalah daftar API terkait dengan Helm Chart.

1. User berhasil mendapatkan informasi nilai konfigurasi pada setiap komponen 5G. Informasi ini digunakan untuk memvalidasi benar atau tidaknya hasil konfigurasi yang dilakukan oleh user.

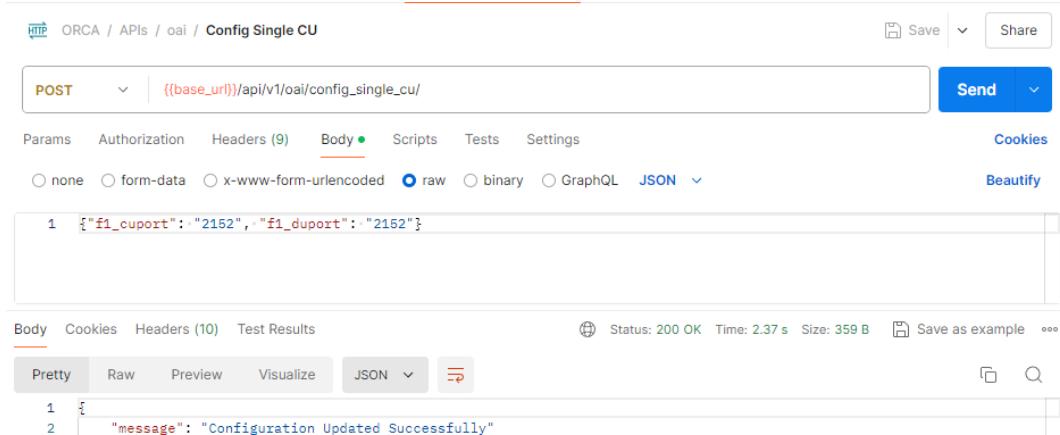


```

1 {
2   "values": [
3     "cuId": "",
4     "cellId": "",
5     "f1InterfaceIPaddr": "",
6     "f1cuPort": "2152",
7     "f1duPort": "2152",
8     "n2InterfaceIPaddr": "",
9     "n3InterfaceIPaddr": "",
10    "mcc": ""
11  ]
12}
  
```

Gambar 5.22 Response API Get Configuration Value

2. User berhasil mengubah nilai konfigurasi pada setiap komponen 5G. Nilai ini akan berpengaruh terhadap keberhasilan konfigurasi setiap komponen.



```

1 {"f1_cuport": "2152", "f1_duport": "2152"}
  
```

```

1 {
2   "message": "Configuration Updated Successfully"
2
  
```

Gambar 5.23 Response API Post Configuration Value

3. User berhasil menjalankan setiap komponen 5G (Deployment Kubernetes). Ini berkaitan dengan manajemen siklus hidup dari komponen 5G.

The screenshot shows the ORCA API interface with the URL `ORCA / APIs / oai / Start Single CU`. A POST request is being made to `((base_url))/api/v1/oai/start_single_cu/`. The Headers tab is selected, showing an Authorization header with a value of `Bearer eyJhbGciOiJIUzI1NilsInR5cCl6IkpXVC...`. The Body tab shows a JSON response with the message `"message": "CU successfully started"`.

Gambar 5.24 Response API Start Component

4. User berhasil menghentikan setiap komponen 5G (Deployment Kubernetes). Ini berkaitan dengan manajemen siklus hidup dari komponen 5G.

The screenshot shows the ORCA API interface with the URL `ORCA / APIs / oai / Stop Single CU`. A POST request is being made to `((base_url))/api/v1/oai/stop_single_cu/`. The Headers tab is selected, showing an Authorization header with a value of `Bearer eyJhbGciOiJIUzI1NilsInR5cCl6IkpXVC...`. The Body tab shows a JSON response with the message `"message": "CU successfully stopped"`.

Gambar 5.25 Response API Stop Component

5.2.1.5 Wireshark APIs

Terakhir ini merupakan REST API yang berkaitan dengan fungsi *sniffing* layaknya aplikasi Wireshark. API ini akan membantu *user* dalam *sniffing* paket data pada setiap komponen 5G dan menyimpan riwayat *sniffing* ke dalam sebuah *file pcap* yang disimpan di dalam *database*. Pada API ini menggunakan tipe *WebSocket* sebagai API utama yang digunakan untuk mengintegrasikan data-data penting yang bersifat *real-time* yang dalam kasus ini adalah data hasil proses *sniffing* paket data. Berikut adalah daftar API terkait fungsi *sniffing*.

Tabel 5.7 List Wireshark API

<i>Category</i>	<i>Method</i>	<i>API Endpoint</i>
<i>List PCAP Files</i>	<i>GET</i>	<code>{base_url}/api/v1/shark/pcap_files/</code>
<i>Download PCAP Files</i>	<i>GET</i>	<code>{base_url}/api/v1/shark/pcap_files/{files_id}/download/</code>
<i>Delete PCAP Files</i>	<i>DELETE</i>	<code>{base_url}/api/v1/shark/pcap_files/{files_id}/remove/</code>

Tabel 5.8 List Wireshark Websocket API

<i>Category</i>	<i>Endpoint</i>
<i>Sniffing Process and Save PCAP Files into Database</i>	<code>ws://{{base_url}}/ws/sniff/</code>

Kedua tabel di atas berisi daftar REST API dan Websocket API *endpoint* yang tersedia terkait dengan fungsionalitas *sniffing*. Setiap API di atas memiliki fungsionalitasnya masing-masing, berikut akan disajikan proses testing seluruh fungsionalitas Wireshark APIs yang dilakukan menggunakan aplikasi Postman untuk tipe REST API dan juga *code editor* dengan Django *libraries* untuk tipe Websocket API.

1. User berhasil mendapatkan informasi paket data dari hasil *sniffing* pada pod (komponen 5G) dan berhasil menyimpan *file PCAP* pada *database*.

Gambar 5.26 Proses Testing API Websocket Sniffing Pod

2. User berhasil mendapatkan informasi daftar *file* PCAP yang telah tersedia pada *database*.

```

1 [
2   {
3     "id": 85,
4     "user": "user1",
5     "filename": "user1_oai-du-level1-user1-764665b6c9-p5nrg_20240623_172812.pcap",
6     "file_size": 128,
7     "created_at": "2024-06-24T00:28:12.744195+07:00"
8   },
9   {
10    "id": 86,
11    "user": "user1",
12    "filename": "user1_oai-nr-ue-level1-user1-ccd7864b7-gd2d2_20240623_172845.pcap",
13    "file_size": 13628,
14    "created_at": "2024-06-24T00:29:03.165935+07:00"
}
  
```

Gambar 5.27 Response API List PCAP File

3. User berhasil men-*download* *file* PCAP yang diinginkan dari *database*

Gambar 5.28 Response API Dowload PCAP File

4. User berhasil menghapus file PCAP yang diinginkan dari database

The screenshot shows the ORCA API testing interface. A DELETE request is made to `[[base_url]]/api/v1/shark/pcap_files/85/remove/`. The Headers tab is selected, showing an Authorization header with a value of `Bearer eyJhbGciOiJIUzI1NilsInR5cCl6IkpXVC...`. The Body tab shows a JSON response with the message `"message": "File deleted successfully"`.

Gambar 5.29 Response API Delete PCAP File

Seluruh APIs telah melalui tahap testing dan mendapatkan hasil sukses. Hal ini menandakan jika seluruh APIs tersebut berfungsi dengan baik dan siap untuk diimplementasi pada sisi frontend. HTTP status code yang muncul apabila respons yang dikeluarkan dari hasil testing mengindikasikan berhasil adalah kode 200 yang berarti OK dan 201 yang berarti Created.

5.2.2 End-to-End (E2E) Testing

End-to-end (E2E) testing bertujuan untuk memastikan bahwa seluruh alur aplikasi, mulai dari *frontend* hingga *backend*, bekerja secara harmonis dan sesuai dengan kebutuhan pengguna akhir. Dengan menguji keterhubungan komponen, kinerja fungsional, serta keamanan data, E2E *testing* dapat mendeteksi masalah yang tidak terdeteksi dalam pengujian unit atau integrasi untuk dapat diperbaiki. Hal ini membantu memastikan aplikasi memenuhi semua persyaratan bisnis dan teknis, serta mengurangi risiko terkait rilis produk ke pengguna akhir. Berikut adalah tabel yang mendeskripsikan alur dari *E2E testing* yang akan dilakukan.

Tabel 5.9 Langkah E2E Testing

Langkah	Deskripsi
Setup Proyek	Siapkan lingkungan pengujian yang mencakup URL akses aplikasi, kredensial login, dan data pengujian yang diperlukan.
Penyusunan Rencana Pengujian	Buat rencana pengujian yang mencakup skenario E2E, seperti membuka halaman, mengisi formulir, dan mengklik tombol, serta hasil yang diharapkan untuk setiap langkah.

Menjalankan Tes Manual	Ikuti rencana pengujian secara manual dengan menjalankan langkah-langkah yang telah ditentukan pada UI aplikasi di browser, mencatat setiap tindakan dan hasil yang diamati.
Pengamatan Hasil	Catat hasil pengujian untuk melihat apakah ada kegagalan atau kesalahan yang terjadi selama pengujian, termasuk tangkapan layar jika diperlukan.
Debugging	Jika ditemukan masalah, analisis dan perbaiki masalah tersebut, kemudian ulangi pengujian untuk memastikan perbaikan.
Dokumentasi Tes	Dokumentasikan hasil pengujian secara rinci, termasuk langkah-langkah yang dilakukan, hasil yang diamati, dan masalah yang ditemukan serta perbaikannya.

Pada tahap ini, proses E2E *testing* yang dilakukan akan memiliki pola berulang sama dengan sebelumnya pada *functional testing*. Hanya saja, kali ini proses *testing* akan dilakukan melalui browser Google Chrome dengan menggunakan tampilan UI yang telah dikembangkan sebagai sisi *frontend*. Berikut adalah poin-poin yang akan dijabarkan dalam bentuk fungsionalitas mewakili setiap APIs yang telah dijelaskan pada *functional testing*.

5.2.2.1 User Management Functions

1. Admin berhasil membuat akun user baru

Username	Completion (%)	Actions
user1	100%	EDIT REMOVE
user2	0%	EDIT REMOVE

Gambar 5.30 Tampilan Admin Membuat User

2. Admin berhasil melihat daftar akun user

User Management		
Create User		
Username	Completion (%)	Actions
user1	100%	EDIT REMOVE
user2	0%	EDIT REMOVE

Gambar 5.31 Tampilan Admin Melihat User

3. Admin berhasil mengubah password akun user

The screenshot shows the User Management page with two users listed: 'user1' (100% completion) and 'user2' (0% completion). A modal dialog is open over the table, titled 'Change your password'. It contains a warning message: 'The password you just used was found in a data breach. Google Password Manager recommends changing your password now.' There are 'OK' and 'Cancel' buttons at the bottom of the modal. At the bottom left of the main page area, there is a green success message: 'User updated successfully!' with a close button.

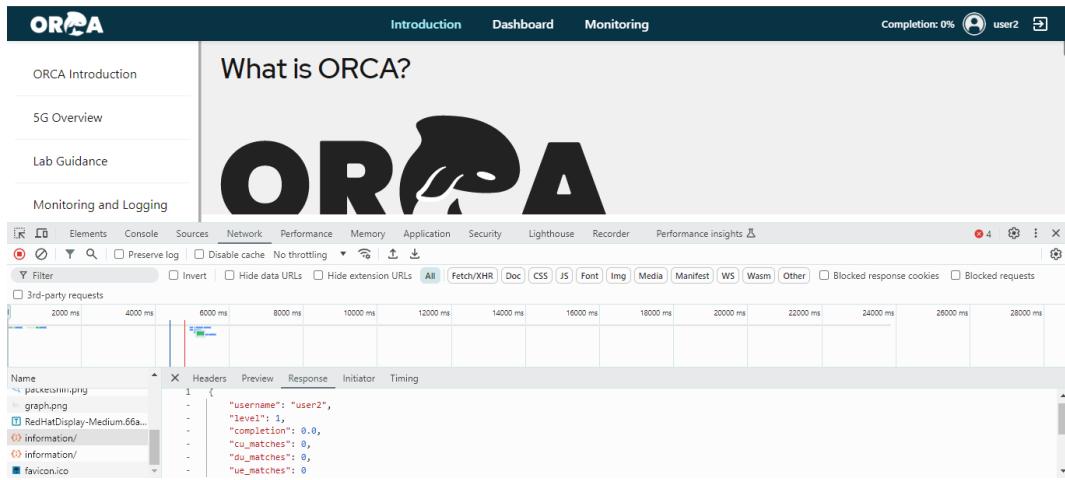
Gambar 5.32 Tampilan Admin Mengubah Password User

4. Admin berhasil menghapus akun user

The screenshot shows the User Management page with one user listed: 'user1' (100% completion). A modal dialog is open over the table, containing a confirmation message: 'Are you sure you want to remove this user?'. There are 'OK' and 'Cancel' buttons at the bottom of the modal. At the bottom left of the main page area, there is a green success message: 'User removed successfully!' with a close button.

Gambar 5.33 Tampilan Admin Menghapus User

5. User berhasil melihat informasi akunnya



Gambar 5.34 Verifikasi Informasi User dari Browser

5.2.2.2 Token Functions

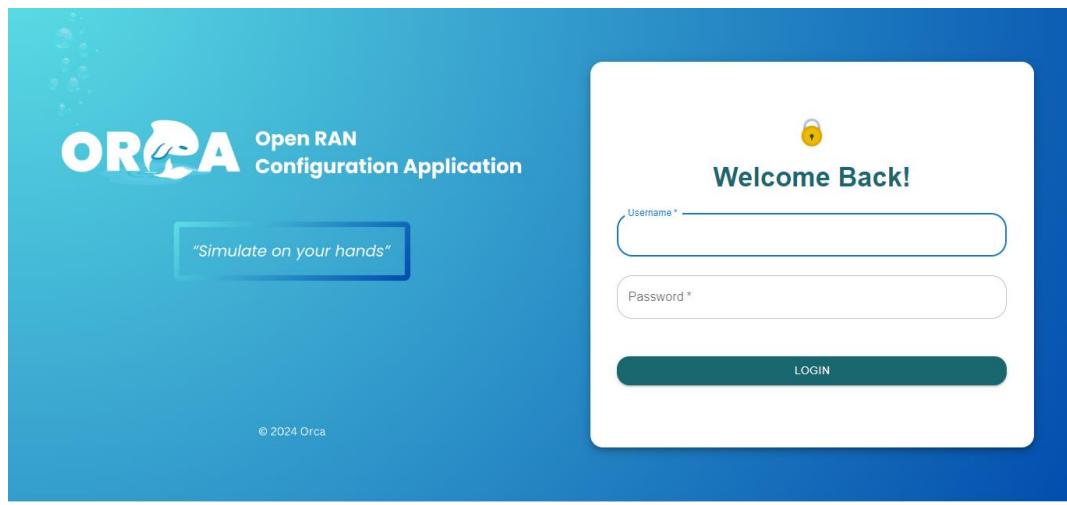
1. User berhasil masuk ke dalam halaman introduction website (*login*)

Gambar 5.35 Verifikasi Informasi User Berhasil Login

2. Admin berhasil masuk ke dalam halaman user management (*login*)

Gambar 5.36 Verifikasi Informasi Admin Berhasil Login

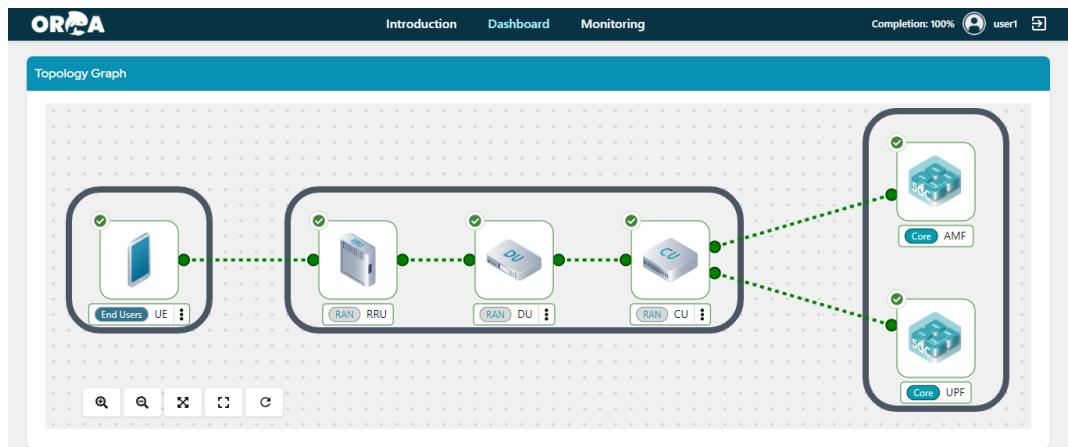
3. Admin/User berhasil keluar dari website dan kembali ke halaman *login* (*logout*)



Gambar 5.37 Verifikasi User dan Admin Berhasil Logout

5.2.2.3 Kubernetes Functions

1. User berhasil mendapatkan daftar informasi pod Kubernetes miliknya ditandai dengan munculnya gambar topologi pada laman dashboard dengan status komponen stopped/running (frame berwarna kuning/hijau).



Gambar 5.38 Koneksi Topologi Terhubung

```
X Headers Preview Response Initiator Timing
1 {
  "pods": [
    {
      "name": "oai-cu-level1-user1-cb4b4dcbb6-kdwlij",
      "ip": "10.16.48.217",
      "network_status": [
        {
          "name": "kube-ovn",
          "interface": "eth0",
          "ips": [
            "10.16.48.217"
          ],
          "mac": "00:00:00:D0:10:57",
          "default": true,
        }
      ]
    }
  ]
}
```

Gambar 5.39 Response API Pod Saat Topologi Terhubung

2. User berhasil mendapatkan daftar informasi deployment Kubernetes miliknya ditandai dengan munculnya gambar topologi pada laman dashboard dengan status komponen stopped/running (frame berwarna kuning/hijau).

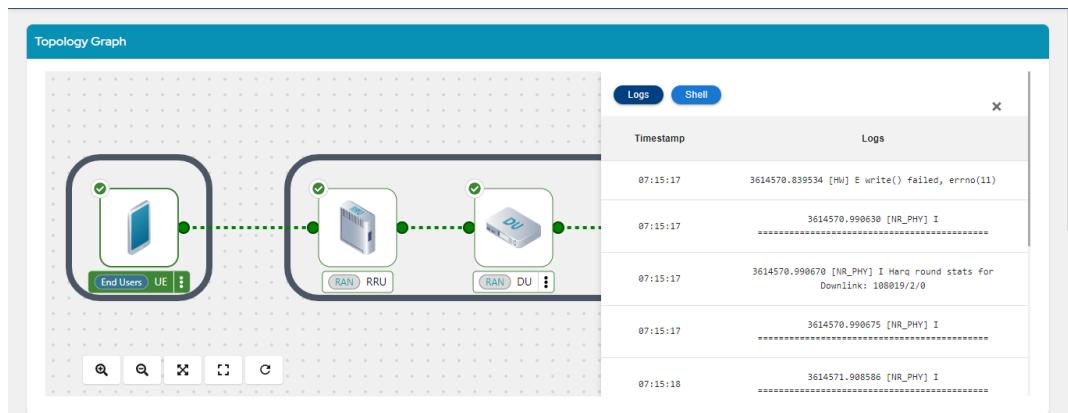
```

1 [
2   {
3     "deployment_name": "oai-cu-level1-user1",
4     "namespace": "user1",
5     "replicas": 1,
6     "available_replicas": 1,
7     "ready_replicas": 1,
8     "updated_replicas": 1,
9     "strategy": "Recreate",
10    "conditions": [
11      {"Progressing", "Available"}
12    ]
13  },
14]

```

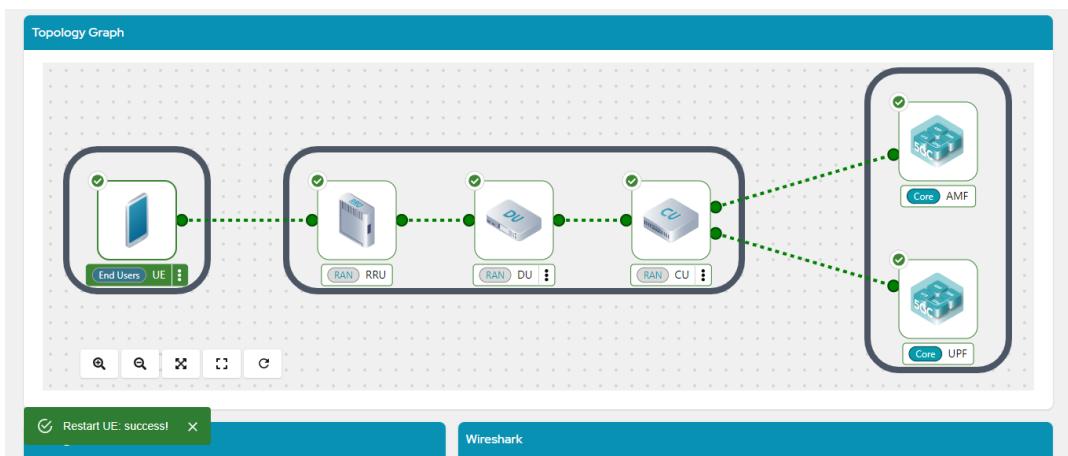
Gambar 5.40 Response API Deployment Saat Topologi Terhubung

3. User berhasil mendapatkan informasi log dari pod Kubernetes miliknya ditandai dengan munculnya informasi log pada setiap komponen ketika menekan ikon komponen dan mengarahkan pada tab Logs yang terdapat pada sidebar yang muncul ketika ikon komponen ditekan.



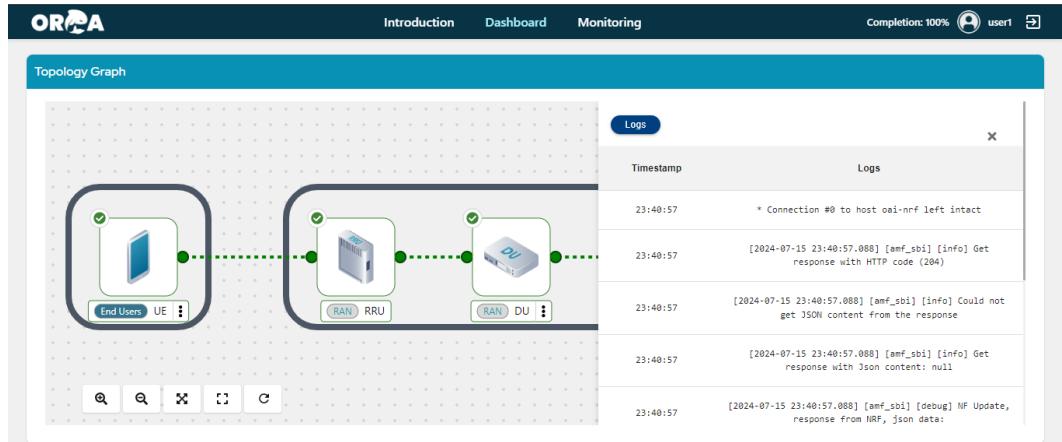
Gambar 5.41 UE Berhasil Menampilkan Log

4. User berhasil me-restart deployment Kubernetes (komponen 5G) sebagai manajemen siklus hidup komponen, ditandai dengan notifikasi sukses proses restart yang dilakukan pada setiap komponen. Proses restart dapat dilakukan dengan menekan kebab button pada setiap komponen ataupun dengan klik kanan pada mouse ataupun touchpad untuk memunculkan pop up pilihan tombol restart.



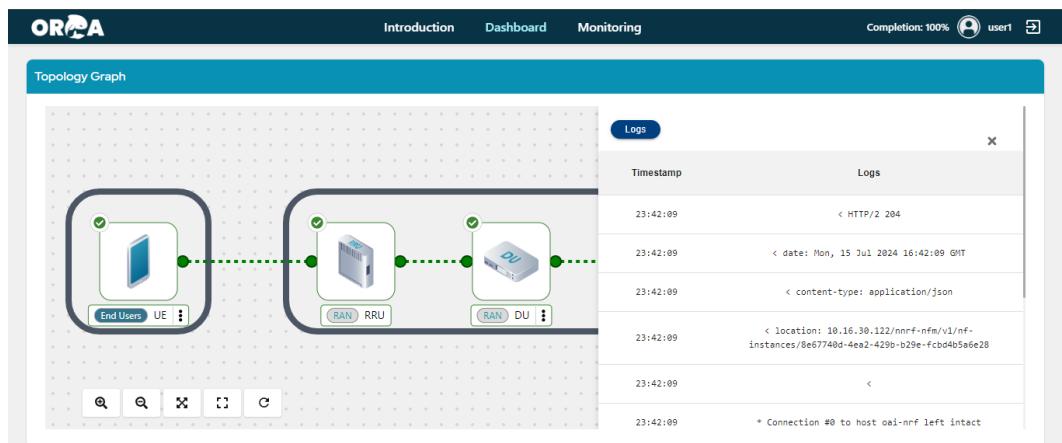
Gambar 5.42 UE Berhasil Di Restart

5. User berhasil mendapatkan informasi log dari komponen AMF ditandai dengan munculnya informasi log pada komponen AMF ketika menekan ikon komponen dan mengarahkan pada tab Logs yang terdapat pada sidebar yang muncul ketika ikon komponen ditekan.



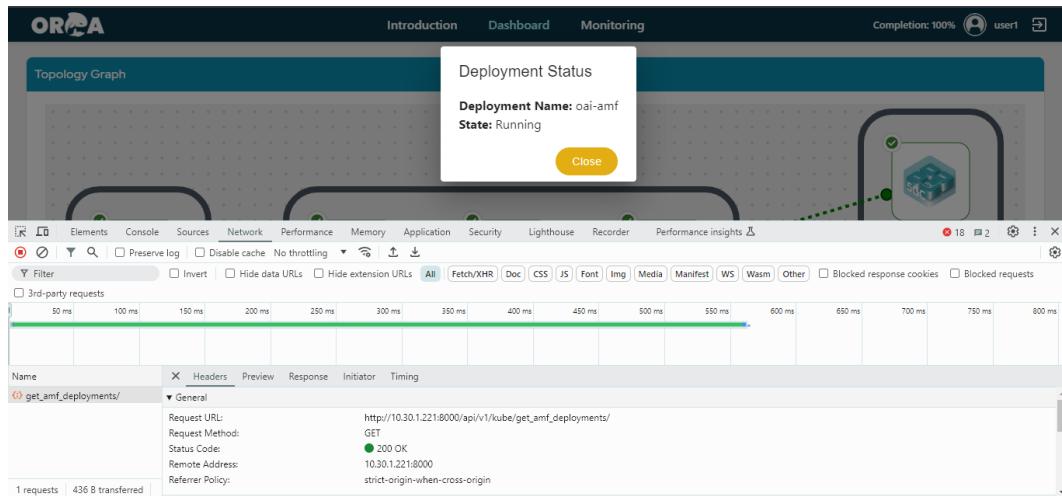
Gambar 5.43 AMF Berhasil Menampilkan Log

6. User berhasil mendapatkan informasi log dari komponen UPF ditandai dengan munculnya informasi log pada komponen UPF ketika menekan ikon komponen dan mengarahkan pada tab Logs yang terdapat pada sidebar yang muncul ketika ikon komponen ditekan.



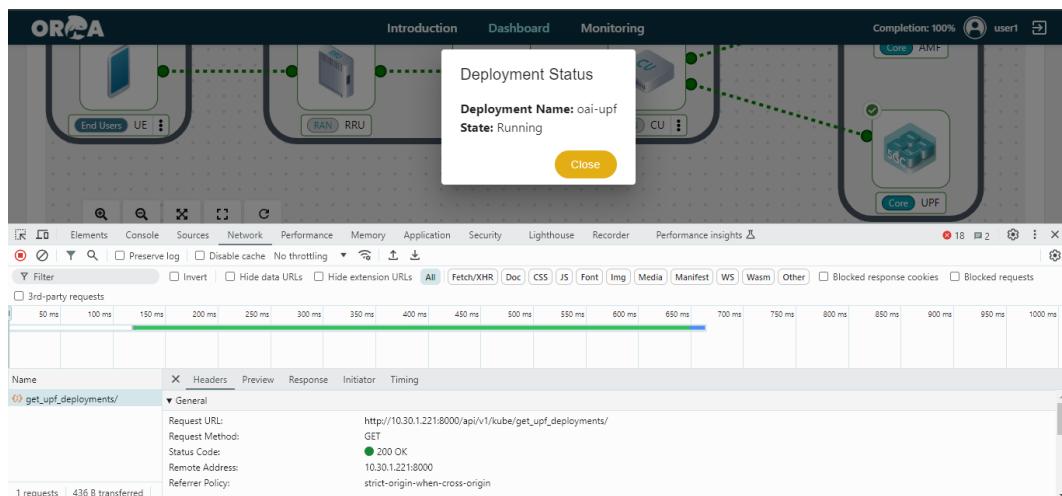
Gambar 5.44 UPF Berhasil Menampilkan Log

7. User berhasil mendapatkan informasi data deployment komponen AMF ditandai dengan munculnya informasi deployment status pada komponen AMF ketika menekan ikon checklist atau silang yang terdapat pada frame komponen AMF.



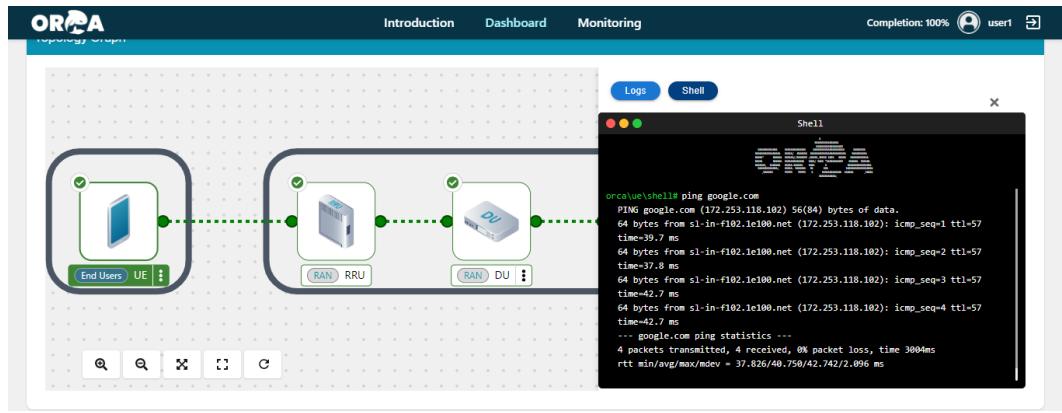
Gambar 5.45 AMF Berhasil Menampilkan State Pod

8. User berhasil mendapatkan informasi data deployment komponen UPF ditandai dengan munculnya informasi deployment status pada komponen UPF ketika menekan ikon checklist atau silang yang terdapat pada frame komponen UPF.



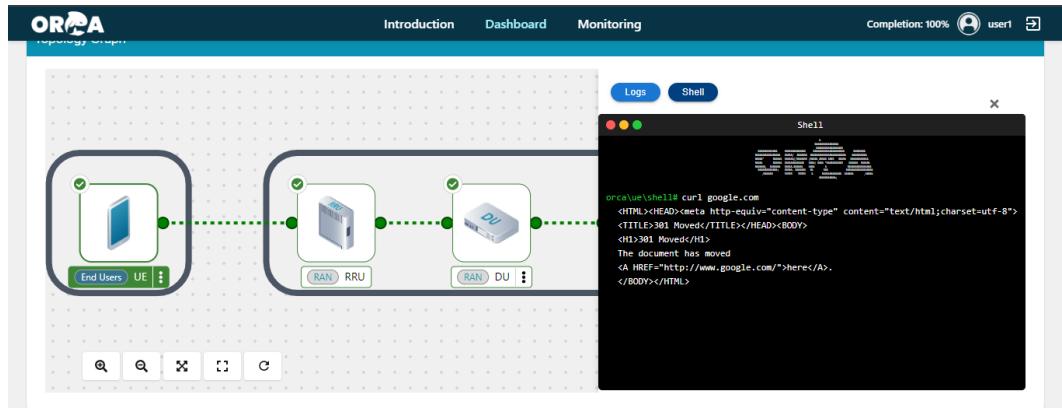
Gambar 5.46 UPF Berhasil Menampilkan State Pod

9. User berhasil melakukan perintah ping dan melihat hasilnya pada komponen UE. Proses ping dapat dilakukan dengan menekan ikon komponen UE, kemudian mengarahkan ke tab Shell pada sidebar yang muncul setelah ikon komponen ditekan. Seperti contoh di bawah, perintah ping google.com menghasilkan output yang menandakan proses ping berjalan dengan baik dan benar. Proses ping ini bertujuan untuk mengecek status koneksi *end-to-end* (E2E) dari komponen UE hingga ke komponen Core dan ke internet berjalan dengan baik atau tidak.



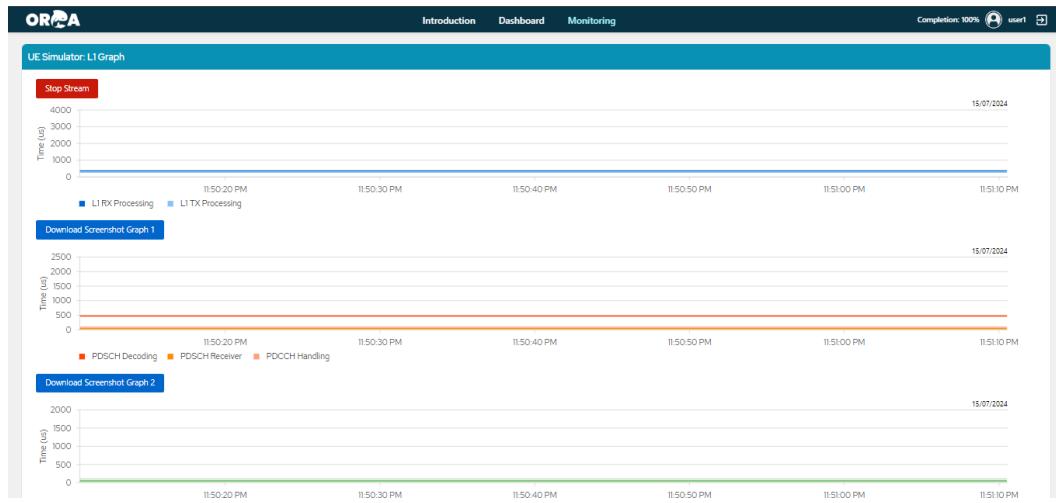
Gambar 5.47 UE Berhasil Menampilkan Hasil Ping

- User berhasil melakukan perintah cURL dan melihat hasilnya pada komponen UE. Proses cURL dapat dilakukan dengan menekan ikon komponen UE, kemudian mengarahkan ke tab Shell pada sidebar yang muncul setelah ikon komponen ditekan. Seperti contoh di bawah, perintah curl google.com menghasilkan output yang menandakan proses cURL berjalan dengan baik dan benar. Proses cURL ini bertujuan untuk mensimulasikan smartphone (UE) dapat mengakses laman website pada internet dengan cara yang simple melalui command-line interface (CLI).



Gambar 5.48 UE Berhasil Menampilkan Hasil CURL

- User berhasil mendapatkan informasi *Key Performance Indicator* (KPI) dari komponen UE, ditandai dengan munculnya nilai-nilai KPI pada table value beserta dengan munculnya grafik monitoring yang bekerja secara real-time untuk memvisualisasikan nilai-nilai KPI tersebut. Informasi ini bisa didapatkan di laman monitoring pada website dashboard.

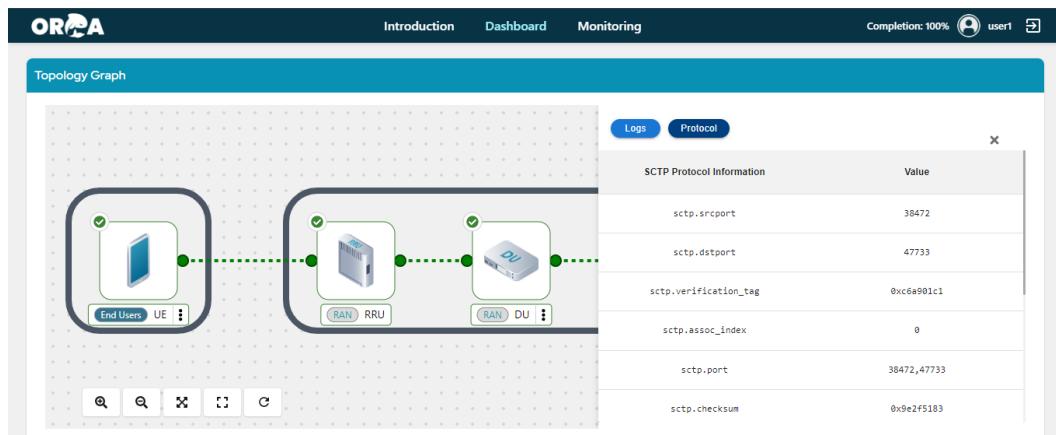


Gambar 5.49 User Berhasil Mendapatkan Hasil Monitoring

Table Value			
Key Performance Indicator	Value (us)	Count	Total Time (us)
L1 TX processing	272.929	3287403	1863.028
ULSCH encoding	48.802	54870	464.716
L1 RX processing	355.457	8706141	3509.706
UL Indication	98.995	3287403	802.546
PDSCH receiver	40.317	54851	833.121
PDSCH decoding	474.975	54850	2956.438
-> Deinterleave	4.837	54851	175.097
-> Rate Unmatch	9.718	54851	1156.099
-> LDPC Decode	294.934	54851	1964.035
PDSCH unscrambling	3.656	54851	75.794

Gambar 5.50 User Berhasil Mendapatkan Key Table Value

12. User berhasil mendapatkan informasi protokol SCTP dari komponen RAN 5G (CU dan DU), ditandai dengan munculnya informasi terkait protocol SCTP ketika menekan ikon komponen CU ataupun DU dan mengarahkan pada tab Protocol yang terdapat pada sidebar yang muncul ketika ikon komponen ditekan.



Gambar 5.51 User Berhasil Mendapatkan Hasil Protocol SCTP

5.2.2.4 Helm Chart Functions

1. User berhasil mendapatkan informasi value konfigurasi pada setiap komponen 5G (CU, DU, dan UE) yang terletak di menu Configuration Panel pada lama dashboard. Hal tersebut ditandai dengan munculnya value pada kolom Current Value setelah user melakukan konfigurasi pada kolom Set Value dan menekan tombol submit untuk mengkonfirmasi pengubahan value pada komponen yang dikonfigurasi.

The screenshot shows the ORCA dashboard with two main sections: Configuration Panel and Wireshark.

Configuration Panel:

- Three tabs at the top: UE, DU, and CU.
- A table with columns: Key, Set Value, and Current Value.
- Values for various configuration keys are listed, such as CU ID (0xe01), Cell ID (12345678L), F1 IP Address (192.168.1.1), etc.
- A "Submit" button at the bottom.

Wireshark:

- Two tabs: Sniff and Files.
- A "Start" button and a dropdown menu "Select a Component".
- A table with columns: No., Timestamp, IP src, IP dst, and Protocol Info.
- A "Filter" button at the bottom.

Gambar 5.52 User Berhasil Mendapatkan Hasil Value Config

2. User berhasil mengubah value konfigurasi pada setiap komponen 5G (CU, DU, dan UE) yang terletak di menu Configuration Panel pada lama dashboard. Hal tersebut ditandai dengan munculnya notifikasi status berhasil konfigurasi sesaat setelah user mengkonfirmasi pengubahan value pada komponen yang dikonfigurasi.

The screenshot shows the ORCA dashboard with the Configuration Panel section.

Configuration Panel:

- Three tabs at the top: UE, DU, and CU.
- A table with columns: Key, Set Value, and Current Value.
- Values for various configuration keys are listed, such as F1 CU Port (2152), F1 DU Port (2152), N2 IP Address (172.21.6.1), etc.
- A green notification bar at the bottom says "Configuration updated successfully!".
- A "Submit" button at the bottom.

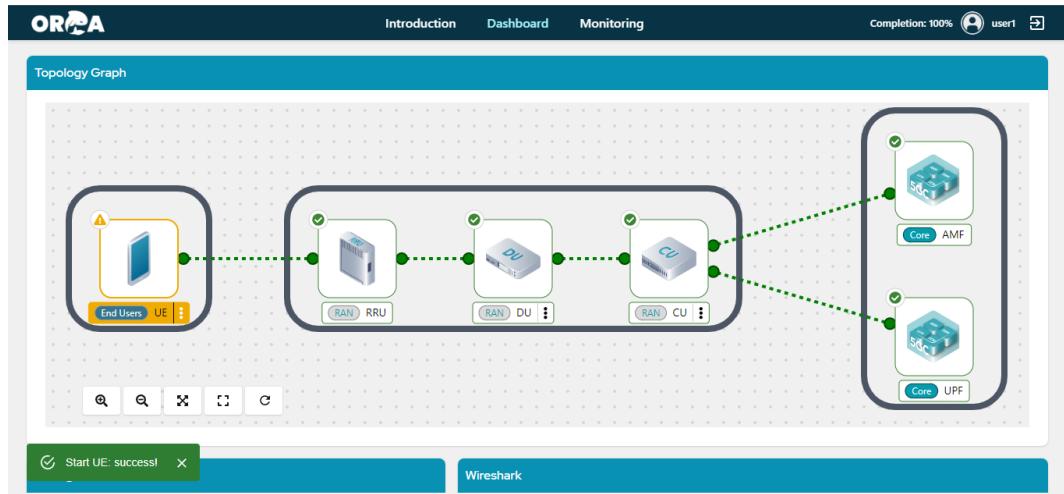
Wireshark:

- Two tabs: Sniff and Files.
- A "Start" button and a dropdown menu "Select a Component".
- A table with columns: No., Timestamp, IP src, IP dst, and Protocol Info.
- A "Filter" button at the bottom.

Gambar 5.53 User Berhasil Mengubah Hasil Value Config

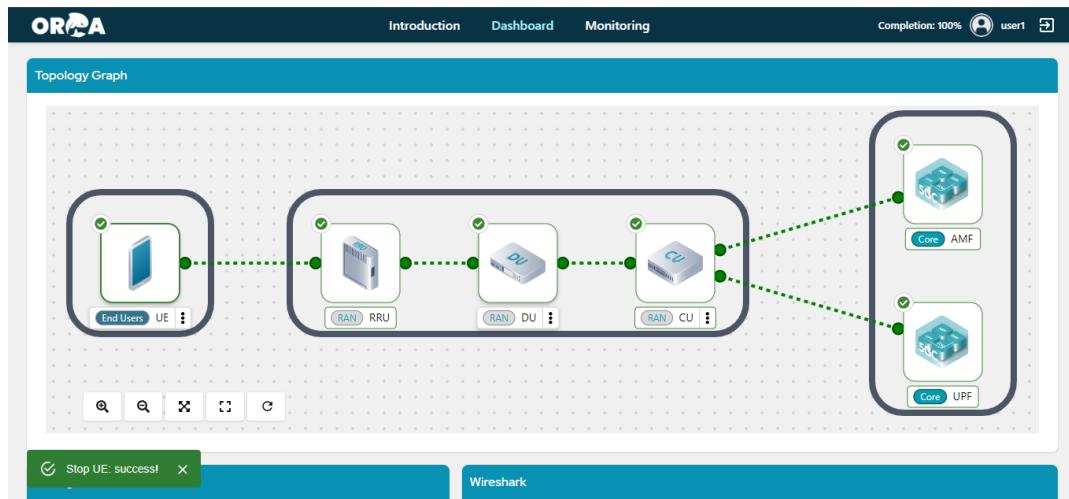
3. User berhasil menjalankan setiap komponen 5G (CU, DU, dan UE) pada manajemen siklus hidup komponen. Hal tersebut ditandai dengan notifikasi sukses pada proses start yang dilakukan pada setiap komponen beserta berubahnya warna frame menjadi hijau

pada komponen tersebut sesaat setelah notifikasi sukses menghilang. Proses ini dapat dilakukan dengan menekan kebab button pada setiap komponen ataupun dengan klik kanan pada mouse ataupun touchpad untuk memunculkan pop up pilihan tombol start.



Gambar 5.54 User Berhasil Menghubungkan State UE

- User berhasil menghentikan setiap komponen 5G (CU, DU, dan UE) pada manajemen siklus hidup komponen. Hal tersebut ditandai dengan notifikasi sukses pada proses stop yang dilakukan pada setiap komponen beserta berubahnya warna frame menjadi kuning pada komponen tersebut sesaat setelah notifikasi sukses menghilang. Proses ini dapat dilakukan dengan menekan kebab button pada setiap komponen ataupun dengan klik kanan pada mouse ataupun touchpad untuk memunculkan pop up pilihan tombol stop.



Gambar 5.55 User Berhasil Menghentikan UE

5.2.2.5 Wireshark Functions

- User berhasil mendapatkan informasi paket data dari hasil *sniffing* pada setiap komponen CU, DU, dan UE serta berhasil menyimpan *file PCAP* pada *database*. Hal tersebut ditandai dengan munculnya informasi paket data hasil sniffing di tab Sniff pada

menu Wireshark yang terdapat pada laman dashboard, tepat disebelah menu Configuration Panel. Informasi yang ditampilkan meliputi nomor, timestamp, IP src, IP dst, dan protocol info.

The screenshot shows the ORPA dashboard with the 'Monitoring' tab selected. On the left, there's a 'Configuration Panel' with tabs for UE, DU, and CU. The 'UE' tab is active. Below it, a message says 'Select a component above'. On the right, the 'Sniff' tab is selected under the 'Files' tab. A search bar shows 'oaitun'. Below it is a table of captured packets:

No.	Timestamp	IP src	IP dst	Protocol Info
13	6.008903318	12.1.1.3	172.253.118.100	ICMP 84 Echo (ping) request id=0x9580, seq=7/1792, ttl=64
14	6.054319579	172.253.118.100	12.1.1.3	ICMP 84 Echo (ping) reply id=0x9580, seq=7/1792, ttl=105 (request in 13)
15	7.010556931	12.1.1.3	172.253.118.100	ICMP 84 Echo (ping) request id=0x9580, seq=8/2048, ttl=64
16	7.051404914	172.253.118.100	12.1.1.3	ICMP 84 Echo (ping) reply id=0x9580, seq=8/2048, ttl=105 (request in 15)
17	8.011636594	12.1.1.3	172.253.118.100	ICMP 84 Echo (ping) request id=0x9580, seq=9/2304, ttl=64
				ICMP 84 Echo (ping) reply id=0x9580,

A blue banner at the bottom left says 'PCAP file stored successfully' with a checkmark icon. A 'Filter' button is at the bottom right of the table.

Gambar 5.56 User Berhasil Mendapatkan Hasil Sniffing UE

Selain itu, notifikasi sukses akan muncul ketika file PCAP hasil dari proses sniffing yang telah dilakukan berhasil disimpan pada database sesaat setelah tombol stop sniffing ditekan. File PCAP ini berisi informasi paket data hasil sniffing namun dalam bentuk yang lebih rinci sehingga user dapat melakukan olah data yang lebih rinci melalui file PCAP tersebut yang dapat dibuka menggunakan aplikasi Wireshark.

2. User berhasil mendapatkan informasi daftar file PCAP yang telah tersedia pada *database*. Hal tersebut ditandai dengan munculnya daftar file PCAP di tab Files pada menu Wireshark yang terdapat pada laman dashboard. Beberapa informasi terkait file tersebut seperti contohnya kapan file tersebut dibuat, nama file, ukuran file, dan juga menu actions yang dapat digunakan untuk mendownload ataupun menghapus file PCAP yang diinginkan dari database.

The screenshot shows the ORPA dashboard with the 'Monitoring' tab selected. On the left, there's a 'Configuration Panel' with tabs for UE, DU, and CU. The 'UE' tab is active. Below it, a table lists configuration parameters like F1 CU Port, F1 DU Port, N2 IP Address, etc. At the bottom is a 'Submit' button. On the right, the 'Wireshark' tab is selected under the 'Files' tab. It shows a table of captured files:

Timestamp	File Name	Size	Actions
7/15/2024 11:59:05 PM	user1_cu-level1_n2_20240715_235831.pcap	816 B	Download Delete
7/15/2024 11:58:10 PM	user1_cu-level1_n2_20240715_225323.pcap	65.88 KB	Download Delete
7/15/2024 11:58:10 PM	user1_cu-level1_f1_20240715_235734.pcap	2.14 KB	Download Delete
7/15/2024 11:58:09 PM	user1_cu-level1_f1_20240715_225222.pcap	72.00 KB	Download Delete

Gambar 5.57 User Berhasil Mendapatkan Hasil File Sniffing

3. User berhasil men-download file PCAP yang diinginkan. Hal tersebut ditandai dengan munculnya pop up pada browser setelah menekan tombol download. File PCAP dapat disimpan pada device user dan dapat digunakan untuk keperluan olah data yang lebih rinci.

Timestamp	File Name	Size	Actions
7/15/2024 11:59:05 PM	user1_cu-level1_n2_20240715_235831.pcap	816 B	Download Delete
7/15/2024 11:58:10 PM	user1_cu-level1_n2_20240715_225323.pcap	65.88 KB	Download Delete
7/15/2024 11:58:10 PM	user1_cu-level1_f1_20240715_235734.pcap	2.14 KB	Download Delete
7/15/2024 11:58:09 PM	user1_cu-level1_f1_20240715_225222.pcap	72.00 KB	Download Delete

Gambar 5.58 User Berhasil Mengunduh Hasil File Sniffing

4. User berhasil menghapus file PCAP yang diinginkan dari database. Hal tersebut ditandai dengan munculnya notifikasi sukses sesaat setelah tombol konfirmasi penghapusan file PCAP yang diinginkan ditekan.

Timestamp	File Name	Size	Actions
7/15/2024 11:58:10 PM	user1_cu-level1_n2_20240715_225323.pcap	65.88 KB	Download Delete
7/15/2024 11:58:10 PM	user1_cu-level1_f1_20240715_235734.pcap	2.14 KB	Download Delete
7/15/2024 11:58:09 PM	user1_cu-level1_f1_20240715_225222.pcap	72.00 KB	Download Delete
7/15/2024 8:11:37 PM	user1_du-level1_f1_20240715_201047.pcap	1.07 KB	Download Delete
7/15/2024	user1_cu-level1_f1_20240715_171640.pcap	1.89 KB	Download Delete

Gambar 5.59 User Berhasil Menghapus Hasil File Sniffing

5.2.3 Performance Testing

Pengujian *performance testing* adalah langkah penting dalam mengukur dan menilai batas kapasitas keseluruhan aplikasi serta performanya di bawah *load* tinggi. Pengujian ini mencakup pengujian *system load capacity* dan koneksi E2E yang dirancang untuk mengidentifikasi batas maksimum resource, serta memastikan aplikasi tetap stabil dan responsif dalam penggunaan.

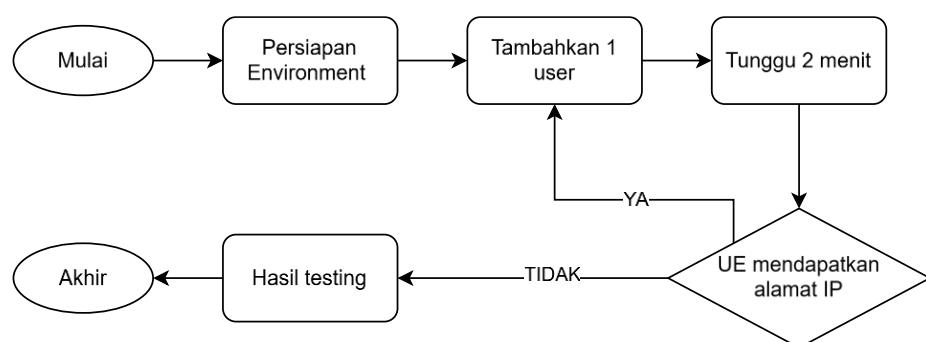
Tabel 5.10 Langkah Performance Testing

Langkah	Deskripsi
Setup Testing Environment	Mempersiapkan lingkungan testing dan memastikan semua komponen klaster berjalan dan siap digunakan.
Inisialisasi Alat Pengujian	Menyiapkan alat pengujian seperti Grafana untuk memonitor dan menguji beban serta kinerja aplikasi.
Pengujian Utilisasi Resource User	Menguji bagaimana aplikasi menggunakan resource selama kondisi normal dan di bawah beban tinggi untuk mengidentifikasi penggunaan rata-rata dan potensi bottleneck.
Pengujian Kapasitas Infra	Menilai batas maksimum kapasitas infrastruktur aplikasi dan memastikan aplikasi tetap responsif saat berada di load maksimum.

Pengujian ini menggunakan `helm` untuk menguji beban dan kinerja aplikasi, serta Grafana untuk visualisasi dan pemantauan performa. akan membantu memonitor penggunaan CPU dan memori dari berbagai komponen aplikasi secara real-time dan Grafana akan menampilkan data performa secara mendetail.

5.2.3.1 System Load Capacity

Pengujian kapasitas beban sistem bertujuan untuk mengidentifikasi batas maksimum jumlah user yang dapat ditampung oleh aplikasi berdasarkan hasil dari pengujian utilisasi sumber daya. Skenario pengujian menggunakan metode *Ramp Up Time* yang meliputi pengujian batas kapasitas maksimum pengguna yang dapat ditampung oleh aplikasi dan pengamatan perubahan beban pada sistem ketika aplikasi digunakan oleh jumlah pengguna yang meningkat hingga mencapai batas maksimum.



Gambar 5.60 Diagram Alir Pengujian System Load Capacity

Gambar 5.60 memperlihatkan diagram alir yang digunakan dalam proses *system load capacity*. Pada inisiasi awal, persiapan *environment* meliputi persiapan konfigurasi beberapa *user* dan memastikan tidak ada *resource* lain yang berjalan pada saat pengujian.

NAME↑	PF	READY	STATUS
oai-cu-level1-user1-cb4b4dcb6-kdwlj	●	2/2	Running
oai-du-level1-user1-79fc6bc764-qkw5s	●	2/2	Running
oai-nr-ue-level1-user1-6d868fcf4-b7xnj	●	2/2	Running

Gambar 5.61 Daftar Pod dan Status Pod per User

Selanjutnya satu akun *user* ditambahkan dengan konfigurasi lengkap dan pada Gambar 5.61 memperlihatkan setiap akun *user* memiliki 3 komponen 5G RAN (CU, DU, dan UE) yang harus dipastikan sudah dalam state *Running*. Setelah 2 menit dapat memeriksa kembali apakah akun *user* dapat terkoneksi 5G secara E2E. Jika iya, maka akun *user* baru akan ditambahkan dan proses ini terus berjalan hingga *user* terbaru menunjukkan penurunan performa yang ditandai dengan tidak terkoneksi secara E2E.

Tabel 5.11 Pengujian System Load Capacity

Menit ke-	Utilisasi CPU	Utilisasi RAM	Keterangan
0	7,2%	23,9%	Idle sistem
1	18,2%	28,5%	Koneksi E2E berhasil pada user ke 1
2	27,9%	32,6%	Koneksi E2E berhasil pada user ke 2
3	37,2%	36,6%	Koneksi E2E berhasil pada user ke 3
4	47,1%	40,6%	Koneksi E2E berhasil pada user ke 4
5	54,4%	44,5%	Koneksi E2E berhasil pada user ke 5
6	62,4%	48,8%	Koneksi E2E berhasil pada user ke 6
7	71,2%	52,7%	Koneksi E2E berhasil pada user ke 7
8	80,2%	56,5%	Koneksi E2E berhasil pada user ke 8
9	86,5%	60,4%	Koneksi E2E berhasil pada user ke 9
10	96,1%	64,9%	Koneksi E2E berhasil pada user ke 10
11	null	null	Koneksi E2E tidak berhasil pada user ke 11

Tabel 5.11 memperlihatkan hasil dari system load capacity dengan metode Ramp Up testing yaitu dengan peningkatan 1 user setiap 2 menit. Pada tabel dapat dilihat total akun user yang telah dibuat berjumlah 11 akun dengan maksimum utilisasi CPU 100% dan utilisasi RAM 75,70%.

Kemudian, pengukuran penggunaan resource per user juga dilakukan untuk mengetahui perkiraan pemakaian resource yang digunakan oleh satu user. Pengukuran dilakukan menggunakan tools K9S dengan parameter pengukuran yang diambil adalah CPU dan RAM.

NAME↑	PF	READY	STATUS	RESTARTS	CPU	MEM
oai-cu-level1-user1-cb4b4dcb6-kdwlj	●	2/2	Running	0	1	116
oai-du-level1-user1-79fc6bc764-qkw5s	●	2/2	Running	0	1641	1493
oai-nr-ue-level1-user1-6d868fcfd4-b7xnj	●	2/2	Running	0	1252	632

Gambar 5.62 Monitoring Resource pada User

Hasil screenshoot menggunakan *tool k9s* pada Gambar 5.62, terdapat pod CU, DU, dan juga UE yang sedang dalam state *Running*. Pod CU menggunakan 1mCPU dan 116MiB RAM, pod DU menggunakan 1641mCPU dan 1493MiB RAM, sedangkan pod UE menggunakan 1252mCPU dan 632MiB RAM. Data tersebut diambil beberapa kali sehingga dapat ditentukan total penggunaan *resource* per satu *user* yang ditampilkan pada tabel 5.12.

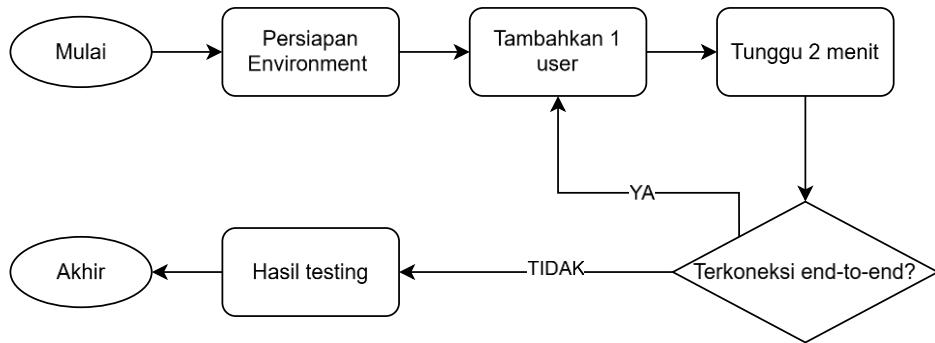
Tabel 5.12 Alokasi CPU dan RAM Satu User

Komponen	CPU (mCPU)	RAM (MiB)
CU	1	119
DU	1569	1304
UE	1211	542
Jumlah	2781	1965

Berdasarkan tabel 5.12 yang menunjukkan penggunaan sumber daya CPU dan RAM, satu akun *user* memerlukan sekitar 2781mCPU dan 1965MiB RAM. Penggunaan sumber daya ini dihitung setelah semua konfigurasi komponen benar dan UE berhasil terhubung ke 5G core. Data ini penting untuk memahami alokasi sumber daya yang diperlukan untuk memastikan koneksi E2E yang stabil.

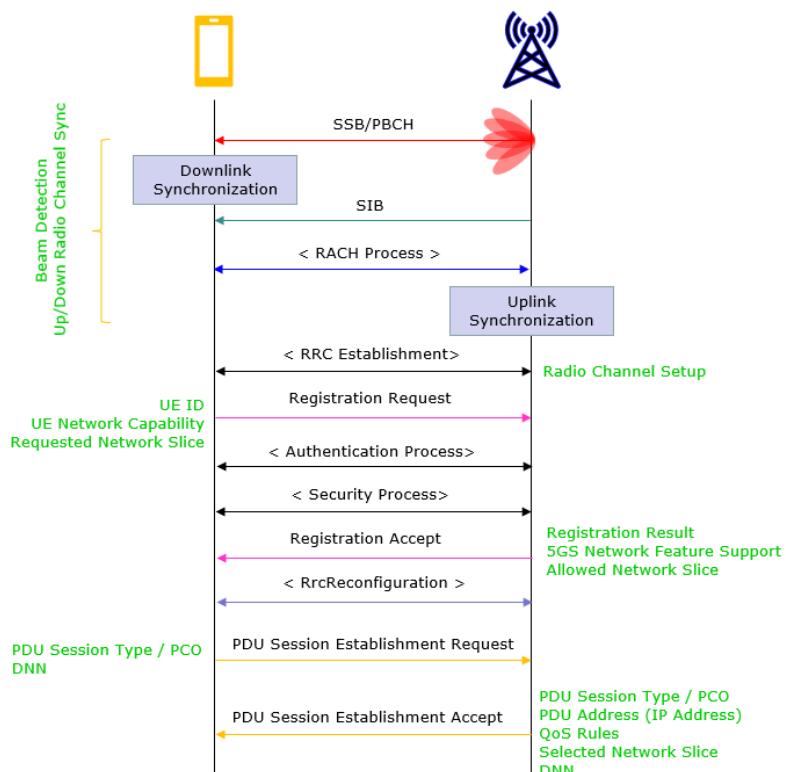
5.2.3.2 End-to-end Connection

Pengujian E2E merupakan proses yang digunakan untuk memastikan bahwa seluruh komponen 5G dapat terhubung dengan baik dari CN hingga ke RAN. Gambar 5.63 memperlihatkan alur skenario pengujian E2E yang dilakukan pada komponen RAN khususnya pada UE.



Gambar 5.63 Diagram Alir Pengujian E2E

Pengujian koneksi E2E dilakukan pada komponen UE untuk memastikan bahwa pengguna akhir dapat terhubung ke jaringan 5G dan mendapatkan alamat IP yang valid. UE mendapatkan alamat IP dan terhubung ke jaringan core 5G dengan melewati berbagai proses dan menggunakan berbagai macam protokol.



Gambar 5.64 Proses UE Mencapai PDU Establishment

Gambar 5.64 memperlihatkan serangkaian proses autentikasi yang dilakukan dari UE ke gNB agar dapat terhubung ke core. Pada akhir proses, terdapat PDU Session Establishment Accept yang menandakan UE sudah terhubung ke core dan mendapatkan alamat IP sehingga UE dapat terhubung ke internet. Ketika dilakukan *sniffing* packet menggunakan wireshark, proses autentikasi menggunakan berbagai macam protokol seperti F1AP, NR RRC, SCTP, dan NAS seperti terlampir pada Gambar 5.65.

No.	Time	Source	Destination	Protocol	Length	Info
9	34.120141	192.168.1.2	192.168.1.1	FIAP/NR RRC	254	InitialULRRCMessageTransfer, RRC Setup Request
10	34.121393	192.168.1.2	192.168.1.2	FIAP/NR RRC	250	SACK (Ack=0, Arvnd=106496) , DLRRCMessageTransfer, RRC Setup
11	34.323962	192.168.1.2	192.168.1.1	SCTP	62	SACK (Ack=0, Arvnd=106496)
12	34.764284	192.168.1.2	192.168.1.1	FIAP/NR RRC/NAS-5GS	138	ULRRCMessageTransfer, RRC Setup Complete, Registration request MAC=0x00000000
13	34.906116	192.168.1.2	192.168.1.2	SCTP	62	SACK (Ack=1, Arvnd=106496)
14	35.439767	192.168.1.2	192.168.1.1	FIAP/NR RRC/NAS-5GS	190	ULRRCMessageTransfer, UL Information Transfer, Authentication request MAC=0x00000000
15	35.243978	192.168.1.2	192.168.1.2	SCTP	62	SACK (Ack=1, Arvnd=106496)
16	35.274389	192.168.1.2	192.168.1.1	FIAP/NR RRC/NAS-5GS	122	ULRRCMessageTransfer, UL Information Transfer, Authentication response MAC=0x00000000
17	35.476921	192.168.1.2	192.168.1.2	SCTP	62	SACK (Ack=2, Arvnd=106496)
18	35.484712	192.168.1.2	192.168.1.1	FIAP/NR RRC/NAS-5GS	290	ULRRCMessageTransfer, DL Information Transfer, Security mode command MAC=0x80000000
19	35.691947	192.168.1.2	192.168.1.1	SCTP	62	SACK (Ack=2, Arvnd=106496)
20	35.712826	192.168.1.2	192.168.1.1	FIAP/NR RRC/NAS-5GS	158	ULRRCMessageTransfer, UL Information Transfer MAC=0x00000000
21	35.782959	192.168.1.1	192.168.1.2	FIAP	174	SACK (Ack=3, Arvnd=106496) , UEContextSetupRequest
22	35.784056	192.168.1.2	192.168.1.1	FIAP	242	SACK (Ack=3, Arvnd=106496) , UEContextSetupResponse
23	35.987983	192.168.1.1	192.168.1.2	SCTP	62	SACK (Ack=4, Arvnd=106496)
24	35.994828	192.168.1.1	192.168.1.2	FIAP/NR RRC	109	ULRRCMessageTransfer, Security Mode Complete MAC=0x3e94b687
25	36.000067	192.168.1.1	192.168.1.2	FIAP/NR RRC	122	SACK (Ack=5, Arvnd=106496) , DLRRCMessageTransfer, UE Capability Enquiry MAC=0x90f17d95
26	36.200067	192.168.1.2	192.168.1.1	SCTP	62	SACK (Ack=4, Arvnd=106496)
27	36.469231	192.168.1.2	192.168.1.1	FIAP/NR RRC	114	ULRRCMessageTransfer, UE Capability Information MAC=0x6e20989d
28	36.479364	192.168.1.1	192.168.1.2	FIAP/NR RRC/NAS-5GS	342	SACK (Ack=6, Arvnd=106496) , DLRRCMessageTransfer, RRC Reconfiguration MAC=0x597b7406
29	36.671976	192.168.1.2	192.168.1.1	SCTP	62	SACK (Ack=5, Arvnd=106496)
30	36.750100	192.168.1.2	192.168.1.1	FIAP/NR RRC	102	ULRRCMessageTransfer, RRC Reconfiguration Complete MAC=0x3361b24e
31	36.751646	192.168.1.1	192.168.1.2	FIAP	118	SACK (Ack=6, Arvnd=106496) , UEContextModificationRequest
32	36.752177	192.168.1.2	192.168.1.1	FIAP	98	SACK (Ack=6, Arvnd=106496) , UEContextModificationResponse
33	36.960010	192.168.1.1	192.168.1.2	SCTP	62	SACK (Ack=8, Arvnd=106496)
34	37.716499	192.168.1.2	192.168.1.1	FIAP/NR RRC/NAS-5GS	114	ULRRCMessageTransfer, UL Information Transfer MAC=0xfc5d2ba

Gambar 5.65 Hasil Wireshark Saat Proses PDU Establishment

Langkah terakhir adalah verifikasi pada UE yang meliputi pengecekan alamat IP dan pengujian koneksi ke internet. Interface oaitun_ue1 merupakan interface yang digunakan untuk mengirimkan trafik data-plane. Sehingga, ketika interface ini sudah mendapatkan alamat IP menandakan sudah bisa terhubung ke internet. Gambar 5.66 merupakan verifikasi alamat IP pada interface oaitun_ue1 dan verifikasi koneksi ke internet.

```
root@oai-nr-ue-level1-user1-6d868fc4-b7xnj:/opt/oai-nr-ue# ip addr show oaitun_ue1
3: oaitun_ue1: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
    link/none
      brd 00:00:00:00:00:00
      inet 12.1.1.2/24 brd 12.1.1.255 scope global oaitun_ue1
        valid_lft forever preferred_lft forever
        inet6 fe80::eaf:25b:4c9:8029/64 scope link stable-privacy
          valid_lft forever preferred_lft forever
root@oai-nr-ue-level1-user1-6d868fc4-b7xnj:/opt/oai-nr-ue# ping www.google.com
PING www.google.com (64.233.170.103) 56(84) bytes of data.
64 bytes from sg-in-f103.1e100.net (64.233.170.103): icmp_seq=1 ttl=57 time=36.4 ms
64 bytes from sg-in-f103.1e100.net (64.233.170.103): icmp_seq=2 ttl=57 time=35.5 ms
64 bytes from sg-in-f103.1e100.net (64.233.170.103): icmp_seq=3 ttl=57 time=34.9 ms
64 bytes from sg-in-f103.1e100.net (64.233.170.103): icmp_seq=4 ttl=57 time=36.5 ms
...
-- www.google.com ping statistics --
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 34.871/35.829/36.513/0.681 ms
```

Gambar 5.66 Pod UE Mendapatkan Interface oaitun_ue1

Tabel 5.13 memperlihatkan hasil pengujian end-to-end dengan menggunakan metode yang sama pada System Load Capacity. Total akun user yang sudah dibuat berjumlah 11 akun dengan total *success rate* 100% sebanyak 10 akun user dan *success rate* 0% sebanyak 2 akun user.

Tabel 5.13 Hasil Pengujian Success Rate

User ke-	Success Rate	Keterangan
0	100%	UE mendapatkan alamat IP
1	100%	UE mendapatkan alamat IP
2	100%	UE mendapatkan alamat IP
3	100%	UE mendapatkan alamat IP
4	100%	UE mendapatkan alamat IP

5	100%	UE mendapatkan alamat IP
6	100%	UE mendapatkan alamat IP
7	100%	UE mendapatkan alamat IP
8	100%	UE mendapatkan alamat IP
9	100%	UE mendapatkan alamat IP
10	100%	UE mendapatkan alamat IP
11	0%	UE tidak mendapatkan alamat IP

5.2.4 User Acceptance Testing

User Acceptance Testing (UAT) bertujuan untuk menilai kepuasan pengguna terhadap aplikasi dan memastikan aplikasi memenuhi kebutuhan mereka. UAT melibatkan *end users*, admin, dan calon pembeli aplikasi untuk memberikan *feedback* terhadap fungsionalitas dan pengalaman pengguna.

Pengujian UAT dilakukan dengan menggunakan server milik Telecom Infra Project (TIP). Jumlah *end user* yang melakukan pengujian ini berjumlah 30 orang dari berbagai *background* seperti mahasiswa, *staff* TIP, dan *engineer*. Untuk dokumentasi pelaksanaan UAT ini dapat dilihat pada lampiran 5.1 – 5.4.

Pelaksanaan UAT dilakukan dengan sesi uji coba yang terstruktur. Pada pengujian ini, user mendapatkan kesempatan untuk melakukan konfigurasi dan mensimulasikan 5G secara langsung pada *dashboard*. Di akhir sesi uji coba, *user* akan mengisi kuisioner *feedback* mengenai pengalaman mereka ketika menggunakan aplikasi. Pengujian UAT memungkinkan *user* maupun *developer* mengidentifikasi kekurangan atau *bug* pada fitur-fitur yang ada. Selain itu juga agar aplikasi lebih sesuai dengan kebutuhan dan ekspektasi *end user*.

Pengujian UAT ini dibagi menjadi dua. Yang pertama *functional testing* bertujuan untuk mengetahui fitur-fitur pada aplikasi berhasil dijalankan oleh *user*. Yang kedua yaitu *usability testing* untuk mendapatkan *feedback* berharga mengenai aplikasi dari *end user*.

Tabel 5.14 Langkah Pengujian UAT

Langkah	Deskripsi
Penyusunan Kuesioner	Menyusun pertanyaan dengan metode System Usability Scale (SUS) untuk mendapatkan feedback dari pengguna
Penyusunan Tabel Functional Testing	Menyusun tabel untuk <i>functional testing</i> tiap fitur
Pelaksanaan UAT	Mengundang 30 pengguna dari berbagai background untuk menguji dashboard. UAT dibagi kedalam enam sesi, di mana tiap sesinya berisi lima orang.
Pengumpulan Kuesioner	Mengumpulkan dan menganalisis feedback dari pengguna yang melakukan pengujian.
Perbaikan berdasarkan feedback dari pengguna	Melakukan perbaikan pada dashboard berdasarkan feedback yang diterima

5.2.4.1 Functional Testing

Functional testing adalah jenis pengujian yang memverifikasi bahwa setiap fitur pada aplikasi beroperasi sesuai dengan spesifikasi yang diinginkan. Pengujian ini menggunakan metode *black box* untuk menguji fitur pada tiap halamannya.

Metode *Black Box* adalah metode pengujian aplikasi di mana struktur internal seperti *source code* aplikasi tidak perlu diketahui oleh penguji sehingga pengujian ini dilakukan dari perspektif *end user*. Penguji hanya fokus pada apa yang dilakukan aplikasi.

Functional testing dengan metode *black box* adalah teknik yang ampuh untuk memastikan bahwa fitur aplikasi memenuhi persyaratan fungsionalnya tanpa menggali struktur internalnya. Hal ini membantu mengidentifikasi perbedaan antara hasil yang diharapkan dengan hasil sebenarnya dan memastikan aplikasi *user-centric*.

Tabel 5.15 Hasil *Functional Testing* pada Halaman Admin

Admin Page						
No	Halaman	Fitur	Langkah Pengujian	Hasil yang Diharapkan	Hasil Aktual	Komentar
1	User Management	Create	Membuat user baru sesuai dengan jumlah yang dinginkan	Akun user baru berhasil ditambahkan	Berhasil	
		Edit	Mengubah password baru pada user	Password baru pada user berhasil diubah	Berhasil	
		Remove	Menghapus akun user	Akun user berhasil dihapus dari sistem	Berhasil	

Tabel 5.16 Hasil Functional Testing pada Halaman Pengguna

User Page						
No	Halaman	Fitur	Langkah Pengujian	Hasil yang Diharapkan	Hasil Aktual	Komentar
1	Login	Login	Memasukkan username dan password lalu klik login	User berhasil login dan diarahkan ke halaman Introduction	Berhasil	
2	Introduction	Introduction	Menekan setiap menu pada sidebar di halaman Introduction	Berhasil menampilkan semua menu pada sidebar	Berhasil	
3	Dashboard	Topology Graph	Menekan tombol decorator status pada setiap komponen	Berhasil menampilkan pop up yang menunjukkan "deployment status"	Berhasil	

		<p>Menekan tombol gambar setiap komponen</p>	Berhasil menampilkan sidebar dan user dapat melihat Protocol Stack dan Log pada sidebar tersebut	Berhasil	
		<p>Menekan tombol "Protocol Stack" dan "Logs" pada sidebar komponen UE</p>	Berhasil menampilkan list protocol dan logs pada komponen UE	Berhasil, tapi belum sempurna	Protocol dan logs terkadang berhasil ditampilkan pada sidebar, terkadang tidak muncul pada sidebar.
		<p>Menekan tombol "Protocol Stack" dan "Logs" pada sidebar komponen DU</p>	Berhasil menampilkan list protocol dan logs pada komponen DU	Berhasil, tapi belum sempurna	Protocol dan logs terkadang berhasil ditampilkan pada sidebar, terkadang tidak muncul pada sidebar.

		Menekan tombol "Protocol Stack" dan "Logs" pada sidebar komponen CU	Berhasil menampilkan list protocol dan logs pada komponen DU	Berhasil, tapi belum sempurna	Protocol dan logs terkadang berhasil ditampilkan pada sidebar, terkadang tidak muncul pada sidebar.
		Melakukan ping dan curl pada terminal di sidebar komponen UE	Berhasil melakukan ping dan curl	Berhasil	
		Menekan tombol kebab menu pada komponen UE. Lalu menekan tombol "Stop", "Start" dan "Restart"	Komponen UE berhasil reload jika terdapat perubahan konfigurasi ketika menekan tombol "Restart". Komponen UE berhenti running ketika menekan tombol "Stop" dan running kembali setelah menekan tombol "Start".	Berhasil	

		<p>Menekan tombol kebab menu pada komponen DU. Lalu menekan tombol "Stop", "Start" dan "Restart"</p>	<p>Komponen DU berhasil reload jika terdapat perubahan konfigurasi ketika menekan tombol "Restart". Komponen DU berhenti running ketika menekan tombol "Stop" dan running kembali setelah menekan tombol "Start".</p>	Berhasil
		<p>Menekan tombol kebab menu pada komponen CU. Lalu menekan tombol "Stop", "Start" dan "Restart"</p>	<p>Komponen CU berhasil reload jika terdapat perubahan konfigurasi ketika menekan tombol "Restart". Komponen CU berhenti running ketika menekan tombol "Stop" dan running kembali setelah menekan tombol "Start".</p>	Berhasil
		<p>Menekan tombol link tiap interface</p>	<p>Berhasil menampilkan</p>	Berhasil

		pop up nama interface		
	Menggeser tombol tiap komponen	Tiap komponen dapat digeser sesuai keinginan User	Berhasil	
	Menggeser topology	Topologi dapat digeser sesuai keinginan User	Berhasil	
	Menekan tombol control bar (zoom in, zoom out, fit to screen, reset view)	Bentuk topologi berhasil zoom in, zoom out, fit to screen, dan kembali ke bentuk semula	Berhasil	
Configuration Panel	Mengisi value pada setiap kolom yang tersedia	User dapat menginput value pada setiap kolom saat mengkonfigurasi	Berhasil	
	Menekan tombol submit setelah mengisi value	Value yang telah terinput berhasil di-submit	Berhasil	
	Dapat melihat value yang sudah terupdate setelah proses submit	User dapat melihat value yang telah ter-submit pada kolom "Current Value"	Berhasil	

	Mengkonfigurasi semua value pada komponen UE	Komponen UE berhasil terkonfigurasi, persentase completion bertambah, terdapat checklist pada config panel, warna pada topologi dan link berubah warna menjadi hijau	Berhasil	
	Mengkonfigurasi semua value pada komponen DU	Komponen DU berhasil terkonfigurasi, persentase completion bertambah, terdapat checklist pada config panel, warna pada topologi dan link berubah warna menjadi hijau	Berhasil	
	Mengkonfigurasi semua value pada komponen CU	Komponen CU berhasil terkonfigurasi, persentase completion bertambah, terdapat checklist pada config panel,	Berhasil	

		warna pada topologi dan link berubah warna menjadi hijau		
Wireshark	Menekan tombol dropdown untuk memilih komponen yang akan di sniff	User dapat memilih komponen CU, DU, dan UE	Berhasil	
	Menekan tombol start untuk memulai proses sniffing paket data pada komponen UE	Wireshark berhasil mengcapture paket data dan menampilkan protokol info pada komponen UE	Berhasil, tapi belum sempurna	Wireshark berhasil mengcapture paket data pada komponen UE jika pengguna memilih komponen UE terlebih dahulu untuk di-sniffing. Jika komponen lain dipilih terlebih dahulu, wireshark

			tidak dapat meng- capture paket data pada komponen UE.
	Menekan tombol start untuk memulai proses sniffing paket data pada komponen DU	Wireshark berhasil mengcapture paket data dan menampilkan protokol info pada komponen DU	Wireshark berhasil mengcapture paket data pada komponen DU jika pengguna memilih komponen DU terlebih dahulu untuk di-sniffing. Jika komponen lain dipilih terlebih dahulu, wireshark tidak dapat meng- capture paket data pada komponen DU.

		<p>Menekan tombol start untuk memulai proses sniffing paket data pada komponen CU</p>	<p>Wireshark berhasil mengcapture paket data dan menampilkan protokol info pada komponen CU</p>	<p>Berhasil, tapi belum sempurna</p>	<p>Wireshark berhasil mengcapture paket data pada komponen CU, namun tidak bisa dilakukan lebih dari 1 user di waktu yang sama.</p>
		<p>Menekan tombol stop untuk menghentikan proses sniffing paket data</p>	<p>Wireshark berhasil menghentikan proses sniffing paket data pada semua komponen</p>	<p>Berhasil</p>	
		<p>Meng-input paket data yang ingin di-filter dan menekan tombol "Filter"</p>	<p>Berhasil memfilter paket data yang diinginkan</p>	<p>Berhasil</p>	
		<p>Menekan tab Files untuk melihat daftar file pcap yang telah di capture sebelumnya</p>	<p>Berhasil menampilkan tab Files pada fitur Wireshark</p>	<p>Berhasil</p>	

			Menekan tombol download untuk mendownload file pcap yang diinginkan	File pcap berhasil diunduh di device user dan file tidak corrupt	Berhasil	
			Menekan tombol remove untuk menghapus file pcap yang diinginkan	File pcap berhasil dihapus pada tab Files	Berhasil	
4	Monitoring	L1 UE Graph	Melihat informasi grafik dari komponen UE secara real-time	Grafik berhasil ditampilkan secara real time	Berhasil	
		UE Protocol Table	Melihat informasi protocol dari komponen UE dalam bentuk tabel secara real-time	Berhasil menampilkan informasi protocol pada komponen UE secara real-time	Berhasil	

5.2.4.2 Usability Testing

Usability testing adalah metode pengujian untuk mengamati dan mengevaluasi bagaimana pengalaman pengguna saat berinteraksi dengan sistem. Usability testing yang digunakan yaitu System Usability Scale (SUS) yang dikembangkan oleh John Brooke di tahun 1996. SUS adalah standar kuesioner yang paling banyak digunakan untuk menilai usabilitas yang dirasakan oleh pengguna. SUS bertujuan untuk mengevaluasi kegunaan berbagai produk dan sistem, seperti situs web, aplikasi *software*, dan *device*. SUS terdiri dari 10 pertanyaan dengan lima pilihan jawaban responden, mulai dari “Sangat Tidak Setuju” hingga “Sangat Setuju”.

10 Pertanyaan SUS yaitu di antaranya:

1. Saya berpikir akan menggunakan sistem ini lagi
2. Saya merasa sistem ini rumit untuk digunakan
3. Saya merasa sistem ini mudah digunakan
4. Saya membutuhkan bantuan dari orang lain atau teknisi dalam menggunakan sistem ini
5. Saya merasa fitur-fitur sistem ini berjalan dengan semestinya
6. Saya merasa ada banyak hal yang tidak konsisten (tidak serasi pada sistem ini)
7. Saya merasa orang lain akan memahami cara menggunakan sistem ini dengan cepat
8. Saya merasa sistem ini membingungkan
9. Saya merasa tidak ada hambatan dalam menggunakan sistem ini
10. Saya perlu membiasakan diri terlebih dahulu sebelum menggunakan sistem ini

Setelah responden mengisi kuesioner, maka perlu melakukan perhitungan terhadap data yang didapatkan. SUS memiliki aturan dalam menghitung skor yang didapat. Berikut adalah aturan skoring pada metode SUS:

- Untuk pertanyaan bermomor ganjil: nilai dari setiap pertanyaan yang didapat dari pengguna akan dikurangi 1 nilai. Contoh nilai yang didapat yaitu 4 maka nilai akhirnya ($4-1=3$)
- Untuk pertanyaan bermomor genap: nilai 5 dikurangi nilai dari setiap pertanyaan yang didapat dari pengguna. Contoh nilai yang didapat yaitu 4 maka nilai akhirnya ($5-4=1$)
- Hal ini bertujuan untuk menskalakan semua nilai dari 0 hingga 4
- Jumlahkan semua nilai, lalu kalikan dengan 2,5 untuk mengkonversi rentang skor menjadi 0 hingga 100

Setiap pertanyaan bermomor ganjil, nilai yang diharapkan adalah 5 (Sangat Setuju) dan nilai yang tidak diharapkan adalah 1 (Sangat Tidak Setuju). Lalu untuk pertanyaan bermomor genap, nilai yang diharapkan adalah 1 (Sangat Tidak Setuju) dan nilai yang tidak diharapkan adalah 5 (Sangat Setuju). Cara di atas adalah cara untuk menentukan skor akhir dari tiap responden. Selanjutnya adalah menghitung data dari keseluruhan responden dengan mencari rata-ratanya. Jadi menjumlahkan skor akhir dari semua responden lalu dibagi dengan jumlah responden.

Rumus untuk menghitung skor SUS sebagai berikut:

$$\bar{x} = \frac{\sum x}{n}$$

(5.1)

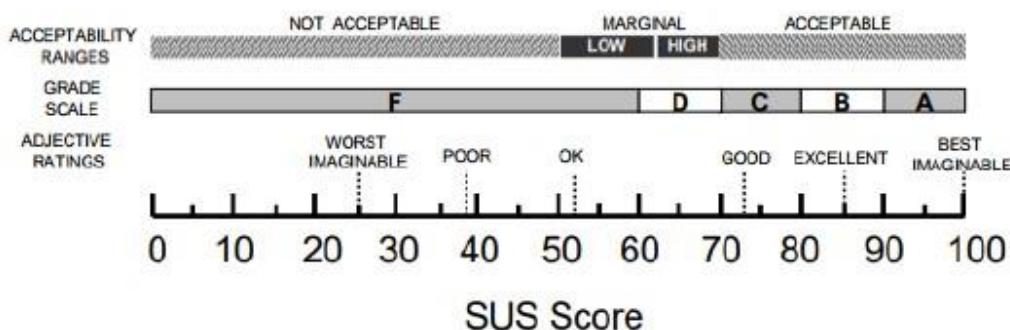
Keterangan:

\bar{x} : skor rata-rata

$\sum x$: jumlah skor SUS

n : jumlah responden

Dari hasil nilai rata-rata skor SUS yang didapat, langkah selanjutnya yaitu membanding nilai skor SUS yang didapat dengan kategori penilaian SUS.



Gambar 5.67 Index Parameter Penilaian System Usability Scale

Nilai rata-rata SUS dari berbagai penelitian adalah 68. Sederhananya, jika di bawah 68 maka nilai SUS dianggap rendah dan aplikasi perlu diperbaiki. Sedangkan jika di atas 68, maka nilai SUS dianggap tinggi. Namun jika mengacu pada gambar, maka skor di atas 80 dianggap “*excellent*”, antara 70 dan 80 dianggap “*good*”, antara 50 dan 69 “*OK*” atau dapat diterima, dan di bawah 50 dianggap “*poor*”.

Selain 10 pertanyaan SUS pada kuesioner, kami menambahkan enam pertanyaan untuk mengetahui apakah dashboard kami meningkatkan *skill* dan menambah pengetahuan pengguna mengenai 5G RAN dan apakah pengguna akan merekomendasikan aplikasi ini ke orang lain. Enam pertanyaan tersebut adalah:

1. Saya merasa fitur *Graphical User Interface* (GUI) pada ORCA *dashboard website* memudahkan saya dalam konfigurasi komponen RAN 5G (CU, DU, dan UE)

2. *Lab guidance* dan materi Open RAN 5G pada halaman Introduction membantu saya dalam memahami Open RAN 5G dan parameter-parameter konfigurasi 5G RAN
3. Visualisasi *topology graph* membantu saya memahami arsitektur 5G
4. Saya dapat mempelajari protokol apa saja pada jaringan 5G dan dapat menganalisis paket data jaringan melalui fitur Wireshark
5. Saya dapat memahami *Key Performance Indicator* (KPI) apa saja yang terdapat pada komponen UE pada fitur monitoring
6. Seberapa besar anda akan merekomendasikan ORCA *dashboard website* ini kepada orang lain?

Kelemahan dari kuesioner SUS adalah kita tidak dapat mengetahui masalah pada aplikasi jika terdapat pertanyaan dengan nilai yang rendah atau nilai akhirnya rendah. Oleh karena itu, terdapat kolom kesan dan saran untuk dapat mengetahui masalah yang dialami pengguna dan mendapatkan masukan dari pengguna terkait aplikasi.

Berikut adalah hasil analisis perhitungan *usability testing* dari kuesioner *feedback* pengguna. Nilai akhir SUS dari UAT 1 adalah 65,42. Untuk perhitungan lebih lengkapnya dapat dilihat pada lampiran 5.5. Lalu untuk hasil akhir SUS dari UAT 2 yaitu 77,1 dan perhitungan lengkapnya dapat dilihat pada lampiran 5.6.

Tabel 5.17 Hasil Perhitungan SUS dari UAT 1

No	Respondent	User Persona	Total	Final Score (Total x 2,5)
1	Respondent 1	Staff TIP	22	55
2	Respondent 2	Staff TIP	30	75
3	Respondent 3	Student	23	57,5
4	Respondent 4	Student	30	75
5	Respondent 5	Student	18	45
6	Respondent 6	Student	23	57,5
7	Respondent 7	Student	23	57,5
8	Respondent 8	Student	20	50
9	Respondent 9	Staff TIP	26	65
10	Respondent 10	Student	21	52,5
11	Respondent 11	Staff TIP	28	70
12	Respondent 12	Student	27	67,5
13	Respondent 13	Student	32	80
14	Respondent 14	Student	22	55
15	Respondent 15	Student	35	87,5

16	Respondent 16	Student	23	57,5
17	Respondent 17	Student	32	80
18	Respondent 18	Student	32	80
19	Respondent 19	Staff TIP	28	70
20	Respondent 20	Student	26	65
21	Respondent 21	Student	26	65
22	Respondent 22	Engineer	22	55
23	Respondent 23	Engineer	31	77,5
24	Respondent 24	Engineer	29	72,5
25	Respondent 25	Engineer	28	70
26	Respondent 26	Engineer	25	62,5
27	Respondent 27	Engineer	18	45
28	Respondent 28	Engineer	32	80
29	Respondent 29	Engineer	22	55
30	Respondent 30	Student	31	77,5
Average				65,42

Tabel 5.18 Hasil Perhitungan SUS dari UAT 2

No	Respondent	User Persona	Total	Final Score (Total x 2,5)
1	Respondent 1	Staff TIP	39	97,5
2	Respondent 2	Staff TIP	34	85
3	Respondent 3	Student	27	67,5
4	Respondent 4	Student	36	90
5	Respondent 5	Student	29	72,5
6	Respondent 6	Student	27	67,5
7	Respondent 7	Student	32	80
8	Respondent 8	Student	33	82,5
9	Respondent 9	Staff TIP	32	80
10	Respondent 10	Student	34	85
11	Respondent 11	Staff TIP	33	82,5
12	Respondent 12	Student	37	92,5
13	Respondent 13	Student	29	72,5
14	Respondent 14	Student	34	85
15	Respondent 15	Student	35	87,5
16	Respondent 16	Student	28	70
17	Respondent 17	Student	30	75
18	Respondent 18	Student	29	72,5
19	Respondent 19	Staff TIP	30	75
20	Respondent 20	Student	27	67,5
21	Respondent 21	Student	26	65

22	Respondent 22	Engineer	22	55
23	Respondent 23	Engineer	33	82,5
24	Respondent 24	Engineer	29	72,5
25	Respondent 25	Engineer	35	87,5
26	Respondent 26	Engineer	25	62,5
27	Respondent 27	Engineer	29	72,5
28	Respondent 28	Engineer	29	72,5
29	Respondent 29	Engineer	30	75
30	Respondent 30	Student	32	80
Average				77,1

5.3 Analisa Hasil Pengujian

5.3.1 Hasil API Testing

Hasil API *testing* dilihat berdasarkan akumulasi keberhasilan seluruh API yang tersedia pada setiap fungsionalitas sesuai dengan spesifikasi yang telah ditentukan. Hasil ini mengacu pada fungsionalitas API yang dites menggunakan aplikasi Postman untuk tipe REST API dan juga *code editor* dengan Django *libraries* untuk tipe WebSocket API. Hasil ini akan disajikan dalam bentuk tabel yang dibagi menjadi lima bagian.

5.3.1.1 Hasil User Management APIs

Tabel 5.19 Hasil User Management APIs

Kategori Fungsi	Hasil yang diharapkan	Status		Kendala
		Sukses	Gagal	
Create user	HTTP status code: 201 Created	✓		-
List all users	HTTP status code: 200 OK	✓		-
Update user's password	HTTP status code: 200 OK	✓		-
Delete user	HTTP status code: 204 No Content	✓		-
Get requested user information	HTTP status code: 200 OK	✓		-

Berdasarkan tabel di atas, seluruh kategori fungsi berstatus sukses, yang berarti tingkat keberhasilannya sebesar 100%. Fungsi untuk manajemen user tidak memerlukan olah data yang rumit pada *database*, sehingga kemungkinan adanya kendala ataupun faktor penghambat sangatlah kecil. Dengan begitu, dapat disimpulkan fungsionalitas API ini bekerja dengan baik.

5.3.1.2 Hasil Token APIs

Tabel 5.20 Hasil Token APIs

Kategori Fungsi	Hasil yang diharapkan	Status		Kendala
		Sukses	Gagal	
<i>Login</i>	<i>HTTP status code: 200 OK</i>	✓		-
<i>Logout</i>	<i>HTTP status code: 200 OK</i>	✓		-
<i>Refresh Token</i>	<i>HTTP status code: 200 OK</i>	✓		-
<i>Verify Token</i>	<i>HTTP status code: 200 OK</i>	✓		-

Berdasarkan tabel di atas, seluruh kategori fungsi berstatus sukses, yang berarti tingkat keberhasilannya sebesar 100%. Fungsi untuk manajemen JSON Web Token (JWT) yang dikembangkan sudah mencakup segala hal yang dibutuhkan untuk mendukung proses otentikasi dan otorisasi. Selain itu, tidak ada faktor penghambat dalam proses pengembangan fungsionalitas API ini yang menandakan API bekerja dengan baik.

5.3.1.3 Hasil Kubernetes APIs

Tabel 5.21 Kubernetes APIs

Kategori Fungsi	Hasil yang diharapkan	Status		Kendala
		Sukses	Gagal	
<i>Get Requested Pod List</i>	<i>HTTP status code: 200 OK</i>	✓		-
<i>Get Requested Deployment List</i>	<i>HTTP status code: 200 OK</i>	✓		-

<i>Get Requested Pod Log</i>	<i>HTTP status code: 200 OK</i>	✓		-
<i>Restart 5G Components</i>	<i>HTTP status code: 200 OK</i>	✓		-
<i>Get Core (AMF & UPF) Log</i>	<i>HTTP status code: 200 OK</i>	✓		-
<i>Get Core (AMF & UPF) Deployment Status</i>	<i>HTTP status code: 200 OK</i>	✓		-
<i>Ping and cURL command</i>	<i>Hasil perintah ping dan cURL dapat tertampilkan dengan benar pada terminal code editor</i>	✓		-
<i>Get UE Monitoring (Key Performance Indicator)</i>	<i>KPI dapat tertampilkan dengan benar pada terminal code editor</i>	✓		-
<i>Get SCTP Protocol Information</i>	<i>Informasi protocol SCTP dapat tertampilkan dengan benar pada terminal code editor</i>	✓		-

Berdasarkan tabel di atas, seluruh kategori fungsi berstatus sukses, yang berarti tingkat keberhasilannya sebesar 100%. Terlepas dari itu, fungsi untuk manajemen kluster Kubernetes yang telah dikembangkan memiliki beberapa faktor penghambat yang sempat muncul meliputi penempatan file config yang tidak tepat, sehingga fungsi manajemen tidak dapat dijalankan dengan benar.

Namun hal itu dapat diatasi dengan menempatkan file config pada direktori yang tepat beserta pemberian hak akses yang benar agar system backend dapat mengaksesnya tanpa

kendala. Dan juga, sebagai faktor pendukung dalam pengembangan fungsi Kubernetes ini, tidak hanya dibutuhkan REST API melainkan juga Websocket API. Websocket API sangat membantu dalam hal integrasi dengan data-data yang bersifat real-time dan dibutuhkan untuk diolah secara real-time. Dengan begitu, dapat disimpulkan jika kendala yang sempat muncul dapat diatasi dengan baik dan keseluruhan fungsi bekerja dengan benar.

Rencana untuk pengembangan lebih lanjut dari fungsionalitas ini adalah dengan memanfaatkan Websocket API dengan maksimal agar pengolahan data yang membutuhkan koneksi real-time dapat dilakukan dengan baik. Meskipun saat ini penggunaan websocket sudah dilakukan, namun dalam implementasinya belum dimanfaatkan dengan maksimal dikarenakan keterbatasan fungsi yang diperlukan hingga saat ini.

5.3.1.4 Hasil Helm Chart APIs

Tabel 5.22 Hasil Helm Chart APIs

Kategori Fungsi	Hasil yang diharapkan	Status		Kendala
		Sukses	Gagal	
<i>Get Configuration Values</i>	<i>HTTP status code: 200 OK</i>	✓		-
<i>Config Component Values</i>	<i>HTTP status code: 200 OK</i>	✓		-
<i>Start Component</i>	<i>HTTP status code: 200 OK</i>	✓		-
<i>Stop Component</i>	<i>HTTP status code: 200 OK</i>	✓		-

Berdasarkan tabel di atas, seluruh kategori fungsi berstatus sukses, yang berarti tingkat keberhasilannya sebesar 100%. Terlepas dari itu, fungsi untuk manajemen Helm Chart sempat memiliki faktor penghambat dalam proses pengembangannya. Faktor tersebut sama halnya dengan fungsi manajemen kluster Kubernetes yang terletak pada kurang tepatnya pemberian hak akses untuk melakukan baris perintah, yang mana dalam kasus ini adalah baris perintah Helm. Dengan begitu, dapat disimpulkan tidak ada kendala pada fungsionalitasnya.

5.3.1.5 Hasil Wireshark APIs

Tabel 5.23 Hasil Wireshark APIs

Kategori Fungsi	Hasil yang diharapkan	Status		Kendala
		Sukses	Gagal	
List PCAP Files	HTTP status code: 200 OK	✓		-
Download PCAP Files	HTTP status code: 200 OK	✓		-
Delete PCAP Files	HTTP status code: 204 No Content	✓		-
Sniffing Process and Save PCAP Files into Database	Hasil sniffing paket data dapat tertampil dengan benar pada terminal code editor	✓		-

Berdasarkan tabel di atas, seluruh kategori fungsi berstatus sukses, yang berarti tingkat keberhasilannya sebesar 100%. Fungsi untuk Wireshark merupakan fungsi penting yang memerlukan koneksi real-time untuk proses pengolahan paket data hasil dari proses sniffing yang dilakukan. Faktor penghambat yang sempat ditemui adalah penggunaan REST API yang kurang efektif dalam proses pengolahan data sniffing secara real-time.

Dengan adanya permasalahan itu, solusi yang digunakan adalah pengimplementasian Websocket API seperti halnya dengan kasus yang terjadi pada fungsi manajemen klaster Kubernetes. Dengan begitu, pengolahan paket data hasil proses sniffing dapat dilakukan secara langsung tanpa harus menunggu data yang dihasilkan dikumpulkan semuanya terlebih dahulu. Rencana kedepannya dalam pengembangan ini adalah untuk mengimplementasikan websocket dengan maksimal untuk fungsi sniffing.

Kendala yang ditemui setelah mengimplementasikan websocket pada fungsi sniffing terdapat pada proses koneksi untuk websocket terkadang memakan waktu yang sedikit lama. Sehingga, terdapat sedikit jeda pada sisi frontend ketika sebuah tombol berfungsi untuk memicu koneksi websocket tersebut dan mengirimkan datanya ke sisi frontend.

5.3.2 Hasil *End-to-End (E2E) Testing*

Hasil ini mengacu pada keberhasilan seluruh alur pada aplikasi yang mengindikasikan sistem frontend dan backend bekerja harmonis pada semua fungsinya. Proses testing dilakukan menggunakan website dashboard yang telah dikembangkan.

5.3.2.1 Hasil User Management Functions

Tabel 5.24 Hasil User Management Functions

Kategori Fungsi	Hasil yang diharapkan	Status		Kendala
		Sukses	Gagal	
Membuat Akun User	Admin berhasil membuat akun user baru	✓		-
Melihat Daftar Akun User	Admin berhasil melihat daftar akun user	✓		-
Mengupdate Password Akun User	Admin berhasil mengubah password akun user	✓		-
Menghapus Akun User	Admin berhasil menghapus akun user	✓		-
Melihat Informasi Akun User	User berhasil melihat informasi akunnya	✓		-

Berdasarkan tabel di atas, seluruh kategori fungsi berstatus sukses, yang berarti tingkat keberhasilannya sebesar 100%. Hal ini menandakan bahwa proses integrasi frontend hingga ke backend pada fungsi manajemen user berjalan dengan benar dan telah memenuhi ekspektasi dari fungsi yang tersedia pada website dashboard yang dikembangkan.

5.3.2.2 Hasil Token Functions

Tabel 5.25 Hasil Token Functions

Kategori Fungsi	Hasil yang diharapkan	Status		Kendala
		Berhasil	Gagal	
Login	User berhasil masuk ke dalam laman introduction, Admin berhasil masuk ke dalam laman user management	✓		-

Logout	Admin/User berhasil keluar dari website dan kembali ke halaman login	✓		-
Refresh Token	Admin/User tetap berhasil mengakses website dashboard ketika masa aktif token sudah usang	✓		-
Verifikasi Token	Admin/User tetap berhasil mengakses website dashboard meskipun telah keluar dari laman dalam kurun waktu masa aktif token	✓		-

Berdasarkan tabel di atas, seluruh kategori fungsi berstatus sukses, yang berarti tingkat keberhasilannya sebesar 100%. Hal ini menandakan bahwa proses integrasi frontend hingga ke backend pada fungsi token terkait otentikasi dan otorisasi berjalan dengan benar dan telah memenuhi ekspektasi dari fungsi yang tersedia pada website dashboard yang dikembangkan.

5.3.2.3 Hasil Kubernetes Functions

Tabel 5.26 Hasil Kubernetes Functions

Kategori Fungsi	Hasil yang diharapkan	Status		Kendala
		Berhasil	Gagal	
User berhasil mendapatkan daftar informasi pod Kubernetes miliknya	Muncul gambar topologi pada laman dashboard dengan status komponen stopped/running	✓		-
User berhasil mendapatkan daftar informasi deployment Kubernetes miliknya	Muncul gambar topologi pada laman dashboard dengan status komponen stopped/running	✓		-
User berhasil mendapatkan informasi log dari	Muncul informasi log pada setiap komponen pada sidebar bagian tab Logs	✓		-

pod Kubernetes miliknya				
User berhasil merestart deployment Kubernetes (komponen 5G) sebagai manajemen siklus hidup komponen	Notifikasi sukses proses restart yang dilakukan pada setiap komponen	✓		-
User berhasil mendapatkan informasi log dari komponen AMF	Muncul informasi log komponen AMF pada sidebar bagian tab Logs	✓		-
User berhasil mendapatkan informasi log dari komponen UPF	Muncul informasi log komponen UPF pada sidebar bagian tab Logs	✓		-
User berhasil mendapatkan informasi data deployment komponen AMF	Muncul informasi deployment status komponen AMF pada tombol decorator status	✓		-
User berhasil mendapatkan informasi data deployment komponen UPF	Muncul informasi deployment status komponen UPF pada tombol decorator status	✓		-
User berhasil melakukan perintah ping dan melihat hasilnya pada komponen UE	Perintah ping menghasilkan output berupa paket diterima dari server dituju	✓		-

User berhasil melakukan perintah cURL dan melihat hasilnya pada komponen UE	Perintah cURL menghasilkan output berupa paket ditampilkan dalam bentuk HTML code dari server dituju	✓		-
User berhasil mendapatkan informasi Key Performance Indicator (KPI) dari komponen UE	Muncul nilai-nilai KPI pada table value beserta dengan munculnya grafik monitoring yang bekerja secara real-time untuk memvisualisasikan nilai-nilai KPI tersebut	✓		-
User berhasil mendapatkan informasi protokol SCTP dari komponen RAN 5G (CU dan DU)	Munculnya informasi terkait protocol SCTP pada sidebar bagian tab Protocol	✓		-

Berdasarkan tabel di atas, seluruh kategori fungsi berstatus sukses, yang berarti tingkat keberhasilannya sebesar 100%. Hal ini menandakan bahwa proses integrasi frontend hingga ke backend pada fungsi manajemen kluster Kubernetes berjalan dengan benar dan telah memenuhi ekspektasi dari fungsi yang tersedia pada website dashboard yang dikembangkan.

Terlepas itu, adapun sedikit anomali yang terkadang muncul ketika menjalankan fungsi ini grafik topologi terkadang tidak tertampil dengan benar pada saat user pertama kali mengakses laman dashboard, dan hal itu dapat teratasi dengan menekan tombol refresh yang terletak di dalam menu Topology Graph yang tersedia pada lama dashboard.

5.3.2.4 Hasil Helm Chart Functions

Tabel 5.27 Hasil Helm Chart Function

Kategori Fungsi	Hasil yang diharapkan	Status		Kendala
		Berhasil	Gagal	
User berhasil mendapatkan	Muncul value pada kolom Current Value	✓		-

informasi value konfigurasi pada setiap komponen 5G (CU, DU, dan UE)	setelah user melakukan konfigurasi pada kolom Set Value			
User berhasil mengubah value konfigurasi pada setiap komponen 5G (CU, DU, dan UE)	Muncul notifikasi status berhasil konfigurasi sesaat setelah user mengkonfirmasi pengubahan value pada komponen yang dikonfigurasi	✓	-	-
User berhasil menjalankan setiap komponen 5G (CU, DU, dan UE) pada manajemen siklus hidup komponen	Notifikasi sukses pada proses start yang dilakukan pada setiap komponen beserta berubahnya warna frame menjadi hijau pada komponen tersebut	✓	-	-
User berhasil menghentikan setiap komponen 5G (CU, DU, dan UE) pada manajemen siklus hidup komponen	Notifikasi sukses pada proses stop yang dilakukan pada setiap komponen beserta berubahnya warna frame menjadi kuning pada komponen tersebut	✓	-	-

Berdasarkan tabel di atas, seluruh kategori fungsi berstatus sukses, yang berarti tingkat keberhasilannya sebesar 100%. Hal ini menandakan bahwa proses integrasi frontend hingga ke backend pada fungsi manajemen Helm Chart berjalan dengan benar dan telah memenuhi ekspektasi dari fungsi yang tersedia pada website dashboard yang dikembangkan.

5.3.2.5 Hasil Wireshark Functions

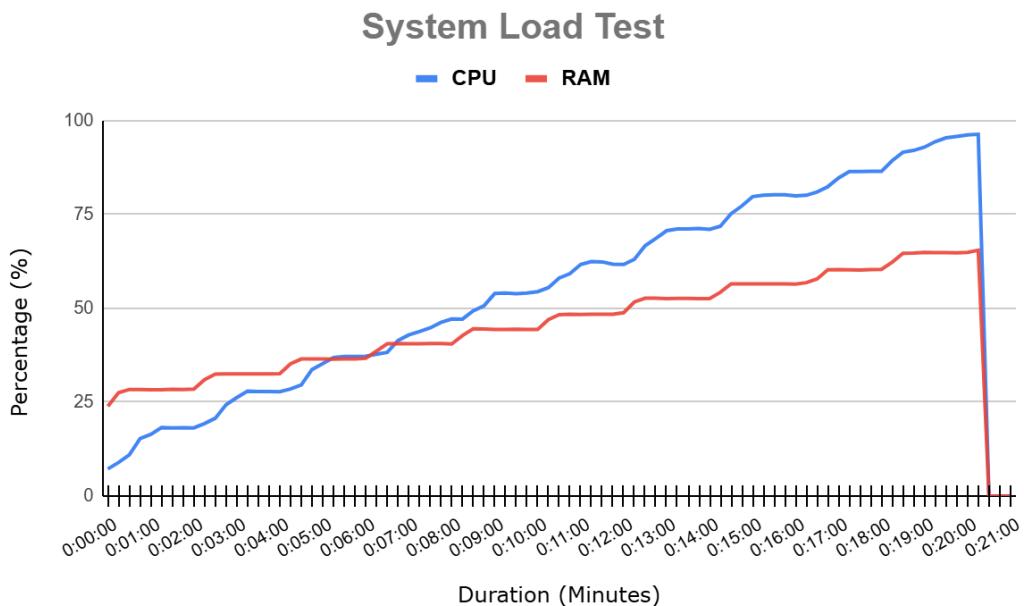
Tabel 5.28 Hasil Wireshark Functions

Kategori Fungsi	Hasil yang diharapkan	Status		Kendala
		Berhasil	Gagal	
User berhasil mendapatkan informasi paket data dari hasil <i>sniffing</i> pada setiap komponen CU, DU, dan UE serta berhasil menyimpan file PCAP pada database	Muncul informasi paket data hasil sniffing di tab Sniff pada menu Wireshark, Notifikasi sukses ketika file PCAP berhasil disimpan pada database	✓	-	-
User berhasil mendapatkan informasi daftar file PCAP yang telah tersedia pada database	Muncul daftar file PCAP di tab Files pada menu Wireshark	✓	-	-
User berhasil men-download file PCAP yang diinginkan	Muncul pop up hasil download pada browser setelah menekan tombol download	✓	-	-
User berhasil menghapus file PCAP yang diinginkan dari database	Muncul notifikasi sukses untuk penghapusan file PCAP yang diinginkan	✓	-	-

Berdasarkan tabel di atas, seluruh kategori fungsi berstatus sukses, yang berarti tingkat keberhasilannya sebesar 100%. Hal ini menandakan bahwa proses integrasi frontend hingga ke backend pada fungsi Wireshark berjalan dengan benar dan telah memenuhi ekspektasi dari fungsi yang tersedia pada website dashboard yang dikembangkan.

Terlepas itu, adapun sedikit anomali yang terkadang muncul ketika menjalankan fungsi ini berkaitan dengan tampilan data yang bersifat real-time. Data tersebut terkadang tidak tertampil dengan benar pada saat proses sniffing berjalan, dan hal itu akan menjadi rencana improvisasi pengembangan untuk kedepannya.

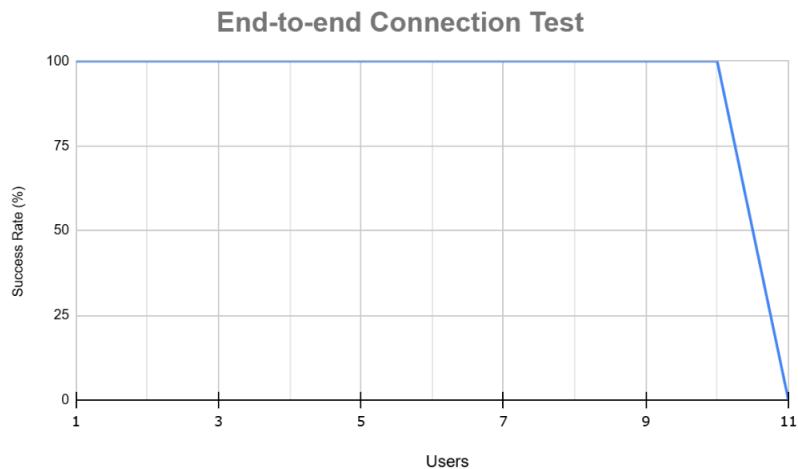
5.3.3 Hasil *Performance Testing*



Gambar 5.68 Grafik System Load Capacity

Berdasarkan grafik pada Gambar 5.68 ditunjukkan pengaruh peningkatan waktu terhadap persentase beban pada CPU dan RAM. Dengan meningkatnya durasi waktu, maka presentase beban yang diperoleh CPU dan RAM juga meningkat. Hal ini ditunjukkan dengan meningkatnya beban setiap menit karena menggunakan metode *Ramp Up Time testing* yang akan meningkatkan 1 *user* per menit ketika kondisi masih terpenuhi.

Pada menit ke-11, CPU dan RAM mencapai beban tertingginya sehingga dimulai *user* 11 sudah tidak dapat terkoneksi ke 5G core. Sedangkan pada menit ke 10, *user* masih dapat terkoneksi ke 5G core dan mendapatkan alamat IP serta mengakses internet. Referensi grafik dapat dilihat pada Gambar 5.69.



Gambar 5.69 Grafik End-to-End Connection

Berdasarkan grafik pada Gambar 5.69 ditunjukkan pengaruh jumlah *user* terhadap *success rate end-to-end connection*. Dapat dilihat pada *user* 1 hingga 10 memiliki *success rate* 100%, artinya UE sudah mendapatkan alamat IP dan sudah bisa terhubung ke internet. Sedangkan dimulai pada *user* 11, UE sudah tidak bisa terkoneksi *end-to-end* yang dipengaruhi oleh System Load Resource. Akibatnya, dimulai dari *user* 11 UE sudah tidak mendapatkan alamat IP sehingga tidak dapat mengakses internet.

Pada pengujian *performance testing* yang dilakukan seperti *System Load Test* dan *End-to-End Connection*, dapat dianalisa bahwa klaster mampu membuat lebih dari 1 koneksi 5G *end-to-end*. Berdasarkan grafik, klaster mampu menangani hingga 10 *user*. Artinya, klaster dapat menghandle 10 *end-to-end connection* secara bersamaan dengan parameter-parameter RAN yang berbeda.

Berdasarkan Tabel 5.12, dapat diambil data bahwa penggunaan *resource* per satu *user* adalah 2781mCPU dan 1965MiB RAM. Dengan mempertimbangkan toleransi 5%, kebutuhan CPU dan RAM per user meningkat menjadi 2920mCPU dan 2063MiB RAM. Penambahan toleransi ini memastikan bahwa sistem memiliki kapasitas yang cukup untuk menangani lonjakan beban sementara. Kemudian, angka-angka tersebut dibulatkan agar lebih mudah digunakan dalam perencanaan sumber daya di masa depan. Hasil akhir dapat dilihat pada Tabel 5.29.

Tabel 5.29 Total Kebutuhan Per User

Resource	Kebutuhan Per User (KPU)	Toleransi 5%	Pembulatan KPU
CPU	2781 mCPU	2920 mCPU	3000 mCPU
RAM	1965 MiB	2063 MiB	2100 MiB

Tabel 5.29 menunjukkan hasil angka akhir setelah dilakukan pembulatan. Berdasarkan angka ini, pengguna dapat menentukan alokasi resource yang digunakan dengan menggunakan rumus berikut ini:

$$Total\ CPU = (Total\ user \times KPU) \quad (5.2)$$

$$Total\ RAM = (Total\ user \times KPU) \quad (5.3)$$

Jika ingin menentukan jumlah *user* yang dapat digunakan oleh sebuah *resource*, dapat menggunakan rumus berikut ini:

$$Total\ user = \frac{(Total\ resource - Idle\ resource)}{KPU} \quad (5.4)$$

Rumus dapat di kalkulasikan pada *resource* CPU ataupun RAM. Agar sistem dapat berjalan dengan baik, hasil dari rumus *total user* sebaiknya dilakukan pembulatan ke bawah dan ambil nilai terkecil dari hasil perhitungan total user berdasarkan KPU CPU atau KPU RAM.

Berdasarkan spesifikasi yang diimplementasikan pada Tabel 4.1, total penggunaan sumber daya dari klaster Kubernetes yang digunakan adalah 40 CPU dan 56 GiB RAM. Berdasarkan hasil pengujian kapasitas beban sistem dan koneksi *end-to-end*, klaster kami mampu menangani hingga 10 *user*. Dengan demikian, jika dihitung menggunakan rumus yang telah dijelaskan dan dikurangi dengan *resource idle system*, hasil perhitungan ini telah terbukti akurat. Rencana pengembangan berkelanjutan memerlukan optimasi *resource* untuk setiap komponen yang telah dijalankan, terutama untuk komponen dengan kebutuhan *resource* tinggi seperti DU dan UE. Optimasi ini sangat penting untuk memastikan penggunaan CPU dan RAM yang lebih efisien, sehingga dapat mendukung lebih banyak *user* dan meningkatkan skalabilitas aplikasi.

5.3.4 Hasil User Acceptance Testing

5.3.4.1 Hasil Functional Testing

Pada pelaksanaan UAT ini, *user* mencoba semua fitur pada *dashboard* seperti *configuration panel* untuk mengkonfigurasi tiap komponen 5G, *topology graph* untuk melihat apakah tiap komponen 5G sudah berhasil terkoneksi, dan fitur-fitur lainnya. Pengujian ini berfungsi untuk mengetahui apakah fitur sudah berjalan sempurna atau belum dari sisi *user*. Dapat dilihat kembali pada tabel 5.14, semua fitur pada halaman *dashboard* admin berhasil berfungsi seperti yang diharapkan. Admin dapat membuat akun *user* baru, mengganti *password* akun *user*, dan menghapus akun *user* tanpa ada kendala ataupun *error*.

Pada halaman *user*, hampir semua fitur berhasil berfungsi secara sempurna. Masih ada beberapa fitur yang terdapat *bug* atau dijalankan secara bersamaan mengalami *error*. Fitur tersebut yaitu fitur *protocol stack* dan *logs* pada *sidebar* tiap komponen di *topology graph*. Fitur ini berhasil berfungsi menampilkan *protocol stack* dan *logs* tiap komponen, tapi terkadang hasil yang ditampilkan hanya ‘null’. Selain itu, ada fitur *sniffing wireshark* yang hanya bisa melakukan *sniffing* paket data satu komponen saja. Hal ini disebabkan oleh *load* yang tinggi di VM di mana kami men-develop *dashboard* tersebut. Semua fitur dapat berjalan sempurna ketika hanya diakses oleh satu *user* saja. Namun ketika banyak *user* mengakses *dashboard* secara bersamaan, beberapa fitur berjalan tidak sempurna.

Fitur *sniffing* pada wireshark membutuhkan waktu yang lama dikarenakan proses *sniffing* berpusat pada *backend*. Proses *sniffing* ini lama dikarenakan memproses data yang telah difilter. Untuk mengatasi masalah ini, proses *sniffing* dipindah ke pod tiap pengguna. Lalu metode *sniffing* yang digunakan pun berbeda. Sebelumnya metode yang digunakan yaitu semua data hasil *sniffing* diambil terlebih dahulu, baru setelah itu data tersebut difilter sesuai dengan protokol yang diinginkan. Metode yang digunakan sekarang yaitu data difilter terlebih dahulu, baru setelah itu data hasil *sniffing* ditampilkan.

5.3.4.2 Hasil Usability Testing

Usability testing digunakan untuk dapat mengetahui dan mengevaluasi pengalaman pengguna ketika menggunakan *dashboard*. Oleh karena itu, pengujian ini menggunakan kuesioner SUS yang diisi oleh pengguna setelah mencoba semua fitur pada dashboard. Dari data yang didapat, dilakukan analisis perhitungan sesuai dengan ketentuan analisis data SUS. Hasil perhitungan data SUS dari UAT 1 mendapatkan nilai rata-rata 65,42. Nilai ini masih di bawah nilai minimal SUS yaitu 68. Namun masih dalam kategori “OK” atau dapat diterima, meskipun perlu perbaikan hingga mencapai nilai minimal.

Terdapat beberapa pertanyaan yang mendapatkan nilai rendah pada kuesioner SUS ini. Dari pertanyaan bernilai rendah ini, kita dapat mengetahui hal-hal yang perlu ditingkatkan yaitu *user* merasa masih banyak hal yang tidak konsisten, *user* masih mengalami hambatan, *user* butuh panduan dan perlu membiasakan diri terlebih dahulu dalam menggunakan *dashboard*.

Pada kuesioner SUS juga terdapat kolom kesan dan saran yang diisi oleh *user*. Berikut adalah kesimpulan dari pendapat dan saran *user* terkait aplikasi:

1. Perlu ditingkatkan dan dioptimalkan fitur yang masih belum berfungsi dengan baik
2. Tampilan UI masih belum responsive terhadap perubahan ukuran layar

Sehingga dapat disimpulkan pada pengujian UAT 1 ini masih terdapat beberapa fitur yang belum berjalan dengan sempurna seperti yang dijelaskan pada sub bab Hasil *Functional Testing*. Selain itu juga tampilan UI masih belum responsive. *Feedback* dari pengguna membantu kami untuk mengetahui apa saja yang perlu diperbaiki dan ditingkatkan pada *dashboard*.

Setelah melakukan beberapa perbaikan pada *dashboard* berdasarkan *feedback* dan pengalaman pengguna pada UAT 1, selanjutnya kami melaksanakan UAT 2 untuk mengetahui *feedback* pengguna terhadap aplikasi kami yang sudah diperbaiki. Nilai akhir dari data perhitungan SUS di UAT 2 ini yaitu 77,1. Skor SUS yang didapat mengalami peningkatan dan berhasil di atas nilai minimal 68. Skor SUS pada UAT 2 ini juga masuk ke kategori *good to excellent*.

Dari enam pertanyaan tambahan, dapat disimpulkan bahwa fitur-fitur pada dashboard kami dapat menambah pengetahuan dan *skill* pengguna mengenai 5G RAN. Nilai rata-rata tiap pertanyaan adalah ≥ 4 dari skala 1-5 di mana nilai 1 berarti “Sangat Tidak Setuju” dan nilai 5 berarti “Sangat Setuju”. Pengguna rata-rata setuju dengan enam pertanyaan tersebut. Analisis perhitungan untuk enam pertanyaan tambahan ada di lampiran 5.7

Tabel 5.30 Nilai Rata-Rata Tiap Pertanyaan

No	Pertanyaan	Nilai Rata-Rata
1	Saya merasa fitur <i>Graphical User Interface</i> (GUI) pada ORCA <i>dashboard website</i> memudahkan saya dalam konfigurasi komponen RAN 5G (CU, DU, dan UE)	4,4
2	<i>Lab guidance</i> dan materi Open RAN 5G pada halaman Introduction membantu saya dalam memahami Open RAN 5G dan parameter-parameter konfigurasi 5G RAN	4,4

3	Visualisasi <i>topology graph</i> membantu saya memahami arsitektur 5G	4,5
4	Saya dapat mempelajari protokol apa saja pada jaringan 5G dan dapat menganalisis paket data jaringan melalui fitur Wireshark	4,2
5	Saya dapat memahami <i>Key Performance Indicator</i> (KPI) apa saja yang terdapat pada komponen UE pada fitur monitoring	4,1
6	Seberapa besar anda akan merekomendasikan ORCA <i>dashboard website</i> ini kepada orang lain?	4,7

5.4 Kesimpulan

Penelitian ini menyoroti beberapa tantangan utama dalam infrastruktur teknologi jaringan 5G konvensional yang mempengaruhi efisiensi, fleksibilitas, dan reliabilitas sistem. Dari aspek manajemen, pengelolaan perangkat secara terpisah meningkatkan risiko *human error* dan kesalahan konfigurasi sehingga mengurangi efisiensi dan reliabilitas. Pada aspek kontrol, keterbatasan dalam pengumpulan metrik dan pemantauan anomali perangkat secara *real-time* menimbulkan kendala signifikan terhadap efisiensi sistem secara keseluruhan. Selain itu, baik perusahaan maupun institusi pendidikan menghadapi keterbatasan sumber daya yang menghambat upaya untuk mengadopsi dan memahami teknologi 5G secara efektif dalam skala kecil untuk keperluan penelitian dan pendidikan.

Untuk mengatasi permasalahan ini, ORCA (Open RAN Configuration Application) diperkenalkan sebagai platform pelatihan berbasis *dashboard* simulasi yang membantu pengguna dalam memahami dan mengkonfigurasi komponen 5G RAN secara sistematis dengan berbagai tingkatan yang meningkatkan pemahaman kasus penggunaan. ORCA menyediakan fitur tambahan seperti monitoring dan *sniffing* yang membantu dalam pemahaman dan konfigurasi komponen-komponen pada 5G RAN. Sistem ORCA terdiri dari empat subsistem, yaitu *frontend*, *backend*, Infrastruktur, dan 5G, dengan fitur-fitur utama seperti grafik topologi, fitur *sniffing*, konfigurasi komponen 5G, dan *user management*.

Pengujian sistem yang dilakukan terdiri dari API *testing*, *End-to-End (E2E) testing*, *performance testing*, dan *user acceptance testing*. API *testing* bertujuan untuk memastikan bahwa setiap WebSocket API dan REST API berjalan sesuai dengan yang diharapkan. *End-to-End testing* bertujuan untuk memastikan bahwa seluruh alur aplikasi, mulai dari *frontend* hingga *backend*, bekerja secara harmonis dan sesuai dengan kebutuhan pengguna akhir. Lalu, *performance testing* adalah langkah penting dalam mengukur dan menilai batas kapasitas

keseluruhan aplikasi serta performanya di bawah *load* tinggi. Sedangkan UAT dilakukan untuk menilai kepuasan pengguna terhadap aplikasi dan memastikan aplikasi memenuhi kebutuhan mereka.

Berdasarkan API *testing*, pengujian dilakukan menggunakan Postman dan Visual Studio Code dengan hasil yang diharapkan dari pengujian ini adalah semua API berfungsi sesuai spesifikasi, interaksi antar modul tidak menghasilkan kesalahan, dan sistem mampu menangani input yang tidak valid dengan tepat. Hasil yang didapatkan setelah pengujian API *testing* adalah semua API pada *backend* dapat berfungsi sesuai dengan respon yang diharapkan.

Selanjutnya untuk *End-to-End testing* dilakukan menggunakan tampilan UI yang telah dikembangkan menggunakan web browser Google Chrome. Hasil yang diharapkan adalah keseluruhan alur aplikasi bekerja dengan harmonis sesuai dengan spesifikasi yang diinginkan. Dan hasil yang didapatkan adalah seluruh alur dapat berfungsi dengan baik dan benar sesuai dengan spesifikasi.

Kemudian dari sisi *performance testing* sesuai dengan implementasi spesifikasi yang sudah disediakan, dapat disimpulkan bahwa spesifikasi sumber daya yang telah disediakan mampu menampung lebih dari 1 koneksi E2E. Berdasarkan hasil pengujian, total user atau koneksi E2E yang dapat dibangun pada infrastruktur mampu mencapai 10 koneksi. Hasil benchmark menunjukkan rekomendasi penggunaan sumber daya yang dibutuhkan 1 *user* adalah 3000CPU dan 2400MiB RAM.

Pada pengujian *user acceptance testing*, terdiri dari *functional testing* and *usability testing*. Hasil *functional testing* pada UAT 1 menunjukkan hampir semua fitur berhasil berfungsi secara sempurna. Masih ada beberapa fitur yang terdapat *bug* atau dijalankan secara bersamaan mengalami *error*. Sedangkan *usability testing* pada UAT 1 memiliki nilai akhir 65,42 di mana nilai ini masih di bawah nilai standar SUS yaitu 68. Nilai ini masuk kategori “OK”, namun masih memerlukan perbaikan. Setelah melakukan perbaikan pada aplikasi dan melaksanakan UAT 2, nilai *usability testing* meningkat menjadi 77,1 dan masuk kategori *good to excellent*. Secara keseluruhan, *dashboard website* untuk mensimulasikan komponen 5G ini berguna untuk pengguna lebih memahami jaringan 5G.

Dari hasil beberapa pengujian yang telah dilakukan, dapat disimpulkan bahwa dashboard yang dikembangkan dari sisi *frontend* dan *backend* sudah berfungsi dengan baik. Dashboard ini mampu menangani hingga 10 user dengan artian mampu menangani hingga 10 koneksi 5G E2E yang berbeda dengan akumulasi. Kemudian berdasarkan pengujian UAT yang

langsung dihadapkan kepada user, hampir semua fitur sudah berjalan dengan baik. Dapat disimpulkan bahwa *dashboard* yang telah dibuat memenuhi segala pengujian dan mampu menjawab permasalahan yang ada.

Dashboard sistem simulasi yang telah dikembangkan tentunya masih memerlukan rencana pengembangan lanjutan. Hal ini ditinjau dari beberapa pengujian yang masih dapat ditingkatkan dari segala sisi. Sehingga diharapkan dari pengembangan *dashboard* ini dapat menjadi referensi dan bisa meningkatkan hasil pengujian yang telah dilakukan. Kemudian diharapkan hasil pengembangan *dashboard* ini dapat memajukan dunia akademik khususnya pada bidang telekomunikasi.

DAFTAR PUSTAKA

- [1] D. Scotece, A. Noor, L. Foschini, and A. Corradi, “5G-Kube: Complex Telco Core Infrastructure Deployment Made Low-Cost,” *IEEE Communications Magazine*, vol. 61, no. 7, pp. 26–30, Jul. 2023, doi: 10.1109/MCOM.006.2200693.
- [2] L. Bonati, M. Polese, S. D’Oro, S. Basagni, and T. Melodia, “Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead,” *Computer Networks*, vol. 182, p. 107516, Dec. 2020, doi: 10.1016/j.comnet.2020.107516.
- [3] T.-Y. Chou, J.-W. Hu, W.-Y. Huang, and T.-L. Liu, “ONOS-based System of TWAREN Virtual Dedicated Network Provisioning in Web UI,” in *2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, IEEE, May 2019, pp. 1–2. doi: 10.1109/ICCE-TW46550.2019.8991931.
- [4] O. Arouk and N. Nikaein, “Kube5G: A Cloud-Native 5G Service Platform,” in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, IEEE, Dec. 2020, pp. 1–6. doi: 10.1109/GLOBECOM42002.2020.9348073.
- [5] K. Cengiz and M. Aydemir, “Next-Generation infrastructure and technology issues in 5G systems,” *Journal of Communications Software and Systems*, vol. 14, no. 1, pp. 33–39, 2018, doi: 10.24138/jcomss.v14i1.422.
- [6] S. Barrachina-Muñoz, M. Payaró, and J. Mangues-Bafalluy, “Cloud-native 5G experimental platform with over-the-air transmissions and end-to-end monitoring,” pp. 1–6, Jul. 2022, [Online]. Available: <http://arxiv.org/abs/2207.11936>
- [7] K. Du, X. Wen, L. Wang, and T. T. Nguyen, “A Cloud-Native Based Access and Mobility Management Function Implementation in 5G Core,” *2020 IEEE 6th International Conference on Computer and Communications, ICCC 2020*, pp. 1251–1256, 2020, doi: 10.1109/ICCC51575.2020.9345262.
- [8] D. Lake, N. Wang, R. Tafazolli, and L. Samuel, “Softwarization of 5G Networks—Implications to Open Platforms and Standardizations,” *IEEE Access*, vol. 9, pp. 88902–88930, 2021, doi: 10.1109/ACCESS.2021.3071649.
- [9] “OAIBOX 5G LAB Manual.” Accessed: Oct. 31, 2023. [Online]. Available: <https://oaibox.com/5g-lab-manual/>

- [10] J. Brassil, “Investigating Integrated Access and Backhaul on the Aether 5G Testbed,” in 2021 IEEE 4th 5G World Forum (5GWF), IEEE, Oct. 2021, pp. 281–286. doi: 10.1109/5GWF52925.2021.00056.
- [11] “Runtime Operational Control (ROC).” Accessed: Oct. 31, 2023. [Online]. Available: <https://docs.aetherproject.org/aether-1.5/amp/roc.html>
- [12] O. Arouk and N. Nikaein, “5G Cloud-Native: Network Management & Automation,” in NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, IEEE, Apr. 2020, pp. 1–2. doi: 10.1109/NOMS47738.2020.9110392.
- [13] Telecomm Infra Project, “OpenRAN ROMA CD/CT OpenRAN Automation White-Paper,” 2023. Accessed: Dec. 09, 2023. [Online]. Available: <https://telecominfraproject.com/openran/>
- [14] J. , Rabcan and Vakhitova A, “MODERN FRAMEWORKS FOR WEB-APPLICATION DEVELOPMENT,” Ģylym žāne bilim, vol. 3, no. 3(72), pp. 33–40, Sep. 2023, doi: 10.52578/2305-9397-2023-3-2-33-40.
- [15] O-RAN ALLIANCE, “O-RAN Working Group 6 Cloudification and Orchestration Use Cases and Requirements for O-RAN Virtualized RAN.” Accessed: Dec. 09, 2023. [Online]. Available: <https://orandownloadsweb.azurewebsites.net/specifications>
- [16] “5G-EPICENTRE : Cloud-Native NetApps for Public Protection and Disaster Relief,” 2023. Accessed: Dec. 07, 2023. [Online]. Available: https://www.5gepicentre.eu/wp-content/uploads/2023/04/5G-EPICENTRE_D2.2_Cloud-Native-Infrastructure-v2.0-compressed.pdf
- [17] D. T. Nguyen and H. T. Nguyen, “Enhancing CNF performance for 5G core network using SR-IOV in Kubernetes,” in 2022 24th International Conference on Advanced Communication Technology (ICACT), IEEE, Feb. 2022, pp. 501–506. doi: 10.23919/ICACT53585.2022.9728817.
- [18] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, “A comprehensive survey of Network Function Virtualization,” Mar. 14, 2018, Elsevier B.V. doi: 10.1016/j.comnet.2018.01.021.
- [19] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, “VNF Placement Optimization at the Edge and Cloud †,” Future Internet, vol. 11, no. 3, p. 69, Mar. 2019, doi: 10.3390/fi11030069.

- [20] W. Attaoui, E. Sabir, H. Elbiaze, and M. Guizani, “VNF and CNF Placement in 5G: Recent Advances and Future Trends,” *IEEE Transactions on Network and Service Management*, 2023, doi: 10.1109/TNSM.2023.3264005.
- [21] J. Shen and J. Brower, “Access and Edge Network Architecture and Management,” in *Future Networks, Services and Management*, Cham: Springer International Publishing, 2021, pp. 157–183. doi: 10.1007/978-3-030-81961-3_5.
- [22] W. Attaoui, E. Sabir, H. Elbiaze, and M. Guizani, “VNF and CNF Placement in 5G: Recent Advances and Future Trends,” *IEEE Transactions on Network and Service Management*, vol. 20, no. 4, pp. 4698–4733, Dec. 2023, doi: 10.1109/TNSM.2023.3264005.
- [23] S. Troia, M. Savi, and G. Maier, “Performance characterization and profiling of chained CPU-bound Virtual Network Functions,” *Computer Networks*, vol. 231, p. 109815, Jul. 2023, doi: 10.1016/j.comnet.2023.109815.
- [24] P. Mandal, “Comparison of Placement Variants of Virtual Network Functions From Availability and Reliability Perspective,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 860–874, Jun. 2022, doi: 10.1109/TNSM.2022.3148006.
- [25] A. De Domenico, Y.-F. Liu, and W. Yu, “Optimal Virtual Network Function Deployment for 5G Network Slicing in a Hybrid Cloud Infrastructure,” *IEEE Trans Wirel Commun*, vol. 19, no. 12, pp. 7942–7956, Dec. 2020, doi: 10.1109/TWC.2020.3017628.
- [26] Q. Zhang, F. Liu, and C. Zeng, “Online Adaptive Interference-Aware VNF Deployment and Migration for 5G Network Slice,” *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2115–2128, Oct. 2021, doi: 10.1109/TNET.2021.3080197.
- [27] H. Taherdoost and M. Madanchian, “Multi-Criteria Decision Making (MCDM) Methods and Concepts,” *Encyclopedia*, vol. 3, no. 1, pp. 77–87, Jan. 2023, doi: 10.3390/encyclopedia3010006.
- [28] M. Gramaglia, P. Serrano, A. Banchs, G. Garcia-Aviles, A. Garcia-Saavedra, and R. Perez, “The case for serverless mobile networking,” in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 779–784.

- [29] P. D. Dutonde, “Website Development Technologies: A Review,” *Int J Res Appl Sci Eng Technol*, vol. 10, no. 1, pp. 359–366, Jan. 2022, doi: 10.22214/ijraset.2022.39839.
- [30] V. Thoutam, “A Study On Python Web Application Framework,” *Journal of Electronics, Computer Networking and Applied Mathematics*, no. 11, pp. 48–55, Sep. 2021, doi: 10.55529/jecnam.11.48.55.
- [31] B. Coll-Perales, M. C. Lucas-Estañ, and M. Sepulcre, “End-to-End V2X Latency Modeling and Analysis in 5G Networks,” *IEEE Trans Veh Technol*, vol. 72, no. 4, pp. 5094–5109, Apr. 2023, doi: 10.1109/TVT.2022.3224614.
- [32] Y. Hu, W. Yang, and Y. Wang, “Fuzzing Method Based on Selection Mutation of Partition Weight Table for 5G Core Network NGAP Protocol,” 2022, pp. 144–155. doi: 10.1007/978-3-030-79728-7_15.
- [33] C.-A. Shen, K.-C. Lu, and S.-Y. Tan, “A Programmable and FPGA-accelerated GTP Offloading Engine for Mobile Edge Computing in 5G Networks,” in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, Apr. 2019, pp. 1021–1022. doi: 10.1109/INFCOMW.2019.8845143.
- [34] E. Sarikaya and E. Onur, “Placement of 5G RAN Slices in Multi-tier O-RAN 5G Networks with Flexible Functional Splits,” in *2021 17th International Conference on Network and Service Management (CNSM)*, IEEE, Oct. 2021, pp. 274–282. doi: 10.23919/CNSM52442.2021.9615541.
- [35] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, “Kubernetes as an Availability Manager for Microservice Applications.”
- [36] N. Singh, A. Singh, and V. Rawat, “Deploying Jenkins, Ansible and Kubernetes to Automate Continuous Integration and Continuous Deployment Pipeline,” in *2022 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, IEEE, Dec. 2022, pp. 1–5. doi: 10.1109/SOLI57430.2022.10294378.
- [37] K. Gallaba, “Improving the Robustness and Efficiency of Continuous Integration and Deployment,” in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, Sep. 2019, pp. 619–623. doi: 10.1109/ICSME.2019.00099.
- [38] A. E. Nocentino and B. Weissman, “Kubernetes Architecture,” in *SQL Server on Kubernetes*, Berkeley, CA: Apress, 2021, pp. 53–70. doi: 10.1007/978-1-4842-7192-6_3.

- [39] V. Sharma, “Managing Multi-Cloud Deployments on Kubernetes with Istio, Prometheus and Grafana,” in 2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS), IEEE, Mar. 2022, pp. 525–529. doi: 10.1109/ICACCS54159.2022.9785124.
- [40] 3GPP 29.510, “5G System; Network function repository services; Stage 3,” 2017.
- [41] Rumah Coding, “Lingkungan Pengembangan Virtual (Virtual Environment) .” Accessed: May 05, 2024. [Online]. Available: <https://rumahcoding.id/belajar/python-fundamental/membuat-dan-mengelola-proyek-python/lingkungan-pengembangan-virtual/>
- [42] “Develop with PatternFly.” Accessed: May 05, 2024. [Online]. Available: <https://www.patternfly.org/get-started/develop/>

LAMPIRAN CD-1

LAMPIRAN CD-2

LAMPIRAN CD-3

LAMPIRAN CD-4

4.1

```
# Penambahan DNS setiap node
$ nano /etc/hosts
10.30.1.212 node1
10.30.1.213 node2
10.30.1.214 node3

# Generate dan Distribusi SSH Key
$ ssh-keygen
$ ssh-copy-id -i ~/.ssh/id_rsa.pub root@node1
$ ssh-copy-id -i ~/.ssh/id_rsa.pub root@node2
$ ssh-copy-id -i ~/.ssh/id_rsa.pub root@node3

# Menyiapkan dependencies
$ apt install python3-pip
$ git clone https://github.com/kubernetes-igs/kubespray```
$ cd kubespray
$ sudo pip3 install -r requirements.txt
$ cp -rfp inventory/sample inventory/mycluster

# Deklarasi IP setiap node
$ declare -a IPS=(10.30.1.212 10.30.1.213 10.30.1.214)
$ CONFIG_FILE=inventory/mycluster/hosts.yaml python3 contrib/inventory_builder/inventory.py
${IPS[@]}

# Deploy menggunakan ansible
$ ansible-playbook -i inventory/mycluster/hosts.yaml --become --become-user=root
cluster.yml
```

4.2

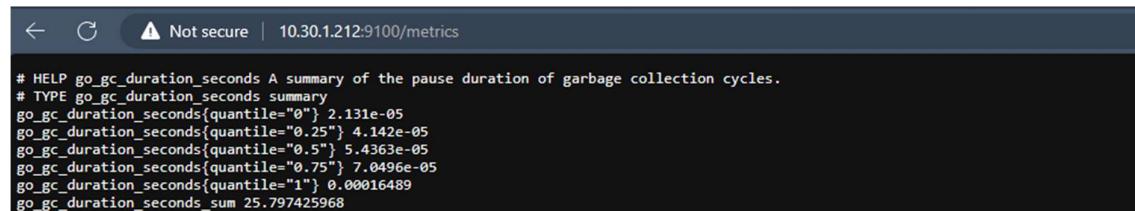
```
ari@node3:~$ kubectl get node
NAME     STATUS   ROLES      AGE     VERSION
node1    Ready    control-plane   179d   v1.28.3
node2    Ready    <none>       179d   v1.28.3
node3    Ready    <none>       145d   v1.28.3
ari@node3:~$ kubectl get all -n nginx-ingress
NAME                           READY   STATUS    RESTARTS   AGE
pod/nginx-ingress-b67db696c-rrbpk   1/1     Running   1 (29d ago)   32d

NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
service/nginx-ingress   LoadBalancer   10.233.31.191   10.30.1.234   80:31735/TCP,443:32008/TCP   155d

NAME          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-ingress   1/1     1           1           155d

NAME          DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-ingress-658bdcbc8   0         0         0         155d
replicaset.apps/nginx-ingress-85784b98df   0         0         0         155d
replicaset.apps/nginx-ingress-b67db696c   1         1         1         155d
```

4.3



The screenshot shows a browser window with the URL `10.30.1.212:9100/metrics`. The page displays a list of metrics, starting with a summary of garbage collection cycles:

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 2.131e-05
go_gc_duration_seconds{quantile="0.25"} 4.142e-05
go_gc_duration_seconds{quantile="0.5"} 5.4363e-05
go_gc_duration_seconds{quantile="0.75"} 7.0496e-05
go_gc_duration_seconds{quantile="1"} 0.00016489
go_gc_duration_seconds_sum 25.797425968
```

4.4

The screenshot shows the Prometheus Targets page. At the top, there are tabs for 'Alerts', 'Graph', 'Status', and 'Help'. Below the tabs is a search bar with the placeholder 'Filter by endpoint or labels'. A blue button labeled 'Collapse All' is visible. The main table has columns for 'Endpoint', 'State', 'Labels', 'Last Scrape', 'Scrape Duration', and 'Error'. There is one entry in the table:

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.16.0.68:9093/metrics	Up	<code>endpoint="alertmanager", instance="10.16.0.68:9093", job="stable-kube-prometheus-sta-alertmanager", namespace="monitoring", pod="alertmanager-stable-kube-prometheus-sta-alertmanager-0", service="stable-kube-prometheus-sta-alertmanager"</code>	10.109s ago	7.945ms	

4.5

The screenshot shows the Grafana login page. The title bar indicates 'Not secure' and the URL 'grafana.orca.edu/login'. The main content features the Grafana logo and the text 'Welcome to Grafana'. It includes fields for 'Email or username' and 'Password', a 'Log in' button, and a 'Forgot your password?' link.

4.6

The screenshot shows the Longhorn dashboard. The top navigation bar includes 'Dashboard', 'Node', 'Volume', 'Recurring Job', 'Backup', and 'Setting'. The main area displays three donut charts and associated data tables.

- Volumes:** Shows 8 Volumes. The data table below shows:

Healthy	4
Degraded	0
In Progress	4
Fault	0
Detached	0
- Storage Schedulable:** Shows 0 Bi. The data table below shows:

Schedulable	0 Bi
Reserved	45.3 Gi
Used	108 Gi
Disabled	0 Bi
Total	151 Gi
- Nodes:** Shows 2 Nodes. The data table below shows:

Schedulable	2
Unschedulable	0
Down	0
Disabled	0
Total	2

4.7

The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', and 'My Views'. Below this is a 'Build Queue' section stating 'No builds in the queue.' To the right is a main panel titled 'Dashboard' with a table showing project details:

S	W	Name	Last Success	Last Failure	Last Duration
		development	N/A	N/A	N/A
		infra	N/A	N/A	N/A
		production	N/A	N/A	N/A
		staging	N/A	N/A	N/A

At the bottom, there are 'Icon legend' and 'Atom feed' links.

4.8

The screenshot shows the Argo CD login interface. It features a large orange octopus character in the center of the page with the text 'Let's get stuff deployed!'. To the right is a form with fields for 'Username' and 'Password', and a 'SIGN IN' button. The Argo logo is at the top right.

4.9

The screenshot shows the Jenkins pipeline stage view for the 'production' pipeline. The pipeline consists of several stages: 'Setup Environment', 'Build Image', 'Push Image and delete locally', 'Change Image Tag on Github', 'ArgoCD Setup', 'ArgoCD Sync', and 'Declarative Post Actions'. The 'Setup Environment' stage took 4s, 'Build Image' took 10s, 'Push Image and delete locally' took 1s, 'Change Image Tag on Github' took 5s, 'ArgoCD Setup' took 4s, 'ArgoCD Sync' took 3s, and 'Declarative Post Actions' took 1s. The total average stage time was 38s. At the bottom, there's a 'Build History' section showing a single commit from July 12 at 07:55.

4.10

```
AN-OPEN-NETRA-INFRA > jasc > backend > staging > Jenkinsfile
 1 pipeline {
 2     agent {
 3         label 'vm4'
 4     }
 5     environment{
 6         GITHUB_ACCESS_TOKEN = credentials("github_access_token")
 7     }
 8     stages {
 9         stage('Setup Environment'){
10             steps{
11                 script{
12                     dir('AN-OPEN-NETRA-BE'){
13                         git branch: 'develop',
14                         //credentialsId: 'github_access_token',
15                         url: "https://github.com/whiwk/netra_backend.git"
16                     script{
17                         env.COMMIT_SHORT_SHA = sh (
18                             script: 'git log --pretty=format:\'%h\' -n 1',
19                             returnStdout: true
20                         ).trim()
21                     }
22                     sh "sed -i \"s/'HOST': '.*'/'HOST': 'backend-db.staging'/g\" netra_backend/settings.py"
23                     sh 'awk '/^DATABASES = /{if (!/^\\s*#/) print}' netra_backend/settings.py"
24
25                 }
26             }
27         }
28     }
29     stage('Build Image') {
30         steps {
31             script {
32                 dir('AN-OPEN-NETRA-BE') {
33                     pwd
34                     sh 'docker build -t quay.tiplab.local/openetra/be:${COMMIT_SHORT_SHA} .'
35                 }
36             }
37         }
38     }
39 }
```

4.11

```
from pathlib import Path
import environ
# Initialise environment variables
env = environ.Env()
# Define the base directory
BASE_DIR = Path(__file__).resolve().parent.parent
# Read the .env file
environ.Env.read_env(BASE_DIR / '.env')
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = env('SECRET_KEY')
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = env.bool('DEBUG', default=False)
ALLOWED_HOSTS = env.list('ALLOWED_HOSTS', default=['*'])
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
#####
    'apps',
    'rest_framework',
    'rest_framework_simplejwt',
    'rest_framework_simplejwt.token_blacklist',
```

```

        'corsheaders',
    ],
    REST_FRAMEWORK = {
        'DEFAULT_AUTHENTICATION_CLASSES': (
            'rest_framework_simplejwt.authentication.JWTAuthentication',
        ),
        'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
        'PAGE_SIZE': 10,
        'DEFAULT_THROTTLE_CLASSES': [
            'rest_framework.throttling.AnonRateThrottle',
            'rest_framework.throttling.UserRateThrottle'
        ],
        'DEFAULT_THROTTLE_RATES': {
            'anon': '100000/day',
            'user': '100000/day',
        },
        'DEFAULT_VERSIONING_CLASS': 'rest_framework.versioning.URLPathVersioning',
        'DEFAULT_RENDERER_CLASSES': [
            'rest_framework.renderers.JSONRenderer',
        ],
        'DEFAULT_PARSER_CLASSES': [
            'rest_framework.parsers.JSONParser',
            'rest_framework.parsers.FormParser',
            'rest_framework.parsers.MultiPartParser'
        ],
        'EXCEPTION_HANDLER': 'rest_framework.views.exception_handler',
    }
    from datetime import timedelta
    SIMPLE_JWT = {
        'ACCESS_TOKEN_LIFETIME': timedelta(minutes=env.int('ACCESS_TOKEN_LIFETIME')),
        'REFRESH_TOKEN_LIFETIME': timedelta(days=env.int('REFRESH_TOKEN_LIFETIME')),
    }
    MIDDLEWARE = [
        'corsheaders.middleware.CorsMiddleware',
        'django.middleware.security.SecurityMiddleware',
        'django.contrib.sessions.middleware.SessionMiddleware',
        'django.middleware.common.CommonMiddleware',
        'django.middleware.csrf.CsrfViewMiddleware',
        'django.contrib.auth.middleware.AuthenticationMiddleware',
        'django.contrib.messages.middleware.MessageMiddleware',
        'django.middleware.clickjacking.XFrameOptionsMiddleware',
        'apps.middleware.SingleSessionMiddleware',
    ]
    ROOT_URLCONF = 'netra_backend.urls'
    TEMPLATES = [
        {
            'BACKEND': 'django.template.backends.django.DjangoTemplates',
            'DIRS': [],
            'APP_DIRS': True,
            'OPTIONS': {
                'context_processors': [
                    'django.template.context_processors.debug',
                    'django.template.context_processors.request',
                    'django.contrib.auth.context_processors.auth',
                    'django.contrib.messages.context_processors.messages',
                ],
            },
        },
    ]
    WSGI_APPLICATION = 'netra_backend.wsgi.application'
    # Database
    DATABASES = {
        'default': {
            'ENGINE': env('DATABASE_ENGINE'),
            'NAME': env('DATABASE_NAME'),
            'USER': env('DATABASE_USER'),
            'PASSWORD': env('DATABASE_PASSWORD'),
            'HOST': env('DATABASE_HOST'),
            'PORT': env.int('DATABASE_PORT'),
        }
    }

```

```

}
# Password validation
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
# Internationalization
LANGUAGE_CODE = 'en-us'
TIME_ZONE = env('TIME_ZONE')    # CONFIGURED
USE_I18N = True
USE_TZ = True
# Static files (CSS, JavaScript, Images)
STATIC_URL = 'static/'
# Default primary key field type
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
CORS_ALLOW_ALL_ORIGINS = env.bool('CORS_ALLOW_ALL_ORIGINS', default=True)
CORS_ALLOW_CREDENTIALS = env.bool('CORS_ALLOW_CREDENTIALS', default=True)
CORS_ALLOW_METHODS = env.list('CORS_ALLOW_METHODS', default=['GET', 'POST', 'PUT', 'PATCH',
'DELETE', 'OPTIONS'])
CORS_ALLOW_HEADERS = env.list('CORS_ALLOW_HEADERS', default=['authorization', 'content-
type', 'x-csrftoken'])

```

4.12

```

from pathlib import Path
import environ
# Initialise environment variables
env = environ.Env()
# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
# Reading .env file
environ.Env.read_env(BASE_DIR / '.env')
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = env('SECRET_KEY')
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = env.bool('DEBUG', default=False)
ALLOWED_HOSTS = env.list('ALLOWED_HOSTS', default=['*'])
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'channels',
    'monitoring',
    'shell',
    'sniff',
    'protocolstack',
    'logs',
]
ASGI_APPLICATION = 'orca_backend_ws.asgi.application'
CHANNEL_LAYERS = {
    'default': {
        'BACKEND': 'channels_redis.core.RedisChannelLayer',

```

```

    'CONFIG': {
        "hosts": [(env('REDIS_HOST'), env.int('REDIS_PORT'))],
    },
},
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
ROOT_URLCONF = 'orca_backend_ws.urls'
TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [],
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
    },
},
]
WSGI_APPLICATION = 'orca_backend_ws.wsgi.application'
# Database
# https://docs.djangoproject.com/en/5.0/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / "db.sqlite3",
    }
}
# Password validation
# https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators
AUTH_PASSWORD_VALIDATORS = [
{
    'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
},
{
    'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
},
{
    'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
},
{
    'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
},
]
# Internationalization
# https://docs.djangoproject.com/en/5.0/topics/i18n/
LANGUAGE_CODE = 'en-us'
TIME_ZONE = env('TIME_ZONE')
USE_I18N = True
USE_TZ = True
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.0/howto/static-files/
STATIC_URL = 'static/'
# Default primary key field type
# https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

```

4.13

<pre>.orca_backend ├── apps/ │ ├── api/v1/ │ │ ├── kube │ │ ├── oai │ │ ├── pcap │ │ ├── shark │ │ ├── token │ │ └── user │ ├── management/ │ │ ├── commands │ │ │ └── data │ │ ├── admin.py │ │ ├── apps.py │ │ ├── kube_utils.py │ │ ├── middleware.py │ │ ├── models.py │ │ ├── serializers.py │ │ ├── tests.py │ │ └── views.py │ ├── deployment │ └── orca_backend ├── .env.example └── .gitignore</pre>	<pre>.orca_backend_ws ├── deployment ├── logs ├── monitoring ├── orca_backend_ws ├── protocolstack ├── shell ├── sniff ├── .env.example ├── .gitignore └── manage.py</pre>
---	--

4.14

<pre>#orca_backend amqp==5.2.0 appdirs==1.4.4 asgiref==3.7.2 async-timeout==4.0.3 attrs==23.2.0 autobahn==23.6.2 Automat==22.10.0 billiard==4.2.0 cachetools==5.3.2 carehare==1.0.2 celery==5.4.0 certifi==2024.2.2 cffi==1.16.0 channels==4.1.0 channels-rabbitmq==4.0.1 channels-redis==4.2.0 charset-normalizer==3.3.2 click==8.1.7 click-didyoumean==0.3.1 click-plugins==1.1.1 click-repl==0.3.0 constantly==23.10.4 cron-descriptor==1.4.3 cryptography==42.0.8 daphne==4.1.2 Django==5.0.2 django-celery-beat==2.6.0 django-celery-results==2.5.1 django-cors-headers==4.3.1 django-environ==0.11.2 django-timezone-field==6.1.0 django-restframework==3.14.0 djngorestframework-simplejwt==5.3.1 et-xmlfile==1.1.0 google-auth==2.28.1 hyperlink==21.0.0 idna==3.6 incremental==22.10.0</pre>	<pre>#orca_backend_ws asgiref==3.8.1 async-timeout==4.0.3 attrs==23.2.0 autobahn==23.6.2 Automat==22.10.0 cffi==1.16.0 channels==4.1.0 channels-redis==4.2.0 constantly==23.10.4 cryptography==42.0.8 daphne==4.1.2 Django==5.0.6 django-environ==0.11.2 hyperlink==21.0.0 idna==3.7 incremental==22.10.0 msgpack==1.0.8 psycopg2-binary==2.9.9 pyasn1==0.6.0 pyasn1_modules==0.4.0 pycparser==2.22 pyOpenSSL==24.1.0 redis==5.0.7 service-identity==24.1.0 six==1.16.0 sqlparse==0.5.0 Twisted==24.3.0 txaio==23.1.1 typing_extensions==4.12.2 websockets==12.0 zope.interface==6.4.post2</pre>
---	--

```

kombu==5.3.7
kubernetes==29.0.0
lxml==5.1.0
msgpack==1.0.8
numpy==2.0.0
oauthlib==3.2.2
openpyxl==3.1.4
packaging==23.2
pamqp==3.3.0
pandas==2.2.2
prompt_toolkit==3.0.47
psycopg2-binary==2.9.9
pyasn1==0.5.1
pyasn1-modules==0.3.0
pycparser==2.22
PyJWT==2.8.0
pyOpenSSL==24.1.0
pyshark==0.6
python-crontab==3.1.0
python-dateutil==2.8.2
pytz==2024.1
PyYAML==6.0.1
redis==5.0.6
requests==2.31.0
requests-oauthlib==1.3.1
rsa==4.9
service-identity==24.1.0
six==1.16.0
sqlparse==0.4.4
termcolor==2.4.0
Twisted==24.3.0
txaio==23.1.1
typing_extensions==4.10.0
tzdata==2024.1
urllib3==2.2.1
vine==5.1.0
wcwidth==0.2.13
websocket-client==1.7.0
websockets==12.0
zope.interface==6.4.post2

```

4.15

```

#orca_backend
from django.db import models
from django.conf import settings
from django.db.models.signals import post_save
from django.dispatch import receiver
import re
class UserProfile(models.Model):
    user = models.OneToOneField(settings.AUTH_USER_MODEL, on_delete=models.CASCADE,
    related_name='profile')
    level = models.IntegerField(default=1)
    completion = models.FloatField(default=0.0)
    session_key = models.CharField(max_length=40, blank=True, null=True)
    cu_matches = models.IntegerField(default=0)
    du_matches = models.IntegerField(default=0)
    ue_matches = models.IntegerField(default=0)
    def __str__(self):
        return self.user.username
class PcapFile(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    namespace = models.CharField(max_length=255) # Add namespace field
    filename = models.CharField(max_length=255)
    file_data = models.BinaryField() # Field to store binary data
    file_size = models.PositiveIntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=True)
    def __str__(self):

```

```

        return f"{self.filename} ({self.user.username})"
class CUConfig(models.Model):
    cuId = models.CharField(max_length=100)
    cellId = models.CharField(max_length=100)
    f1InterfaceIPadd = models.CharField(max_length=100)
    f1cuPort = models.CharField(max_length=100)
    f1duPort = models.CharField(max_length=100)
    n2InterfaceIPadd = models.CharField(max_length=100)
    n3InterfaceIPadd = models.CharField(max_length=100)
    mcc = models.CharField(max_length=100)
    mnc = models.CharField(max_length=100)
    tac = models.CharField(max_length=100)
    sst = models.CharField(max_length=100)
    amfhost = models.CharField(max_length=100)
    def __str__(self):
        return f"CU Config: {self.id}"
class DUConfig(models.Model):
    gnbId = models.CharField(max_length=100)
    duId = models.CharField(max_length=100)
    cellId = models.CharField(max_length=100)
    f1InterfaceIPadd = models.CharField(max_length=100)
    f1cuPort = models.CharField(max_length=100)
    f1duPort = models.CharField(max_length=100)
    mcc = models.CharField(max_length=100)
    mnc = models.CharField(max_length=100)
    tac = models.CharField(max_length=100)
    sst = models.CharField(max_length=100)
    usrp = models.CharField(max_length=100)
    cuHost = models.CharField(max_length=100)
    def __str__(self):
        return f"DU Config: {self.id}"
class UEConfig(models.Model):
    multusIPadd = models.CharField(max_length=100)
    rfsimServer = models.CharField(max_length=100)
    fullImsi = models.CharField(max_length=100)
    fullKey = models.CharField(max_length=100)
    opc = models.CharField(max_length=100)
    dnn = models.CharField(max_length=100)
    sst = models.CharField(max_length=100)
    sd = models.CharField(max_length=100)
    usrp = models.CharField(max_length=100)
    def __str__(self):
        return f"UE Config: {self.id}"
class UserConfiguration(models.Model):
    user = models.OneToOneField(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    cu_config = models.ForeignKey(CUConfig, on_delete=models.CASCADE)
    du_config = models.ForeignKey(DUConfig, on_delete=models.CASCADE)
    ue_config = models.ForeignKey(UEConfig, on_delete=models.CASCADE)
    def __str__(self):
        return f"{self.user.username}'s Configuration"
def get_config_by_user_number(model_class, user_number):
    try:
        return model_class.objects.get(id=user_number)
    except model_class.DoesNotExist:
        return None
def extract_user_number(username):
    match = re.match(r'user(\d+)', username)
    return int(match.group(1)) if match else None
@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def create_user_profile(sender, instance, created, **kwargs):
    if created:
        UserProfile.objects.create(user=instance)
        # Extract the user number from the username
        user_number = extract_user_number(instance.username)
        if user_number:
            # Get the configurations based on the user number
            cu_config = get_config_by_user_number(CUConfig, user_number)
            du_config = get_config_by_user_number(DUConfig, user_number)
            ue_config = get_config_by_user_number(UEConfig, user_number)
            if cu_config and du_config and ue_config:

```

```

        UserConfiguration.objects.create(user=instance, cu_config=cu_config,
du_config=du_config, ue_config=ue_config)
@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def save_user_profile(sender, instance, **kwargs):
    instance.profile.save()

```

4.16

```

#orca_backend
from django.contrib.auth.models import User
from django.contrib.auth.hashers import make_password
from rest_framework import serializers
from .models import UserProfile, PcapFile
class UserCreateSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['id', 'username', 'password']
        extra_kwargs = {'password': {'write_only': True}}
    def create(self, validated_data):
        validated_data['password'] = make_password(validated_data.get('password'))
        return super(UserCreateSerializer, self).create(validated_data)
class UserProfileSerializer(serializers.ModelSerializer):
    class Meta:
        model = UserProfile
        fields = ['level', 'completion', 'cu_matches', 'du_matches', 'ue_matches']
class UserListSerializer(serializers.ModelSerializer):
    profile = UserProfileSerializer(read_only=True)
    class Meta:
        model = User
        fields = ['id', 'username', 'profile']
class UserUpdateSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['username', 'password']
        extra_kwargs = {
            'password': {'write_only': True, 'required': False},
            'username': {'required': False}
        }
    def update(self, instance, validated_data):
        instance.username = validated_data.get('username', instance.username)
        password = validated_data.get('password')
        if password:
            instance.set_password(password)
        instance.save()
        return instance
class UserInformationSerializer(serializers.ModelSerializer):
    level = serializers.IntegerField(source='profile.level')
    completion = serializers.FloatField(source='profile.completion')
    cu_matches = serializers.IntegerField(source='profile.cu_matches')
    du_matches = serializers.IntegerField(source='profile.du_matches')
    ue_matches = serializers.IntegerField(source='profile.ue_matches')
    class Meta:
        model = User
        fields = ['username', 'level', 'completion', 'cu_matches', 'du_matches',
'ue_matches']
        class Meta:
            model = User
            fields = ['username', 'level', 'completion', 'cu_matches', 'du_matches',
'ue_matches']
class PcapFileSerializer(serializers.ModelSerializer):
    user = serializers.CharField(source='user.username', read_only=True) # Use username
instead of user ID
    class Meta:
        model = PcapFile
        fields = ['id', 'user', 'namespace', 'filename', 'file_size', 'created_at'] #
Include namespace field
        read_only_fields = ['id', 'user', 'namespace', 'filename', 'file_size',
'created_at']

```

4.17

```
from django.contrib.auth.models import User
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import IsAdminUser, IsAuthenticated
from rest_framework.response import Response
from rest_framework import status
from apps.serializers import UserCreateSerializer, UserListSerializer,
UserUpdateSerializer, UserInformationSerializer
from django.db import IntegrityError, transaction
from kubernetes import client, config
import re
from apps.models import UserProfile
from apps.kube_utils import get_role_yaml, get_role_binding_yaml
import subprocess
from apps.api.v1.oai.views import create_all_components, delete_all_components
import subprocess
@api_view(['POST'])
@permission_classes([IsAdminUser])
def modified_create_user(request):
    # Attempt to get 'number' from POST data; default to 0 if not found or invalid
    try:
        number_of_users = int(request.data.get('number', 0))
    except ValueError:
        return Response({'error': 'Please provide a valid integer value for "number".'},
status=status.HTTP_400_BAD_REQUEST)
    if number_of_users < 1:
        return Response({'error': 'Please provide a positive integer value.'},
status=status.HTTP_400_BAD_REQUEST)
    config.load_kube_config()
    api_instance = client.CoreV1Api()
    rbac_api_instance = client.RbacAuthorizationV1Api()
    try:
        user_count = max(number_of_users, 1)  # Ensure at least one user is created
        highest_number = 0
        created_users = []  # Track successfully created users
        # Find the highest existing username number
        for user in User.objects.filter(username__startswith='user'):
            match = re.match(r'user(\d+)', user.username)
            if match:
                number = int(match.group(1))
                highest_number = max(highest_number, number)
        # Start creating users from the next available number
        for i in range(highest_number + 1, highest_number + user_count + 1):
            username = f'user{i}'
            password = f'user{i}'
            namespace = f'user{i}'
            role_name = f'user{i}-role'
            role_binding_name = f'user{i}-rolebinding'
            if not User.objects.filter(username=username).exists():
                try:
                    with transaction.atomic():
                        new_user = User.objects.create_user(username=username,
password=password)
                        UserProfile.objects.get_or_create(user=new_user, defaults={'level':
1, 'completion': 0.0})
                        # Create the user's namespace
                        subprocess.run(["kubectl", "create", "namespace", namespace])
                        # Generate the role and role binding YAML
                        role_yaml = get_role_yaml(namespace, role_name)
                        role_binding_yaml = get_role_binding_yaml(namespace,
role_binding_name, username, role_name)
                        # Apply the role and role binding
                        subprocess.run(["kubectl", "apply", "-f", "-"],
input=role_yaml.encode('utf-8'))
                        subprocess.run(["kubectl", "apply", "-f", "-"],
input=role_binding_yaml.encode('utf-8'))
                        response = create_all_components(request, namespace)
                        if response != "Success":
                            return HttpResponse(response)
                except IntegrityError:
                    continue
    except Exception as e:
        print(f"An error occurred: {e}")
        return Response({'error': 'An error occurred during user creation.'},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

```

        created_users.append(username)
    except IntegrityError as e:
        return Response({'error': f"Failed to create {username}: {str(e)}"}, status=status.HTTP_400_BAD_REQUEST)
    return Response({'message': f"Users created successfully: {created_users}"}, status=status.HTTP_201_CREATED)
except Exception as e:
    return Response({'error': f"An error occurred: {str(e)}"}, status=status.HTTP_400_BAD_REQUEST)
@api_view(['GET'])
@permission_classes([IsAdminUser])
def list_users(request):
    users = User.objects.filter(is_staff=False)
    serializer = UserListSerializer(users, many=True)
    return Response(serializer.data)
@api_view(['PUT', 'PATCH'])
@permission_classes([IsAdminUser])
def update_user(request, pk):
    try:
        user = User.objects.get(pk=pk)
    except User.DoesNotExist:
        return Response({'error': 'User not found.'}, status=status.HTTP_404_NOT_FOUND)
    old_username = user.username
    serializer = UserUpdateSerializer(user, data=request.data, partial=True)  # partial=True allows for partial updates
    if serializer.is_valid():
        serializer.save()
        # Check if the username has been updated
        new_username = serializer.data.get('username')
        if old_username != new_username:
            try:
                # Run the subprocess command to update the Kubernetes namespace
                subprocess.run(['kubectl', 'rename', 'namespace', old_username, new_username], check=True)
            except subprocess.CalledProcessError as e:
                return Response({'error': f'Failed to update Kubernetes namespace: {str(e)}'}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
            return Response(serializer.data)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
@api_view(['DELETE'])
@permission_classes([IsAdminUser])
def delete_user(request, pk):
    config.load_kube_config()
    api_instance = client.CoreV1Api()
    try:
        user = User.objects.get(pk=pk)
        username = user.username

        # Derive namespace, role, and rolebinding names
        namespace = f"{username}"
        role_name = f"{username}-role"
        role_binding_name = f"{username}-rolebinding"
        # First, delete all components for the user
        response = delete_all_components(request, namespace)
        if response != "Success":
            return HttpResponse(response)
        # Then, proceed with deleting the Kubernetes resources
        try:
            subprocess.run(["kubectl", "delete", "rolebinding", role_binding_name, "--namespace", namespace], check=True)
            subprocess.run(["kubectl", "delete", "role", role_name, "--namespace", namespace], check=True)
            subprocess.run(["kubectl", "delete", "namespace", namespace], check=True)
        except subprocess.CalledProcessError as e:
            return Response({'error': f'Failed to delete Kubernetes resources: {e}'}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
        # Delete the user after cleaning up associated resources
        user.delete()
        return Response({'message': 'User and associated Kubernetes resources deleted successfully.'}, status=status.HTTP_204_NO_CONTENT)
    except User.DoesNotExist:

```

```

        return Response({'error': 'User not found.'}, status=status.HTTP_404_NOT_FOUND)
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_user_information(request):
    user = request.user
    serializer = UserInformationSerializer(user)
    return Response(serializer.data)
MATCH_INCREMENT = 100.0 / 33 # Increment per successful match
def update_completion(user):
    user_profile = UserProfile.objects.get(user=user)
    total_matches = user_profile.cu_matches + user_profile.du_matches +
    user_profile.ue_matches
    user_profile.completion = total_matches * MATCH_INCREMENT
    user_profile.save()
def compare_values(user, specific_values, db_values, component):
    matches = {key: specific_values[key] == db_values[key] for key in
    specific_values.keys()}
    match_count = sum(matches.values())
    user_profile = UserProfile.objects.get(user=user)
    if component == 'cu':
        user_profile.cu_matches = match_count
    elif component == 'du':
        user_profile.du_matches = match_count
    elif component == 'ue':
        user_profile.ue_matches = match_count
    user_profile.save()
    update_completion(user)
    if match_count == 0:
        return 'failure', matches
    elif match_count == len(matches):
        return 'success', matches
    else:
        return 'partial success', matches
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def compare_cu_config(request):
    user_namespace = f"{request.user.username}"
    command = ["helm", "get", "values", "single-cu", "--namespace", user_namespace]
    try:
        helm_output = subprocess.check_output(command, stderr=subprocess.STDOUT)
        values_yaml = helm_output.decode('utf-8')
        values_json = yaml.safe_load(values_yaml)
        specific_values = {
            'cuId': values_json.get('config', {}).get('cuId', ''),
            'cellId': values_json.get('config', {}).get('cellId', ''),
            'f1InterfaceIPadd': values_json.get('multus', {}).get('f1Interface',
            {}).get('IPadd', ''),
            'f1cuPort': values_json.get('config', {}).get('f1cuPort', ''),
            'f1duPort': values_json.get('config', {}).get('f1duPort', ''),
            'n2InterfaceIPadd': values_json.get('multus', {}).get('n2Interface',
            {}).get('IPadd', ''),
            'n3InterfaceIPadd': values_json.get('multus', {}).get('n3Interface',
            {}).get('IPadd', ''),
            'mcc': values_json.get('config', {}).get('mcc', ''),
            'mnc': values_json.get('config', {}).get('mnc', ''),
            'tac': values_json.get('config', {}).get('tac', ''),
            'sst': values_json.get('config', {}).get('sst', ''),
            'amfhost': values_json.get('config', {}).get('amfhost', '')
        }
        user = request.user
        try:
            user_configuration = UserConfiguration.objects.get(user=user)
        except UserConfiguration.DoesNotExist:
            return Response({"error": "User configuration not found"}, status=status.HTTP_404_NOT_FOUND)
        db_values = {
            'cuId': user_configuration.cu_config.cuId,
            'cellId': user_configuration.cu_config.cellId,
            'f1InterfaceIPadd': user_configuration.cu_config.f1InterfaceIPadd,
            'f1cuPort': user_configuration.cu_config.f1cuPort,
            'f1duPort': user_configuration.cu_config.f1duPort,

```

```

'n2InterfaceIPadd': user_configuration.cu_config.n2InterfaceIPadd,
'n3InterfaceIPadd': user_configuration.cu_config.n3InterfaceIPadd,
'mcc': user_configuration.cu_config.mcc,
'mnc': user_configuration.cu_config.mnc,
'tac': user_configuration.cu_config.tac,
'sst': user_configuration.cu_config.sst,
'amfhost': user_configuration.cu_config.amfhost
}
status, matches = compare_values(user, specific_values, db_values, 'cu')
return JsonResponse({'status': status, 'matches': matches})
except subprocess.CalledProcessError as e:
    return JsonResponse({'error': 'Failed to retrieve Helm release values', 'details': e.output.decode('utf-8')}, status=500)
except Exception as e:
    return JsonResponse({'error': 'An unexpected error occurred', 'details': str(e)}, status=500)
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def compare_du_config(request):
    user_namespace = f'{request.user.username}'
    command = ["helm", "get", "values", "single-du", "--namespace", user_namespace]
    try:
        helm_output = subprocess.check_output(command, stderr=subprocess.STDOUT)
        values_yaml = helm_output.decode('utf-8')
        values_json = yaml.safe_load(values_yaml)
        specific_values = {
            'gnbId': values_json.get('config', {}).get('gnbId', ''),
            'duId': values_json.get('config', {}).get('duId', ''),
            'cellId': values_json.get('config', {}).get('cellId', ''),
            'flInterfaceIPadd': values_json.get('multus', {}).get('flInterface',
            {}).get('IPadd', ''),
            'flcuPort': values_json.get('config', {}).get('flcuPort', ''),
            'flduPort': values_json.get('config', {}).get('flduPort', ''),
            'mcc': values_json.get('config', {}).get('mcc', ''),
            'mnc': values_json.get('config', {}).get('mnc', ''),
            'tac': values_json.get('config', {}).get('tac', ''),
            'sst': values_json.get('config', {}).get('sst', ''),
            'usrp': values_json.get('config', {}).get('usrp', ''),
            'cuHost': values_json.get('config', {}).get('cuHost', '')
        }
        user = request.user
        try:
            user_configuration = UserConfiguration.objects.get(user=user)
        except UserConfiguration.DoesNotExist:
            return Response({"error": "User configuration not found"}, status=status.HTTP_404_NOT_FOUND)

        db_values = {
            'gnbId': user_configuration.du_config.gnbId,
            'duId': user_configuration.du_config.duId,
            'cellId': user_configuration.du_config.cellId,
            'flInterfaceIPadd': user_configuration.du_config.flInterfaceIPadd,
            'flcuPort': user_configuration.du_config.flcuPort,
            'flduPort': user_configuration.du_config.flduPort,
            'mcc': user_configuration.du_config.mcc,
            'mnc': user_configuration.du_config.mnc,
            'tac': user_configuration.du_config.tac,
            'sst': user_configuration.du_config.sst,
            'usrp': user_configuration.du_config.usrp,
            'cuHost': user_configuration.du_config.cuHost
        }
        status, matches = compare_values(user, specific_values, db_values, 'du')
        return JsonResponse({'status': status, 'matches': matches})
    except subprocess.CalledProcessError as e:
        return JsonResponse({'error': 'Failed to retrieve Helm release values', 'details': e.output.decode('utf-8')}, status=500)
    except Exception as e:
        return JsonResponse({'error': 'An unexpected error occurred', 'details': str(e)}, status=500)
@api_view(['GET'])
@permission_classes([IsAuthenticated])

```

```

def compare_ue_config(request):
    user_namespace = f"{request.user.username}"
    command = ["helm", "get", "values", "single-ue", "--namespace", user_namespace]
    try:
        helm_output = subprocess.check_output(command, stderr=subprocess.STDOUT)
        values_yaml = helm_output.decode('utf-8')
        values_json = yaml.safe_load(values_yaml)
        specific_values = {
            'multusIPadd': values_json.get('multus', {}).get('ipadd', ''),
            'rfSimServer': values_json.get('config', {}).get('rfSimServer', ''),
            'fullImsi': values_json.get('config', {}).get('fullImsi', ''),
            'fullKey': values_json.get('config', {}).get('fullKey', ''),
            'opc': values_json.get('config', {}).get('opc', ''),
            'dnn': values_json.get('config', {}).get('dnn', ''),
            'sst': values_json.get('config', {}).get('sst', ''),
            'sd': values_json.get('config', {}).get('sd', ''),
            'usrp': values_json.get('config', {}).get('usrp', '')
        }
        user = request.user
        try:
            user_configuration = UserConfiguration.objects.get(user=user)
        except UserConfiguration.DoesNotExist:
            return Response({"error": "User configuration not found"}, status=status.HTTP_404_NOT_FOUND)
        db_values = {
            'multusIPadd': user_configuration.ue_config.multusIPadd,
            'rfSimServer': user_configuration.ue_config.rfSimServer,
            'fullImsi': user_configuration.ue_config.fullImsi,
            'fullKey': user_configuration.ue_config.fullKey,
            'opc': user_configuration.ue_config.opc,
            'dnn': user_configuration.ue_config.dnn,
            'sst': user_configuration.ue_config.sst,
            'sd': user_configuration.ue_config.sd,
            'usrp': user_configuration.ue_config.usrp
        }
        status, matches = compare_values(user, specific_values, db_values, 'ue')
        return JsonResponse({'status': status, 'matches': matches})
    except subprocess.CalledProcessError as e:
        return JsonResponse({'error': 'Failed to retrieve Helm release values', 'details': e.output.decode('utf-8')}, status=500)
    except Exception as e:
        return JsonResponse({'error': 'An unexpected error occurred', 'details': str(e)}, status=500)

```

4.18

```

from django.shortcuts import render, HttpResponseRedirect
from rest_framework.response import Response
from rest_framework.decorators import api_view
import subprocess
import yaml
import os
from apps.models import UserProfile
import json
from django.http import JsonResponse
from pathlib import Path
from rest_framework.permissions import IsAuthenticated
from rest_framework.decorators import api_view, permission_classes
from rest_framework import status
BASE_DIR = Path(__file__).resolve().parent.parent
SINGLE_CU_BASE_DIR = os.path.join(BASE_DIR, 'oai/oai-e2e/oai-cu/')
SINGLE_DU_BASE_DIR = os.path.join(BASE_DIR, 'oai/oai-e2e/oai-du/')
SINGLE_UE_BASE_DIR = os.path.join(BASE_DIR, 'oai/oai-e2e/oai-nr-ue/')
SINGLE_CU_DEPLOYMENT_DIR = os.path.join(BASE_DIR, 'oai/oai-e2e/oai-cu/templates/')
SINGLE_DU_DEPLOYMENT_DIR = os.path.join(BASE_DIR, 'oai/oai-e2e/oai-du/templates/')
SINGLE_UE_DEPLOYMENT_DIR = os.path.join(BASE_DIR, 'oai/oai-e2e/oai-nr-ue/templates/')
SINGLE_CU_VALUES_FILE_PATH = os.path.join(SINGLE_CU_BASE_DIR, 'values.yaml')
SINGLE_DU_VALUES_FILE_PATH = os.path.join(SINGLE_DU_BASE_DIR, 'values.yaml')
SINGLE_UE_VALUES_FILE_PATH = os.path.join(SINGLE_UE_BASE_DIR, 'values.yaml')

```

```

SINGLE_CU_CHART_FILE_PATH = os.path.join(SINGLE_CU_BASE_DIR, 'Chart.yaml')
SINGLE_DU_CHART_FILE_PATH = os.path.join(SINGLE_DU_BASE_DIR, 'Chart.yaml')
SINGLE_UE_CHART_FILE_PATH = os.path.join(SINGLE_UE_BASE_DIR, 'Chart.yaml')
SINGLE_CU_DEPLOYMENT_FILE_PATH = os.path.join(SINGLE_CU_DEPLOYMENT_DIR, 'deployment.yaml')
SINGLE_DU_DEPLOYMENT_FILE_PATH = os.path.join(SINGLE_DU_DEPLOYMENT_DIR, 'deployment.yaml')
SINGLE_UE_DEPLOYMENT_FILE_PATH = os.path.join(SINGLE_UE_DEPLOYMENT_DIR, 'deployment.yaml')
def update_chart_name(chart_file_path, username):
    with open(chart_file_path, 'r') as file:
        chart_yaml = yaml.safe_load(file)
    # Append the username to the existing chart name with a hyphen
    chart_yaml['name'] = f"{chart_yaml['name']}-{username}"
    with open(chart_file_path, 'w') as file:
        yaml.dump(chart_yaml, file)
def revert_chart_name(chart_file_path, original_name):
    with open(chart_file_path, 'r') as file:
        chart_yaml = yaml.safe_load(file)
    chart_yaml['name'] = original_name
    with open(chart_file_path, 'w') as file:
        yaml.dump(chart_yaml, file)
###CREATE ALL 5G COMPONENT NEEDED BY THE USER###
def create_all_components(request, namespace):
    original_names = {}
    # Define paths to your chart files (assuming these are defined elsewhere)
    chart_files = {
        "single_cu": SINGLE_CU_CHART_FILE_PATH,
        "single_du": SINGLE_DU_CHART_FILE_PATH,
        "single_ue": SINGLE_UE_CHART_FILE_PATH,
    }
    try:
        # Store original names and update with new names
        for key, chart_file_path in chart_files.items():
            with open(chart_file_path, 'r') as file:
                chart_yaml = yaml.safe_load(file)
            original_names[chart_file_path] = chart_yaml['name']
            update_chart_name(chart_file_path, namespace)
        # Define the Helm install commands
        helm_commands = {
            "single_cu": [SINGLE_CU_BASE_DIR, "single-cu"],
            "single_du": [SINGLE_DU_BASE_DIR, "single-du"],
            "single_ue": [SINGLE_UE_BASE_DIR, "single-ue"],
        }
        # Install Helm charts
        for key, value in helm_commands.items():
            base_dir, release_name = value
            subprocess.run([
                "helm", "install", release_name, "--values", "values.yaml",
                ".", "--namespace", namespace
            ], cwd=base_dir)

        # Define and scale deployments for each level
        leve1_deployments = ["cu", "du", "nr-ue"]
        for component in leve1_deployments:
            deployment_name = f"oai-{component}-leve1-{namespace}"
            subprocess.run([
                "kubectl", "scale", "deployment", deployment_name, "--replicas=0",
                "--namespace=" + namespace
            ])
        return "Success"
    except subprocess.CalledProcessError as e:
        return Response({"error": f"An error occurred: {e}"}, status=500)
    finally:
        # Revert the names back to original
        for chart_file_path, original_name in original_names.items():
            revert_chart_name(chart_file_path, original_name)
###DELETE ALL 5G COMPONENT ALONGSIDE USER ACCOUNT DELETION###
def delete_all_components(request, namespace):
    try:
        #SINGLE - CU
        subprocess.run([
            "helm", "delete", "single-cu", "--namespace", namespace
        ])

```

```

#SINGLE - DU
subprocess.run([
    "helm", "delete", "single-du", "--namespace", namespace
])
#SINGLE - UE
subprocess.run([
    "helm", "delete", "single-ue", "--namespace", namespace
])
return "Success"
except subprocess.CalledProcessError as e:
    # Handle errors in the subprocesses
    return f"An error occurred: {e}"
###SINGLE - CU##
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def values_single_cu(request):
    user_namespace = f"{request.user.username}"
    # Execute helm get values command
    command = ["helm", "get", "values", "single-cu", "--namespace", user_namespace]
    try:
        helm_output = subprocess.check_output(command, stderr=subprocess.STDOUT)
        values_yaml = helm_output.decode('utf-8')
        # Convert YAML to JSON
        values_json = yaml.safe_load(values_yaml)
        # Assuming 'values' is the top-level key as per your example (confirm this path in
        # your actual YAML structure).
        specific_values = {
            'cuid': values_json.get('config', {}).get('cuid', ''),
            'cellId': values_json.get('config', {}).get('cellId', ''),
            'f1InterfaceIPadd': values_json.get('multus', {}).get('f1Interface',
            {}).get('IPadd', ''),
            'f1cuPort': values_json.get('config', {}).get('f1cuPort', ''),
            'f1duPort': values_json.get('config', {}).get('f1duPort', ''),
            'n2InterfaceIPadd': values_json.get('multus', {}).get('n2Interface',
            {}).get('IPadd', ''),
            'n3InterfaceIPadd': values_json.get('multus', {}).get('n3Interface',
            {}).get('IPadd', ''),
            'mcc': values_json.get('config', {}).get('mcc', ''),
            'mnc': values_json.get('config', {}).get('mnc', ''),
            'tac': values_json.get('config', {}).get('tac', ''),
            'sst': values_json.get('config', {}).get('sst', ''),
            'amfhost': values_json.get('config', {}).get('amfhost', '')
        }
        return JsonResponse({'values': specific_values})
    except subprocess.CalledProcessError as e:
        return JsonResponse({'error': 'Failed to retrieve Helm release values',
        'details': e.output.decode('utf-8')}, status=500)
    except Exception as e:
        return JsonResponse({'error': 'An unexpected error occurred',
        'details': str(e)}, status=500)
###SINGLE - DU##
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def values_single_du(request):
    user_namespace = f"{request.user.username}"
    # Execute helm get values command
    command = ["helm", "get", "values", "single-du", "--namespace", user_namespace]
    try:
        helm_output = subprocess.check_output(command, stderr=subprocess.STDOUT)
        values_yaml = helm_output.decode('utf-8')
        # Convert YAML to JSON
        values_json = yaml.safe_load(values_yaml)  # Assumes PyYAML or similar package is
        used
        # Extract specific values
        specific_values = {
            'gnbId': values_json.get('config', {}).get('gnbId', ''),
            'duId': values_json.get('config', {}).get('duId', ''),
            'cellId': values_json.get('config', {}).get('cellId', ''),
            'f1InterfaceIPadd': values_json.get('multus', {}).get('f1Interface',
            {}).get('IPadd', ''),
            'f1cuPort': values_json.get('config', {}).get('f1cuPort', ''),
            'f1duPort': values_json.get('config', {}).get('f1duPort', ''),
            'n2InterfaceIPadd': values_json.get('multus', {}).get('n2Interface',
            {}).get('IPadd', ''),
            'n3InterfaceIPadd': values_json.get('multus', {}).get('n3Interface',
            {}).get('IPadd', ''),
            'mcc': values_json.get('config', {}).get('mcc', ''),
            'mnc': values_json.get('config', {}).get('mnc', ''),
            'tac': values_json.get('config', {}).get('tac', ''),
            'sst': values_json.get('config', {}).get('sst', ''),
            'amfhost': values_json.get('config', {}).get('amfhost', '')
        }
        return JsonResponse({'values': specific_values})
    except subprocess.CalledProcessError as e:
        return JsonResponse({'error': 'Failed to retrieve Helm release values',
        'details': e.output.decode('utf-8')}, status=500)
    except Exception as e:
        return JsonResponse({'error': 'An unexpected error occurred',
        'details': str(e)}, status=500)

```

```

        'f1duPort': values_json.get('config', {}).get('f1duPort', ''),
        'mcc': values_json.get('config', {}).get('mcc', ''),
        'mnc': values_json.get('config', {}).get('mnc', ''),
        'tac': values_json.get('config', {}).get('tac', ''),
        'sst': values_json.get('config', {}).get('sst', ''),
        'usrp': values_json.get('config', {}).get('usrp', ''),
        'cuHost': values_json.get('config', {}).get('cuHost', '')
    }
    return JsonResponse({'values': specific_values})
except subprocess.CalledProcessError as e:
    return JsonResponse({'error': 'Failed to retrieve Helm release values',
                        'details': e.output.decode('utf-8')}, status=500)
except Exception as e:
    return JsonResponse({'error': 'An unexpected error occurred',
                        'details': str(e)}, status=500)
###SINGLE - UE###
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def values_single_ue(request):
    user_namespace = f"{request.user.username}"
    # Execute helm get values command
    command = ["helm", "get", "values", "single-ue", "--namespace", user_namespace]
    try:
        helm_output = subprocess.check_output(command, stderr=subprocess.STDOUT)
        values_yaml = helm_output.decode('utf-8')
        # Convert YAML to JSON
        values_json = yaml.safe_load(values_yaml)  # Assumes PyYAML or similar package is used
        # Extract specific values
        specific_values = {
            'multusIPadd': values_json.get('multus', {}).get('ipadd', ''),
            'rfSimServer': values_json.get('config', {}).get('rfSimServer', ''),
            'fullImsi': values_json.get('config', {}).get('fullImsi', ''),
            'fullKey': values_json.get('config', {}).get('fullKey', ''),
            'opc': values_json.get('config', {}).get('opc', ''),
            'dnn': values_json.get('config', {}).get('dnn', ''),
            'sst': values_json.get('config', {}).get('sst', ''),
            'sd': values_json.get('config', {}).get('sd', ''),
            'usrp': values_json.get('config', {}).get('usrp', '')
        }
        return JsonResponse({'values': specific_values})
    except subprocess.CalledProcessError as e:
        return JsonResponse({'error': 'Failed to retrieve Helm release values',
                            'details': e.output.decode('utf-8')}, status=500)
    except Exception as e:
        return JsonResponse({'error': 'An unexpected error occurred',
                            'details': str(e)}, status=500)
###SINGLE - CU###
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def config_single_cu(request):
    username = request.user.username
    namespace = f"{username}"
    # Store the original name and update with the new name
    with open(SINGLE_CU_CHART_FILE_PATH, 'r') as file:
        chart_yaml = yaml.safe_load(file)
        original_name = chart_yaml['name']
    update_chart_name(SINGLE_CU_CHART_FILE_PATH, username)
    try:
        # Get current values
        get_values_command = ["helm", "get", "values", "single-cu", "--namespace",
                             namespace, "--output", "yaml"]
        current_values_yaml = subprocess.check_output(get_values_command).decode("utf-8")
        current_values = yaml.safe_load(current_values_yaml)
        # Iterate over expected fields and update if provided in JSON body
        expected_fields = {
            'cu_id': ['config', 'cuId'],
            'cell_id': ['config', 'cellId'],
            'f1_int': ['multus', 'f1Interface', 'IPadd'],
            'f1_cuport': ['config', 'f1cuPort'],
            'f1_duport': ['config', 'f1duPort'],
        }
        for field, paths in expected_fields.items():
            if field in current_values:
                for path in paths:
                    if path in current_values[field]:
                        current_values[field][path] = request.data.get(path, current_values[field][path])
        with open(SINGLE_CU_CHART_FILE_PATH, 'w') as file:
            yaml.safe_dump(current_values, file)
    except subprocess.CalledProcessError as e:
        return JsonResponse({'error': 'Failed to retrieve Helm release values',
                            'details': e.output.decode('utf-8')}, status=500)
    except Exception as e:
        return JsonResponse({'error': 'An unexpected error occurred',
                            'details': str(e)}, status=500)

```

```

        'n2_int': ['multus', 'n2Interface', 'IPadd'],
        'n3_int': ['multus', 'n3Interface', 'IPadd'],
        'mcc': ['config', 'mcc'],
        'mnc': ['config', 'mnc'],
        'tac': ['config', 'tac'],
        'sst': ['config', 'sst'],
        'amf_host': ['config', 'amfhost']
    }
    # Update the current_values based on the provided JSON data
    for field, path in expected_fields.items():
        value = request.data.get(field)
        if value:
            target = current_values
            for key in path[:-1]:
                target = target.setdefault(key, {})
            target[path[-1]] = value
    # Save the updated values to a YAML file
    updated_values_yaml = yaml.dump(current_values)
    with open('updated_values.yaml', 'w') as temp_file:
        temp_file.write(updated_values_yaml)
    # Execute Helm upgrade command with the updated values
    upgrade_command = [
        "helm", "upgrade", "single-cu", SINGLE CU BASE DIR,
        "--namespace", namespace,
        "-f", "updated_values.yaml"
    ]
    subprocess.run(upgrade_command, check=True)
    # Scale the deployment to 1 replica
    deployment_name = f"oai-cu-level1-{username}"
    subprocess.run([
        "kubectl", "scale", "deployment", deployment_name, "--replicas=1",
        "--namespace=" + namespace
    ], check=True)
    os.remove('updated_values.yaml')
    return Response({"message": "Configuration Updated Successfully"}, status=status.HTTP_200_OK)
except subprocess.CalledProcessError as e:
    return Response({"message": f"Error upgrading Helm chart: {str(e)}"}, status=status.HTTP_400_BAD_REQUEST)
finally:
    # Revert the chart name back to its original
    revert_chart_name(SINGLE CU CHART FILE PATH, original_name)
###SINGLE - DU###
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def config_single_du(request):
    username = request.user.username
    namespace = f"{username}"

    # Store the original name and update with the new name
    with open(SINGLE DU CHART FILE PATH, 'r') as file:
        chart_yaml = yaml.safe_load(file)
        original_name = chart_yaml['name']
    update_chart_name(SINGLE DU CHART FILE PATH, username)
    try:
        # Get current values
        get_values_command = ["helm", "get", "values", "single-du", "--namespace",
        namespace, "--output", "yaml"]
        current_values_yaml = subprocess.check_output(get_values_command).decode("utf-8")
        current_values = yaml.safe_load(current_values_yaml)
        # Dictionary to map JSON keys to their path in the YAML data structure
        field_mappings = {
            'gnb_id': ['config', 'gnbId'],
            'du_id': ['config', 'duId'],
            'cell_id': ['config', 'cellId'],
            'f1_int': ['multus', 'f1Interface', 'IPadd'],
            'f1_cuport': ['config', 'f1cuPort'],
            'f1_duport': ['config', 'f1duPort'],
            'mcc': ['config', 'mcc'],
            'mnc': ['config', 'mnc'],
            'tac': ['config', 'tac'],
        }
    
```

```

'sst': ['config', 'sst'],
'usrp': ['config', 'usrp'],
'cu_host': ['config', 'cuHost']
}
# Update the YAML data structure based on the provided JSON data
for field, path in field_mappings.items():
    value = request.data.get(field)
    if value:
        target = current_values
        for key in path[:-1]:
            target = target.setdefault(key, {})
        target[path[-1]] = value
# Save the updated values to a YAML file
updated_values_yaml = yaml.dump(current_values)
with open('updated_values.yaml', 'w') as temp_file:
    temp_file.write(updated_values_yaml)
# Execute Helm upgrade command with the updated values
upgrade_command = [
    "helm", "upgrade", "single-du", SINGLE_DU_BASE_DIR,
    "--namespace", namespace,
    "-f", 'updated_values.yaml'
]
subprocess.run(upgrade_command, check=True)
# Scale the deployment to 1 replica
deployment_name = f"oai-du-level1-{username}"
subprocess.run([
    "kubectl", "scale", "deployment", deployment_name, "--replicas=1",
    "--namespace=" + namespace
], check=True)
os.remove('updated_values.yaml')
return Response({"message": "Configuration Updated Successfully"}, status=status.HTTP_200_OK)
except subprocess.CalledProcessError as e:
    return Response({"message": f"Error upgrading Helm chart: {str(e)}"}, status=status.HTTP_400_BAD_REQUEST)
finally:
    # Revert the chart name back to its original
    revert_chart_name(SINGLE_DU_CHART_FILE_PATH, original_name)
###SINGLE - UE##
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def config_single_ue(request):
    username = request.user.username
    namespace = f"{username}"
    # Store the original name and update with the new name
    with open(SINGLE_UE_CHART_FILE_PATH, 'r') as file:
        chart_yaml = yaml.safe_load(file)
        original_name = chart_yaml['name']
    update_chart_name(SINGLE_UE_CHART_FILE_PATH, username)
    try:
        # Get current values
        get_values_command = ["helm", "get", "values", "single-ue", "--namespace",
        namespace, "--output", "yaml"]
        current_values_yaml = subprocess.check_output(get_values_command).decode("utf-8")
        current_values = yaml.safe_load(current_values_yaml)
        # Dictionary to map JSON keys to their path in the YAML data structure
        field_mappings = {
            'multus_ipadd': ['multus', 'ipadd'],
            'rfsimserver': ['config', 'rfSimServer'],
            'fullimsi': ['config', 'fullImsi'],
            'fullkey': ['config', 'fullKey'],
            'opc': ['config', 'opc'],
            'dnn': ['config', 'dnn'],
            'sst': ['config', 'sst'],
            'sd': ['config', 'sd'],
            'usrp': ['config', 'usrp']
        }
        # Update the YAML data structure based on the provided JSON data
        for field, path in field_mappings.items():
            value = request.data.get(field)
            if value:

```

```

        target = current_values
        for key in path[:-1]:
            target = target.setdefault(key, {})
        target[path[-1]] = value
    # Save the updated values to a YAML file
    updated_values_yaml = yaml.dump(current_values)
    with open('updated_values.yaml', 'w') as temp_file:
        temp_file.write(updated_values_yaml)
    # Execute Helm upgrade command with the updated values
    upgrade_command = [
        "helm", "upgrade", "single-ue", SINGLE_UE_BASE_DIR,
        "--namespace", namespace,
        "-f", 'updated_values.yaml'
    ]
    subprocess.run(upgrade_command, check=True)
    # Scale the deployment to 1 replica
    deployment_name = f"oai-nr-ue-level1-{username}"
    subprocess.run([
        "kubectl", "scale", "deployment", deployment_name, "--replicas=1",
        "--namespace=" + namespace
    ], check=True)
    os.remove('updated_values.yaml')
    return Response({"message": "Configuration Updated Successfully"}, status=status.HTTP_200_OK)
except subprocess.CalledProcessError as e:
    return Response({"message": f"Error upgrading Helm chart: {str(e)}"}, status=status.HTTP_400_BAD_REQUEST)
finally:
    # Revert the chart name back to its original
    revert_chart_name(SINGLE_UE_CHART_FILE_PATH, original_name)
###SINGLE CU - START###
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def start_single_cu(request):
    try:
        username = request.user.username # Get the currently logged-in user's username
        namespace = f"{username}" # Construct the namespace based on the username
        deployment_name = f"oai-cu-level1-{username}" # Dynamically create the deployment name
        subprocess.run([
            "kubectl", "scale", "deployment", deployment_name, "--replicas=1",
            "--namespace=" + namespace
        ])
        return Response({"message": "CU successfully started"}, status=status.HTTP_200_OK)
    except subprocess.CalledProcessError as e:
        return Response({"error": f"An error occurred: {e}"}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)

###SINGLE DU - START###
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def start_single_du(request):
    try:
        username = request.user.username # Get the currently logged-in user's username
        namespace = f"{username}" # Construct the namespace based on the username
        deployment_name = f"oai-du-level1-{username}" # Dynamically create the deployment name
        subprocess.run([
            "kubectl", "scale", "deployment", deployment_name, "--replicas=1",
            "--namespace=" + namespace
        ])
        return Response({"message": "DU successfully started"}, status=status.HTTP_200_OK)
    except subprocess.CalledProcessError as e:
        return Response({"error": f"An error occurred: {e}"}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
###SINGLE UE - START###
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def start_single_ue(request):
    try:
        username = request.user.username # Get the currently logged-in user's username

```

```

namespace = f"{username}" # Construct the namespace based on the username
deployment_name = f"oai-nr-ue-level1-{username}" # Dynamically create the deployment name
subprocess.run([
    "kubectl", "scale", "deployment", deployment_name, "--replicas=1",
    "--namespace=" + namespace
])
return Response({"message": "UE successfully started"}, status=status.HTTP_200_OK)
except subprocess.CalledProcessError as e:
    return Response({"error": f"An error occurred: {e}"}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
###SINGLE CU - STOP###
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def stop_single_cu(request):
    try:
        username = request.user.username # Get the currently logged-in user's username
        namespace = f"{username}" # Construct the namespace based on the username
        deployment_name = f"oai-cu-level1-{username}" # Dynamically create the deployment name
        subprocess.run([
            "kubectl", "scale", "deployment", deployment_name, "--replicas=0",
            "--namespace=" + namespace
        ])
        return Response({"message": "CU successfully stopped"}, status=status.HTTP_200_OK)
    except subprocess.CalledProcessError as e:
        return Response({"error": f"An error occurred: {e}"}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
    ###SINGLE DU - STOP###
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def stop_single_du(request):
    try:
        username = request.user.username # Get the currently logged-in user's username
        namespace = f"{username}" # Construct the namespace based on the username
        deployment_name = f"oai-du-level1-{username}" # Dynamically create the deployment name
        subprocess.run([
            "kubectl", "scale", "deployment", deployment_name, "--replicas=0",
            "--namespace=" + namespace
        ])
        return Response({"message": "DU successfully stopped"}, status=status.HTTP_200_OK)
    except subprocess.CalledProcessError as e:
        return Response({"error": f"An error occurred: {e}"}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
    ###SINGLE UE - STOP###
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def stop_single_ue(request):
    try:
        username = request.user.username # Get the currently logged-in user's username
        namespace = f"{username}" # Construct the namespace based on the username
        deployment_name = f"oai-nr-ue-level1-{username}" # Dynamically create the deployment name
        subprocess.run([
            "kubectl", "scale", "deployment", deployment_name, "--replicas=0",
            "--namespace=" + namespace
        ])
        return Response({"message": "UE successfully stopped"}, status=status.HTTP_200_OK)
    except subprocess.CalledProcessError as e:
        return Response({"error": f"An error occurred: {e}"}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)

```

4.19

```

from kubernetes import client, config
from rest_framework.decorators import api_view, permission_classes

```

```

from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
import json
from django.http import JsonResponse
import subprocess
from rest_framework import status
from apps.models import UserProfile
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_user_pods(request):
    try:
        config.load_kube_config()
        v1 = client.CoreV1Api()
        # Derive namespace from the current user's username
        user_namespace = f"{request.user.username}"
        # List pods in the specific namespace
        pods_list = v1.list_namespaced_pod(namespace=user_namespace)
        # Retrieve all network statuses in advance
        network_statuses = get_network_status_annotations()
        # Convert the pods_list to a list of dictionaries
        pods_info = []
        for pod in pods_list.items:
            # Ensure annotations are not None
            annotations = pod.metadata.annotations or {}
            # Retrieve the network status for the current pod
            network_status_json = network_statuses.get(pod.metadata.name, 'Not available')
            pod_info = {
                'name': pod.metadata.name,
                'ip': pod.status.pod_ip,
                'network_status': network_status_json, # Inserting network status in JSON
format
                'state': pod.status.phase,
                'namespace': pod.metadata.namespace,
                'node': pod.spec.node_name,
                # Add more fields as needed
            }
            pods_info.append(pod_info)
        # Return JsonResponse with the list of pod information
        return JsonResponse({'pods': pods_info})
    except client.rest.ApiException as e:
        return Response({"error": f"Failed to retrieve pods: {str(e)}"}, status=e.status)
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_user_deployments(request):
    config.load_kube_config()
    apps_v1 = client.AppsV1Api()
    namespace = f"{request.user.username}"
    try:
        deployments = apps_v1.list_namespaced_deployment(namespace)
        deployment_data = []
        for deployment in deployments.items:
            deployment_info = {
                "deployment_name": deployment.metadata.name,
                "namespace": deployment.metadata.namespace,
                "replicas": deployment.spec.replicas,
                "available_replicas": deployment.status.available_replicas,
                "ready_replicas": deployment.status.ready_replicas,
                "updated_replicas": deployment.status.updated_replicas,
                "strategy": deployment.spec.strategy.type,
                "conditions": [condition.type for condition in
deployment.status.conditions] if deployment.status.conditions else []
            }
            deployment_data.append(deployment_info)
        return Response(deployment_data)
    except client.rest.ApiException as e:
        return Response({"error": f"Failed to retrieve deployments: {str(e)}"}, status=e.status)
    ###RELATED TO 5G COMPONENTS MANAGEMENT FEATURE###
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def restart_single_cu(request):

```

```

try:
    # Derive the namespace from the current user's username
    namespace = f"{request.user.username}"
    deployment_name = f"oai-cu-level1-{namespace}" # Dynamically create the deployment
name
    result = subprocess.run(
        ["kubectl", "rollout", "restart", "deployment", deployment_name, "--namespace",
    namespace],
        capture_output=True, text=True
    )
    # If the subprocess call was successful, return a success response
    return Response({"message": "Deployment restarted successfully.", "details": result.stdout}, status=200)
except subprocess.CalledProcessError as e:
    # Return an error response if the subprocess call failed
    return Response({"error": "An error occurred while restarting the deployment.", "details": str(e)}, status=500)
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def restart_single_du(request):
    try:
        # Derive the namespace from the current user's username
        namespace = f"{request.user.username}"
        deployment_name = f"oai-du-level1-{namespace}" # Dynamically create the deployment
name
        result = subprocess.run(
            ["kubectl", "rollout", "restart", "deployment", deployment_name, "--namespace",
    namespace],
            capture_output=True, text=True
        )
        # If the subprocess call was successful, return a success response
        return Response({"message": "Deployment restarted successfully.", "details": result.stdout}, status=200)
    except subprocess.CalledProcessError as e:
        # Return an error response if the subprocess call failed
        return Response({"error": "An error occurred while restarting the deployment.", "details": str(e)}, status=500)
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def restart_single_ue(request):
    try:
        # Derive the namespace from the current user's username
        namespace = f"{request.user.username}"
        deployment_name = f"oai-nr-ue-level1-{namespace}" # Dynamically create the deployment
name
        result = subprocess.run(
            ["kubectl", "rollout", "restart", "deployment", deployment_name, "--namespace",
    namespace],
            capture_output=True, text=True
        )
        # If the subprocess call was successful, return a success response
        return Response({"message": "Deployment restarted successfully.", "details": result.stdout}, status=200)
    except subprocess.CalledProcessError as e:
        # Return an error response if the subprocess call failed
        return Response({"error": "An error occurred while restarting the deployment.", "details": str(e)}, status=500)
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_pod_logs(request, pod_name):
    namespace = f"{request.user.username}"
    try:
        cmd = [
            "kubectl", "logs", pod_name,
            "-n", namespace,
            "--tail=10", # Fetch the last 10 lines of logs
            "--timestamps"
        ]
        result = subprocess.run(cmd, capture_output=True, text=True, check=True)
        logs = result.stdout.strip().split('\n')
        structured_logs = []

```

```

        for line in logs:
            parts = line.split(maxsplit=1) # Split only on the first space
            if len(parts) == 2:
                timestamp_str, log_message = parts
                # Parse the timestamp and reformat it to show only the time
                timestamp = parse_kubernetes_timestamp(timestamp_str)
                time_only_str = timestamp.strftime('%H:%M:%S')
                structured_logs.append({"timestamp": time_only_str, "log": log_message})
        return JsonResponse(structured_logs, safe=False)
    except subprocess.CalledProcessError as e:
        error_message = f"Failed to fetch logs: {e.stderr}"
        return JsonResponse({"error": error_message}, status=400)
def get_pod_core(identifier, namespace="core-network"):
    try:
        cmd = ["kubectl", "get", "pods", "-n", namespace, "-o", "json"]
        result = subprocess.run(cmd, capture_output=True, text=True, check=True)
        pods = json.loads(result.stdout)
        for pod in pods['items']:
            pod_name = pod['metadata']['name']
            if identifier in pod_name:
                return pod_name
    return None
    except subprocess.CalledProcessError as e:
        raise RuntimeError(f"Failed to get pods: {e.stderr}")
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_amf_logs(request):
    try:
        pod_name = get_pod_core("amf")
        if not pod_name:
            return JsonResponse({"error": "AMF pod not found"}, status=404)
        logs = get_core_logs(pod_name)
        return JsonResponse(logs, safe=False)
    except RuntimeError as e:
        return JsonResponse({"error": str(e)}, status=400)
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_upf_logs(request):
    try:
        pod_name = get_pod_core("upf")
        if not pod_name:
            return JsonResponse({"error": "UPF pod not found"}, status=404)
        logs = get_core_logs(pod_name)
        return JsonResponse(logs, safe=False)
    except RuntimeError as e:
        return JsonResponse({"error": str(e)}, status=400)
def get_core_logs(pod_name, namespace="core-network"):
    try:
        cmd = [
            "kubectl", "logs", pod_name,
            "-n", namespace,
            "--tail=10", # Fetch the last 10 lines of logs
            "--timestamps"
        ]
        result = subprocess.run(cmd, capture_output=True, text=True, check=True)
        logs = result.stdout.strip().split('\n')
        structured_logs = []
        for line in logs:
            parts = line.split(maxsplit=1) # Split only on the first space
            if len(parts) == 2:
                timestamp_str, log_message = parts
                # Parse the timestamp and reformat it to show only the time
                timestamp = parse_kubernetes_timestamp(timestamp_str)
                time_only_str = timestamp.strftime('%H:%M:%S')
                structured_logs.append({"timestamp": time_only_str, "log": log_message})
    return structured_logs
    except subprocess.CalledProcessError as e:
        raise RuntimeError(f"Failed to fetch logs: {e.stderr}")
def get_deployment_core(identifier, namespace="core-network"):
    try:
        cmd = ["kubectl", "get", "deployments", "-n", namespace, "-o", "json"]

```

```

result = subprocess.run(cmd, capture_output=True, text=True, check=True)
deployments = json.loads(result.stdout)
filtered_deployments = []
for deployment in deployments['items']:
    deployment_name = deployment['metadata']['name']
    if identifier in deployment_name:
        replicas = deployment['status']['replicas']
        filtered_deployments.append({
            'name': deployment_name,
            'replicas': replicas
        })
return filtered_deployments
except subprocess.CalledProcessError as e:
    raise RuntimeError(f"Failed to get deployments: {e.stderr}")
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_amf_deployments(request):
    try:
        deployments = get_deployment_core("amf")
        return JsonResponse(deployments, safe=False)
    except RuntimeError as e:
        return JsonResponse({'error': str(e)}, status=500)
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_upf_deployments(request):
    try:
        deployments = get_deployment_core("upf")
        return JsonResponse(deployments, safe=False)
    except RuntimeError as e:
        return JsonResponse({'error': str(e)}, status=500)

```

4.20

```

from rest_framework_simplejwt.views import TokenObtainPairView
from apps.api.v1.token.serializers import MyTokenObtainPairSerializer
from rest_framework.permissions import IsAuthenticated
class MyTokenObtainPairView(TokenObtainPairView):
    serializer_class = MyTokenObtainPairSerializer

```

4.21

```

from django.urls import path
from . import views

urlpatterns = [
    path('create/', views.modified_create_user, name='create_user'),
    path('list/', views.list_users, name='list_users'),
    path('update/<int:pk>/', views.update_user, name='update_user'),
    path('delete/<int:pk>/', views.delete_user, name='delete_user'),
    path('information/', views.get_user_information, name='get_user_information'),
    path('compare/cu/', views.compare_cu_config, name='compare_cu_config'),
    path('compare/du/', views.compare_du_config, name='compare_du_config'),
    path('compare/ue/', views.compare_ue_config, name='compare_ue_config'),
]

```

4.22

```

from django.urls import path, include
from . import views
urlpatterns = [
    path('values_single_cu/', views.values_single_cu, name='values_single_cu'),
    path('values_single_du/', views.values_single_du, name='values_single_du'),
    path('values_single_ue/', views.values_single_ue, name='values_single_ue'),
    path('config_single_cu/', views.config_single_cu, name='config_single_cu'),
    path('config_single_du/', views.config_single_du, name='config_single_du'),
]

```

```

        path('config_single_ue/', views.config_single_ue, name='config_single_ue'),
        path('start_single_cu/', views.start_single_cu, name='start_single_cu'),
        path('start_single_du/', views.start_single_du, name='start_single_du'),
        path('start_single_ue/', views.start_single_ue, name='start_single_ue'),
        path('stop_single_cu/', views.stop_single_cu, name='stop_single_cu'),
        path('stop_single_du/', views.stop_single_du, name='stop_single_du'),
        path('stop_single_ue/', views.stop_single_ue, name='stop_single_ue'),
    ]

```

4.23

```

from rest_framework_simplejwt.serializers import TokenObtainPairSerializer
from django.contrib.sessions.models import Session
import uuid
from apps.models import UserProfile
class MyTokenObtainPairSerializer(TokenObtainPairSerializer):
    def validate(self, attrs):
        # First, call the superclass method that validates the user and returns the token
        data = super().validate(attrs)
        # Get the user object
        user = self.user
        # Generate a new session identifier
        new_session_id = str(uuid.uuid4())
        # Retrieve or create the user profile
        profile, created = UserProfile.objects.get_or_create(user=user)
        # Invalidate the previous session if it exists
        if profile.session_key:
            Session.objects.filter(session_key=profile.session_key).delete()
        # Update the user profile with the new session identifier
        profile.session_key = new_session_id
        profile.save()
        # Add custom claims to the token
        token = self.get_token(user) # This retrieves the already generated token
        token['is_staff'] = user.is_staff
        token['is_superuser'] = user.is_superuser
        token['session_id'] = new_session_id # Include the new session identifier in the
        token
        token['level'] = profile.level # Include the user profile level in the token
        # Return the token and additional data
        return {
            'access': str(token.access_token),
            'refresh': str(token),
            'session_id': new_session_id,
            'is_staff': user.is_staff,
            'is_superuser': user.is_superuser,
            'level': profile.level
        }

```

4.24

```

#Consumers Monitoring
import json
import asyncio
import subprocess
from channels.generic.websocket import AsyncWebsocketConsumer
class MonitoringConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.monitoring_task = None
        await self.accept()
    async def disconnect(self, close_code):
        if self.monitoring_task:
            self.monitoring_task.cancel()
    async def receive(self, text_data):
        data = json.loads(text_data)
        pod_name = data.get('pod_name')
        namespace = data.get('namespace')
        if not pod_name or not namespace:

```

```

        await self.send(text_data=json.dumps({'error': 'Invalid pod name or
namespace'})))
    return
    # Start the monitoring task
    self.monitoring_task = asyncio.create_task(self.monitor_logs(pod_name, namespace))
async def monitor_logs(self, pod_name, namespace):
    while True:
        try:
            command_list = ['kubectl', 'exec', '-it', pod_name, '-n', namespace, '--',
'cat', 'nrL1_UE_stats-0.log']
            process = await asyncio.create_subprocess_exec(
                *command_list,
                stdout=asyncio.subprocess.PIPE,
                stderr=asyncio.subprocess.PIPE
            )
            while True:
                line = await process.stdout.readline()
                if line:
                    await self.send(text_data=json.dumps({'monitoring_output':
line.decode('utf-8')}))
                else:
                    break
            await process.wait()
            # Sleep for a short duration before the next read to simulate `watch` behavior
            await asyncio.sleep(2)
        except asyncio.CancelledError:
            # Handle task cancellation
            if process:
                process.kill()
            break
#Consumers ProtocolStack
class ProtocolStackConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.sniffing_process = None
        await self.accept()

    async def disconnect(self, close_code):
        if self.sniffing_process:
            self.sniffing_process.kill()

    async def receive(self, text_data):
        data = json.loads(text_data)
        pod_name = data.get('pod_name')
        namespace = data.get('namespace')

        if not pod_name or not namespace:
            await self.send(text_data=json.dumps({'error': 'Invalid pod name or
namespace'})))
        return

        # Command to capture SCTP packets and parse them with tshark
        sniff_command = [
            'kubectl', 'exec', '-it', pod_name, '-n', namespace, '-c', 'tcpdump', '--',
            'sh', '-c', 'tshark -i f1 -Y "sctp" -T fields -e sctp.srcport -e sctp.dstport -
            e sctp.verification_tag -e sctp.assoc_index -e sctp.port -e sctp.checksum -e
            sctp.checksum.status -e sctp.chunk_type -e sctp.chunk_flags -e sctp.chunk_length -e
            sctp.parameter_type -e sctp.parameter_length -e sctp.parameter_heartbeat_information'
        ]

        self.sniffing_process = await asyncio.create_subprocess_exec(
            *sniff_command,
            stdout=asyncio.subprocess.PIPE,
            stderr=asyncio.subprocess.PIPE
        )

        while True:
            chunk = await self.sniffing_process.stdout.readline() # Read line by line
            if chunk:
                formatted_output = chunk.decode('utf-8').strip()
                await self.send(text_data=json.dumps({'data': formatted_output}))

```

```

        else:
            break

        await self.sniffing_process.wait()
        self.sniffing_process = None
#Consumers Shell
class ShellConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.command_process = None
        await self.accept()
    async def disconnect(self, close_code):
        if self.command_process:
            self.command_process.kill()
    async def receive(self, text_data):
        data = json.loads(text_data)
        pod_name = data.get('pod_name')
        namespace = data.get('namespace')
        command = data.get('command')
        if not pod_name or not namespace or not command:
            await self.send(text_data=json.dumps({'error': 'Invalid pod name, namespace, or
command'}))
        return
        if command == 'stop':
            if self.command_process:
                # Find the PID of the running process inside the pod and kill it
                find_pid_command = f"kubectl exec -it {pod_name} -n {namespace} -- pkill -f
'{self.current_command}'"
                subprocess.run(find_pid_command, shell=True)
                await self.send(text_data=json.dumps({'message': 'Command stopped'})))
        return
        self.current_command = command
        # Check if the command is 'ping' and if '-c' is missing
        if command.startswith('ping') and '-c' not in command:
            command += ' -c 4'
        command_list = ['kubectl', 'exec', '-it', pod_name, '-n', namespace, '--'] +
command.split()
        self.command_process = await asyncio.create_subprocess_exec(
            *command_list,
            stdout=asyncio.subprocess.PIPE,
            stderr=asyncio.subprocess.PIPE
        )
        while True:
            line = await self.command_process.stdout.readline()
            if line:
                await self.send(text_data=json.dumps({'command_output': line.decode('utf-
8')})))
            else:
                break
        await self.command_process.wait()
        self.command_process = None
#Consumers Sniff
class SniffConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.sniffing_process = None
        self.pcap_process = None
        self.pcap_filepath = None
        await self.accept()
        signal.signal(signal.SIGTERM, self.handle_termination)
        signal.signal(signal.SIGINT, self.handle_termination)

    async def disconnect(self, close_code):
        await self.cleanup()

    async def receive(self, text_data):
        data = json.loads(text_data)
        pod_name = data.get('pod_name')
        namespace_name = data.get('namespace')
        interface = data.get('interface')  # Get the interface from user input

        if not pod_name or not namespace_name:

```

```

        await self.send(text_data=json.dumps({'error': 'Invalid pod name or
namespace'})))
        return

    component = '-'.join(pod_name.split('-')[1:3])
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')

    # Include the interface in the pcap filename
    if interface:
        pcap_filename = f"{namespace_name}_{component}_{interface}_{timestamp}.pcap"
    else:
        pcap_filename = f"{namespace_name}_{component}_{timestamp}.pcap"

    pcap_directory = 'home'
    os.makedirs(pcap_directory, exist_ok=True)
    self.pcap_filepath = f'{pcap_directory}/{pcap_filename}'

    if 'oai-cu-level1' in pod_name:
        if interface not in ['f1', 'n2', 'n3']:
            await self.send(text_data=json.dumps({'error': 'Invalid interface for CU
component'}))
            return
        sniff_command = [
            'kubectl', 'exec', '-it', pod_name, '-n', namespace_name, '-c', 'tcpdump',
            '--',
            'tshark', '-i', interface, 'sctp or udp port 2152'
        ]
        pcap_command = [
            'kubectl', 'exec', '-it', pod_name, '-n', namespace_name, '-c', 'tcpdump',
            '--',
            'tshark', '-i', interface, '-w', self.pcap_filepath, 'sctp or udp port
2152'
        ]
    elif 'oai-du-level1' in pod_name:
        sniff_command = [
            'kubectl', 'exec', '-it', pod_name, '-n', namespace_name, '-c', 'tcpdump',
            '--',
            'tshark', '-i', 'f1', 'sctp or udp port 2152'
        ]
        pcap_command = [
            'kubectl', 'exec', '-it', pod_name, '-n', namespace_name, '-c', 'tcpdump',
            '--',
            'tshark', '-i', 'f1', '-w', self.pcap_filepath, 'sctp or udp port 2152'
        ]
    elif 'oai-nr-ue-level1' in pod_name:
        if interface == 'oaitun':
            sniff_command = [
                'kubectl', 'exec', '-it', pod_name, '-n', namespace_name, '-c',
                'tcpdump', '--',
                'tshark', '-i', 'oaitun_ue1'
            ]
            pcap_command = [
                'kubectl', 'exec', '-it', pod_name, '-n', namespace_name, '-c',
                'tcpdump', '--',
                'tshark', '-i', 'oaitun_ue1', '-w', self.pcap_filepath
            ]
        elif interface == 'net1':
            sniff_command = [
                'kubectl', 'exec', '-it', pod_name, '-n', namespace_name, '-c',
                'tcpdump', '--',
                'tshark', '-i', 'net1', 'sctp or udp port 2152'
            ]
            pcap_command = [
                'kubectl', 'exec', '-it', pod_name, '-n', namespace_name, '-c',
                'tcpdump', '--',
                'tshark', '-i', 'net1', '-w', self.pcap_filepath, 'sctp or udp port
2152'
            ]
        else:
            await self.send(text_data=json.dumps({'error': 'Invalid interface for UE
component'}))

```

```

        return
    else:
        await self.send(text_data=json.dumps({'error': 'Unsupported component type'}))

    self.sniffing_process = await asyncio.create_subprocess_exec(
        *sniff_command,
        stdout=asyncio.subprocess.PIPE,
        stderr=asyncio.subprocess.PIPE
    )

    self.pcap_process = await asyncio.create_subprocess_exec(
        *pcap_command,
        stdout=asyncio.subprocess.PIPE,
        stderr=asyncio.subprocess.PIPE
    )

    try:
        while True:
            chunk = await self.sniffing_process.stdout.readline()
            if chunk:
                formatted_output = chunk.decode('utf-8').strip()
                await self.send(text_data=json.dumps({'data': formatted_output}))
            else:
                break

        await self.sniffing_process.wait()
        self.sniffing_process = None

        await self.pcap_process.wait()
        self.pcap_process = None

    except asyncio.CancelledError:
        await self.cleanup()

    async def cleanup(self):
        if self.sniffing_process:
            self.sniffing_process.terminate()
            await self.sniffing_process.wait()
            self.sniffing_process = None

        if self.pcap_process:
            self.pcap_process.terminate()
            await self.pcap_process.wait()
            self.pcap_process = None

    def handle_termination(self, signum, frame):
        asyncio.create_task(self.cleanup())

```

4.25

```

#Routing Monitoring
from django.urls import path
from .consumers import MonitoringConsumer
websocket_urlpatterns = [
    path('ws/monitoring/', MonitoringConsumer.as_asgi()),
]

#Routing Protocol Stack
from django.urls import path
from .consumers import ProtocolStackConsumer
websocket_urlpatterns = [
    path('ws/protocolstack/', ProtocolStackConsumer.as_asgi()),
]

#Routing Shell
from django.urls import path
from .consumers import ShellConsumer
websocket_urlpatterns = [
    path('ws/shell/', ShellConsumer.as_asgi()),
]

```

```
#Routing Sniff
from django.urls import path
from .consumers import SniffConsumer
websocket_urlpatterns = [
    path('ws/sniff/', SniffConsumer.as_asgi()),
]
```

4.26

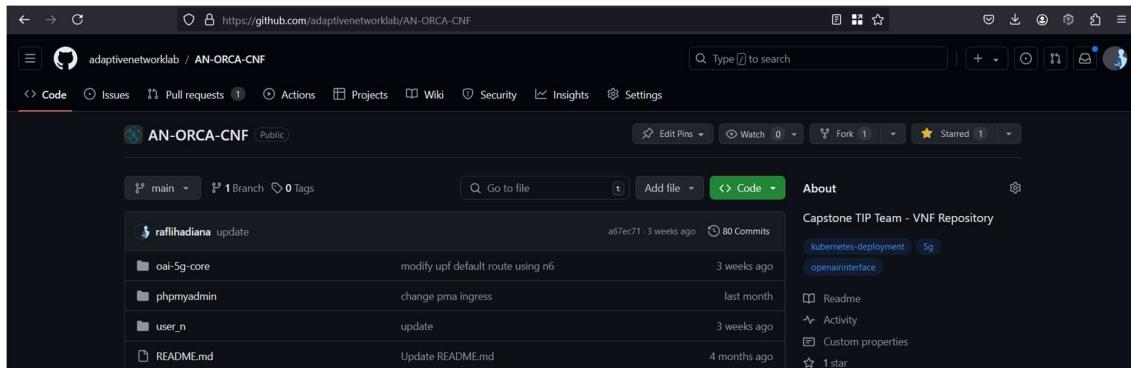
```
#Test Monitoring
import asyncio
import websockets
import json
async def test_websocket():
    uri = "ws://10.30.1.221:8020/ws/monitoring/"
    async with websockets.connect(uri) as websocket:
        # Send initial data to start the monitoring command
        message = json.dumps({
            'pod_name': 'oai-nr-ue-level1-user3-695c6c944c-vvj5x', # Replace with your
actual pod name
            'namespace': 'user3', # Replace with your actual namespace
        })
        await websocket.send(message)
        # Listen for messages from the server
        while True:
            try:
                response = await websocket.recv()
                data = json.loads(response)
                print(f"Received: {data}")
            except websockets.ConnectionClosed:
                print("Connection closed")
                break
# Replace 'your_pod_name' and 'your_namespace' with actual values
asyncio.get_event_loop().run_until_complete(test_websocket())
#Test Protocol Stack
import asyncio
import websockets
import json
async def test_websocket():
    uri = "ws://10.30.1.221:8020/ws/protocolstack/"
    async with websockets.connect(uri) as websocket:
        # Send initial data to start the tcpdump command
        message = json.dumps({
            'pod_name': 'oai-du-level1-user1-865645d8f4-xjg8r', # Replace with your actual
pod name
            'namespace': 'user1' # Replace with your actual namespace
        })
        await websocket.send(message)
        # Listen for messages from the server
        while True:
            try:
                response = await websocket.recv()
                response_data = json.loads(response)
                if 'data' in response_data:
                    print(response_data['data'])
                elif 'message' in response_data:
                    print(response_data['message'])
                elif 'error' in response_data:
                    print(response_data['error'])
            except websockets.ConnectionClosed:
                print("Connection closed")
                break
asyncio.get_event_loop().run_until_complete(test_websocket())
#Test Shell
import asyncio
import websockets
import json
async def test_websocket():
    uri = "ws://10.30.1.221:8020/ws/shell/"
```

```

        async with websockets.connect(uri) as websocket:
            # Send initial data to start the command
            message = json.dumps({
                'pod_name': 'oai-nr-ue-level1-user3-7ddf4d8466-ttz2s', # Replace with your
actual pod name
                'namespace': 'user3', # Replace with your actual namespace
                'command': 'curl google.com' # User-defined command
            })
            await websocket.send(message)
            # Listen for messages from the server
            while True:
                try:
                    response = await websocket.recv()
                    data = json.loads(response)
                    print(f"Received: {data}")
                except websockets.ConnectionClosed:
                    print("Connection closed")
                    break
# Replace 'your_pod_name' and 'your_namespace' with actual values
asyncio.get_event_loop().run_until_complete(test_websocket())
#Test Sniff
import asyncio
import websockets
import json
async def test_websocket():
    uri = "ws://10.30.1.221:8020/ws/sniff/"
    async with websockets.connect(uri) as websocket:
        # Send initial data to start the tcpdump command
        message = json.dumps({
            'pod_name': 'oai-nr-ue-level1-user1-85c76b9949-21kqk', # Replace with your
actual pod name
            'namespace': 'user1', # Replace with your actual namespace
            'interface': 'net1' # Uncomment and replace with actual interface if needed
(e.g., 'f1', 'n2', 'n3', 'oaitun', 'net1')
        })
        await websocket.send(message)
        # Listen for messages from the server
        while True:
            try:
                response = await websocket.recv()
                response_data = json.loads(response)
                if 'data' in response_data:
                    print(response_data['data'])
                elif 'message' in response_data:
                    print(response_data['message'])
            except websockets.ConnectionClosed:
                print("Connection closed")
                break
# Replace 'your_pod_name' and 'your_namespace' with actual values
asyncio.get_event_loop().run_until_complete(test_websocket())

```

4.27

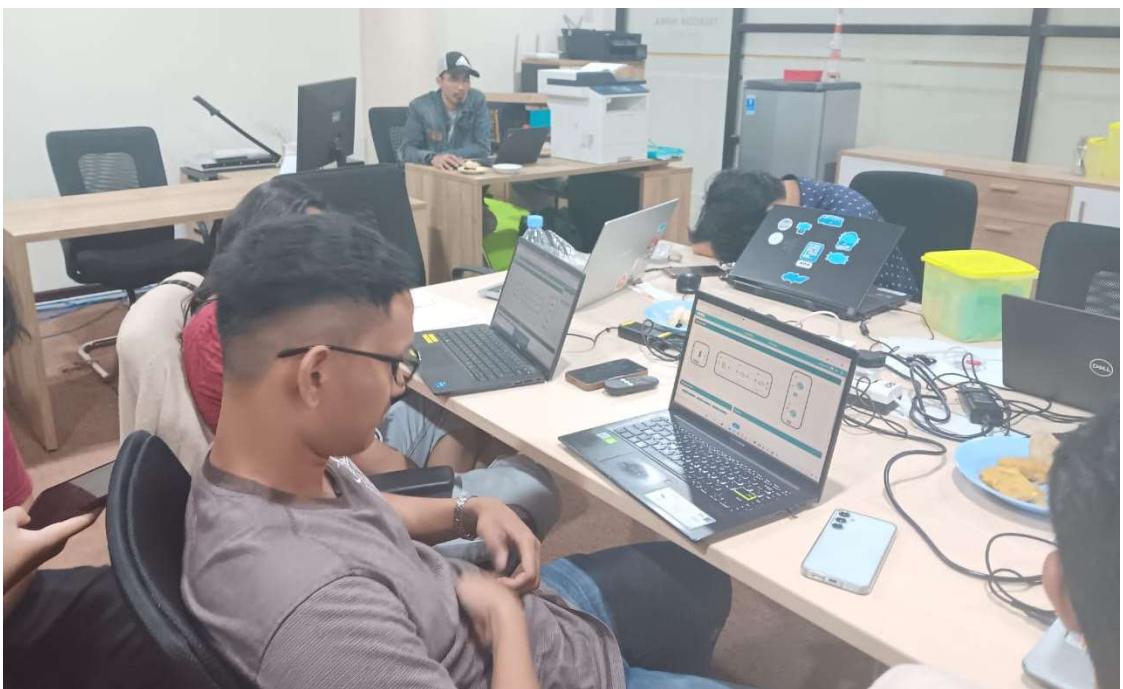


LAMPIRAN CD-5

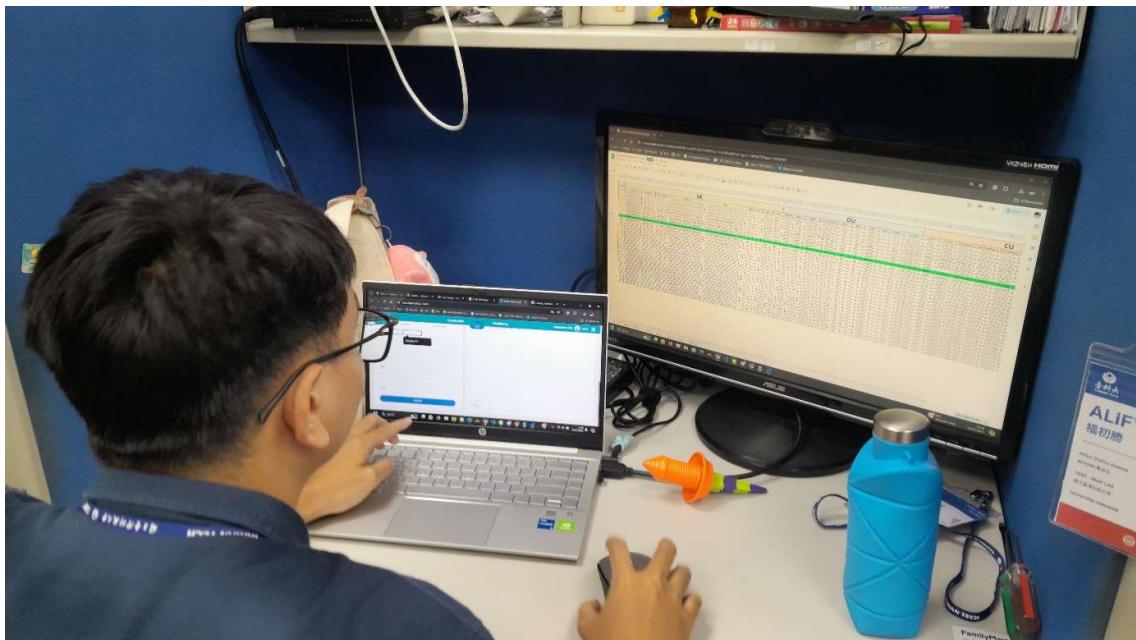
5.1



5.2



5.3



5.4



5.5

No	Responden	User Persona	Skor Asli										Skor Hasil Hitung										Jumlah	Nilai (Jumlah x 2,5)
			Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10		
1	Responden 1	Staff TIP	4	4	4	4	4	2	4	2	2	4	3	1	3	3	3	3	1	1	1	22	55	
2	Responden 2	Staff TIP	5	2	4	3	4	2	4	1	3	2	4	3	3	2	3	3	3	0	30	75		
3	Responden 3	Mahasiswa	5	3	3	5	4	2	4	2	4	5	4	2	2	0	3	3	3	3	0	23	57,5	
4	Responden 4	Mahasiswa	5	2	4	3	4	2	4	1	4	3	4	3	3	2	3	3	3	4	3	2	30	75
5	Responden 5	Mahasiswa	5	5	5	5	5	5	5	5	3	5	4	0	4	0	4	0	4	0	2	0	18	45
6	Responden 6	Mahasiswa	5	2	4	4	4	3	4	2	2	5	4	3	3	1	3	2	1	3	1	0	23	57,5
7	Responden 7	Mahasiswa	5	3	3	4	5	2	3	3	4	5	4	2	2	1	4	3	2	2	3	0	23	57,5
8	Responden 8	Mahasiswa	5	5	5	5	4	4	5	5	4	4	4	0	4	0	3	1	4	0	3	1	20	50
9	Responden 9	Staff TIP	5	1	5	5	4	4	4	1	3	4	4	4	4	0	3	1	3	4	2	1	26	65
10	Responden 10	Mahasiswa	4	2	4	5	3	4	4	2	3	4	3	3	3	0	2	1	3	3	2	1	21	52,5
11	Responden 11	Staff TIP	5	1	5	3	3	2	2	1	3	3	4	4	4	2	2	3	1	4	2	2	28	70
12	Responden 12	Mahasiswa	4	2	5	4	5	5	5	1	4	4	3	3	4	1	4	0	4	4	3	1	27	67,5
13	Responden 13	Mahasiswa	5	1	5	4	5	2	5	1	3	3	4	4	4	1	4	3	4	4	2	2	32	80
14	Responden 14	Mahasiswa	4	2	3	3	3	2	2	3	4	4	3	3	2	2	3	1	2	3	1	22	55	
15	Responden 15	Mahasiswa	5	1	5	1	3	1	5	1	4	3	4	4	4	2	4	4	4	3	2	35	87,5	
16	Responden 16	Mahasiswa	4	3	4	4	3	3	5	1	3	5	3	2	3	1	2	2	4	4	2	0	23	57,5
17	Responden 17	Mahasiswa	4	1	5	4	4	2	5	1	5	3	3	4	4	1	3	3	4	4	2	32	80	
18	Responden 18	Mahasiswa	5	2	4	2	4	2	4	5	2	4	3	3	3	3	3	3	3	4	3	32	80	
19	Responden 19	Staff TIP	5	2	5	3	5	3	4	1	2	4	4	3	4	2	4	2	3	4	1	1	28	70
20	Responden 20	Mahasiswa	5	2	4	4	4	2	4	1	3	5	4	3	3	2	2	3	1	2	3	0	26	65
21	Responden 21	Mahasiswa	5	2	2	4	5	2	4	2	4	4	4	3	1	1	4	3	3	3	3	1	26	65
22	Responden 22	Engineer	5	2	4	5	4	2	2	3	3	4	4	3	3	0	3	3	1	2	2	1	22	55
23	Responden 23	Engineer	4	2	4	2	3	2	4	2	5	1	3	3	3	2	3	3	4	4	3	1	31	77,5
24	Responden 24	Engineer	5	1	4	4	5	1	4	2	3	4	3	4	4	3	4	4	3	3	2	1	29	72,5
25	Responden 25	Engineer	4	2	5	3	3	3	4	2	4	2	3	3	3	2	2	3	3	3	3	3	28	70
26	Responden 26	Engineer	4	3	4	3	5	2	4	2	3	4	3	4	3	2	2	3	3	3	2	25	62,5	
27	Responden 27	Engineer	4	2	3	5	2	4	2	2	4	4	3	3	2	0	1	1	3	3	1	1	18	45
28	Responden 28	Engineer	5	2	5	2	4	2	5	2	3	2	4	3	3	3	3	4	3	2	3	32	80	
29	Responden 29	Engineer	5	3	3	4	3	3	4	3	4	4	4	2	2	1	2	2	3	2	3	1	22	55
30	Responden 30	Mahasiswa	5	3	5	4	4	1	4	1	5	3	4	2	4	1	3	4	3	4	2	31	77,5	

Nilai Akhir SUS

65,42

5.6

No	Responden	User Persona	Skor Asli										Skor Hasil Hitung										Jumlah	Nilai (Jumlah x 2,5)
			Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10		
1	Responden 1	Staff TIP	5	1	5	2	5	1	5	1	5	1	4	4	4	4	4	4	4	4	4	4	39	97,5
2	Responden 2	Staff TIP	5	1	5	2	4	1	5	1	4	4	4	4	4	4	3	3	4	4	4	3	34	85
3	Responden 3	Mahasiswa	3	3	5	3	3	2	4	3	5	2	2	2	4	2	2	3	3	2	4	3	27	67,5
4	Responden 4	Mahasiswa	5	1	5	2	5	1	5	1	4	3	4	4	4	3	4	4	4	4	3	2	36	90
5	Responden 5	Mahasiswa	4	2	5	3	4	2	4	3	4	2	3	3	4	2	3	3	3	2	3	3	29	72,5
6	Responden 6	Mahasiswa	5	3	5	3	4	3	4	3	4	3	4	2	4	2	3	2	3	2	3	2	27	67,5
7	Responden 7	Mahasiswa	4	2	5	1	4	1	3	4	2	5	2	3	3	4	4	3	2	3	3	4	32	80
8	Responden 8	Mahasiswa	4	2	5	2	3	1	4	1	4	1	3	3	3	4	3	2	4	3	4	3	33	82,5
9	Responden 9	Staff TIP	5	2	4	1	4	2	4	3	5	2	4	4	3	3	4	3	3	2	4	3	32	80
10	Responden 10	Mahasiswa	5	2	4	1	5	2	4	2	4	1	4	3	3	4	4	3	3	3	4	3	34	85
11	Responden 11	Staff TIP	4	1	4	1	4	2	5	1	3	2	3	4	3	3	4	3	3	4	2	3	33	82,5
12	Responden 12	Mahasiswa	5	2	4	1	5	1	5	2	5	1	4	3	3	4	4	4	4	3	4	4	37	92,5
13	Responden 13	Mahasiswa	5	2	3	2	5	3	4	2	4	3	4	3	2	3	4	2	3	3	3	2	29	72,5
14	Responden 14	Mahasiswa	5	1	4	1	4	1	4	2	3	1	4	4	3	4	3	4	3	3	2	4	34	85
15	Responden 15	Mahasiswa	5	1	5	1	3	1	5	1	4	3	4	4	4	2	4	4	4	3	3	2	35	87,5
16	Responden 16	Mahasiswa	4	2	4	3	3	2	3	1	4	2	3	3	3	2	2	3	2	3	3	3	28	70
17	Responden 17	Mahasiswa	4	2	4	2	4	2	4	2	4	2	3	3	3	3	3	3	3	3	3	3	30	75
18	Responden 18	Mahasiswa	4	1	4	3	4	3	5	3	3	1	3	4	3	2	3	2	4	2	2	4	29	72,5
19	Responden 19	Staff TIP	4	2	4	1	3	3	4	3	5	1	3	3	3	4	2	2	3	2	4	4	30	75
20	Responden 20	Mahasiswa	5	2	4	3	3	2	5	3	3	3	4	3	3	2	2	3	4	2	2	2	27	67,5
21	Responden 21	Mahasiswa	5	2	4	5	4	2	4	2	4	4	4	3	3	0	3	3	3	3	3	1	26	65
22	Responden 22	Engineer	5	2	4	5	4	2	2	3	3	4	4	3	3	0	3	3	1	2	2	1	22	55
23	Responden 23	Engineer	4	2	5	2	5	1	4	3	5	2	3	3	4	3	4	4	3	2	4	3	33	82,5
24	Responden 24	Engineer	5	1	4	4	5	1	4	2	3	4	4	4	3	1	4	4	3	3	2	1	29	72,5
25	Responden 25	Engineer	4	1	5	2	4	2	5	2	5	1	3	4	4	3	3	3	3	4	4	4	35	87,5
26	Responden 26	Engineer	4	2	4	3	4	1	2	4	3	3	4	3	3	2	3	3	3	3	2	1	25	62,5
27	Responden 27	Engineer	5	2	4	3	4	2	4	2	4	3	4	3	3	2	3	3	3	3	2	2	29	72,5
28	Responden 28	Engineer	4	2	4	4	5	1	4	2	4	3	3	3	3	1	4	4	3	3	3	2	29	72,

5.7

No	Responden	User Persona	Skor Asli					
			Q1	Q2	Q3	Q4	Q5	Q6
1	Responden 1	Staff TIP	5	5	5	5	5	5
2	Responden 2	Staff TIP	5	4	5	4	5	5
3	Responden 3	Mahasiswa	4	5	4	5	4	5
4	Responden 4	Mahasiswa	4	5	5	4	5	5
5	Responden 5	Mahasiswa	4	4	4	4	4	4
6	Responden 6	Mahasiswa	5	5	5	3	4	5
7	Responden 7	Mahasiswa	3	4	4	5	4	5
8	Responden 8	Mahasiswa	5	5	5	5	4	5
9	Responden 9	Staff TIP	4	3	5	4	4	4
10	Responden 10	Mahasiswa	4	4	5	4	5	5
11	Responden 11	Staff TIP	4	5	3	4	3	4
12	Responden 12	Mahasiswa	4	5	5	5	4	4
13	Responden 13	Mahasiswa	5	4	5	3	4	4
14	Responden 14	Mahasiswa	4	5	5	4	5	5
15	Responden 15	Mahasiswa	4	4	4	4	3	5
16	Responden 16	Mahasiswa	5	3	4	5	4	5
17	Responden 17	Mahasiswa	5	4	4	3	5	4
18	Responden 18	Mahasiswa	5	5	4	4	4	4
19	Responden 19	Staff TIP	4	5	4	4	3	5
20	Responden 20	Mahasiswa	4	4	5	5	4	5
21	Responden 21	Mahasiswa	4	5	4	4	5	5
22	Responden 22	Engineer	4	4	4	3	3	4
23	Responden 23	Engineer	4	4	5	4	3	4
24	Responden 24	Engineer	5	5	5	4	4	5
25	Responden 25	Engineer	4	4	5	5	5	5
26	Responden 26	Engineer	4	4	4	4	4	5
27	Responden 27	Engineer	5	4	5	4	4	5
28	Responden 28	Engineer	5	5	5	5	5	5
29	Responden 29	Engineer	5	4	5	5	4	5
30	Responden 30	Mahasiswa	5	4	4	3	3	4
Rata-Rata			4,4	4,4	4,5	4,2	4,1	4,7