IMPLEMENTASI OBJECT-RELATIONAL MAPPING (ORM) PRISMA DALAM PENGEMBANGAN WEB SDA DIVISION DI PT TELKOM INDONESIA TBK



Disusun Oleh:

N a m a	: Arya Danuarta Ramadhan
NIM	: 20523245

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA

2025

HALAMAN PENGESAHAN DOSEN PEMBIMBING

IMPLEMENTASI OBJECT-RELATIONAL MAPPING (ORM) PRISMA DALAM PENGEMBANGAN WEB SDA DIVISION DI PT TELKOM INDONESIA TBK

TUGAS AKHIR JALUR MAGANG

Disusun Oleh:

N a m a	: Arya Danuarta Ramadhan
NIM	: 20523245

Yogyakarta, 21 Mei 2025

Pembimbing,



(Dr. Yudi Prayudi, S.Si., M.Kom.)

HALAMAN PENGESAHAN DOSEN PENGUJI

IMPLEMENTASI OBJECT-RELATIONAL MAPPING (ORM) PRISMA DALAM PERANCANGAN RESTFUL API UNTUK WEB SDA DIVISION DI PT TELKOM INDONESIA TBK

TUGAS AKHIR JALUR MAGANG

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 21 Mei 2025

Tim Penguji

Dr. Yudi Prayudi, S.Si., M.Kom _____

Anggota 1

Ir. Chandra Kusuma Dewa, S.Kom., M.Cs., Ph.D. _____

Anggota 2

Erika Ramadhani, S.T., M.Eng. _____

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

(Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama	:	Arya Danuarta Ramadhan
NIM	:	20523245

Tugas akhir dengan judul:

IMPLEMENTASI OBJECT-RELATIONAL MAPPING (ORM) PRISMA DALAM PERANCANGAN RESTFUL API UNTUK WEB SDA DIVISION DI PT TELKOM INDONESIA TBK

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 21 Mei 2025



(Arya Danuarta Ramadhan)

HALAMAN PERSEMBAHAN

Tugas akhir ini dipersembahkan kepada kedua orangtua saya yang telah memberikan dukungan penuh disaat susah maupun senang. Tanpa doa dari kedua orangtua, saya tidak akan pernah sampai dititik ini. Terimakasih atas perhatian yang diberikan setiap saat sehingga saya dapat menjalani semuanya dengan baik.

HALAMAN MOTO

“ In the end, the love you take is equal to make the love you make”

(The Beatles)

KATA PENGANTAR

Assalamu’alaikum Warrahmatullahi Wabarakatuh

Puji syukur saya panjatkan ke hadirat Allah SWT atas limpahan rahmat, karunia, serta petunjuk-Nya, sehingga saya dapat menyelesaikan tugas akhir yang berjudul " IMPLEMENTASI OBJECT-RELATIONAL MAPPING (ORM) PRISMA PNGEMBANGAN WEB SDA DIVISION DI PT TELKOM INDONESIA TBK".

Tugas akhir ini disusun sebagai salah satu syarat untuk menyelesaikan pendidikan Strata Satu (S1) di Universitas Islam Indonesia, Fakultas Teknologi Industri, Program Studi Informatika. Pada proses penyusunan tugas akhir ini, saya menyadari tanpa dukungan, bimbingan, dan arahan yang diberikan dari berbagai pihak, tugas akhir ini tidak akan terselesaikan dengan baik. Oleh karena itu, saya ingin mengucapkan terimakasih kepada :

1. Allah SWT yang telah membantu disetiap langkahnya.
2. Ayah (Gusti Djabbar) Ibu (Jumarni) Adik (Arini Aulia) dan Kerabat(Aisyah), yang selalu memberikan dukungan serta semangat disaat susah maupun senang.
3. Bapak Dr. Yudi Prayudi, S.Si., M.Kom. Selaku dosen pembimbing tugas akhir atas arahan,bantuan, serta bimbingannya dalam proses pengerjaan tugas akhir hingga selesai.
4. Bapak Dr. Raden Teduh Dirgahayu, S.T., M.Sc. Ketua Jurusan Informatika Universitas Islam Indonesia.
5. Bapak Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D Ketua Program Studi Informatika Universitas Islam Indonesia.
6. Teman-teman dari Kendari yang telah memberikan motivasi serta semangat saat saya mengalami kesulitan. Secara khusus saya ucapkan terimakasih kepada Shafwan, Riziq, Alif, Mustari, Gina, Tya, Padang, Gabriel dan yang lainnya yang tidak dapat saya tuliskan satu persatu.

7. Rekan-rekan diperkuliahan yang telah menemani dalam pengerjaan tugas akhir. Saya menyadari akan banyaknya kekurangan dalam tugas akhir yang saya kerjakan. Oleh karena itu, saya menerima kritik dan saran yang membangun untuk penyempurnaan penelitian kedepannya. Semoga Allah SWT memberikan taufik dan hidayah-Nya kepada kita semua. Amin. *Wassalamu'alaikum Warrahmatullahi Wabarakatuh*

Yogyakarta, 21 Mei 2025



(Arya Danuarta Ramadhan)

INTISARI

Tujuan dari penelitian ini adalah untuk mengembangkan aplikasi manajemen proyek berbasis web yang dapat meningkatkan efektivitas dan efisiensi pengelolaan proyek di PT Telkom Indonesia Tbk. Proyek pengembangan web ini dilaksanakan dengan menggunakan metode *Scrum*. Program magang yang dilaksanakan di PT Telekomunikasi Indonesia Tbk ini berlangsung selama tujuh bulan, dimulai dari 17 Oktober hingga 1 Juli. Magang berlangsung selama jam kerja perusahaan, yaitu dari Senin hingga Jumat, mulai pukul 08.00 WIB hingga 17.00 WIB. Hasil penelitian menunjukkan bahwa pemagang yang mengikuti program magang di Departemen Penyelesaian, Pengiriman, dan Keamanan (SDA) Regional II PT Telkom Indonesia mengalami banyak kemajuan dan pembelajaran yang signifikan dalam hal profesionalisme dan pengembangan diri. Pemagang mengembangkan aplikasi berbasis web yang mencakup berbagai elemen dari front-end hingga back-end selama masa magang. Pemagang dapat menggunakan teknologi terkini seperti Vue.js untuk pengembangan antarmuka pengguna dan Express.js untuk back-end untuk secara aktif menerapkan teori dalam proyek nyata. Pemagang belajar mengatur waktu dengan baik, memprioritaskan tugas yang ada, dan menyelesaikan masalah selama proses pengembangan. Pemagang berkomitmen untuk memperluas pengetahuan dan keterampilan mereka dalam teknologi dan struktur industri terbaru.

Keywords: *Object-Relational Mapping (ORM), PT. Telkom Indonesia Tbk, Express.jss, Metode Scrum.*

GLOSARIUM

API Documentation	Dokumen yang menjelaskan fungsi, parameter, dan respon dari endpoint API untuk mempermudah pengembang lain memahami dan menggunakannya.
Backend	Bagian dari sistem atau aplikasi yang bekerja di server, bertanggung jawab untuk logika aplikasi, pengelolaan data, dan komunikasi dengan database.
CRUD	Operasi dasar dalam pengelolaan data yang mencakup Create (membuat), Read (membaca), Update (memperbarui), dan Delete (menghapus).
Debugging	Proses menemukan dan memperbaiki kesalahan atau bug dalam kode program.
Deployment	Proses pengiriman atau pengimplementasian aplikasi ke lingkungan produksi sehingga dapat digunakan oleh pengguna akhir.
Error Handling	Proses mendeteksi, menangani, dan memberikan respons terhadap kesalahan yang terjadi selama aplikasi berjalan.
Frontend	Bagian antarmuka pengguna dari sebuah aplikasi atau website yang terlihat dan digunakan oleh pengguna akhir.
HTTP	HyperText Transfer Protocol; protokol komunikasi yang digunakan untuk pertukaran data di internet.
JSON Web Token (JWT)	Metode autentikasi tanpa status (stateless) yang menghasilkan token unik untuk memverifikasi identitas pengguna dalam setiap permintaan API.
Node.js	Lingkungan runtime JavaScript yang memungkinkan pengembangan backend aplikasi berbasis web dengan performa tinggi.
Object-Relational Mapping (ORM)	Teknik pemrograman yang memetakan data dari database relasional ke dalam objek pada bahasa pemrograman berbasis objek, sehingga mempermudah pengelolaan data tanpa penulisan manual query SQL.

Prisma	Sebuah ORM modern untuk Node.js yang menyediakan fitur type-safe dan kemudahan pengelolaan database relasional melalui model skema yang intuitif.
RESTful API	Representational State Transfer API; arsitektur yang memanfaatkan metode HTTP (GET, POST, PUT, DELETE) untuk memungkinkan komunikasi dan manipulasi data antara klien dan server.
Role-Based Access Control (RBAC)	Sistem kontrol akses berbasis peran yang memberikan hak akses sesuai peran pengguna, seperti admin atau user biasa.
Scalability	Kemampuan sistem atau aplikasi untuk menangani peningkatan jumlah pengguna atau data tanpa menurunkan performa.
Schema	Struktur atau kerangka tabel dalam database yang mendefinisikan atribut-atribut data dan hubungan antar tabel.
Sprint	Siklus kerja pendek dalam metodologi Scrum yang fokus pada penyelesaian tugas spesifik dalam dua hingga empat minggu.
Token-Based Authentication	Metode autentikasi di mana pengguna diberikan token unik setelah berhasil login, yang kemudian digunakan untuk semua permintaan selanjutnya.
Vue.js	Framework JavaScript untuk pengembangan antarmuka pengguna (frontend) yang interaktif dan responsif.

DAFTAR ISI

HALAMAN JUDUL 1
HALAMAN PENGESAHAN DOSEN PEMBIMBING 2
HALAMAN PENGESAHAN DOSEN PENGUJI 3
HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR 4
HALAMAN PERSEMBAHAN 5
HALAMAN MOTO 6
KATA PENGANTAR 7
SARI 9
GLOSARIUM 10
DAFTAR ISI 12
DAFTAR TABEL 14
DAFTAR GAMBAR 15

BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Ruang Lingkup 4
1.3 Tujuan 8
1.4 Manfaat 9
1.5 Sistematika Penulisan ....

BAB II LANDASAN TEORI DAN TINJAUAN PUSTAKA 11
2.1 *Website* 11
2.2 Aplikasi Manajemen Proyek 12
2.4 *Object-Relational Mapping* (ORM) 14
2.5 ORM *Prisma* 15
2.6 Pengelolaan *Database* Relasional dalam Pengembangan *API* 16
2.7 Keamanan dalam Pengembangan *API* 18
2.8 Tinjauan Pustaka 19

BAB III PELAKSANAAN MAGANG 24
3.1 Tempat dan Waktu Pelaksaan Magang 24
3.2 Peran dan Tanggung Jawab 25
3.3 Metode Pelaksanaan 28

3.4 Pelaksanaan Magang 30
3.4.1 *Onboarding* dan Pengenalan Lingkungan Kerja 30
3.4.2 Memahami Struktur *Database* pada Aplikasi SDA 32
3.4.3 Implementasi *ORM Prisma* untuk *Database* SDA 34
3.4.4 Pengembangan dan Pembuatan *RESTful API* 38
3.4.5 *Testing* dan Optimisasi *API* dengan *Prisma* 38
3.4.6 *Deployment* dan Dokumentasi *API* 39
3.5 Tahapan *Sprint* dalam Pengembangan Proyek 41
3.5.1 *Sprint* 1 – Inisiasi 41
3.5.2 *Sprint* 2 – Perencanaan 42
3.5.3 *Sprint* 3 - Pengembangan 43
3.5.4 *Sprint* 4 - API Dasar 48
3.5.5 *Sprint* 5 - *Testing* dan Optimisasi API 49
3.5.6 *Sprint* 6 - *Deployment* dan Dokumentasi 49
3.6 Konfigurasi ORM Prisma ....
3.7 Dokumentasi Prisma 50
BAB IV REFLEKSI PELAKSANAAN MAGANG 56
4.1 Relevansi Akademik 56
4.2 Pembelajaran Magang 60
4.2.1 Pembelajaran dari Penerapan Teknologi Terkini 61
4.2.2 Pembelajaran dari Penerapan Metodologi Agile dan Scrum 64
BAB V PENUTUP 69
5.1 Kesimpulan 69
5.2 Saran 70
DAFTAR PUSTAKA 73

DAFTAR TABEL

Tabel 3.1 Konfigurasi *schema Prisma* 44
Tabel 3.2 Model Schema Type 50
Tabel 3.3 Model Division 50
Tabel 3.4 Model Unit 51
Tabel 3.5 Model Partner 51
Tabel 3.6 Model Customer 52
Tabel 3.7 Model Project 52
Tabel 3.8 Model Produk 53
Tabel 3.9 Model Produk Kategori 54
Tabel 3.10 Model User 54
Tabel 3.11 Detail Impact Based Group 54
Tabel 3.12 Model Status YTD 54
Tabel 3.13 Project Category 55
Tabel 3.14 Model Roles 55

DAFTAR GAMBAR

Gambar.1.1 Pengumpulan Data dan Analisis Permasalahan Pengumpulan Data dan Analisis Permasalahan.....4

Gambar 3.1 Onboarding dan Pengenalan Lingkungan.....31
Gambar 3.2 Deployment dan Dokumentasi API.....40
Gambar 3.3 Tahapan Sprint Dalam Pengembangan ProyekTabel 3 1 41

BAB I
PENDAHULUAN

1.1 Latar Belakang

Di era digital yang terus berkembang pesat, organisasi di seluruh dunia dihadapkan pada tantangan untuk meningkatkan efisiensi dan efektivitas dalam pengelolaan proyek. Di Indonesia, PT Telkom Indonesia Tbk, sebagai salah satu perusahaan telekomunikasi terbesar, memainkan peran penting dalam mendukung pembangunan infrastruktur digital di seluruh negeri. Divisi Sumber Daya Alam (SDA) di PT Telkom Indonesia memiliki tanggung jawab strategis untuk mengoptimalkan penggunaan sumber daya alam serta mendukung proyek-proyek yang terkait dengan lingkungan dan keberlanjutan. Namun, pengelolaan proyek di divisi ini sering kali mengalami kendala yang signifikan, termasuk keterbatasan dalam sistem yang terfragmentasi, kesulitan dalam berkomunikasi antar tim, dan kurangnya transparansi dalam pengelolaan data (Atlassian, 2023; Kerzner, 2017).

PT Telkom Indonesia Tbk, yang didirikan pada tahun 1961, telah berkomitmen untuk menjadi pemimpin dalam layanan telekomunikasi dan solusi digital di Indonesia. Perusahaan ini terlibat dalam berbagai proyek yang berkaitan dengan pembangunan infrastruktur telekomunikasi, pengembangan teknologi informasi, serta penyediaan layanan internet dan data. Divisi SDA berfokus pada proyek-proyek yang memanfaatkan dan mengelola sumber daya alam, termasuk proyek-proyek yang berkontribusi pada keberlanjutan lingkungan. Beberapa proyek yang sedang berjalan melibatkan pengembangan aplikasi untuk pengelolaan data lingkungan, pengawasan sumber daya, dan pelaporan terkait dampak lingkungan dari kegiatan operasional. Dalam hal ini, penulis laporan ini terlibat dalam proyek pengembangan aplikasi manajemen proyek berbasis web yang berfokus pada pengelolaan sumber daya alam. Proyek ini memiliki durasi enam bulan, dimulai dari Oktober 2023 hingga April 2024, dengan beberapa aktivitas penting, termasuk pengumpulan data mengenai masalah yang dihadapi divisi, pengembangan backend aplikasi, dan perancangan RESTful API.

Masalah-masalah yang dihadapi divisi SDA dapat menyebabkan tantangan serius dalam mencapai target proyek, berpotensi mengakibatkan keterlambatan dalam penyelesaian dan meningkatkan risiko pemborosan sumber daya. Dalam situasi seperti ini, penggunaan aplikasi manajemen proyek berbasis web telah muncul sebagai solusi yang sangat relevan. Aplikasi ini tidak hanya menyediakan platform yang memungkinkan kolaborasi yang lebih baik antar

anggota tim, tetapi juga memfasilitasi akses yang lebih mudah terhadap informasi proyek, dan kemampuan untuk memantau kemajuan proyek secara real-time. Penelitian menunjukkan bahwa penerapan perangkat lunak manajemen proyek dapat meningkatkan produktivitas tim hingga 30%, serta mengurangi waktu yang dibutuhkan untuk komunikasi dan koordinasi (Serrador & Pinto, 2015). Oleh karena itu, pengembangan aplikasi manajemen proyek berbasis web di Divisi SDA PT Telkom Indonesia Tbk menjadi suatu kebutuhan mendesak untuk meningkatkan kinerja dan efisiensi operasional, serta memastikan keberhasilan proyek-proyek yang dijalankan.

Salah satu pendekatan yang sangat relevan dalam pengembangan aplikasi web adalah penggunaan Representational State Transfer (REST) melalui RESTful API. RESTful API menyediakan metode komunikasi yang efisien antara klien dan server dengan menggunakan protokol HTTP, yang memudahkan pengembang dalam mengakses dan mengelola data dari berbagai sumber (Fielding, 2000). Kelebihan dari RESTful API terletak pada kesederhanaan dan kemudahan penggunaannya, menjadikannya pilihan yang ideal untuk aplikasi manajemen proyek yang memerlukan interaksi data yang cepat dan responsif. Dengan implementasi RESTful API, tim proyek di Divisi SDA dapat mengelola data secara lebih efektif, meningkatkan transparansi, serta mempercepat proses pengambilan keputusan. Hal ini tentunya akan berkontribusi pada pengurangan risiko dan peningkatan kepuasan stakeholder yang terlibat dalam proyek.

Dalam pengembangan aplikasi ini, penggunaan Object-Relational Mapping (ORM) seperti Prisma menjadi sangat penting untuk mempermudah interaksi dengan basis data. ORM Prisma memungkinkan pengembang untuk berinteraksi dengan basis data menggunakan kode pemrograman yang lebih bersih dan terstruktur, sehingga mengurangi kemungkinan kesalahan dalam penulisan query SQL (Bauer & King, 2021). Dengan menggunakan ORM, pengembang dapat lebih fokus pada pengembangan logika bisnis aplikasi dan fitur-fitur baru tanpa terjebak dalam kerumitan pengelolaan basis data. Selain itu, penerapan ORM juga membantu menjaga integritas data dan meningkatkan performa sistem, yang merupakan faktor kunci dalam pengelolaan proyek yang sukses (Pernici et al., 2018). Dengan demikian, penggabungan ORM dalam aplikasi manajemen proyek akan memudahkan tim dalam mengelola informasi proyek dan memungkinkan mereka untuk melakukan analisis data dengan lebih efisien.

Aspek keamanan juga merupakan faktor penting yang tidak boleh diabaikan dalam pengembangan RESTful API. Dalam praktiknya, langkah-langkah keamanan seperti otorisasi, autentikasi, dan validasi input sangat krusial untuk melindungi aplikasi dari potensi ancaman

dan serangan siber yang dapat merugikan (Mansfield-Devine, 2013). Menerapkan protokol keamanan yang baik tidak hanya melindungi data sensitif tetapi juga meningkatkan kepercayaan pengguna terhadap aplikasi yang dikembangkan. Dalam konteks ini, penting untuk mengimplementasikan praktik terbaik dalam keamanan API, termasuk penggunaan token untuk autentikasi dan enkripsi data untuk melindungi informasi yang dikirimkan melalui jaringan. Keamanan yang kuat akan memastikan bahwa data yang dikelola dalam aplikasi tidak hanya aman dari pencurian atau penyalahgunaan, tetapi juga mematuhi regulasi dan standar industri yang berlaku.

Pelaksanaan proyek ini dilakukan dengan menggunakan metode pengembangan perangkat lunak Scrum, yang telah terbukti efektif dalam meningkatkan kolaborasi, transparansi, dan fleksibilitas dalam pengembangan proyek perangkat lunak. Metode Scrum memungkinkan tim untuk beradaptasi dengan cepat terhadap perubahan kebutuhan dan prioritas, serta meningkatkan kolaborasi antar anggota tim dan memungkinkan identifikasi masalah lebih awal dalam proses pengembangan (Schwaber & Sutherland, 2020). Setiap sprint dalam metode ini memiliki durasi sekitar dua minggu, dengan tujuan untuk menghasilkan peningkatan yang dapat diukur dalam pengembangan API. Pendekatan ini diyakini dapat membantu Divisi SDA untuk menghadapi dinamika perubahan dalam pengelolaan proyek, serta memberikan solusi yang lebih responsif terhadap kebutuhan pengguna.

Dengan mempertimbangkan tantangan yang dihadapi oleh Divisi SDA, serta potensi solusi yang ditawarkan oleh RESTful API dan ORM Prisma, penelitian ini bertujuan untuk mengembangkan aplikasi manajemen proyek berbasis web yang dapat meningkatkan efisiensi dan efektivitas pengelolaan proyek di PT Telkom Indonesia Tbk. Proyek ini diharapkan tidak hanya memberikan manfaat signifikan bagi perusahaan, tetapi juga menjadi kontribusi yang berarti dalam pengembangan ilmu pengetahuan di bidang teknologi informasi dan manajemen proyek. Dengan demikian, penelitian ini tidak hanya berfokus pada aspek teknis pengembangan, tetapi juga pada dampak sosial dan ekonomi yang dihasilkan dari implementasi sistem yang lebih baik dalam pengelolaan proyek di sektor telekomunikasi. Pemilihan topik ini juga didasarkan pada keinginan penulis untuk berkontribusi dalam meningkatkan proses manajemen proyek dan memanfaatkan teknologi terkini untuk mencapai hasil yang lebih baik.

1.2 Ruang Lingkup

Proyek *Implementasi Object-Relational Mapping (ORM) Prisma dalam Perancangan RESTful API untuk Web Divisi SDA di PT Telkom Indonesia Tbk* mencakup serangkaian aktivitas dan peran yang dilaksanakan selama enam bulan, mulai dari pertengahan Oktober 2023 hingga pertengahan April 2024. Adapun cakupan kegiatan dan tanggung jawab penulis dalam proyek ini mencakup beberapa aspek utama, sebagai berikut:

1. Pengumpulan Data dan Analisis Permasalahan di Divisi SDA TREG II

Pada tahap awal, penulis melakukan kegiatan pengumpulan data untuk mengidentifikasi permasalahan, hambatan, dan kebutuhan spesifik dari Divisi SDA TREG II. Proses ini dilakukan dengan beberapa metode, termasuk:



Gambar.1.1 Pengumpulan Data dan Analisis Permasalahan Pengumpulan Data dan Analisis Permasalahan

- **Wawancara Mendalam**: Melakukan wawancara terstruktur dengan pemangku kepentingan utama, seperti manajer proyek, tim teknis, dan pengguna akhir, untuk memperoleh pemahaman mendalam mengenai kendala kerja yang dihadapi dan ekspektasi dari aplikasi yang akan dikembangkan.
- **Observasi Langsung**: Mengamati alur kerja dan proses bisnis yang dilakukan dalam pengelolaan proyek di divisi ini untuk memetakan arus data, identifikasi kendala yang sering muncul, dan memahami potensi area untuk perbaikan.

- **Peninjauan Dokumentasi**: Menganalisis dokumen internal terkait proyek, SOP (Standard Operating Procedure), dan laporan performa untuk memahami kebijakan data yang diterapkan, serta mengevaluasi apakah alur kerja telah berjalan efisien.

Hasil dari kegiatan pengumpulan data ini dijadikan acuan dalam menyusun spesifikasi aplikasi dan menentukan fitur-fitur utama yang dapat mendukung penyelesaian masalah dan meningkatkan produktivitas di Divisi SDA.

2. Studi dan Penguasaan Teknologi Relevan (*Vue.js*, *Node.js*, dan ORM Prisma)

Setelah memahami kebutuhan dari sisi bisnis, penulis kemudian berfokus pada mempelajari dan menguasai teknologi yang dibutuhkan untuk pengembangan aplikasi ini, yaitu Vue.js, Node.js, dan ORM Prisma, melalui langkah-langkah berikut:

- **Pengenalan Framework *Vue.js* dan *Node.js***: Memahami struktur dan prinsip kerja *Vue.js* untuk pengembangan *frontend* aplikasi dan *Node.js* untuk *backend*. *Vue.js* dipilih karena kemampuannya dalam membangun *UI* yang interaktif dan responsif, sementara *Node.js* menawarkan performa yang baik dalam mengelola permintaan dari aplikasi.
- **Pembelajaran *ORM Prisma***: Mempelajari *ORM Prisma* secara mendalam, termasuk pemetaan tabel database dengan entitas pada aplikasi. *Prisma* memungkinkan penggunaan query yang lebih sederhana dan aman melalui *ORM*, serta memudahkan dalam mengelola operasi *CRUD* dan migrasi data. Tahap ini juga mencakup uji coba integrasi *Prisma* dengan *Node.js* untuk memastikan fungsionalitas dan kestabilan aplikasi.
- **Eksplorasi Best Practices**: Penulis mempelajari praktik terbaik dalam penggunaan *ORM* dan *RESTful API*, seperti desain skema database yang efisien, manajemen koneksi database, serta optimasi performa *API*. Pemahaman ini penting agar aplikasi dapat berjalan optimal dan memenuhi standar keamanan.

3. Pengembangan *Back-End* Aplikasi Divisi SDA

Pada tahap ini, penulis mengembangkan back-end aplikasi web Divisi SDA dengan fokus pada desain arsitektur yang skalabel dan mudah dikelola. Tahapan pengembangan ini mencakup beberapa kegiatan:

- **Desain Struktur Database dengan *ORM Prisma***: Menyusun struktur database untuk memenuhi kebutuhan penyimpanan data Divisi SDA, seperti data proyek, tim, dan progres proyek. Penggunaan *Prisma* memudahkan penulis dalam membuat dan

memodifikasi skema database, serta menambahkan validasi untuk memastikan konsistensi data.

- **Pengembangan Fitur *CRUD***: Mengimplementasikan fungsi *CRUD* untuk mengelola data proyek. Fitur ini mencakup *Create* untuk pembuatan data baru, *Read* untuk menampilkan data, *Update* untuk memperbarui data, dan *Delete* untuk menghapus data. *ORM Prisma* digunakan sebagai lapisan pemrosesan data, memungkinkan akses data yang lebih efisien dan aman.
- **Keamanan dan Autentikasi**: Menambahkan lapisan keamanan, seperti autentikasi *JWT* (*JSON Web Token*) dan otorisasi berbasis peran (*role-based authorization*), guna membatasi akses terhadap data sensitif. Pengguna hanya dapat mengakses informasi sesuai dengan peran dan kewenangannya.
- **Pengujian Unit dan Integrasi**: Melakukan pengujian pada setiap komponen *back-end*, termasuk validasi endpoint *API* dan uji integrasi database untuk memastikan semua fitur bekerja sesuai yang diharapkan dan respons terhadap permintaan berlangsung efisien.

4. Perancangan dan Implementasi *RESTful API* untuk Aplikasi Divisi SDA

Tahap akhir dalam pengembangan aplikasi adalah merancang dan mengimplementasikan *RESTful API* yang memungkinkan komunikasi antara aplikasi web Divisi SDA dan sistem lain yang relevan. Proses ini terdiri dari beberapa bagian penting berikut:

1) Desain *Endpoint API*

Penyusunan *endpoint* dilakukan berdasarkan kebutuhan fungsional aplikasi, seperti:

a. /projects untuk data proyek,
b. /teams untuk informasi tim, dan
c. /tasks untuk detail tugas.

Setiap *endpoint* dirancang untuk menangani operasi *CRUD* (*Create, Read, Update, Delete*) menggunakan metode *HTTP* standar seperti *GET*, *POST*, *PUT*, dan *DELETE*. Pendekatan ini memastikan struktur komunikasi yang konsisten dan dapat diakses oleh berbagai komponen *frontend* maupun *external services*.

2) Dokumentasi *API*

Untuk mendukung kemudahan pemahaman dan penggunaan oleh tim pengembang lain, setiap *endpoint* didokumentasikan menggunakan alat seperti *Swagger* atau *Postman*. Dokumentasi mencakup:

a. Struktur permintaan dan respons (*request & response structure*)
b. Parameter input, baik *path parameters*, *query parameters*, maupun *request body*
c. Format *response codes* (200, 400, 401, 500, dll.)
d. Prosedur penanganan kesalahan (*error handling* guidelines)

Dokumentasi ini sangat penting untuk memastikan bahwa *API* bersifat *developer-friendly* dan dapat digunakan secara konsisten di lingkungan pengembangan maupun produksi.

3) Implementasi *Error Handling* dan *Responsiveness*

Agar *API* tetap andal, dilakukan penerapan strategi *error handling* menyeluruh. Ini mencakup:

a. Penanganan masukan tidak valid (*invalid input validation*)
b. Penanganan kegagalan koneksi dengan *database*
c. Penanganan *unauthorized access* melalui *authentication middleware*

Selain itu, performa *API* dioptimalkan untuk memastikan waktu respons tetap cepat dengan menerapkan teknik *query optimization*, *pagination*, serta pembatasan ukuran data dalam *payload*.

4) Pengujian *API* dan *Monitoring*

Setelah seluruh *endpoint* berhasil diimplementasikan, dilakukan pengujian menyeluruh (*end-to-end testing*) untuk memastikan fungsionalitas, keamanan, dan performa berjalan sesuai rencana. Beberapa jenis pengujian yang dilakukan meliputi:

a. *Unit testing* dan *integration testing*
b. Simulasi *stress testing* menggunakan alat seperti *Postman* Runner atau *JMeter*
c. Implementasi alat pemantauan (*monitoring*) seperti *New Relic* atau *Prometheus* untuk memantau performa *API* di lingkungan produksi

5) Validasi Rancangan *User Interface*

Validasi terhadap *user interface (UI)* dilakukan untuk memastikan bahwa tampilan aplikasi sesuai dengan kebutuhan pengguna dan mendukung pengalaman pengguna (*user experience/UX*) yang optimal. Proses validasi dilakukan melalui:

a. Uji coba prototipe dengan pengguna nyata: Prototipe rancangan dibuat menggunakan alat seperti *Figma*, lalu diuji kepada pengguna dari Divisi SDA untuk mendapatkan masukan langsung terkait *layout*, navigasi, dan kenyamanan visual.
b. Pengujian kegunaan (*usability testing*): Menggunakan metode *think-aloud protocol* untuk memahami bagaimana pengguna menavigasi antarmuka.

c. Penyesuaian iteratif berdasarkan feedback: Desain diperbaiki secara iteratif berdasarkan hasil uji coba dan masukan dari pengguna hingga mencapai kesesuaian antara kebutuhan bisnis dan kenyamanan pengguna.

6) Validasi Rancangan *Basis Data*

Agar performa dan integritas sistem tetap terjaga, rancangan *basis data* juga divalidasi melalui beberapa tahapan penting:

a. Peninjauan skema oleh *developer senior*: Struktur entitas dan relasi antar tabel diperiksa untuk memastikan tidak terjadi *redundansi* dan bahwa semua *foreign keys*, *constraints*, serta indeks telah didefinisikan dengan benar.

b. Pengujian integritas data (*data integrity testing*): Menguji apakah validasi *relational constraints* (misalnya *ON DELETE CASCADE*, *NOT NULL*, dan *UNIQUE*) bekerja sesuai yang diharapkan dalam simulasi penggunaan nyata.

c. Evaluasi performa query: Melakukan uji performa terhadap *SQL query* hasil *ORM Prisma* untuk mengidentifikasi potensi *bottleneck* serta mengoptimalkan *JOIN* dan *indexing*.

Validasi ini sangat penting karena *basis data* merupakan tulang punggung dari aplikasi, dan kesalahan dalam perancangannya dapat berdampak langsung pada kestabilan sistem serta keandalan *API*.

1.3 Tujuan

Penelitian ini bertujuan untuk mengembangkan dan menerapkan *Object-Relational Mapping* (ORM) *Prisma* dalam perancangan *RESTful API* untuk aplikasi web di Divisi SDA PT Telkom Indonesia Tbk. Adapun tujuan spesifik dari penelitian ini adalah sebagai berikut:

1. Untuk menganalisis dan mengidentifikasi permasalahan utama yang dihadapi oleh Divisi SDA dalam proses pengelolaan proyek, khususnya terkait efisiensi, keamanan, dan kemudahan akses data.

2. Untuk mempelajari teknologi *Vue.js*, *Node.js*, dan ORM *Prisma*, serta menerapkannya dalam pengembangan aplikasi berbasis web yang mampu mendukung kebutuhan Divisi SDA secara efektif.

3. Untuk merancang dan mengembangkan *back-end* aplikasi web yang skalabel, responsif, dan aman dengan memanfaatkan ORM *Prisma* untuk memudahkan integrasi dan manajemen database yang efisien.

4. Untuk membangun *RESTful API* yang terintegrasi dengan aplikasi web Divisi SDA guna mendukung komunikasi data yang aman, cepat, dan konsisten antara aplikasi dengan sistem-sistem lain.
5. Untuk mendokumentasikan proses pengembangan API secara terstruktur, yang dapat digunakan sebagai panduan oleh pengembang lain di PT Telkom Indonesia Tbk atau sebagai bahan pelatihan dalam penerapan teknologi serupa.
6. Untuk memberikan contoh nyata penerapan ORM *Prisma* dan *RESTful API* yang dapat dijadikan referensi bagi penelitian dan pengembangan lebih lanjut, baik di lingkungan akademik maupun industri.

1.4 Manfaat

Penelitian ini diharapkan memberikan manfaat yang signifikan, baik secara praktis maupun teoritis, sebagai berikut:

1. **Manfaat Praktis**
    a. Bagi Divisi SDA PT Telkom Indonesia Tbk: Aplikasi web yang dihasilkan dari penelitian ini diharapkan dapat meningkatkan efisiensi operasional Divisi SDA dalam pengelolaan data proyek, memudahkan pemantauan progres proyek, dan mempercepat akses informasi yang akurat dan aman melalui *RESTful API*.
    b. Bagi Pengembang dan Tim Teknologi Informasi di PT Telkom Indonesia Tbk: Dokumentasi dan implementasi API berbasis ORM *Prisma* yang disusun dalam penelitian ini dapat menjadi panduan untuk pengembangan aplikasi serupa di masa mendatang, terutama dalam penerapan ORM untuk meningkatkan kinerja dan keamanan pengelolaan data.
    c. Bagi PT Telkom Indonesia Tbk secara keseluruhan: Hasil penelitian ini dapat menjadi dasar pengembangan standar teknologi di PT Telkom Indonesia, yang dapat diterapkan pada divisi-divisi lain yang membutuhkan sistem pengelolaan data yang terintegrasi, aman, dan efisien, sehingga mendukung transformasi digital yang efektif di seluruh organisasi.
2. **Manfaat Teoritis**

Penelitian ini diharapkan dapat memperkaya literatur terkait penggunaan ORM *Prisma* dalam pengembangan *RESTful API*, khususnya pada konteks pengembangan aplikasi web berbasis data dalam jumlah besar dan real-time.

Hasil penelitian ini dapat berkontribusi pada kajian-kajian di bidang *web development* dan *database management*, sebagai panduan dan referensi bagi penelitian lanjutan yang berfokus pada optimalisasi kinerja dan keamanan aplikasi berbasis web melalui ORM.

Penelitian ini diharapkan memberikan kontribusi dalam pemahaman tentang praktik terbaik (*best practices*) dalam penggunaan ORM dan *RESTful API*, yang dapat dijadikan dasar dalam pengembangan modul pembelajaran atau bahan kuliah di bidang Teknologi Informasi.

3. **Manfaat Bagi Institusi Pendidikan/Kampus**

    a. Bagi Mahasiswa: Penelitian ini dapat menjadi sumber referensi bagi mahasiswa yang ingin memahami penerapan ORM *Prisma* dalam pengembangan aplikasi, serta sebagai contoh implementasi teknologi terkini dalam dunia kerja. Hasil penelitian ini dapat digunakan sebagai panduan dalam tugas akhir atau proyek akademik yang membutuhkan integrasi data dalam aplikasi berbasis web.

    b. Bagi Dosen dan Peneliti: Penelitian ini dapat digunakan sebagai studi kasus dalam proses pembelajaran dan pengajaran teknologi *back-end* di kampus. Selain itu, temuan-temuan dari penelitian ini dapat dijadikan sebagai landasan penelitian lanjutan, yang fokus pada penerapan teknologi ORM atau pengembangan *RESTful API* untuk keperluan penelitian atau publikasi ilmiah.

    c. Bagi Program Studi Teknologi Informasi: Dengan adanya penelitian ini, program studi dapat memperoleh contoh penerapan teknologi yang relevan dan terkini, yang dapat menjadi bahan pengayaan kurikulum di mata kuliah *Web Development*, *Database Management*, atau *Backend Development*. Hasil penelitian ini juga diharapkan dapat mendukung kampus dalam memperkuat kemitraan dengan industri, seperti PT Telkom Indonesia, guna menyediakan akses bagi mahasiswa untuk berpartisipasi dalam proyek nyata dan memahami penerapan teknologi langsung di lapangan.

BAB II
LANDASAN TEORI DAN TINJAUAN PUSTAKA

2.1 *Website*

Website adalah serangkaian halaman digital yang saling terkait dan biasanya di-host pada server, yang dapat diakses oleh pengguna melalui jaringan internet. Setiap *website* dibangun menggunakan kombinasi bahasa pemrograman dasar, seperti *HTML* (HyperText Markup Language) untuk struktur halaman, *CSS* (Cascading Style Sheets) untuk gaya dan tampilan, serta *JavaScript* untuk interaktivitas dan dinamisme (Anderson et al., 2022). Website sendiri telah berkembang secara signifikan dalam beberapa dekade terakhir, dimulai dari *website* statis yang hanya menampilkan informasi sederhana, hingga kini menjadi *website* dinamis yang interaktif dan terintegrasi dengan berbagai aplikasi lain. Website dinamis tidak hanya menyediakan konten tetap tetapi juga memungkinkan pengguna untuk berinteraksi, mengirimkan data, dan melakukan berbagai tindakan yang dikustomisasi sesuai kebutuhan.

Dalam konteks *website* modern, kehadiran berbagai *framework* dan *library* telah memungkinkan pengembang untuk membangun antarmuka pengguna yang lebih interaktif dan responsif. *Vue.js*, *React.js*, dan *Angular* adalah beberapa contoh *framework* yang sering digunakan untuk membangun *frontend* modern. Menurut penelitian Zhang (2023), penggunaan *framework* seperti *Vue.js* dalam pengembangan *frontend* tidak hanya memudahkan manajemen *user interface* tetapi juga memungkinkan pemisahan logika dan desain, sehingga kode menjadi lebih mudah dipelihara. Dengan struktur komponen dalam *Vue.js*, setiap bagian halaman (seperti *header*, *footer*, atau sidebar) dapat diatur secara modular dan independen, yang memudahkan proses pengembangan dan perawatan.

Struktur arsitektur yang baik juga memainkan peran penting dalam *website* untuk meningkatkan performa dan efisiensi. Rivera (2021) mengemukakan bahwa arsitektur yang optimal memungkinkan kecepatan akses yang lebih baik dan pengalaman pengguna yang lancar. Struktur yang rapi tidak hanya meningkatkan *user experience* tetapi juga memudahkan *SEO* (Search Engine Optimization). Dengan adanya elemen-elemen seperti *meta-tags*, *canonical tags*, dan pengaturan kata kunci yang tepat, mesin pencari dapat mengindeks *website* lebih baik, meningkatkan visibilitas situs dalam hasil pencarian. Penelitian Johnson (2021) mengungkapkan bahwa performa *website*, termasuk kecepatan akses, responsivitas, dan kualitas navigasi, sangat memengaruhi *bounce rate* atau tingkat keluarnya pengguna dari situs.

Semakin cepat dan mudah diakses sebuah *website*, semakin besar kemungkinan pengguna tetap berada di situs dan berinteraksi lebih lama.

Pengembangan *website* dinamis saat ini juga semakin tergantung pada integrasi dengan *backend* dan *database*, terutama untuk aplikasi yang melibatkan banyak data atau membutuhkan personalisasi konten. Dengan menggunakan API, khususnya *RESTful API*, *website* modern dapat menampilkan data secara real-time tanpa perlu memuat ulang halaman. Zhang (2023) menyebutkan bahwa integrasi antara *frontend* dan *backend* memungkinkan penyajian data yang lebih cepat, responsif, dan terstruktur, terutama untuk situs e-commerce atau aplikasi yang memerlukan informasi terbaru dan personalisasi. API memungkinkan komunikasi dua arah antara *website* dan server, sehingga memudahkan pengguna untuk berinteraksi dan melakukan tindakan secara langsung.

2.2 Aplikasi Manajemen Proyek

Aplikasi manajemen proyek adalah perangkat lunak yang bertujuan untuk memfasilitasi perencanaan, pelaksanaan, pemantauan, dan pengelolaan proyek secara keseluruhan. Aplikasi ini memungkinkan tim untuk mengoordinasikan tugas, mengalokasikan sumber daya, serta memantau kemajuan proyek, membantu dalam menjaga agar proyek tetap sesuai dengan jadwal dan anggaran yang ditetapkan (Williams et al., 2021). Manajemen proyek yang efektif membutuhkan alat yang dapat mengintegrasikan berbagai aspek proyek, seperti penugasan, pelaporan, dan komunikasi, dalam satu platform.

Penelitian oleh Lee dan Park (2022) menyoroti fitur-fitur utama aplikasi manajemen proyek berbasis *web*, seperti *task management*, *time tracking*, *resource allocation*, dan *performance monitoring*. *Task management* adalah fitur yang memungkinkan pengguna untuk memecah proyek menjadi tugas-tugas kecil dan menugaskannya kepada anggota tim. Fitur ini mempermudah pemantauan tugas individual dan memperjelas peran dan tanggung jawab setiap anggota. Sementara itu, *time tracking* memungkinkan pemantauan waktu yang digunakan untuk setiap tugas atau proyek, membantu tim untuk mengidentifikasi potensi hambatan dan mengelola waktu dengan lebih efektif. Dalam konteks ini, *resource allocation* berperan dalam pengelolaan sumber daya seperti anggaran, peralatan, atau tenaga kerja, memungkinkan pemanfaatan yang optimal. Terakhir, *performance monitoring* berfungsi untuk melacak kinerja tim dan proyek secara keseluruhan, memberikan metrik kinerja untuk membantu tim memahami apakah proyek berada di jalur yang benar.

Aplikasi manajemen proyek yang lebih canggih sering kali terintegrasi dengan *RESTful API* dan *ORM* untuk memfasilitasi akses dan pengelolaan data secara real-time (Kumar et al., 2020). API memungkinkan aplikasi untuk terhubung dengan berbagai sumber data dan mengumpulkan informasi dari sistem eksternal tanpa memerlukan pengelolaan data secara manual. Dalam penelitian oleh Choi (2021), ditemukan bahwa integrasi API dalam aplikasi manajemen proyek mempercepat proses pengambilan data dan meningkatkan akurasi informasi yang ditampilkan. Sebagai contoh, dalam proyek besar, API memungkinkan data mengenai status proyek, anggaran, dan alokasi sumber daya dapat diperbarui secara otomatis dan disajikan kepada pengguna tanpa keterlibatan langsung.

2.3 RESTful API

RESTful API (*Representational State Transfer* API) adalah suatu arsitektur API yang memfasilitasi komunikasi antara sistem atau aplikasi dengan menggunakan metode HTTP. API ini mendukung operasi dasar seperti *CRUD* (Create, Read, Update, Delete), yang memungkinkan pengembang untuk mengelola dan memanipulasi data dengan menggunakan metode HTTP seperti *GET*, *POST*, *PUT*, dan *DELETE* (Fielding, 2000). *RESTful API* memungkinkan aplikasi untuk saling terhubung dan berkomunikasi secara efisien, terlepas dari perbedaan platform atau teknologi yang digunakan (Hill et al., 2022).

Dalam pengembangan aplikasi, *RESTful API* memiliki sejumlah keuntungan. Salah satu keunggulan utamanya adalah fleksibilitas dan skalabilitas, yang memungkinkan berbagai layanan atau komponen sistem berinteraksi secara modular. API yang dirancang dengan baik memungkinkan komunikasi yang lebih efisien antara berbagai sistem, sehingga integrasi dapat dilakukan secara terstruktur dan mudah diperbarui. Martin et al. (2023) menyatakan bahwa API yang baik memungkinkan tim *frontend* dan *backend* untuk bekerja secara independen, sehingga mempercepat pengembangan. Sebagai contoh, sebuah aplikasi e-commerce yang menggunakan *RESTful API* dapat memperbarui data produk dan harga di *backend* tanpa mengubah struktur atau tampilan *frontend*, yang menghemat waktu dan biaya.

Penggunaan *RESTful API* juga memungkinkan penerapan *data caching* di sisi klien, yang berarti data dapat disimpan sementara untuk mengurangi beban pada server. *Data caching* ini memungkinkan pengguna mengakses informasi dengan lebih cepat, terutama jika data tersebut sering diminta. Studi Zhang (2023) menekankan pentingnya *RESTful API* dalam meningkatkan performa dan responsivitas aplikasi, khususnya pada aplikasi berbasis *web* yang

memiliki jumlah pengguna besar. *Data caching* tidak hanya mempercepat akses data tetapi juga meningkatkan stabilitas sistem, mengurangi potensi *downtime* akibat beban berlebih pada server.

Dalam konteks aplikasi manajemen proyek, *RESTful API* memungkinkan data proyek seperti tugas, sumber daya, dan progres dapat diakses dan dimodifikasi secara real-time. Hal ini memungkinkan tim proyek untuk melihat informasi terbaru dan membuat keputusan berdasarkan data yang akurat. Choi (2021) menyebutkan bahwa *RESTful API* yang terintegrasi dengan baik memungkinkan pengguna untuk berinteraksi langsung dengan data tanpa perlu mengakses database atau sistem internal, sehingga menjaga keamanan dan kerahasiaan data.

2.4 *Object-Relational Mapping* (ORM)

Object-Relational Mapping (ORM) adalah teknik pemrograman yang memungkinkan pemetaan data dari *database* relasional ke dalam bahasa pemrograman berbasis objek. Dengan ORM, tabel dalam *database* dapat dipetakan menjadi objek, yang kemudian dapat diakses langsung oleh kode program tanpa perlu menulis *SQL query* secara manual. Turner et al. (2021) menjelaskan bahwa ORM sangat memudahkan pengembang dalam pengelolaan data, karena menghilangkan kompleksitas dalam penulisan dan pemahaman *query SQL*, serta mengurangi risiko kesalahan kode.

ORM menawarkan berbagai keuntungan dalam pengembangan aplikasi, terutama dari sisi efisiensi dan keamanan. Menurut Gomez (2020), ORM memungkinkan pengembang untuk menulis lebih sedikit kode dan meningkatkan produktivitas dengan menghilangkan kebutuhan untuk menulis *query SQL* untuk setiap operasi data. Sebagai contoh, ORM memungkinkan pengembang untuk melakukan operasi *CRUD* (Create, Read, Update, Delete) pada data hanya dengan memanggil fungsi yang sudah tersedia dalam ORM, tanpa perlu memahami detail teknis dari *query SQL*. Ini tidak hanya menghemat waktu pengembangan tetapi juga memastikan bahwa data tetap konsisten dan valid.

Selain itu, ORM juga memiliki fitur *schema migration*, yang memungkinkan pengembang untuk mengubah struktur tabel dalam *database* tanpa harus memodifikasi kode program secara langsung. Fitur ini sangat berguna dalam aplikasi yang membutuhkan penyesuaian struktur data secara berkala, seperti aplikasi manajemen proyek atau e-commerce. Studi Wang et al. (2022) menemukan bahwa ORM dapat mengurangi waktu pengembangan

hingga 30% dibandingkan dengan *raw SQL query*, karena ORM secara otomatis menangani berbagai operasi dasar, seperti validasi data, hubungan antar tabel, dan migrasi skema.

Keunggulan lainnya adalah dalam hal keamanan. ORM mengurangi risiko serangan *SQL injection* dengan mengamankan data melalui objek, bukan *raw SQL query*. Dengan ORM, pengembang dapat menghindari risiko injeksi SQL yang sering kali menjadi kerentanan utama dalam aplikasi *web* (Turner et al., 2021). Selain itu, ORM menyediakan metode validasi data yang otomatis, memastikan bahwa data yang diterima oleh *database* adalah data yang telah diverifikasi.

2.5 *ORM Prisma*

ORM *Prisma* adalah ORM modern yang dirancang khusus untuk *Node.js*, yang menawarkan integrasi yang mudah dengan berbagai *database* dan fitur *type-safe* yang memastikan keandalan dan konsistensi data (Anderson & Smith, 2023). Salah satu keunggulan *Prisma* adalah kemampuannya untuk menghasilkan *schema* yang langsung dapat digunakan dalam aplikasi, yang memudahkan pemetaan data dan meningkatkan efisiensi pengembangan. Fitur *type-safe* pada *Prisma* memungkinkan kode yang ditulis lebih terstruktur, karena memverifikasi tipe data yang digunakan dalam setiap operasi.

Penelitian oleh Brown et al. (2022) menunjukkan bahwa *Prisma* memiliki keunggulan dalam hal kompatibilitas dan performa. *Prisma* memungkinkan pengembang untuk mendefinisikan skema *database* melalui file konfigurasi, yang kemudian diubah menjadi tabel dalam *database*. Ini memungkinkan pengelolaan skema *database* dan migrasi data yang lebih mudah, karena perubahan dalam skema dapat dilakukan secara otomatis tanpa perlu mengubah *SQL query*. Dengan antarmuka yang intuitif, *Prisma* memberikan kemudahan bagi pengembang dalam mengelola skema *database* tanpa memerlukan pengetahuan mendalam tentang SQL.

Prisma juga mendukung integrasi dengan *RESTful API*, sehingga data dapat diakses dan diolah melalui objek yang telah dipetakan dari skema *database*. Menurut Jones (2023), penggunaan ORM *Prisma* dalam aplikasi meningkatkan keamanan dan struktur data karena setiap objek diatur berdasarkan skema yang telah didefinisikan. Dalam pengembangan aplikasi manajemen proyek, *Prisma* memungkinkan akses data proyek, tugas, dan pengguna dilakukan

secara langsung melalui objek-objek ini, yang tidak hanya meningkatkan keamanan tetapi juga mempercepat proses pengembangan aplikasi.

2.6 Pengelolaan Database Relasional dalam Pengembangan API

Pengelolaan *database* relasional adalah proses pengelolaan data dengan menyusun informasi ke dalam tabel-tabel yang saling terhubung melalui relasi yang ditentukan dalam struktur *database*. Dalam sistem ini, data disusun dengan prinsip normalisasi yang memecah data ke dalam tabel-tabel yang lebih kecil untuk meminimalisir redundansi dan meningkatkan integritas data. Setiap tabel memiliki hubungan dengan tabel lain berdasarkan kunci utama (*primary key*) dan kunci asing (*foreign key*), yang membentuk struktur relasional antar data di seluruh *database* (Stevens et al., 2021). Melalui pendekatan relasional ini, data dapat diakses, diubah, dan dikelola secara efisien serta meminimalkan inkonsistensi. Struktur relasional memberikan kemampuan untuk melakukan operasi *CRUD* (Create, Read, Update, Delete) dengan mengoptimalkan integritas dan konsistensi data dalam sistem.

Dalam aplikasi yang kompleks, seperti manajemen proyek, e-commerce, atau sistem ERP (*Enterprise Resource Planning*), pengelolaan *database* relasional sangat penting karena aplikasi-aplikasi tersebut biasanya menyimpan data dalam jumlah besar dan dengan struktur yang rumit. Pengelolaan yang baik memungkinkan akses data yang lebih terstruktur dan mudah diatur, serta memungkinkan pembaruan dan penghapusan data tanpa merusak integritas sistem secara keseluruhan. Pada aplikasi dengan data besar, kemampuan *database* relasional untuk menjaga integritas data dengan konsisten melalui penggunaan transaksi (*transaction*) sangat penting untuk mendukung performa aplikasi (Lin & Lee, 2023).

Penggunaan *Object-Relational Mapping* (ORM) dalam pengelolaan *database* relasional menawarkan berbagai keuntungan, terutama dalam hal efisiensi pengelolaan data, pengurangan kode yang berulang, serta kemudahan dalam pengembangan API. ORM adalah pendekatan yang mengonversi data dalam *database* relasional ke dalam bentuk objek yang dapat diakses dalam kode program. Dengan ORM, pengembang tidak perlu menulis *query SQL* secara manual untuk setiap operasi data, sehingga memudahkan mereka dalam mengelola dan memanipulasi data. ORM memungkinkan pengembang untuk menggunakan fungsi-fungsi yang sudah disediakan, seperti *find*, *save*, atau *delete*, yang secara otomatis akan diterjemahkan menjadi *query SQL* yang kompleks di *backend*.

Lin dan Lee (2023) menunjukkan bahwa penggunaan ORM dalam pengelolaan *database* relasional tidak hanya mempercepat akses data, tetapi juga memudahkan pengembang dalam memetakan data ke dalam kode program melalui model-model yang merepresentasikan setiap tabel dalam *database*. Setiap relasi antar tabel dapat didefinisikan dalam model ORM, yang kemudian diterjemahkan menjadi tabel dan relasi dalam *database*. Sebagai contoh, jika ada tabel "users" dan tabel "projects", ORM memungkinkan relasi "one-to-many" di antara keduanya dengan mendefinisikannya dalam model. Pada akhirnya, ORM akan memastikan bahwa setiap data pengguna terkait dengan proyek-proyek yang dimiliki oleh pengguna tersebut.

ORM seperti *Prisma* semakin menyederhanakan proses ini dengan menyediakan antarmuka yang intuitif dan memungkinkan pengembang mendefinisikan relasi antar tabel secara langsung dalam model. *Prisma* menyediakan fitur *schema definition*, di mana pengembang dapat menuliskan relasi antar tabel seperti "one-to-many" atau "many-to-many" langsung dalam model skema *Prisma*. ORM seperti ini secara otomatis akan melakukan pembaruan pada skema *database* setiap kali ada perubahan model yang dilakukan di kode. Proses ini, yang dikenal sebagai *schema migration*, memungkinkan pengembang untuk memperbarui struktur *database* secara otomatis sesuai dengan kebutuhan aplikasi tanpa perlu menulis ulang *query* atau merusak data yang ada.

Keuntungan lain dari penggunaan ORM dalam *database* relasional adalah kemampuannya untuk meningkatkan keamanan dan konsistensi data. Dengan ORM, setiap operasi yang berhubungan dengan *database* berjalan melalui lapisan objek yang terstruktur, yang membatasi risiko terjadinya kesalahan atau celah keamanan, seperti *SQL injection*. *SQL injection* adalah salah satu bentuk serangan yang memanfaatkan kelemahan dalam *query* SQL untuk mendapatkan akses tidak sah ke *database*. Karena ORM mengelola semua *query* SQL di dalam model yang sudah didefinisikan, risiko *SQL injection* dapat diminimalkan. *Prisma* misalnya, memiliki fitur *type-safe* yang memastikan bahwa setiap data yang dimasukkan ke dalam *database* sesuai dengan tipe yang diharapkan, sehingga mengurangi kemungkinan adanya data yang tidak valid.

Penggunaan ORM seperti *Prisma* dalam integrasi data API juga mempermudah proses pengembangan, terutama dalam aplikasi yang membutuhkan akses data yang cepat dan aman. ORM memfasilitasi integrasi antara *backend* dan *frontend*, memungkinkan komunikasi yang cepat melalui API. Pada aplikasi dengan arsitektur *RESTful API*, ORM menyediakan lapisan

objek yang dapat digunakan untuk memetakan data dalam *database* ke dalam bentuk JSON, yang kemudian dikirimkan ke klien. Hal ini memungkinkan pengembang untuk mengambil data secara langsung dalam format yang mudah diakses, tanpa harus menulis *query SQL* setiap kali permintaan data dilakukan.

Secara keseluruhan, penggunaan ORM dalam pengelolaan *database* relasional membantu pengembangan aplikasi yang kompleks menjadi lebih efisien, terstruktur, dan aman. ORM memudahkan pengembang dalam memetakan data, mengelola skema *database*, dan melakukan migrasi secara otomatis. ORM juga membantu mengurangi risiko kesalahan manusia dalam penulisan *query SQL*, serta meningkatkan keamanan dan performa aplikasi.

2.7 *Keamanan dalam Pengembangan* API

Keamanan merupakan aspek krusial dalam pengembangan *Application Programming Interface* (API), terutama karena API memungkinkan interaksi antara sistem yang berbeda dan sering kali mengakses data sensitif yang memerlukan perlindungan ketat. Dalam konteks ini, keamanan API mencakup tiga komponen utama: otorisasi, autentikasi, dan validasi input, yang bekerja bersama untuk memastikan bahwa hanya pengguna sah yang dapat mengakses atau memodifikasi data yang diperlukan. Otorisasi adalah proses yang memastikan pengguna memiliki hak untuk mengakses sumber daya tertentu, dan ini sangat penting dalam aplikasi yang mengelola data sensitif atau memiliki banyak pengguna dengan hak akses berbeda. Menurut Martin et al. (2021), otorisasi yang tepat dapat mengurangi risiko akses tidak sah ke data dan melindungi data sensitif dalam aplikasi web. Misalnya, dengan menggunakan metode *Role-Based Access Control* (RBAC), sistem dapat menentukan hak akses pengguna berdasarkan peran mereka dalam aplikasi, seperti admin, pengguna biasa, atau tamu, yang memungkinkan sistem untuk membatasi akses secara efektif.

Selain otorisasi, autentikasi merupakan langkah penting yang bertujuan untuk memverifikasi identitas pengguna, memastikan bahwa setiap permintaan yang dilakukan ke API berasal dari sumber yang sah. Salah satu metode autentikasi yang umum digunakan dalam *RESTful API* adalah *JSON Web Token* (JWT), yang memberikan keamanan tambahan melalui pengeluaran token unik untuk setiap sesi pengguna. Token ini bekerja sebagai bukti identitas pengguna dalam setiap permintaan, yang mencegah akses tidak sah karena hanya pengguna yang memiliki token valid yang dapat mengakses API. Nguyen et al. (2022) menjelaskan bahwa JWT memungkinkan sistem untuk tetap aman tanpa harus menyimpan kredensial

pengguna setiap kali permintaan diajukan, memberikan lapisan keamanan tambahan yang sangat berguna dalam aplikasi dengan banyak pengguna aktif.

Selain otorisasi dan autentikasi, validasi input juga memainkan peran penting dalam mengamankan API. Validasi input memastikan bahwa data yang diterima oleh sistem sudah sesuai dengan format yang diinginkan, sehingga mencegah potensi serangan seperti *SQL Injection*, yang dapat merusak atau mengambil data dari database. Dalam hal ini, Liu et al. (2022) menekankan bahwa validasi input membantu menjaga integritas data dan melindungi sistem dari ancaman keamanan. Dengan menggunakan *Object-Relational Mapping* (ORM) seperti *Prisma*, validasi input dapat dilakukan secara otomatis di tingkat model, mengurangi risiko *SQL Injection* karena setiap operasi data dilakukan melalui objek yang sudah terstruktur dan tervalidasi. ORM juga mempermudah implementasi keamanan dengan mengelola data melalui model yang telah disesuaikan dengan skema *database*, yang menjaga konsistensi data sekaligus mematuhi aturan keamanan yang telah ditetapkan.

2.8 Tinjauan Pustaka

Penelitian Martin et al. (2021) berfokus pada pentingnya otorisasi dalam API, terutama untuk aplikasi yang melibatkan banyak pengguna dengan akses terhadap data yang sensitif. Dalam konteks aplikasi berbasis *web*, otorisasi diperlukan untuk membatasi hak akses pengguna, menjaga keamanan data, dan menghindari penyalahgunaan informasi. Penelitian Martin et al. menggarisbawahi pentingnya *Role-Based Access Control* (RBAC) sebagai metode otorisasi yang efektif, di mana hak akses diatur berdasarkan peran pengguna dalam sistem, seperti admin, pengguna biasa, dan tamu. Dengan RBAC, setiap pengguna hanya diberikan akses yang sesuai dengan peran dan tanggung jawabnya, mengurangi risiko akses tidak sah. Namun, penelitian ini tidak menjelaskan secara rinci bagaimana otorisasi dapat diintegrasikan dengan ORM, khususnya dalam konteks ORM *Prisma*, untuk memastikan keamanan API yang lebih baik. Penelitian ini berkontribusi dengan mengintegrasikan otorisasi berbasis RBAC dalam ORM *Prisma* untuk memastikan bahwa hak akses pengguna dapat dikelola secara lebih efektif dalam API yang aman dan efisien. Integrasi ini memberikan kontrol akses yang lebih kuat, di mana otorisasi dikelola langsung di tingkat objek yang dipetakan dalam ORM, mengurangi risiko pelanggaran hak akses.

Nguyen et al. (2022) melakukan penelitian tentang autentikasi API dengan menggunakan *JSON Web Token* (JWT), yang merupakan metode autentikasi tanpa status

(*stateless*) yang populer dalam API berbasis *web*. JWT mengeluarkan token unik yang berfungsi sebagai bukti identitas pengguna, di mana token ini disertakan dalam setiap permintaan API sebagai bentuk autentikasi. JWT memberikan beberapa keuntungan keamanan, seperti meminimalkan risiko penyalahgunaan kredensial karena token hanya berlaku selama waktu tertentu dan setiap token yang tidak valid akan otomatis ditolak. Dengan JWT, server API tidak perlu menyimpan informasi sesi pengguna, yang meningkatkan efisiensi dan skalabilitas API. Namun, penelitian ini tidak mendalami integrasi JWT dengan ORM untuk memastikan bahwa autentikasi dapat dikelola dengan lebih baik pada aplikasi yang memiliki skala data besar dan banyak pengguna. Penelitian ini mengisi kekosongan ini dengan mengimplementasikan JWT bersama dengan ORM *Prisma*, di mana token autentikasi akan dikaitkan langsung dengan objek pengguna dalam ORM, memastikan bahwa hanya pengguna yang terverifikasi yang dapat mengakses data yang sesuai. Integrasi JWT dan ORM dalam penelitian ini memungkinkan autentikasi yang lebih aman dalam API berbasis *web*, serta mengurangi kebutuhan untuk mengelola data sesi yang kompleks.

Validasi input adalah elemen lain yang sangat penting dalam keamanan API, karena validasi input membantu memastikan bahwa data yang diterima oleh API sudah sesuai dengan standar yang ditentukan dan mencegah data berbahaya masuk ke dalam sistem. Liu et al. (2022) mengungkapkan bahwa validasi input yang baik dapat melindungi API dari serangan seperti *SQL Injection*, di mana penyerang menyisipkan perintah SQL berbahaya ke dalam input yang tidak tervalidasi. Dengan menggunakan ORM seperti *Prisma*, validasi input dapat dilakukan secara otomatis, karena ORM menyediakan lapisan validasi bawaan yang mengonversi data menjadi objek yang sesuai dengan model. ORM *Prisma* memungkinkan pengembang untuk menentukan tipe data dan batasan input langsung di dalam model, sehingga data yang tidak sesuai akan otomatis ditolak. Namun, penelitian Liu et al. tidak mengeksplorasi validasi input untuk aplikasi yang membutuhkan pembaruan data real-time dan keamanan data yang konsisten dalam API. Penelitian ini mengisi celah tersebut dengan memanfaatkan fitur validasi bawaan ORM *Prisma* dalam RESTful API, yang memungkinkan setiap data yang diterima selalu tervalidasi dengan baik sebelum disimpan dalam *database*, menjaga integritas data dalam aplikasi manajemen proyek.

Stevens et al. (2021) menyoroti pentingnya pengelolaan *database* relasional dalam pengembangan API, terutama dalam konteks aplikasi yang kompleks. *Database* relasional menyediakan struktur data yang memungkinkan setiap elemen data saling terhubung melalui

relasi yang terdefinisi dengan baik, seperti hubungan antar tabel. Hal ini sangat penting dalam aplikasi berskala besar, di mana data yang terstruktur memungkinkan proses akses dan modifikasi data yang efisien dan aman. Penelitian Stevens et al. menunjukkan bahwa relasi antar tabel memungkinkan integritas data tetap terjaga dan meminimalkan inkonsistensi data, namun penelitian ini tidak membahas penggunaan ORM dalam pengelolaan *database* relasional yang terintegrasi dalam RESTful API. Penelitian ini mengisi kekosongan tersebut dengan menggunakan ORM *Prisma* untuk memetakan relasi antar tabel ke dalam objek di kode program. Dalam penelitian ini, ORM *Prisma* tidak hanya membantu mempercepat akses data tetapi juga memastikan bahwa semua relasi dalam *database* dikelola secara aman dan konsisten dalam API, sehingga data dalam aplikasi manajemen proyek dapat diakses secara terstruktur dan efisien.

Lin dan Lee (2023) mengeksplorasi keuntungan penggunaan ORM dalam pengelolaan *database* relasional, di mana ORM memungkinkan data dalam *database* dipetakan langsung ke dalam kode program sebagai objek. ORM seperti *Prisma* memungkinkan pengembang untuk mendefinisikan relasi antar tabel langsung di dalam model, yang kemudian diterjemahkan menjadi relasi dalam *database* relasional. Ini sangat membantu dalam mengelola data dalam aplikasi yang kompleks, karena pengembang tidak perlu menulis *query SQL* secara manual untuk setiap operasi data. Penelitian ini juga menunjukkan bahwa penggunaan ORM dapat mempercepat proses pengambilan data dan memudahkan integrasi data dalam API, namun tidak membahas bagaimana ORM dapat diterapkan dalam aplikasi yang membutuhkan akses data real-time. Penelitian ini mengisi celah tersebut dengan mengimplementasikan ORM *Prisma* dalam RESTful API, memastikan akses data yang cepat dan aman dalam aplikasi manajemen proyek.

Turner et al. (2021) membahas penggunaan ORM sebagai solusi untuk mengelola data dalam aplikasi skala besar, terutama dalam hal efisiensi dan keamanan data. ORM menyediakan lapisan pemetaan data yang memudahkan pengembang untuk mengelola data tanpa harus menulis *query SQL* secara manual. Dalam ORM, setiap tabel dalam *database* direpresentasikan sebagai objek di dalam kode program, yang memungkinkan operasi data dilakukan langsung melalui objek tersebut. Penelitian Turner et al. menunjukkan bahwa ORM mengurangi kompleksitas pengelolaan data dan meningkatkan keamanan, karena operasi data dilakukan melalui objek yang sudah tervalidasi. Namun, penelitian ini tidak mengeksplorasi bagaimana ORM dapat diintegrasikan dengan RESTful API untuk memfasilitasi pengelolaan

data yang aman dan efisien dalam aplikasi. Penelitian ini berkontribusi dengan menerapkan ORM *Prisma* dalam RESTful API, memanfaatkan model objek untuk memetakan data dalam aplikasi manajemen proyek dengan lebih aman dan efisien.

Brown et al. (2022) mengkaji ORM *Prisma* dalam konteks pengelolaan skema *database* dan fitur *type-safe* yang memungkinkan pengelolaan data yang lebih aman. *Prisma* memungkinkan pengelolaan skema secara otomatis, di mana setiap perubahan dalam struktur data dapat langsung diimplementasikan tanpa perlu melakukan migrasi manual, serta memastikan tipe data yang digunakan sudah sesuai dengan spesifikasi dalam model. Hal ini sangat berguna dalam aplikasi berbasis *web* yang membutuhkan pembaruan data secara dinamis. Namun, penelitian ini tidak mengeksplorasi bagaimana ORM *Prisma* dapat diintegrasikan dalam API yang membutuhkan validasi input ketat untuk menjaga keamanan data. Penelitian ini memperluas konteks Brown et al. dengan menerapkan *Prisma* dalam pengembangan RESTful API yang mengutamakan validasi input secara otomatis, memastikan bahwa setiap data yang diterima API sesuai dengan skema yang telah ditentukan dan memenuhi standar keamanan aplikasi.

Jones (2023) mengkaji keunggulan ORM *Prisma* dalam integrasinya dengan RESTful API, terutama untuk pengelolaan data yang konsisten dan aman. *Prisma* memudahkan pengembang untuk mengelola data dalam aplikasi berbasis API karena memungkinkan data diakses langsung melalui objek yang sudah dipetakan dari model *database*. Penelitian ini menunjukkan bahwa integrasi *Prisma* dengan RESTful API memungkinkan pengelolaan data yang lebih efisien dan aman, namun kurang mendalami penggunaan otorisasi dalam konteks aplikasi yang kompleks. Penelitian ini melengkapi kajian Jones dengan menerapkan otorisasi berbasis peran pada ORM *Prisma*, memastikan bahwa setiap permintaan data diatur sesuai hak akses pengguna, menjaga keamanan dan integritas data dalam aplikasi manajemen proyek.

Choi (2021) membahas pentingnya RESTful API dalam aplikasi manajemen proyek, yang memungkinkan komunikasi data real-time antara berbagai komponen dalam aplikasi. RESTful API menyediakan antarmuka komunikasi yang memungkinkan akses data yang efisien dan memudahkan pengguna dalam melihat dan memperbarui data proyek tanpa harus langsung terhubung dengan *database*. Namun, penelitian ini tidak mengeksplorasi penggunaan ORM untuk mempercepat dan meningkatkan keamanan akses data dalam API. Penelitian ini melengkapi perspektif Choi dengan mengintegrasikan RESTful API dan ORM *Prisma*, memungkinkan akses data yang cepat dan aman sesuai kebutuhan aplikasi manajemen proyek,

di mana setiap permintaan data melalui API dikelola oleh ORM yang memastikan data tervalidasi sebelum disimpan.

Rivera (2021) menyoroti pentingnya arsitektur API yang optimal dalam meningkatkan *user experience*, terutama dalam aplikasi berbasis *web* yang membutuhkan akses data cepat. Rivera menunjukkan bahwa *data caching* dapat membantu meningkatkan performa API dengan menyimpan data yang sering diakses secara sementara di sisi klien. Meskipun demikian, Rivera tidak mengeksplorasi bagaimana *data caching* dapat diintegrasikan dengan ORM dalam aplikasi real-time. Penelitian ini mengisi celah tersebut dengan memanfaatkan ORM *Prisma* yang terintegrasi dengan RESTful API dalam aplikasi manajemen proyek, memungkinkan *data caching* untuk akses data yang lebih cepat tanpa mengorbankan keamanan.

Dari kajian literatur ini, terlihat bahwa meskipun terdapat berbagai penelitian yang membahas aspek keamanan, otorisasi, autentikasi, validasi input, dan pengelolaan *database*, masih ada beberapa celah yang belum terisi, terutama terkait integrasi ORM *Prisma* dalam pengembangan RESTful API yang aman dan efisien di aplikasi manajemen proyek berbasis *web*. Penelitian ini memberikan kontribusi dengan menunjukkan bagaimana ORM *Prisma* dapat digunakan untuk meningkatkan keamanan, efisiensi akses data, dan mendukung fitur-fitur keamanan seperti otorisasi berbasis peran, autentikasi JWT, serta validasi input dalam API. Integrasi ORM *Prisma* dengan RESTful API dalam penelitian ini memberikan solusi komprehensif yang memenuhi kebutuhan aplikasi berbasis *web* yang membutuhkan keamanan tingkat tinggi dalam pengelolaan data yang kompleks.

BAB III
PELAKSANAAN MAGANG

3.1 Tempat dan Waktu Pelaksaan Magang

Program magang ini dilaksanakan di SDA Division PT Telkom Indonesia Tbk, yang berlokasi di alamat lengkap lokasi kantor atau tempat magang]. Divisi SDA PT Telkom Indonesia Tbk merupakan bagian dari perusahaan yang berfokus pada pengembangan dan manajemen sistem data serta aplikasi yang mendukung operasional perusahaan di bidang telekomunikasi. Selama masa magang, peserta memiliki akses langsung ke fasilitas dan infrastruktur perusahaan, yang memberikan pengalaman kerja yang sesuai dengan standar industri.

Pelaksanaan magang dimulai pada tanggal 17 Oktober 2024 dan berlangsung selama enam bulan penuh hingga 15 April 2025. Dalam kurun waktu ini, peserta magang mendapatkan pengalaman kerja yang intensif di lingkungan profesional dengan bimbingan langsung dari para ahli di bidangnya. Jadwal kegiatan magang mengikuti jam kerja perusahaan, yaitu dari hari Senin hingga Jumat, mulai pukul 08.00 WIB hingga 17.00 WIB. Setiap harinya, peserta bekerja di bawah supervisi pembimbing teknis dari perusahaan yang ditunjuk khusus untuk mendampingi dan mengarahkan peserta dalam menjalankan berbagai aktivitas yang telah direncanakan.

Selama jam kerja tersebut, peserta terlibat dalam berbagai aktivitas, mulai dari perkenalan terhadap budaya perusahaan, pemahaman sistem yang digunakan, hingga pelaksanaan proyek pengembangan perangkat lunak. Pembimbing teknis bertanggung jawab untuk memberikan arahan dan supervisi pada setiap tahapan proyek, serta memastikan bahwa peserta memperoleh pemahaman yang baik terhadap proses yang dijalani. Pertemuan evaluasi rutin diadakan setiap akhir pekan untuk mengevaluasi progres yang telah dicapai, mendiskusikan kendala yang mungkin muncul, dan memberikan masukan untuk perbaikan di minggu berikutnya.

Selain kegiatan sehari-hari di kantor, terdapat beberapa sesi pelatihan dan workshop khusus yang diadakan oleh perusahaan. Sesi-sesi ini mencakup pelatihan keterampilan teknis, seperti penggunaan *framework* dan *tools* khusus yang diterapkan dalam proyek, serta pelatihan soft skills, seperti komunikasi dan kolaborasi tim. Setiap peserta magang juga diwajibkan untuk mengikuti kegiatan-kegiatan pengembangan diri ini, yang bertujuan untuk meningkatkan

keterampilan peserta dan membekali mereka dengan pengetahuan tambahan yang relevan untuk mendukung kesuksesan dalam proyek.

Dengan pelaksanaan yang sistematis dan terstruktur ini, magang di SDA Division PT Telkom Indonesia Tbk memberikan pengalaman kerja nyata dalam industri telekomunikasi yang sangat berharga. Kegiatan yang dijalankan tidak hanya berfokus pada pengembangan keterampilan teknis, tetapi juga pada peningkatan kemampuan interpersonal peserta, yang penting untuk membentuk tenaga kerja yang profesional dan siap menghadapi tantangan di dunia kerja yang sesungguhnya.

3.2 Peran dan Tanggung Jawab

Selama program magang di SDA Division PT Telkom Indonesia Tbk, peran yang diemban adalah sebagai seorang *Back-end Developer* dengan tanggung jawab utama pada pengembangan dan implementasi *RESTful API* untuk pengelolaan data sumber daya alam (SDA). *RESTful API* ini dirancang untuk menyediakan antarmuka yang dapat diakses oleh berbagai sistem dan memungkinkan pertukaran data secara aman dan efisien. Dalam proses pengembangan, digunakan *Object-Relational Mapping* (*ORM*) *Prisma*, yang memudahkan interaksi dengan *database* relasional, menjaga integritas data, dan memastikan keamanan serta efisiensi operasional. *Prisma* digunakan sebagai lapisan penghubung antara aplikasi dan *database*, memungkinkan pengelolaan data yang lebih mudah, lebih aman, dan lebih terstruktur. Tanggung jawab sebagai *Back-end Developer* mencakup berbagai aspek penting dari pengembangan *API*, mulai dari desain dan implementasi *CRUD operations*, integrasi *ORM Prisma*, pengujian ketat, dokumentasi menyeluruh, hingga optimasi performa dan keamanan.

Pertama, tanggung jawab utama dalam peran ini adalah mengembangkan *CRUD* (*Create, Read, Update, Delete*) API untuk aplikasi SDA Division. Setiap operasi *CRUD* ini merupakan komponen dasar dalam pengelolaan data, memungkinkan aplikasi untuk membuat, membaca, memperbarui, dan menghapus data SDA secara langsung melalui *API*. Proses pengembangan *CRUD* API dilakukan dengan memanfaatkan metode *HTTP* seperti *POST*, *GET*, *PUT*, dan *DELETE*, yang masing-masing memiliki peran spesifik dalam komunikasi data. Sebagai contoh, metode *POST* digunakan untuk menambahkan data baru, *GET* untuk mengambil data, *PUT* untuk memperbarui data yang ada, dan *DELETE* untuk menghapus data. Dalam mengembangkan *CRUD operations*, perhatian khusus diberikan pada struktur data dan alur proses yang optimal, memastikan bahwa setiap permintaan dan respons memenuhi standar kecepatan dan efisiensi, serta meminimalkan penggunaan sumber daya. Selain itu, desain *API*

mengikuti standar *RESTful*, yang memastikan bahwa *endpoint* API mudah dipahami dan diakses oleh pengembang lain yang mungkin menggunakan API ini di masa mendatang.

Kedua, sebagai bagian dari tanggung jawab pengelolaan data, dilakukan integrasi *ORM Prisma* dengan *database* relasional yang digunakan dalam aplikasi. *Prisma* memberikan kemudahan dalam memetakan tabel dalam *database* ke dalam objek yang dapat digunakan langsung dalam kode program. Dengan ORM *Prisma*, operasi pada *database* dapat dilakukan tanpa harus menulis *query SQL* manual yang kompleks. Misalnya, untuk operasi seperti pembuatan tabel baru, pembaruan data, atau pengelolaan relasi antar tabel, *Prisma* secara otomatis menghasilkan *query* yang sesuai, menghemat waktu pengembangan sekaligus mengurangi risiko kesalahan sintaks. *Prisma* juga dilengkapi dengan fitur *type-safe*, yang memastikan bahwa setiap data yang diproses sudah sesuai dengan tipe data yang didefinisikan dalam model, sehingga meminimalkan risiko kesalahan tipe data dan meningkatkan kestabilan aplikasi. Integrasi *Prisma* juga memungkinkan pengelolaan relasi antar tabel, seperti relasi satu-ke-banyak (*one-to-many*) atau banyak-ke-banyak (*many-to-many*), yang sangat penting dalam aplikasi yang membutuhkan data terstruktur dengan hubungan yang kompleks.

Ketiga, aspek keamanan menjadi fokus utama dalam pengembangan *API*. Sebagai *Back-end Developer*, tanggung jawab utama dalam konteks ini adalah memastikan bahwa setiap *endpoint* API memiliki lapisan keamanan yang memadai. Salah satu metode yang diterapkan adalah autentikasi menggunakan *JSON Web Token* (*JWT*), di mana setiap pengguna yang ingin mengakses API harus melewati proses autentikasi terlebih dahulu. *JWT* bekerja dengan cara menghasilkan token unik yang menyimpan informasi tentang pengguna, seperti identitas dan hak akses mereka. Token ini kemudian disertakan dalam setiap permintaan API sebagai bukti autentikasi. *JWT* juga memiliki masa berlaku yang dapat diatur, sehingga mengurangi risiko penyalahgunaan token dalam jangka panjang. Selain autentikasi, aspek keamanan lainnya adalah otorisasi, yaitu memastikan bahwa hanya pengguna dengan izin yang sesuai yang dapat mengakses atau memodifikasi data tertentu. Otorisasi ini diterapkan melalui kontrol berbasis peran (*Role-Based Access Control* atau *RBAC*), di mana setiap pengguna diberikan peran tertentu (misalnya, admin, editor, atau viewer) yang menentukan hak akses mereka di dalam sistem. RBAC membantu menjaga keamanan data dengan membatasi akses sesuai dengan peran dan tanggung jawab masing-masing pengguna.

Keempat, pengujian dan optimasi *API* merupakan bagian penting dari tanggung jawab sebagai *Back-end Developer*. Pengujian dilakukan dengan menggunakan alat seperti *Postman*

dan *Jest* untuk memastikan bahwa setiap *endpoint* API berfungsi sesuai dengan spesifikasi dan memberikan respons yang benar. Pengujian dilakukan dalam beberapa tahap, termasuk pengujian fungsional untuk memastikan setiap operasi *CRUD* berjalan dengan benar, serta pengujian keamanan untuk mengidentifikasi dan memperbaiki celah yang mungkin ada. Selain itu, dilakukan pengujian beban (*load testing*) untuk mengevaluasi kinerja *API* dalam situasi dengan permintaan tinggi, memastikan bahwa API mampu menangani banyak pengguna tanpa menurunkan performa. Hasil dari pengujian ini digunakan untuk mengoptimalkan *endpoint* dan proses di dalam *API*, seperti memperbaiki waktu respons dan mengurangi penggunaan memori. Selain pengujian, dokumentasi yang menyeluruh juga disusun untuk memudahkan pemeliharaan dan pemahaman *API* di masa depan. Dokumentasi ini mencakup deskripsi setiap *endpoint*, parameter yang dibutuhkan, format respons, serta contoh penggunaan *API*. Dokumentasi yang baik sangat penting untuk memudahkan pengembang lain dalam mengintegrasikan *API* ini ke dalam sistem mereka dan memastikan keberlanjutan proyek.

Kelima, tugas optimasi performa *API* juga menjadi bagian penting dari tanggung jawab. Optimasi dilakukan dengan berbagai cara, mulai dari menyesuaikan arsitektur *database* untuk meningkatkan kecepatan akses data hingga memperbaiki algoritma yang digunakan dalam logika *backend*. Salah satu teknik optimasi yang diterapkan adalah *data caching*, di mana data yang sering diakses disimpan sementara untuk mempercepat waktu respons pada permintaan berikutnya. Selain itu, fitur optimisasi bawaan dari *Prisma* juga dimanfaatkan untuk mempercepat operasi *query* pada *database*. Teknik lain yang digunakan termasuk *pagination* untuk mengelola jumlah data yang ditampilkan dalam satu waktu, serta penyesuaian *indexing* pada *database* untuk mempercepat pencarian data.

Secara keseluruhan, peran sebagai *Back-end Developer* dalam pengembangan *RESTful API* ini memberikan wawasan mendalam tentang pengelolaan sisi server dari aplikasi berbasis *web*, terutama dalam konteks industri yang memerlukan keamanan dan performa tinggi. Dengan memanfaatkan *ORM Prisma* untuk mengelola *database* dan menerapkan standar keamanan yang ketat, *API* yang dikembangkan tidak hanya mampu memenuhi kebutuhan aplikasi SDA Division tetapi juga dapat diandalkan dalam lingkungan produksi yang memiliki banyak pengguna. Keterlibatan dalam berbagai tahapan, mulai dari desain dan implementasi *CRUD API*, integrasi dengan *Prisma*, pengujian dan dokumentasi yang teliti, hingga optimasi performa, memberikan pengalaman berharga dalam proses pengembangan perangkat lunak yang komprehensif dan memenuhi standar industri.

3.3 Metode Pelaksanaan

Proyek pengembangan *RESTful API* ini dilaksanakan dengan menggunakan metode *Scrum*, sebuah kerangka kerja dalam pengembangan perangkat lunak yang bertujuan untuk meningkatkan kolaborasi, transparansi, dan fleksibilitas dalam proses pengembangan. *Scrum* dipilih karena sifatnya yang adaptif, memungkinkan tim untuk merespons perubahan kebutuhan dan memperbaiki produk secara bertahap. Dengan mengadopsi *Scrum*, proyek ini dibagi menjadi beberapa siklus kerja yang disebut *sprint*, di mana setiap *sprint* berlangsung selama dua minggu. Setiap *sprint* memiliki tujuan spesifik dan hasil yang dapat diukur, sehingga memungkinkan tim untuk mengidentifikasi progres yang dicapai serta memperbaiki masalah yang ditemukan di setiap tahap. Pendekatan ini memastikan bahwa pengembangan API dapat berjalan secara efisien dan memenuhi kebutuhan yang ditentukan oleh SDA Division PT Telkom Indonesia Tbk.

Pelaksanaan *Scrum* dimulai dengan tahap *Sprint Planning*, yang dilakukan di awal setiap *sprint*. Dalam pertemuan *Sprint Planning*, seluruh anggota tim, termasuk *Back-end Developer*, *Scrum Master*, dan pemangku kepentingan dari SDA Division, berkumpul untuk menetapkan tujuan *sprint* berikutnya dan mendefinisikan *user stories* serta tugas-tugas yang harus diselesaikan dalam dua minggu ke depan. *User stories* adalah deskripsi singkat tentang kebutuhan pengguna yang dijabarkan dari perspektif pengguna akhir. Misalnya, untuk pengembangan API dalam proyek ini, *user stories* mungkin mencakup kebutuhan seperti, “Sebagai pengguna, saya ingin dapat mengakses data SDA dengan cepat dan aman” atau “Sebagai administrator, saya ingin dapat memperbarui data SDA melalui API dengan autentikasi.” Setiap *user story* kemudian dipecah menjadi tugas-tugas yang lebih kecil dan dimasukkan ke dalam *backlog* untuk *sprint* yang akan datang.

Selama pelaksanaan *sprint*, diadakan pertemuan harian yang disebut *Daily Stand-up*, yang berlangsung selama 15 menit. Dalam pertemuan ini, setiap anggota tim melaporkan progres yang telah dicapai, kendala yang dihadapi, serta rencana kerja untuk hari tersebut. *Daily Stand-up* memungkinkan tim untuk tetap selaras dengan tujuan *sprint* dan mengidentifikasi masalah atau hambatan yang mungkin mengganggu kemajuan proyek. Misalnya, jika ada kendala teknis dalam integrasi *ORM Prisma* dengan *database*, tim dapat segera mendiskusikannya dan menemukan solusi bersama. Transparansi dalam pertemuan ini memungkinkan semua anggota tim untuk mengetahui kondisi proyek secara real-time dan membantu dalam menjaga ritme kerja yang konsisten selama *sprint* berlangsung.

Setelah *sprint* berjalan selama dua minggu, proyek memasuki tahap *Sprint Review*, di mana hasil kerja selama *sprint* dipresentasikan dan dievaluasi. Pada tahap ini, *Back-end Developer* mempresentasikan hasil pengembangan *API* yang telah dicapai, seperti implementasi *CRUD operations*, integrasi *ORM Prisma*, atau pengujian API. Tim bersama pemangku kepentingan kemudian mengevaluasi apakah tujuan *sprint* telah tercapai dan mengidentifikasi area yang memerlukan peningkatan. *Sprint Review* juga memungkinkan tim untuk mendapatkan umpan balik langsung dari pemangku kepentingan, sehingga mereka dapat menyesuaikan pengembangan di *sprint* berikutnya agar lebih sesuai dengan kebutuhan dan harapan pengguna. Umpan balik ini sangat penting dalam menjaga kualitas dan relevansi produk yang dikembangkan, terutama untuk proyek seperti API yang harus memenuhi standar keamanan dan performa tinggi.

Setelah *Sprint Review*, tim melakukan *Sprint Retrospective* yang berfokus pada proses dan kolaborasi tim selama *sprint* tersebut. Dalam *Sprint Retrospective*, setiap anggota tim didorong untuk mengidentifikasi aspek-aspek yang berjalan dengan baik, hal-hal yang perlu diperbaiki, serta ide-ide untuk meningkatkan efektivitas tim di *sprint* berikutnya. Misalnya, jika terdapat hambatan teknis dalam integrasi API dengan *database*, atau kesulitan dalam menjaga waktu respons *endpoint*, maka tim akan berdiskusi tentang cara-cara untuk mengatasi masalah tersebut. *Sprint Retrospective* bertujuan untuk meningkatkan proses kerja tim secara berkesinambungan dan memastikan bahwa setiap *sprint* berikutnya dapat dijalankan dengan lebih efektif.

Dalam proyek ini, setiap *sprint* memiliki fokus dan tujuan yang spesifik, sesuai dengan kebutuhan pengembangan API. Sebagai contoh, *sprint* pertama berfokus pada pengembangan dasar *CRUD operations*, di mana setiap fungsi dasar untuk membuat, membaca, memperbarui, dan menghapus data SDA dikembangkan dan diuji secara internal. *Sprint* kedua mungkin berfokus pada integrasi *ORM Prisma* dengan *database* relasional untuk mempermudah manajemen data dan meningkatkan keamanan. Sementara itu, *sprint* ketiga difokuskan pada pengujian keamanan dan optimasi *endpoint* untuk memastikan bahwa API dapat menangani permintaan dalam jumlah besar tanpa mengalami penurunan performa. Setiap *sprint* disusun untuk membangun fondasi yang kuat bagi pengembangan API secara bertahap, sehingga setiap iterasi menghasilkan produk yang lebih stabil dan siap digunakan.

Keuntungan lain dari penggunaan metode *Scrum* dalam proyek ini adalah fleksibilitas untuk menyesuaikan rencana berdasarkan kebutuhan yang berubah. Karena API merupakan

bagian penting dalam aplikasi yang terus berkembang, sering kali terdapat kebutuhan baru atau perubahan spesifikasi yang perlu segera diakomodasi. Dengan *Scrum*, tim dapat menyesuaikan prioritas dan tujuan pada *sprint* berikutnya untuk memastikan bahwa pengembangan API tetap relevan dan memenuhi kebutuhan terbaru dari SDA Division. Hal ini mencakup adaptasi pada keamanan *endpoint*, tambahan fitur untuk pengelolaan data SDA, atau optimasi performa API.

Seluruh pelaksanaan metode *Scrum* ini difasilitasi oleh peran *Scrum Master*, yang bertanggung jawab untuk memastikan bahwa proses *Scrum* diikuti dengan baik oleh semua anggota tim. *Scrum Master* membantu tim untuk tetap fokus pada tujuan *sprint*, mengatasi hambatan yang mungkin mengganggu alur kerja, serta memastikan bahwa setiap anggota tim memahami tanggung jawab dan tugas mereka. Di akhir setiap *sprint*, *Scrum Master* juga mengevaluasi proses kerja secara keseluruhan dan memberikan saran untuk peningkatan proses di *sprint* berikutnya. Dengan adanya *Scrum Master*, tim pengembangan dapat bekerja dengan lebih terorganisir dan terarah, memastikan bahwa setiap *sprint* memberikan nilai tambah bagi pengembangan API.

Metode *Scrum* yang diterapkan dalam proyek pengembangan API ini tidak hanya meningkatkan kolaborasi tim tetapi juga memungkinkan tim untuk beradaptasi secara cepat terhadap kebutuhan baru dan perubahan spesifikasi. Dengan pembagian proyek menjadi *sprint* yang terstruktur, setiap anggota tim memiliki kejelasan terhadap peran dan tanggung jawab mereka, serta dapat berkolaborasi secara efektif untuk mencapai tujuan bersama. Pendekatan iteratif ini memberikan fleksibilitas dan memungkinkan pengembangan API yang berkualitas tinggi, sesuai dengan standar industri dan kebutuhan khusus dari SDA Division PT Telkom Indonesia Tbk.

3.4 Pelaksanaan Magang

Program magang yang dilaksanakan di PT Telekomunikasi Indonesia Tbk ini berlangsung selama tujuh bulan, dimulai dari 17 Oktober hingga 1 Juli. Aktivitas magang ini mengikuti tahapan manajemen proyek yang terstruktur

3.4.1 *Onboarding* dan Pengenalan Lingkungan Kerja

Tahap *onboarding* dimulai pada tanggal 17 Oktober dan berlangsung selama lima hari. Tahap awal ini berfokus pada perkenalan peserta magang dengan lingkungan kerja di PT Telkom Indonesia Tbk, khususnya di SDA Division. *Onboarding* mencakup serangkaian

kegiatan yang bertujuan untuk memperkenalkan budaya, nilai-nilai, serta struktur organisasi di PT Telkom Indonesia Tbk. Peserta diberikan informasi tentang visi, misi, dan etos kerja perusahaan, yang dirancang untuk membekali mereka dengan pemahaman yang baik mengenai ekspektasi dan standar yang diterapkan di perusahaan.



Gambar 3.1 Onboarding dan Pengenalan Lingkungan

Peserta juga diajak untuk memahami peran SDA Division dalam mendukung berbagai operasi perusahaan, serta kontribusi spesifik divisi ini dalam mencapai tujuan besar PT Telkom Indonesia Tbk di industri telekomunikasi. Selain orientasi mengenai perusahaan, peserta diperkenalkan dengan anggota tim yang akan bekerja bersama mereka selama proyek berlangsung, termasuk supervisor dan mentor yang akan menjadi panduan selama proses magang. Melalui sesi interaktif, diskusi, dan aktivitas *team building*, peserta didorong untuk mengenal satu sama lain dan membangun hubungan kerja yang harmonis, yang esensial untuk kolaborasi yang efektif di fase-fase berikutnya.

Kegiatan *onboarding* juga mencakup pengenalan teknis terhadap proyek yang akan dikerjakan, termasuk pemaparan awal mengenai tujuan proyek, lingkup kerja, serta teknologi dan alat yang akan digunakan. Peserta diberi gambaran tentang alat-alat pengembangan seperti *Prisma ORM*, *RESTful API*, dan alat pengelolaan proyek seperti *Jira*

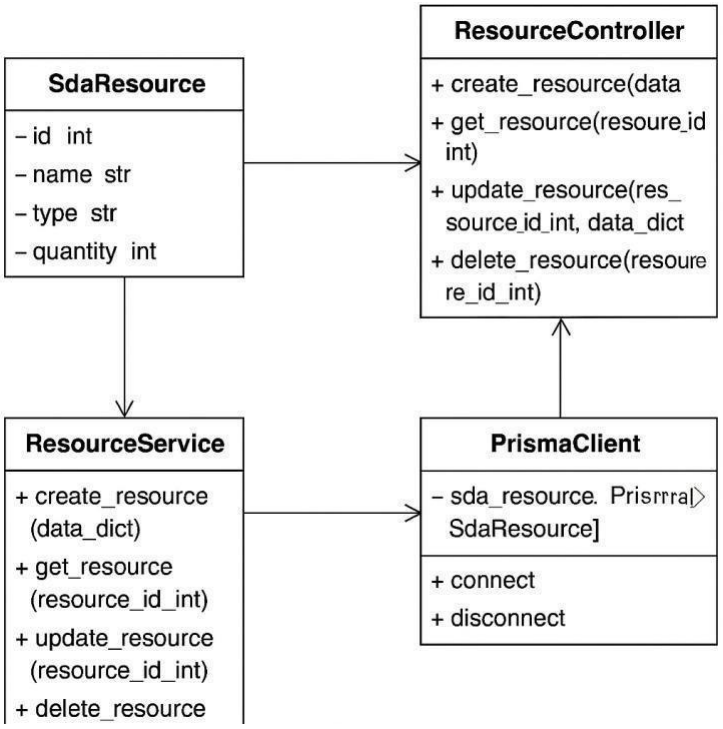
atau *Trello*. Dengan adanya *onboarding* ini, peserta dapat memulai proyek dengan pemahaman yang jelas tentang peran mereka, ekspektasi dari perusahaan, dan lingkungan kerja yang mendukung. Tahap ini juga bertujuan untuk membangun fondasi pengetahuan dasar yang kuat, sehingga peserta lebih siap untuk menangani tantangan teknis yang akan dihadapi dalam proyek.

3.4.2 Memahami Struktur *Database* pada Aplikasi SDA

Setelah menyelesaikan *onboarding*, peserta memasuki tahap berikutnya yaitu memahami struktur *database* yang digunakan dalam aplikasi SDA, yang dimulai pada tanggal 24 Oktober dan berlangsung selama satu bulan. Fase ini sangat penting karena struktur *database* yang baik adalah fondasi dari aplikasi yang efisien dan aman. Peserta mempelajari desain skema *database* SDA secara mendalam, termasuk tabel-tabel yang digunakan, kolom-kolom dalam tabel, dan tipe data yang ada. Selain itu, peserta juga belajar tentang relasi antar tabel dalam *database*, seperti relasi satu-ke-banyak (*one-to-many*), banyak-ke-banyak (*many-to-many*), dan satu-ke-satu (*one-to-one*).

Dalam fase ini, peserta diperkenalkan dengan konsep *normalisasi*, yaitu proses yang digunakan untuk mengatur data dalam *database* agar efisien dan mengurangi redundansi data. Dengan *normalisasi*, *database* dioptimalkan untuk menjaga efisiensi penyimpanan dan kecepatan akses data, sekaligus meminimalkan duplikasi data. Peserta juga belajar cara menjaga integritas referensial dengan memastikan bahwa setiap relasi antar tabel mengikuti aturan kunci asing (*foreign key*) yang tepat. Pemahaman yang baik mengenai desain dan struktur *database* memungkinkan peserta untuk lebih mudah dalam mengembangkan *API* yang dapat berinteraksi dengan *database* secara efisien.

Selain mempelajari struktur dan desain *database*, peserta juga diberikan latihan dalam menulis *query SQL* dasar untuk melakukan operasi *CRUD* (*Create, Read, Update, Delete*) pada data SDA. Peserta diperkenalkan dengan alat seperti *MySQL Workbench* atau *pgAdmin*, yang membantu mereka memahami bagaimana data disimpan, diambil, dan dimodifikasi dalam *database*. Pengetahuan tentang *query SQL* ini akan sangat berguna ketika peserta beralih ke tahap berikutnya, yaitu mengimplementasikan *ORM Prisma* yang secara otomatis dapat memetakan data dalam *database* menjadi objek yang dapat diakses dalam kode.



Gambar 3.1 Skema *Class Diagram*

Class diagram yang ditampilkan pada Gambar 3.1 menggambarkan struktur dan hubungan antar komponen utama dalam sistem manajemen sumber daya SDA (SdaResource). Pada diagram ini, *class* SdaResource merepresentasikan entitas utama dengan atribut dasar seperti *id* bertipe *int*, *name* dan *type* yang bertipe *string*, serta *quantity* bertipe *int* untuk menunjukkan jumlah sumber daya yang tersedia. Selanjutnya, *class* ResourceService berfungsi sebagai lapisan *service* yang mengelola logika bisnis terkait sumber daya, dengan metode CRUD (Create, Read, Update, Delete) yang ditandai oleh fungsi seperti *create_resource(data_dict)*, *get_resource(resource_id_int)*, *update_resource(resource_id_int)*, dan *delete_resource*. Fungsi-fungsi ini bertanggung jawab dalam melakukan operasi terhadap data sumber daya dengan parameter yang jelas dan menggunakan tipe data yang tepat untuk menjaga konsistensi. Di sisi lain, *class* ResourceController bertugas sebagai penghubung antara pengguna atau *client* dengan *service*, menerima permintaan dan meneruskannya ke *ResourceService* melalui metode yang serupa, seperti *create_resource(data)*, *get_resource(resource_id_int)*, *update_resource(resource_id_int, data_dict)*, dan *delete_resource(resource_id_int)*. Kontroler ini juga berfungsi untuk mengatur alur data masuk dan keluar sehingga aplikasi dapat berjalan secara terstruktur dan responsif. Terakhir, PrismaClient berperan sebagai *ORM* (Object-Relational Mapping) yang menghubungkan aplikasi dengan database, di

mana entitas sda_resource pada PrismaClient merupakan representasi dari *class* SdaResource di dalam sistem yang memungkinkan operasi *connect* dan *disconnect* ke database secara efisien. Penggunaan PrismaClient ini memberikan kemudahan dalam melakukan manipulasi data database melalui objek-objek dalam kode aplikasi tanpa harus menulis query SQL secara manual, sekaligus memastikan integritas dan keamanan data melalui fitur-fitur modern seperti *type-safe*. Dengan skema *class diagram* ini, keseluruhan arsitektur sistem menjadi lebih modular, mudah dikembangkan, dan terjaga maintainability-nya karena setiap komponen memiliki tanggung jawab yang terdefinisi dengan baik.

3.4.3 Validasi Rancangan Antarmuka dan Basis Data

Sebelum memasuki tahap implementasi pengembangan RESTful API, dilakukan serangkaian proses *validation* yang menyeluruh terhadap rancangan teknis, khususnya pada *user interface* (UI) dan rancangan basis data. Validasi ini bertujuan untuk memastikan bahwa kedua rancangan tersebut benar-benar memenuhi kebutuhan pengguna serta dapat diimplementasikan secara efektif dalam sistem yang dikembangkan di PT Telekomunikasi Indonesia Tbk, khususnya di SDA Division.

1. Validasi Rancangan Antarmuka Pengguna (*User Interface/UI*)

Validasi rancangan UI dilakukan secara komprehensif untuk memastikan desain memenuhi prinsip-prinsip *usability*, *accessibility*, dan *user experience* (UX) yang baik. Proses ini dimulai dengan sesi *review* desain yang melibatkan pembimbing lapangan dari tim pengembang PT Telkom Indonesia Tbk, serta pihak terkait yang berperan sebagai *stakeholders* pengguna akhir.

Review desain mencakup pemeriksaan aspek-aspek berikut:

a. Konsistensi tata letak visual dan elemen grafis, agar sesuai dengan *corporate branding* PT Telkom Indonesia Tbk.

b. Kelengkapan fitur dan elemen interaktif yang mendukung kebutuhan operasional SDA Division, seperti formulir input data, tampilan daftar resource, dan mekanisme navigasi yang intuitif.

c. Responsivitas desain pada berbagai perangkat, terutama pada *mobile* dan *desktop* yang umum digunakan oleh pengguna.

d. Kepatuhan terhadap standar *accessibility* agar aplikasi dapat diakses dengan mudah oleh semua pengguna, termasuk mereka yang memiliki keterbatasan fisik.

Selain sesi *review*, dilakukan pula *interactive simulation testing* menggunakan prototipe UI berbasis *wireframe* atau *mockup* yang dikembangkan menggunakan alat bantu seperti Figma atau Adobe XD. Simulasi ini memungkinkan peserta magang dan tim pembimbing untuk melakukan *usability testing* awal, mengidentifikasi potensi masalah seperti *user flow* yang membingungkan, *button placement* yang tidak efektif, serta kemungkinan *user error*. Feedback yang diperoleh dari simulasi ini kemudian didokumentasikan secara rinci dan digunakan untuk melakukan perbaikan desain sebelum masuk ke tahap pengembangan kode.

2. Validasi Rancangan Basis Data *(Database Design)*

Validasi rancangan basis data dilakukan untuk memastikan struktur data yang telah dirancang mampu memenuhi kebutuhan penyimpanan dan pengelolaan data SDA dengan efisien dan aman. Tahap ini meliputi pemeriksaan menyeluruh terhadap aspek-aspek berikut:

a. Skema database secara keseluruhan, termasuk definisi tabel, kolom, dan tipe data yang sesuai dengan kebutuhan aplikasi.

b. Implementasi prinsip *database normalization* untuk menghilangkan redundansi data dan menjaga konsistensi antar tabel.

c. Hubungan antar tabel yang direpresentasikan dengan *foreign key constraints*, seperti relasi *one-to-many*, *many-to-many*, dan *one-to-one*, yang sudah sesuai dengan kebutuhan relasi data dalam aplikasi SDA.

d. Pengaturan *indexing* pada kolom-kolom yang sering diakses untuk meningkatkan performa *query* database.

e. Penentuan tipe data dan panjang kolom yang tepat agar menghindari pemborosan ruang penyimpanan dan meningkatkan efisiensi akses data.

Proses validasi ini dilakukan bersama pembimbing lapangan dengan menggunakan alat bantu seperti MySQL Workbench dan pgAdmin, yang memungkinkan untuk melakukan *visual schema review* dan pengujian skenario *CRUD* (*Create*, *Read*, *Update*, *Delete*) menggunakan data contoh. Pengujian ini bertujuan untuk memeriksa integritas

data, kecepatan akses, serta memastikan tidak adanya konflik atau inkonsistensi data yang dapat mengganggu proses pengembangan API selanjutnya.

Selain itu, dilakukan analisis skenario *transaction management* dan *concurrency control* untuk memastikan bahwa rancangan basis data dapat menangani akses data simultan tanpa terjadi deadlock atau inkonsistensi, hal yang krusial dalam lingkungan produksi.

Semua temuan dan rekomendasi dari proses validasi rancangan UI dan basis data didokumentasikan secara rinci dalam laporan evaluasi. Dokumentasi ini mencakup deskripsi masalah yang ditemukan, usulan perbaikan teknis, serta rencana tindak lanjut untuk memastikan perbaikan dilakukan secara menyeluruh. Laporan ini juga menjadi referensi utama untuk memastikan bahwa rancangan teknis yang telah tervalidasi dapat langsung diimplementasikan dengan minim risiko kegagalan.

Proses validasi rancangan ini menjadi fondasi utama yang meminimalisir risiko kegagalan di tahap implementasi, serta memastikan produk akhir yang dikembangkan dapat memenuhi standar kualitas tinggi yang diharapkan oleh PT Telkom Indonesia Tbk, khususnya dari sisi *user satisfaction*, *system reliability*, dan *data integrity*.

Proses validasi rancangan antarmuka pengguna (*user interface/UI*) dan rancangan basis data dilakukan secara menyeluruh oleh penulis bersama pembimbing lapangan dari tim developer Divisi SDA PT Telekomunikasi Indonesia Tbk. Validasi ini dilakukan secara bertahap, dimulai dari sesi *review* desain yang melibatkan pemeriksaan aspek *usability*, *accessibility*, konsistensi visual, dan kesesuaian fitur UI dengan kebutuhan pengguna, serta pengujian prototipe menggunakan alat seperti Figma untuk memastikan *user experience* yang optimal. Pada rancangan basis data, validasi mencakup evaluasi skema database meliputi definisi tabel, tipe data, relasi antar tabel (*foreign key*), dan prinsip *normalization* untuk menjaga efisiensi dan integritas data. Pengujian juga dilakukan menggunakan MySQL Workbench dan pgAdmin dengan simulasi operasi *CRUD* untuk memastikan struktur data memenuhi kebutuhan aplikasi SDA secara efektif dan aman. Seluruh hasil validasi didokumentasikan dan menjadi acuan sebelum memasuki tahap implementasi, sehingga rancangan teknis yang telah tervalidasi dapat mengurangi risiko kesalahan dan mendukung pengembangan sistem yang handal dan sesuai standar perusahaan.

Pengujian rancangan basis data dilakukan dengan pembuatan *mock data* yang mensimulasikan kondisi nyata untuk menguji kelayakan struktur tabel, integritas *foreign*

key constraints, serta performa operasi *CRUD* pada database. Pengujian ini bertujuan memastikan bahwa rancangan basis data dapat menampung data secara konsisten dan aman, serta mendukung kebutuhan aplikasi secara optimal tanpa adanya inkonsistensi atau *redundancy* data. Sementara itu, pengujian rancangan antarmuka pengguna (*user interface/UI*) dilakukan melalui metode *prototype testing* menggunakan Figma, di mana sejumlah calon pengguna dari tim internal melakukan simulasi alur navigasi, interaksi elemen UI, dan memberikan *feedback* langsung terkait kenyamanan penggunaan, kemudahan akses, serta estetika desain. Metode ini membantu mengidentifikasi potensi masalah *usability* dan melakukan penyempurnaan sebelum tahap implementasi penuh, sehingga rancangan antarmuka yang dihasilkan lebih *user-friendly* dan sesuai dengan kebutuhan pengguna akhir.

3.4.4 Implementasi *ORM Prisma* untuk *Database* SDA

Pada tahap ini, peserta mulai mengimplementasikan *Object-Relational Mapping* (*ORM*) *Prisma* pada *database* SDA, yang berlangsung selama dua minggu mulai dari tanggal 1 Desember. *Prisma* adalah alat ORM modern yang memungkinkan interaksi dengan *database* relasional secara lebih efisien dan terstruktur. Implementasi *Prisma* memudahkan peserta dalam mengelola *database*, karena setiap operasi data dapat dilakukan melalui objek-objek dalam kode, tanpa harus menulis *query SQL* manual yang kompleks.

Peserta belajar bagaimana *Prisma* memetakan tabel-tabel dalam *database* ke dalam model yang dapat diakses langsung di dalam aplikasi. Dengan menggunakan *schema definition* pada *Prisma*, peserta dapat mendefinisikan tabel, kolom, dan relasi antar tabel secara deklaratif dalam file konfigurasi, yang kemudian diterjemahkan menjadi tabel dalam *database*. Selain itu, peserta memanfaatkan fitur *type-safe* pada *Prisma* yang memungkinkan validasi tipe data secara otomatis, sehingga data yang diambil atau disimpan ke dalam *database* sudah sesuai dengan tipe yang diharapkan, mengurangi risiko kesalahan dan meningkatkan keandalan aplikasi.

Fase ini juga mencakup integrasi *Prisma* dengan *CRUD API* yang akan dikembangkan. Setiap operasi *CRUD* untuk data SDA (seperti membuat, membaca, memperbarui, dan menghapus data) diimplementasikan menggunakan *Prisma*, yang secara otomatis menghasilkan *query* yang efisien sesuai dengan spesifikasi yang diberikan. Dengan *Prisma*,

peserta dapat melakukan operasi data yang kompleks dengan sintaks yang lebih sederhana dan aman, menghemat waktu pengembangan dan memudahkan pemeliharaan di masa depan.

3.4.5 Pengembangan dan Pembuatan *RESTful API*

Tahap pengembangan dan pembuatan *RESTful API* dimulai pada tanggal 15 Januari dan berlangsung selama 3,5 bulan. Pada fase ini, peserta bertanggung jawab untuk mengembangkan *endpoint API* yang memungkinkan aplikasi untuk berkomunikasi dengan *database* secara aman dan efisien. *RESTful API* ini menggunakan metode *HTTP* seperti *POST*, *GET*, *PUT*, dan *DELETE* untuk melakukan berbagai operasi data, termasuk operasi *CRUD* untuk mengelola data SDA.

Selain pengembangan fungsi dasar, peserta juga fokus pada penerapan standar keamanan dalam setiap *endpoint API*. Salah satu langkah keamanan yang diterapkan adalah autentikasi menggunakan *JSON Web Token* (*JWT*), yang memastikan bahwa hanya pengguna yang terverifikasi yang dapat mengakses data tertentu. *JWT* memberikan token unik untuk setiap sesi pengguna, yang kemudian disertakan dalam setiap permintaan ke *API*. Selain itu, peserta juga menerapkan otorisasi berbasis peran (*Role-Based Access Control* atau *RBAC*) untuk mengatur hak akses sesuai peran pengguna, seperti admin atau pengguna biasa, guna memastikan bahwa data hanya dapat diakses atau dimodifikasi oleh pengguna yang memiliki izin.

Pada fase ini, peserta melakukan pengujian unit untuk setiap *endpoint* yang dikembangkan, memastikan bahwa setiap fungsi berjalan sesuai dengan spesifikasi dan memberikan respons yang sesuai. Pengujian ini membantu dalam mendeteksi bug atau kesalahan lebih awal, sehingga mempercepat proses perbaikan dan mengurangi risiko masalah pada tahap akhir proyek.

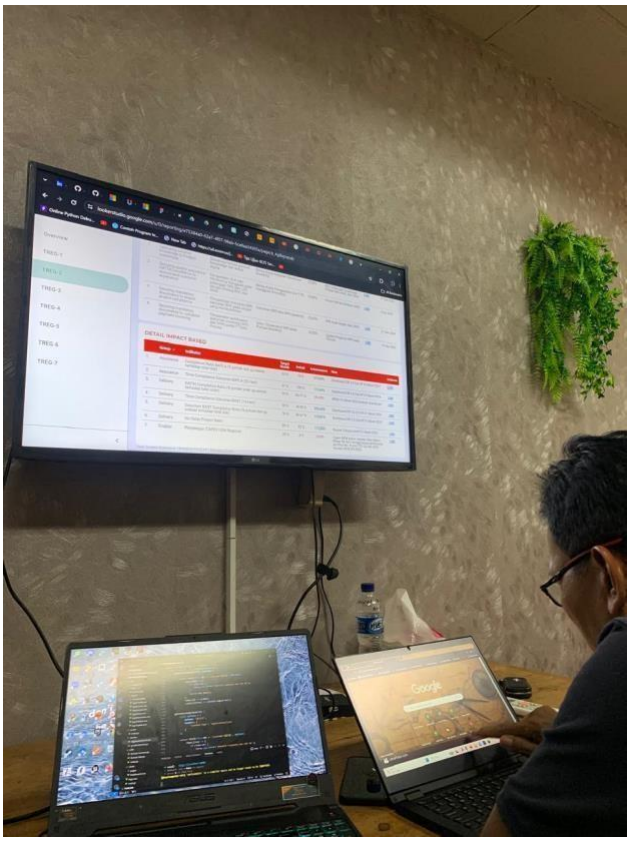
3.4.6 *Testing* dan Optimisasi *API* dengan *Prisma*

Setelah pengembangan *API* selesai, peserta memasuki tahap *testing* dan optimisasi, yang dimulai pada tanggal 1 Mei dan berlangsung selama satu bulan. Pada fase ini, dilakukan pengujian komprehensif untuk memastikan bahwa setiap *endpoint* API berjalan dengan baik dan aman. Pengujian dilakukan menggunakan alat seperti *Postman* dan *Jest*, yang memungkinkan peserta untuk memeriksa setiap fungsi API, melakukan pengujian

fungsional, dan melakukan *load testing* untuk mengevaluasi performa API di bawah beban tinggi.

Optimisasi juga menjadi fokus utama pada tahap ini. Peserta memanfaatkan fitur *Prisma* untuk mempercepat waktu akses data dengan memanfaatkan mekanisme *caching* serta teknik *pagination* untuk mengelola jumlah data yang ditampilkan. Selain itu, optimisasi pada level *database* juga dilakukan, seperti menambahkan indeks pada kolom yang sering diakses untuk mempercepat proses pencarian data. Tujuan dari optimisasi ini adalah untuk memastikan bahwa API mampu memberikan waktu respons yang cepat dan mendukung pengguna dalam jumlah besar tanpa terjadi penurunan performa.

3.4.7 *Deployment* dan Dokumentasi *API*



Gambar 3.2 Deployment dan Dokumentasi API

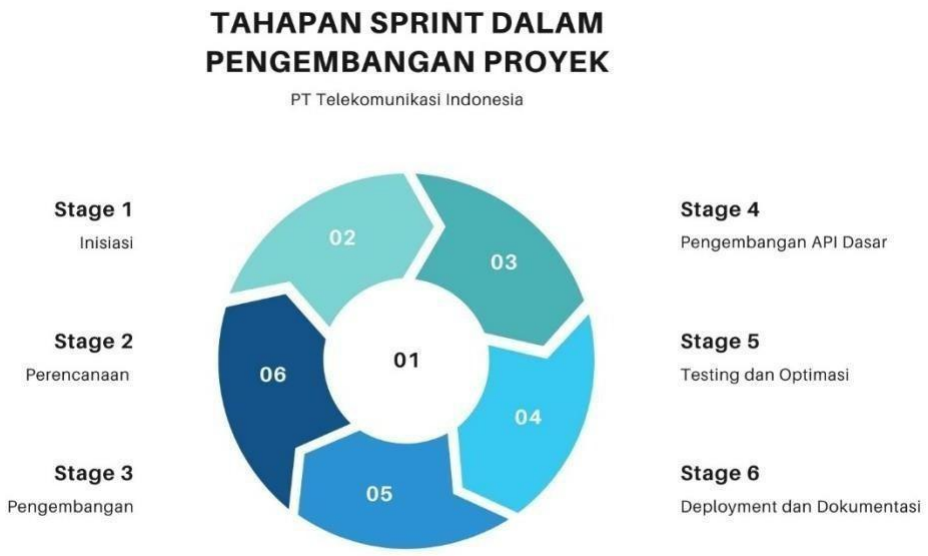
terakhir adalah *deployment* dan dokumentasi *API*, yang dimulai pada tanggal 1 Juni dan berlangsung selama dua minggu. Pada tahap ini, *API* yang telah dikembangkan dan diuji diterapkan ke lingkungan produksi, sehingga dapat diakses oleh pengguna akhir. Peserta bertanggung jawab untuk merencanakan dan menjalankan proses *deployment* dengan hati-hati, memastikan bahwa tidak ada gangguan pada layanan selama transisi ke lingkungan produksi.

Selain *deployment*, dokumentasi menyeluruh disusun untuk setiap *endpoint API*, mencakup deskripsi fungsi, parameter yang dibutuhkan, format permintaan dan respons, serta contoh penggunaan. Dokumentasi ini sangat penting untuk memudahkan tim lain dalam memahami dan menggunakan *API* yang dikembangkan. Dokumentasi juga mencakup panduan pemeliharaan dan cara menangani error yang mungkin muncul di lingkungan produksi. Dengan dokumentasi yang baik, tim pengembang berikutnya dapat dengan mudah memahami struktur dan fungsi *API*, sehingga memudahkan proses pemeliharaan dan pengembangan lebih lanjut di masa mendatang.

Keseluruhan tahapan ini memastikan bahwa pengembangan *RESTful API* dilakukan secara sistematis dan terstruktur, menghasilkan API yang memenuhi kebutuhan SDA Division PT

Telkom Indonesia Tbk dengan standar kualitas yang tinggi, baik dari sisi keamanan, efisiensi, maupun kemudahan penggunaan.

3.5 Tahapan *Sprint* dalam Pengembangan Proyek



Gambar 3.3 Tahapan Sprint Dalam Pengembangan ProyekTabel 3 1

3.5.1 *Sprint* 1 – Inisiasi

Tahap pertama dalam pengembangan proyek ini adalah *Sprint* Inisiasi, yang dimulai pada tanggal 17 Oktober dan berlangsung selama lima hari. *Sprint* ini berfokus pada *onboarding* dan pengenalan lingkungan kerja di PT Telkom Indonesia Tbk, khususnya di SDA Division. Tahap ini sangat penting untuk memberikan pemahaman awal bagi peserta magang mengenai visi, misi, nilai-nilai, dan struktur organisasi perusahaan. Peserta diperkenalkan pada tujuan utama SDA Division dan peran yang diemban oleh divisi ini dalam mendukung operasi dan inovasi teknologi di PT Telkom Indonesia Tbk.

Selama fase *onboarding*, peserta menjalani orientasi yang mencakup pengenalan terhadap kebijakan perusahaan, etika kerja, dan prosedur standar operasional (SOP) yang harus diikuti selama masa magang. Peserta juga diperkenalkan pada sistem komunikasi dan kolaborasi yang akan digunakan selama proyek berlangsung, seperti alat manajemen

proyek (*Jira* atau *Trello*) yang akan membantu dalam melacak tugas dan kemajuan pekerjaan, serta alat komunikasi tim (*Microsoft Teams*, *Slack*, atau *Zoom*) untuk koordinasi antar anggota tim. Selain itu, peserta diberikan akun untuk mengakses repositori kode perusahaan di platform seperti *GitHub* atau *GitLab*, yang digunakan untuk mengelola kontrol versi dan kolaborasi kode secara terstruktur.

Di tahap ini, peserta juga berkesempatan untuk berkenalan dengan anggota tim yang akan bekerja bersama mereka, termasuk mentor, supervisor, dan rekan kerja lainnya. Pengenalan ini dilakukan melalui sesi perkenalan dan kegiatan *team building* yang dirancang untuk membangun hubungan kerja yang solid dan saling percaya antar anggota tim. Memiliki hubungan yang baik dengan rekan kerja diharapkan dapat membantu peserta dalam beradaptasi lebih cepat dan memudahkan mereka untuk meminta bantuan atau klarifikasi ketika menghadapi kendala.

Selain itu, peserta mendapatkan penjelasan mendetail mengenai proyek yang akan mereka kerjakan, termasuk ruang lingkup, tujuan, dan hasil yang diharapkan. Proyek ini bertujuan untuk mengembangkan dan mengimplementasikan *RESTful API* untuk pengelolaan data SDA menggunakan *Object-Relational Mapping* (*ORM*) *Prisma*. Pada akhir *sprint* ini, peserta diharapkan memiliki pemahaman yang jelas mengenai konteks proyek, peran dan tanggung jawab mereka, serta ekspektasi yang harus dipenuhi selama masa magang. Output dari *sprint* ini adalah kesiapan peserta untuk memasuki fase berikutnya dengan pemahaman dasar yang kuat mengenai lingkungan kerja dan proyek yang akan dikerjakan.

3.5.2 *Sprint* 2 – Perencanaan

Tahap kedua adalah *Sprint* Perencanaan, yang dimulai pada tanggal 24 Oktober dan berlangsung selama satu bulan. Pada tahap ini, peserta melakukan analisis kebutuhan proyek untuk memperoleh pemahaman yang lebih dalam tentang struktur *database* dan spesifikasi API yang akan dikembangkan. Langkah pertama dalam *sprint* ini adalah mengidentifikasi kebutuhan pengguna dan sistem melalui diskusi dengan pemangku kepentingan, seperti manajer proyek, pengguna akhir, dan anggota tim lainnya. Dalam diskusi ini, peserta berupaya untuk memahami masalah yang dihadapi oleh pengguna akhir dan bagaimana API yang akan dikembangkan dapat membantu dalam menyelesaikan masalah tersebut.

Selanjutnya, peserta mempelajari konsep dasar *database* yang digunakan di SDA Division, termasuk struktur tabel, jenis data yang dikelola, serta relasi antar tabel yang ada. Dalam fase ini, peserta melakukan analisis terhadap struktur *database* yang dibutuhkan untuk menyimpan data SDA. *Database* harus dirancang dengan mempertimbangkan berbagai faktor seperti integritas data, efisiensi penyimpanan, dan kecepatan akses. Untuk itu, peserta memanfaatkan konsep *normalisasi* dalam mendesain *database*, yang membantu dalam mengurangi redundansi data dan memaksimalkan efisiensi penyimpanan.

Dalam perencanaan ini, peserta juga membuat *Entity Relationship Diagram* (ERD) yang menunjukkan entitas utama dalam *database*, kolom-kolom dalam tabel, dan hubungan antar entitas. ERD ini merupakan representasi visual dari struktur data yang akan diimplementasikan, yang sangat penting untuk memudahkan pemahaman tentang bagaimana data akan disimpan dan dikelola. Peserta mempelajari berbagai jenis relasi, seperti relasi satu-ke-banyak (*one-to-many*), banyak-ke-banyak (*many-to-many*), dan satu-ke-satu (*one-to-one*), yang akan menjadi dasar dalam menyusun skema *database*.

Selain itu, peserta menyusun dokumen spesifikasi API yang mencakup definisi *endpoint*, metode *HTTP* yang digunakan (*GET*, *POST*, *PUT*, *DELETE*), serta parameter yang diperlukan untuk setiap operasi. Setiap *endpoint* API dirancang untuk menangani permintaan tertentu dari klien, seperti pengambilan data, penyimpanan data baru, pembaruan data, dan penghapusan data. Dokumen spesifikasi ini akan menjadi panduan utama dalam tahap pengembangan, memastikan bahwa API yang dibuat dapat memenuhi kebutuhan pengguna. Output dari *sprint* ini adalah ERD, spesifikasi API awal, dan analisis kebutuhan yang komprehensif untuk menjadi acuan di *sprint* berikutnya.

3.5.3 *Sprint* 3 - Pengembangan

Sprint ketiga berfokus pada pengembangan awal dengan melakukan *setup ORM Prisma* dan mengintegrasikannya dengan *database* yang telah dirancang. Tahap ini dimulai pada 1 Desember dan berlangsung selama dua minggu. *ORM Prisma* digunakan sebagai lapisan penghubung antara aplikasi dan *database* relasional, memungkinkan peserta untuk mengelola data dengan lebih efisien tanpa perlu menulis *query SQL* secara manual. *Prisma* memungkinkan pemetaan data dalam tabel menjadi objek yang dapat diakses langsung dalam kode program.

```
generator client {
```

```
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url      =
env("mysql://username:password@localhost:5432/nama_databas
e")
}
```

Tabel 3.1 Konfigurasi *schema Prisma*

Peserta mulai dengan membuat *schema* dalam *Prisma* yang mencakup definisi tabel, kolom, dan relasi antar tabel. *Schema* ini merupakan representasi dari struktur *database* yang sudah dirancang dalam ERD. Setiap entitas dalam *schema* mewakili tabel di dalam *database*, dan setiap kolom pada entitas mewakili kolom dalam tabel. *Prisma* juga dilengkapi dengan fitur *type-safe*, yang memungkinkan data yang dimasukkan dan diambil dari *database* sesuai dengan tipe data yang telah ditentukan, sehingga mencegah terjadinya kesalahan tipe data yang sering menyebabkan bug.



Gambar 3.4 *Entity Relationship Diagram*

ERD (Entity Relationship Diagram) yang ditampilkan menunjukkan struktur data untuk pengelolaan proyek dengan beberapa entitas utama yang berhubungan satu sama lain. Berikut adalah penjelasan untuk masing-masing entitas dan relasi antar-entitas dalam ERD ini:

1. **Entitas `unit`**
    - **Atribut:**
        - `id` (int): Kunci utama yang mengidentifikasi setiap unit secara unik.

- `nama` (string): Nama dari unit.

- **Relasi:**
  - Memiliki hubungan dengan entitas `divisi`, menunjukkan bahwa setiap `unit` terkait dengan satu `divisi`.

2. **Entitas `divisi`**
   - **Atribut:**
     - `id` (int): Kunci utama yang mengidentifikasi setiap divisi.
     - `nama` (string): Nama dari divisi.
   - **Relasi:**
     - Terhubung dengan `unit`, menunjukkan bahwa sebuah `divisi` memiliki satu atau beberapa `unit`.
3. **Entitas `unit_proyek`**
   - **Atribut:**
     - `id` (int): Kunci utama untuk mengidentifikasi unit proyek.
     - `nama` (string): Nama dari unit proyek.
   - **Relasi:**
     - Berelasi dengan entitas `proyek`, menunjukkan bahwa sebuah `unit_proyek` mungkin terlibat dalam sebuah `proyek`.
4. **Entitas `proyek`**
   - **Atribut:**
     - `id` (int): Kunci utama untuk setiap proyek.
     - `name` (varchar): Nama proyek.
     - `category` (enum): Kategori proyek, kemungkinan disimpan dalam bentuk *enumeration* untuk pilihan yang terbatas.
     - `document_status` (enum): Status dokumen proyek, juga disimpan dalam bentuk *enumeration*.

- `value` (int): Nilai dari proyek.
- `request_date` (date): Tanggal permintaan proyek.
- `no_order` (varchar): Nomor order terkait proyek.
- `quote_number` (varchar): Nomor kutipan terkait proyek.
- `no_contract` (varchar): Nomor kontrak untuk proyek.
- `idLop` (varchar): ID terkait LOP (Line of Project).
- `last_position` (varchar): Posisi terakhir proyek.
- `next_position` (varchar): Posisi berikutnya dari proyek.
- `schema_active` (varchar): Status schema yang aktif.
- `customer_id` (int): Merujuk ke `id` dari entitas `customer`.

o **Relasi:**

- Terhubung dengan `customer` melalui `customer_id`, menunjukkan bahwa sebuah proyek dimiliki atau dikaitkan dengan satu `customer`.
- Berhubungan dengan `unit_proyek`, yang menunjukkan bahwa setiap proyek memiliki keterlibatan dari `unit_proyek` tertentu.

5. **Entitas `skema`**

o **Atribut:**

- `id` (int): Kunci utama yang mengidentifikasi setiap skema.
- `nama` (string): Nama dari skema.

o **Relasi:**

- Berhubungan dengan entitas `proyek`, mungkin untuk mengindikasikan bahwa setiap proyek memiliki skema yang relevan.

6. **Entitas `customer`**

o **Atribut:**

- `id` (int): Kunci utama untuk mengidentifikasi setiap customer.
- `name` (string): Nama dari customer.

        - `partner_id` (string): Mengacu ke `id` dari entitas `partner`.
    - **Relasi:**
        - Terhubung dengan `partner`, menunjukkan bahwa setiap customer memiliki satu partner.

7. **Entitas `partner`**
    - **Atribut:**
        - `id` (int): Kunci utama untuk mengidentifikasi setiap partner.
        - `nama` (string): Nama dari partner.
    - **Relasi:**
        - Berelasi dengan `customer`, menunjukkan bahwa sebuah partner mungkin memiliki beberapa customer yang terkait dengannya.

Ringkasan Relasi Utama:

- `unit` berelasi dengan `divisi`, menunjukkan bahwa setiap divisi memiliki beberapa unit.
- `proyek` terhubung ke `customer`, yang menunjukkan bahwa setiap proyek dimiliki atau terkait dengan customer tertentu.
- `customer` berelasi dengan `partner`, sehingga setiap customer dapat memiliki partner terkait.
- `proyek` juga berhubungan dengan `unit_proyek` dan `skema`, menunjukkan bahwa proyek terlibat dengan berbagai unit proyek dan skema yang relevan.

Diagram ini menggambarkan bagaimana entitas-entitas utama dalam sistem saling berinteraksi dan mendukung pengelolaan proyek dengan struktur data yang kompleks dan terorganisir.

Selama fase ini, peserta mempelajari berbagai fitur tambahan dalam *Prisma*, seperti fitur migrasi, yang memungkinkan perubahan pada struktur *database* secara otomatis saat terjadi perubahan dalam *schema*. Fitur ini sangat berguna ketika ada penambahan atau pengurangan kolom dalam tabel selama proses pengembangan. Peserta juga mempelajari

cara melakukan *seeding* data, yaitu mengisi *database* dengan data uji awal yang digunakan untuk pengujian dan pengembangan lebih lanjut.

Pada akhir *sprint* ini, *database* telah terhubung dengan *Prisma*, dan skema API awal sudah siap untuk diuji. Output dari *sprint* ini adalah *database* yang sudah terintegrasi dengan *Prisma*, dan model *schema* awal API yang dapat digunakan untuk melanjutkan ke tahap pengembangan API lebih lanjut.

3.5.4 *Sprint* 4 - API Dasar

Sprint keempat dimulai pada tanggal 15 Januari dan berlangsung selama 3,5 bulan, dengan fokus pada pengembangan API dasar untuk mendukung operasi *CRUD* di modul utama SDA. Pada tahap ini, peserta mengembangkan berbagai *endpoint* API yang memungkinkan aplikasi untuk mengakses dan mengelola data SDA melalui operasi dasar *CRUD* (Create, Read, Update, Delete). *CRUD* adalah inti dari setiap API yang dikelola, memungkinkan aplikasi untuk membuat, mengambil, memperbarui, dan menghapus data.

Setiap *endpoint* dikembangkan dengan menggunakan metode *HTTP* yang sesuai. Misalnya, metode *POST* digunakan untuk membuat data baru, *GET* untuk mengambil data yang sudah ada, *PUT* untuk memperbarui data yang ada, dan *DELETE* untuk menghapus data. Peserta mengembangkan setiap *endpoint* dengan memastikan bahwa data yang diterima dari klien memenuhi standar validasi sebelum diproses. Validasi ini bertujuan untuk mencegah adanya data yang tidak valid atau berbahaya yang dapat merusak *database* atau menyebabkan kesalahan pada aplikasi.

Selama *sprint* ini, peserta juga menerapkan langkah-langkah keamanan pada setiap *endpoint*, termasuk autentikasi dan otorisasi. Autentikasi menggunakan *JSON Web Token* (*JWT*) diterapkan untuk memastikan bahwa hanya pengguna yang terverifikasi yang dapat mengakses API. Setiap kali pengguna melakukan permintaan ke API, *JWT* yang sudah diterima pada proses *login* harus disertakan dalam *header* permintaan sebagai bukti autentikasi. Sementara itu, otorisasi berbasis peran (*Role-Based Access Control* atau *RBAC*) diterapkan untuk mengatur hak akses berdasarkan peran pengguna, seperti admin, editor, atau pengguna biasa, sehingga data hanya dapat diakses atau diubah oleh pengguna yang berwenang.

Pada akhir *sprint* ini, semua fungsi dasar untuk mengelola data SDA telah siap dan berfungsi sepenuhnya. Output dari *sprint* ini adalah *endpoint CRUD* yang lengkap, terintegrasi dengan *database* melalui *Prisma*, dan siap untuk diuji di tahap berikutnya.

3.5.5 *Sprint* 5 - *Testing* dan Optimisasi API

Setelah pengembangan API dasar selesai, *Sprint* kelima dimulai pada tanggal 1 Mei dan berlangsung selama satu bulan. Pada fase ini, peserta melakukan pengujian terhadap API secara menyeluruh untuk memastikan bahwa semua *endpoint* berfungsi sesuai spesifikasi. Pengujian dilakukan menggunakan alat seperti *Postman* untuk memeriksa fungsi dasar dan memastikan bahwa setiap *endpoint* memberikan respons yang diharapkan. Selain itu, *unit testing* juga dilakukan untuk setiap fungsi dalam API guna memastikan bahwa setiap komponen API dapat bekerja secara independen dan tanpa kesalahan.

Load testing juga dilakukan untuk mengevaluasi kinerja API di bawah beban tinggi. Dalam *load testing*, API diuji dengan banyak permintaan simultan untuk menilai seberapa baik API dapat menangani lonjakan permintaan dan bagaimana performa API di bawah tekanan. Berdasarkan hasil pengujian ini, peserta melakukan berbagai optimisasi, termasuk penerapan *caching* untuk data yang sering diakses, penggunaan *pagination* untuk mengelola data yang ditampilkan dalam jumlah besar, dan penambahan indeks pada kolom penting untuk mempercepat proses pencarian data di *database*.

Output dari *sprint* ini adalah API yang telah diuji, dioptimalkan, dan siap untuk *deployment*. API kini memiliki kinerja yang lebih baik dan siap untuk digunakan dalam lingkungan produksi.

3.5.6 *Sprint* 6 - *Deployment* dan Dokumentasi

Tahap terakhir, yaitu *Sprint Deployment* dan Dokumentasi, dimulai pada tanggal 1 Juni dan berlangsung selama dua minggu. Pada tahap ini, API yang telah diuji dan dioptimalkan diterapkan ke server produksi, memungkinkan akses oleh pengguna akhir. Proses *deployment* dilakukan dengan hati-hati untuk memastikan bahwa API berfungsi dengan baik di lingkungan produksi, termasuk uji coba tambahan untuk memastikan semua *endpoint* berfungsi dengan baik di server.

Selain itu, peserta menyusun dokumentasi yang komprehensif untuk setiap *endpoint*, mencakup informasi seperti deskripsi fungsi, parameter yang diperlukan, contoh permintaan dan respons, serta panduan pemecahan masalah. Dokumentasi ini juga berisi

panduan untuk pemeliharaan API dan cara menangani error yang mungkin terjadi di lingkungan produksi. Dokumentasi ini sangat penting untuk memudahkan tim lain dalam memahami dan menggunakan API, serta mendukung pemeliharaan dan pengembangan lanjutan di masa depan.

Output dari *sprint* ini adalah API yang sudah di-*deploy* di server produksi dan dokumentasi lengkap yang siap digunakan oleh tim teknis lainnya. Dokumentasi ini menjadi panduan yang berguna untuk memastikan bahwa API dapat digunakan dan dikembangkan lebih lanjut dengan standar yang telah ditetapkan.

3.6 Dokumentasi Prisma

a. Model

1. **SchemaType**:

```
model SchemaType {
  id         Int        @id @default(autoincrement())
  name       String
  createdAt DateTime  @default(now())
  updatedAt DateTime  @default(now())
  project    Project[]
}
```

Tabel 3.2 Model Schema Type

- Menyimpan informasi tipe skema.
- **Relasi**: Terhubung dengan model Project.

2. **Division**:

```
model Division {
  id         Int       @id @default(autoincrement())
  name       String
  units      Unit[]
  createdAt DateTime @default(now())
  updatedAt DateTime @default(now())
}
```

Tabel 3.3 Model Division

- Menyimpan informasi divisi.

- **Relasi**: Memiliki koleksi Unit.

3. **Unit**:

```
model Unit {
  id          Int        @id @default(autoincrement())
  name        String
  divisionId Int
  division    Division  @relation(fields: [divisionId],
references: [id])
  projects    Project[]
  createdAt   DateTime  @default(now())
  updatedAt   DateTime   @default(now())
}
```

Tabel 3.4 Model Unit

- **Relasi**: Terhubung dengan Division dan memiliki koleksi Project.

4. **Partner**:

```
model Partner {
  id         Int         @id @default(autoincrement())
  name       String
  customers Customer[]
  createdAt DateTime    @default(now())
  updatedAt DateTime    @default(now())
}
```

Tabel 3.5 Model Partner

- **Relasi**: Terhubung dengan Customer.

5. **Customer**:

```
model Customer {
  id         Int        @id @default(autoincrement())
  name       String
  partnerId Int
  partner    Partner    @relation(fields: [partnerId],
references: [id])
  createdAt DateTime  @default(now())
  updatedAt DateTime  @default(now())
```

```
  project    Project[]
}
```

Tabel 3.6 Model Customer

- **Relasi**: Terhubung dengan Partner dan Project.

6. **Project**:

```
model Project {
  id              Int                     @id
@default(autoincrement())
  name            String
  Category        ProjectCategory
  documentStatus ProjectDocumentStatus
  value           Int
  requestDate     String
  orderNum        String
  quoteNum        String
  contractNum     String
  IDLOP           String
  lastPosition    String
  nextPosition    String
  activeSchema    String
  schemaId    Int
  schema          SchemaType              @relation(fields:
[schemaId], references: [id])
  unitsId         Int
  units           Unit                    @relation(fields:
[unitsId], references: [id])
  customerId      Int
  customer        Customer                @relation(fields:
[customerId], references: [id])
  createdAt       DateTime                @default(now())
  updatedAt       DateTime                @default(now())
}
```

Tabel 3.7 Model Project

- Mewakili proyek dengan berbagai atribut seperti kategori (ProjectCategory) dan status dokumen (ProjectDocumentStatus).

- **Relasi**: Terhubung dengan SchemaType, Unit, dan Customer.

7. **Product**:

```
model Product {
  id                    Int             @id
@default(autoincrement())
  name                  String
  videoMeta             String?
  description           String
  benefit               String
  valueProposition      String
  specificationFeatures String?
  customerSTP           String
  configuration         String
  SLA_SLUG              String
  productCategoryId     Int
  parentId              Int?
  productCategory       ProductCategory @relation(fields:
[productCategoryId], references: [id])
  parentProduct         Product?        @relation("parentChilds",
fields: [parentId], references: [id], onDelete: Cascade)
  childProducts         Product[]       @relation("parentChilds")
  createdAt             DateTime        @default(now())
  updatedAt             DateTime        @default(now())
}
```

Tabel 3.8 Model Produk

- **Relasi**: Terhubung dengan ProductCategory dan memiliki hubungan induk/anak dengan produk lain.

8. **ProductCategory**:

```
model ProductCategory {
  id        Int       @id @default(autoincrement())
  name      String
  desc      String?
  videoMeta String?
  createdAt DateTime  @default(now())
```

```
  updatedAt DateTime @default(now())
  products Product[]
}
```

Tabel 3.9 Model Produk Kategori

- **Relasi**: Memiliki koleksi Product.

9. **User**:

```
model User {
  email    String @id
  password String
  salt     String
  roles    Roles
}
```

Tabel 3.10 Model User

- Menyimpan data pengguna dengan peran (Roles).

b. Enum

1. **DetailImpactBasedGroup**:

```
enum DetailImpactBasedGroup {
  assurance
  delivery
  enabler
  solution
}
```

Tabel 3.11 Detail Impact Based Group

- Kategori grup dampak.

2. **StatusYTD**:

```
enum StatusYTD {
  achieved
  not_started @map("not started")
  close
}
```

Tabel 3.12 Model Status YTD

- Status pencapaian aktivitas/proyek.

3. **ProjectCategory**:

```
enum ProjectCategory {
  pengadaan_baru @map("pengadaan baru")
  pembaruan
}
```

Tabel 3.13 Project Category

- Kategori proyek.

4. **Roles**:

```
enum Roles {
  admin
  user
}
```

Tabel 3.14 Model Roles

Peran pengguna.

BAB IV
REFLEKSI PELAKSANAAN MAGANG

4.1 Relevansi Akademik

Pengalaman magang di PT Telkom Indonesia memungkinkan pemagang untuk menguji relevansi teori-teori yang diperoleh selama perkuliahan dalam konteks praktis yang kompleks dan dinamis. Analisis ini memeriksa beberapa area utama di mana terdapat gap atau perbedaan signifikan antara teori yang dipelajari di kelas dan praktik lapangan selama magang. Identifikasi perbedaan ini memberikan wawasan yang berharga dan menawarkan kontribusi akademik yang dapat meningkatkan relevansi kurikulum dengan kebutuhan industri.

1. Penerapan Framework Pengembangan Perangkat Lunak

   Dalam lingkungan akademik, pemahaman tentang framework pengembangan perangkat lunak seperti *Vue.js* dan *Express.js* umumnya difokuskan pada aspek fungsional dan arsitekturalnya. Vue.js, misalnya, dipelajari sebagai framework *front-end* yang ringan dengan konsep reaktif yang memudahkan pembaruan data di sisi pengguna, terutama untuk aplikasi yang memerlukan pembaruan *real-time* (Sarcar et al., 2019). Di sisi lain, Express.js, yang sering kali dipelajari sebagai bagian dari teknologi stack *JavaScript full-stack*, dikenal karena kecepatannya dalam membangun *back-end* dengan REST API yang mendukung skalabilitas dan efisiensi, terutama dalam sistem berbasis layanan mikro (Burns et al., 2021).

   Namun, dalam lingkungan kerja di PT Telkom Indonesia, pemagang menemukan bahwa aspek efisiensi dan skalabilitas framework bukan satu-satunya pertimbangan dalam pemilihan teknologi. Faktor-faktor seperti ketersediaan sumber daya, waktu pelatihan, dan kebutuhan pengguna menjadi alasan yang tidak selalu dijelaskan dalam kurikulum akademik. Sebagai contoh, meskipun Vue.js memungkinkan pembaruan data yang efisien dan penanganan komponen berbasis UI yang terstruktur, pada beberapa kasus, kebutuhan untuk mencapai hasil dengan cepat memerlukan pendekatan praktis yang bisa mengorbankan fleksibilitas Vue.js untuk solusi yang lebih sederhana dan langsung. Implementasi seperti ini bertujuan untuk mengurangi kompleksitas yang berpotensi memperlambat pengembangan.

Di sini, pemagang menyadari adanya *gap* antara pemahaman teoritis yang ideal dengan kebutuhan praktis untuk menyesuaikan metode pengembangan berdasarkan kendala waktu dan skala proyek. Pemahaman ini menjadi sumbangan akademik yang penting karena mengindikasikan bahwa pemilihan framework tidak dapat sepenuhnya didasarkan pada keunggulan teknis saja, tetapi harus dipertimbangkan bersama dengan konteks spesifik proyek yang sedang dikerjakan, termasuk tim dan klien.

2. Implementasi Metodologi Agile dan Scrum

Dalam teori, metodologi Agile dan Scrum diajarkan sebagai solusi terbaik untuk meningkatkan fleksibilitas dan produktivitas proyek perangkat lunak. Agile didasarkan pada prinsip iterasi pendek yang memungkinkan pengembangan secara bertahap, di mana hasil setiap tahap dapat dievaluasi dan diperbaiki sesuai umpan balik pengguna, menjadikannya metodologi yang sangat adaptif (Beck et al., 2001). Scrum, sebagai salah satu implementasi Agile, memiliki struktur tahapan yang jelas, seperti *sprint planning*, *daily stand-ups*, *sprint review*, dan *sprint retrospective* (Schwaber & Sutherland, 2017). Di kelas, mahasiswa biasanya diajarkan untuk mengikuti tahapan ini secara disiplin untuk menjamin efisiensi kerja dan kualitas hasil.

Selama magang, pemagang menemukan bahwa metode Scrum di PT Telkom Indonesia mengalami adaptasi yang signifikan. Misalnya, waktu yang dialokasikan untuk *sprint retrospective*, yang bertujuan untuk mengevaluasi proses kerja dan mengidentifikasi perbaikan, sering kali tidak seideal yang dijelaskan dalam teori. Dalam praktik, *sprint retrospective* sering kali dilakukan secara singkat, bahkan terkadang hanya melibatkan diskusi informal. Selain itu, tidak semua tahapan Scrum diimplementasikan secara konsisten. Pada kondisi di mana tenggat waktu ketat, beberapa tahapan Scrum diakselerasi atau dihilangkan demi fokus pada penyelesaian fitur utama. Hal ini mengajarkan pemagang bahwa metodologi Agile dan Scrum dapat dipraktikkan secara fleksibel, sesuai kebutuhan tim dan proyek.

Perbedaan ini memberi sumbangan pemahaman akademik bahwa di lapangan, Agile dan Scrum sering kali diterapkan secara hibrida atau bahkan dimodifikasi sesuai tuntutan proyek dan budaya kerja tim. Ini menunjukkan bahwa meskipun struktur teoretis Scrum penting untuk meningkatkan produktivitas, praktiknya perlu mempertimbangkan konteks organisasi. Oleh karena itu, mahasiswa perlu dilatih untuk menghadapi variasi implementasi Agile dan Scrum di lingkungan kerja yang beragam,

serta mengembangkan keterampilan adaptasi untuk menyesuaikan dengan kebutuhan organisasi.

3. Manajemen Waktu dan Prioritas

Di kelas, konsep manajemen waktu dan prioritas sering kali diajarkan menggunakan metode yang berstruktur, seperti penggunaan Gantt Chart, CPM (Critical Path Method), atau matriks prioritas Eisenhower untuk mengelola tugas berdasarkan tingkat kepentingan dan urgensi (Covey, 1989). Metode ini mengajarkan mahasiswa untuk mengatur tugas secara berurutan dan tetap fokus pada prioritas yang telah ditetapkan di awal proyek. Namun, selama magang, pemagang menemukan bahwa dinamika proyek di lapangan membuat perencanaan yang statis sulit dipertahankan.

Pada praktiknya, proyek pengembangan perangkat lunak sering kali mengalami perubahan prioritas yang tidak dapat diprediksi, baik karena perubahan kebutuhan dari klien maupun adanya penemuan teknis baru yang harus segera diterapkan. Hal ini memaksa pemagang untuk cepat beradaptasi, mengelola kembali prioritas dengan fleksibilitas yang tinggi, dan menjaga produktivitas dalam kondisi yang serba berubah. Misalnya, ketika pengembangan antarmuka pengguna (UI) hampir selesai, masukan pengguna yang baru dapat mengubah seluruh rencana UI, dan tim harus melakukan revisi besar yang sebelumnya tidak terencana. Situasi ini menunjukkan bahwa dalam dunia kerja, manajemen waktu harus fleksibel dan responsif terhadap perubahan yang dinamis.

Perbedaan ini memberikan kontribusi akademik yang penting bahwa dalam kurikulum, manajemen waktu perlu dilatih bukan hanya dalam konteks perencanaan awal, tetapi juga bagaimana mahasiswa dapat berlatih merespons perubahan tak terduga. Kemampuan untuk menilai kembali prioritas, re-scheduling, dan adaptasi terhadap situasi yang berubah cepat adalah aspek kunci dalam pengembangan karier profesional di industri yang cepat berubah.

4. Komunikasi dan Kolaborasi dalam Tim

Dalam dunia akademik, komunikasi tim diajarkan sebagai dasar kolaborasi yang efektif, sering kali dengan teori-teori tentang pentingnya komunikasi yang terbuka, mendengarkan aktif, dan cara bernegosiasi dalam kelompok (Adair, 2009). Mahasiswa diajarkan bahwa komunikasi yang baik meningkatkan kinerja dan harmoni

dalam tim, serta memastikan setiap anggota memahami tujuan yang sama. Namun, di PT Telkom Indonesia, pemagang menemukan bahwa komunikasi dalam praktiknya tidak selalu terstruktur dan membutuhkan keterampilan interpersonal yang lebih dari sekadar mendengarkan dan berbicara dengan baik.

Di lapangan, komunikasi dalam tim melibatkan adaptasi dengan anggota tim yang memiliki latar belakang, pengalaman, dan tingkat keterampilan yang berbeda. Sering kali, tantangan muncul karena setiap anggota memiliki pandangan yang berbeda dalam memahami prioritas proyek. Misalnya, ada situasi di mana seorang anggota dengan lebih banyak pengalaman berusaha menyederhanakan proses kerja, sementara anggota lain yang baru belajar ingin mengikuti tahapan sesuai prosedur teoritis. Di sini, kemampuan untuk menyeimbangkan antara mendengarkan pandangan anggota lain dan menyampaikan pendapat secara diplomatis menjadi kunci keberhasilan komunikasi.

Perbedaan antara teori dan praktik ini memberikan kontribusi akademik bahwa kurikulum perlu memberikan porsi lebih pada keterampilan komunikasi interpersonal yang lebih mendalam. Mahasiswa dapat dilatih untuk menghadapi perbedaan pandangan dan keterampilan dengan lebih realistis, melalui simulasi atau proyek lintas disiplin yang membutuhkan kerja sama lintas bidang.

Dari hasil analisis di atas, terdapat beberapa rekomendasi akademik yang dapat meningkatkan kesiapan mahasiswa untuk menghadapi situasi nyata di lapangan:

1. Pengajaran Studi Kasus Nyata dan Project-based Learning

   Kurikulum pengembangan perangkat lunak dapat lebih banyak menyertakan studi kasus yang berdasarkan pengalaman nyata dari industri. Misalnya, menggunakan contoh proyek yang melibatkan pengembangan aplikasi dengan kendala waktu dan sumber daya. Ini dapat memberi mahasiswa gambaran realistis tentang kompleksitas dan kompromi yang sering diperlukan dalam pemilihan teknologi dan metode.

2. Latihan Adaptasi Metodologi Agile dan Scrum

   Dalam pengajaran metodologi Agile dan Scrum, simulasi proyek dapat disesuaikan untuk memberikan pengalaman adaptasi dengan berbagai skenario. Dengan demikian, mahasiswa tidak hanya mempelajari tahapan Scrum secara teoretis, tetapi juga berlatih untuk menyesuaikan tahapan tersebut dalam kondisi yang mungkin memerlukan perubahan jadwal atau diskusi informal.

3. Keterampilan Manajemen Waktu yang Dinamis

   Latihan manajemen waktu dan prioritas dapat ditingkatkan dengan simulasi situasi proyek di mana prioritas sering berubah. Dengan cara ini, mahasiswa akan lebih siap menghadapi kondisi di mana perencanaan yang dibuat di awal harus disesuaikan kembali di tengah proyek untuk mengikuti perubahan kebutuhan atau kendala teknis.

4. Peningkatan Latihan Komunikasi Antar-Personal dan Negosiasi

   Kurikulum dapat memberikan porsi latihan komunikasi yang lebih mendalam, seperti negosiasi antar anggota tim yang memiliki latar belakang berbeda, atau simulasi proyek yang memerlukan kolaborasi lintas disiplin. Ini akan membantu mahasiswa mengembangkan keterampilan komunikasi yang efektif dan fleksibel dalam lingkungan kerja yang kolaboratif.

Dengan demikian, temuan ini memberikan gambaran yang lebih komprehensif tentang kebutuhan industri dan memberikan rekomendasi yang dapat diterapkan dalam pengembangan kurikulum. Dengan adanya adaptasi ini, diharapkan lulusan tidak hanya memahami teori tetapi juga memiliki kesiapan yang tinggi untuk mengatasi tantangan di lapangan.

4.2 Pembelajaran Magang

Pengalaman magang di PT Telkom Indonesia memberikan pemagang wawasan yang mendalam dalam mengembangkan keterampilan teknis serta pemahaman tentang dinamika kerja profesional. Program ini memfasilitasi pemagang untuk menerapkan teori-teori yang dipelajari di perkuliahan dan mengintegrasikannya dengan praktik nyata di lapangan. Selama magang, pemagang memperoleh berbagai manfaat, menghadapi tantangan, serta berhadapan dengan hambatan yang menguji keterampilan adaptasi dalam lingkungan kerja yang dinamis. Proses ini mencakup pemahaman mendalam mengenai penggunaan teknologi terkini, implementasi metodologi manajemen proyek modern seperti Agile dan Scrum, serta pentingnya keterampilan komunikasi dan kolaborasi dalam tim. Melalui pengalaman ini, pemagang mampu mengembangkan wawasan praktis tentang penerapan teknologi, manajemen proyek, dan pengelolaan prioritas waktu dalam situasi dunia kerja yang sesungguhnya. Ini menjadi dasar penting bagi pemagang dalam memadukan teori akademik dengan tantangan dan kebutuhan dunia industri, membangun kesiapan profesional untuk menghadapi peran yang lebih besar dalam karier ke depan.

4.2.1 Pembelajaran dari Penerapan Teknologi Terkini

1. Pengalaman Praktis dalam Penggunaan Framework Vue.js dan Express.js

   Selama magang, pemagang mendapat kesempatan untuk menggunakan Vue.js sebagai framework *front-end* dan Express.js sebagai framework *back-end*. Dalam teori, Vue.js dikenal karena pendekatan berbasis komponen, yang mempermudah pengelolaan elemen UI dan mendukung *data binding*, sehingga perubahan data di *back-end* langsung tercermin pada *front-end* (Sarcar et al., 2019). Namun, dalam praktik, pemagang belajar bahwa implementasi Vue.js memerlukan keterampilan dalam strukturisasi komponen agar tetap modular dan efisien, terutama dalam aplikasi yang kompleks. Di sisi *back-end*, Express.js memungkinkan pembuatan RESTful API dengan efisien, mendukung pengembangan aplikasi berbasis layanan mikro dengan kecepatan tinggi (Burns et al., 2021). Pengalaman ini membuka wawasan pemagang tentang pentingnya memahami arsitektur aplikasi secara menyeluruh, bukan hanya sebatas penerapan framework secara teori.

2. Implementasi Modular dan *Reusable Component*

   Salah satu manfaat besar dari penggunaan Vue.js adalah kemampuannya untuk membangun komponen yang modular dan *reusable*. Dalam teori, modularisasi dianggap sebagai metode untuk menjaga kode tetap terstruktur dan mudah dikelola. Di lapangan, pemagang menyadari bahwa kemampuan untuk membuat komponen *reusable* bukan hanya menghemat waktu, tetapi juga memungkinkan tim pengembangan untuk lebih fokus pada pengembangan fitur baru. Penggunaan komponen yang dapat digunakan ulang ini membantu mengatasi kendala dalam pengembangan proyek besar, karena modul yang telah diuji dan diterapkan dapat digunakan di beberapa bagian aplikasi tanpa perlu membangun ulang dari awal.

3. Sinkronisasi *Front-end* dan *Back-end* dalam Pengembangan Aplikasi

   Dalam pengembangan aplikasi berbasis Vue.js dan Express.js, sinkronisasi data antara *front-end* dan *back-end* menjadi tantangan utama yang harus diatasi. Pada teori, koneksi antar-layer dalam aplikasi sering diajarkan dengan pendekatan sederhana, namun di lapangan pemagang menemukan bahwa sinkronisasi ini membutuhkan konfigurasi dan debugging yang intensif untuk mengatasi perbedaan data yang sering terjadi. Pemagang harus memastikan bahwa setiap perubahan data pada *back-end* bisa langsung terlihat pada *front-end*, terutama dalam aplikasi yang memerlukan pembaruan

data real-time. Pengalaman ini memberikan pemahaman praktis bahwa teori sinkronisasi data memerlukan adaptasi yang lebih rinci di lingkungan kerja.

4. Pemahaman Mendalam tentang RESTful API

   RESTful API memfasilitasi komunikasi antara aplikasi *front-end* dan *back-end*. Selama magang, pemagang mempraktikkan cara mengimplementasikan API dan berlatih menggunakan metode HTTP seperti GET, POST, PUT, dan DELETE untuk mengakses dan memanipulasi data. Selain itu, pemagang menghadapi tantangan dalam menangani kesalahan, seperti kegagalan koneksi atau kesalahan pemrosesan data. Pengalaman ini menegaskan pentingnya pemahaman mendalam mengenai RESTful API dalam memastikan integrasi yang lancar antara berbagai komponen aplikasi, serta pentingnya menangani kesalahan dalam komunikasi data dengan cepat dan efisien.

5. Debugging dan Pemecahan Masalah pada Kode yang Kompleks

   Salah satu keterampilan yang sangat diperlukan dalam pengembangan perangkat lunak adalah kemampuan debugging. Dalam teori, debugging sering kali dianggap sebagai langkah dasar yang melibatkan identifikasi dan perbaikan kesalahan. Namun, di lapangan, pemagang menemukan bahwa debugging memerlukan teknik yang lebih lanjut, seperti *console logging*, analisis stack trace, dan penggunaan alat debugging di browser. Menghadapi bug yang kompleks, pemagang belajar untuk mengembangkan kemampuan troubleshooting yang efektif, bekerja sama dengan tim untuk menemukan solusi, dan mengidentifikasi akar masalah. Hal ini memberikan pemahaman bahwa debugging merupakan proses iteratif yang membutuhkan ketelitian dan ketangguhan mental.

6. Pengelolaan Status Aplikasi dengan Vuex

   Dalam aplikasi yang kompleks, pengelolaan status (*state management*) menjadi tantangan penting. Vuex adalah pustaka untuk pengelolaan status di aplikasi Vue.js, dan pemagang mempelajari cara menggunakannya untuk mengatur data aplikasi secara terpusat. Teori tentang pengelolaan status sering kali mengajarkan dasar-dasar penyimpanan data, tetapi di lapangan, pemagang belajar bahwa Vuex memungkinkan data antar-komponen tetap sinkron tanpa menyebabkan konflik. Melalui Vuex, pemagang belajar menangani data dengan lebih efisien dan menjaga konsistensi status aplikasi meskipun terjadi perubahan dinamis.

7. Optimalisasi Komponen Berdasarkan Keterbatasan Waktu

   Dalam proyek yang memiliki tenggat waktu ketat, pemagang sering kali harus membuat keputusan untuk menyederhanakan beberapa komponen atau modul agar aplikasi tetap berfungsi sesuai jadwal. Meskipun teori sering kali mengajarkan pengembangan optimal, praktik di lapangan menunjukkan bahwa optimalisasi terkadang harus dikompromikan demi efisiensi waktu. Pemagang belajar tentang fleksibilitas dalam pengembangan aplikasi, serta kapan harus menerapkan solusi pragmatis yang efisien tanpa mengorbankan fungsionalitas inti aplikasi.

8. Adaptasi Framework untuk Kebutuhan Khusus Proyek

   Sering kali, framework Vue.js dan Express.js perlu disesuaikan dengan pustaka tambahan untuk memenuhi kebutuhan khusus proyek. Dalam beberapa kasus, pemagang harus mencari pustaka tambahan untuk menambah fungsionalitas tertentu yang tidak disediakan oleh Vue.js atau Express.js secara bawaan. Pengalaman ini memberikan wawasan tentang fleksibilitas dan kreativitas dalam pemilihan alat pengembangan yang disesuaikan dengan kebutuhan pengguna, serta membuka wawasan bahwa tidak ada solusi yang sepenuhnya "satu ukuran untuk semua" dalam pengembangan perangkat lunak.

9. Pengalaman Uji Coba pada Berbagai Perangkat

   Pengujian aplikasi untuk memastikan kompatibilitas di berbagai perangkat menjadi tantangan nyata dalam pengembangan *front-end*. Dalam teori, pengujian responsivitas sering kali dilakukan dengan menggunakan emulator, tetapi di lapangan pemagang belajar bahwa pengujian pada perangkat nyata, seperti desktop, tablet, dan ponsel, sangat penting untuk memastikan aplikasi berjalan dengan baik pada resolusi layar yang berbeda. Pengalaman ini menekankan pentingnya memperhatikan aspek teknis seperti tampilan, kinerja, dan responsivitas dalam kondisi nyata, tidak hanya secara teoritis.

10. Penerapan Prinsip MVC di Lapangan

    Model-View-Controller (MVC) adalah arsitektur yang membagi kode aplikasi menjadi tiga bagian utama: model untuk data, view untuk tampilan, dan controller untuk logika. Dalam teori, MVC diajarkan sebagai metode untuk memisahkan tanggung jawab dalam kode. Namun, di lapangan, pemagang menemukan bahwa

penerapan MVC juga membantu dalam memudahkan proses debugging dan modifikasi fitur. MVC memberikan struktur yang jelas pada aplikasi dan memastikan bahwa setiap perubahan dapat dilakukan tanpa merusak keseluruhan kode, membantu pemagang memahami pentingnya organisasi kode dalam proyek besar.

4.2.2 Pembelajaran dari Penerapan Metodologi Agile dan Scrum

1. Implementasi Iteratif dalam Proses Pengembangan Proyek

   Pemagang belajar bahwa Scrum, sebagai bagian dari metodologi Agile, memungkinkan tim untuk bekerja dalam siklus pengembangan singkat yang disebut *sprint*. Setiap *sprint* biasanya berlangsung selama dua hingga empat minggu dan terdiri dari serangkaian tugas yang fokus pada pengembangan fitur tertentu. Dalam pendekatan tradisional, pengembangan perangkat lunak umumnya dilakukan secara linear atau *waterfall*, di mana setiap fase, seperti analisis, desain, pengkodean, pengujian, dan implementasi, berlangsung secara berurutan dan hanya dimulai ketika fase sebelumnya selesai. Pendekatan linear ini sering kali memakan waktu lebih lama karena perubahan sulit diterapkan jika suatu tahapan telah selesai.

   Dengan Scrum, pemagang belajar bahwa iterasi pada setiap *sprint* memungkinkan tim untuk terus mengembangkan dan menyempurnakan aplikasi berdasarkan umpan balik dari pengguna atau perubahan kebutuhan. Jika ada fitur baru atau perbaikan yang dibutuhkan, Scrum memungkinkan tim untuk langsung menerapkan perubahan tersebut pada *sprint* berikutnya, membuat proses pengembangan menjadi lebih fleksibel dan responsif terhadap kebutuhan pengguna. Pemagang juga memahami bahwa pendekatan iteratif ini meningkatkan keterlibatan pengguna dalam proses pengembangan, karena pengguna dapat melihat hasil dari setiap *sprint* dan memberikan masukan yang relevan.

2. Pengalaman dalam Pembagian Tugas Sprint

   Pembagian tugas yang efektif adalah salah satu fondasi penting dalam Scrum. Dalam setiap *sprint planning*, pemagang belajar bagaimana tugas-tugas proyek dipecah menjadi bagian-bagian kecil yang disebut *user stories* atau *tasks*. Tugas-tugas ini memiliki tujuan yang spesifik dan batasan waktu tertentu sehingga dapat diselesaikan dalam jangka waktu *sprint* yang telah ditentukan. Di lapangan, pemagang menyadari

bahwa pembagian tugas ini bukan hanya soal membagi pekerjaan, tetapi juga tentang mengidentifikasi prioritas serta kemampuan setiap anggota tim.

Pembagian tugas yang efektif memungkinkan pemagang untuk mengelola waktu dengan lebih baik, menghindari pekerjaan yang berlebihan, dan meningkatkan akurasi perencanaan. Selain itu, pemagang memahami bahwa komunikasi dan koordinasi antara anggota tim sangat penting selama *sprint planning*. Setiap anggota tim harus memahami peran masing-masing, sehingga setiap bagian dari proyek berjalan sesuai prioritas dan saling melengkapi dalam mencapai tujuan akhir dari *sprint*. Pengalaman ini memperkaya pemahaman pemagang tentang pentingnya perencanaan yang detail dan terstruktur agar tim dapat bekerja dengan lebih produktif dan efisien.

3. Adaptasi *Daily Stand-Up*

Daily stand-up merupakan pertemuan singkat yang diadakan setiap hari selama *sprint*. Dalam teori Scrum, pertemuan ini hanya berlangsung 10-15 menit untuk memungkinkan tim menyampaikan progres harian, kendala, dan rencana kerja masing-masing. Namun, dalam praktiknya, pemagang mendapati bahwa pertemuan ini sering berlangsung lebih lama karena beberapa masalah teknis atau kendala yang muncul memerlukan pembahasan mendalam. Meskipun ada pedoman untuk menjaga *stand-up* tetap singkat, fleksibilitas sangat dibutuhkan, terutama saat tim menghadapi masalah yang kompleks.

Adaptasi ini mengajarkan pemagang pentingnya komunikasi yang efektif dan fleksibilitas untuk mengatasi kendala secara tepat waktu. Selama *stand-up*, anggota tim dapat mengidentifikasi masalah yang perlu segera diselesaikan dan bertukar ide tentang cara terbaik untuk mengatasinya. Pemagang memahami bahwa keterbukaan dan kesediaan untuk beradaptasi menjadi kunci dalam menangani dinamika dan tantangan yang mungkin muncul setiap hari.

4. Kolaborasi Antar-Fungsi dalam *Sprint Review*

Sprint review adalah kesempatan bagi tim untuk mempresentasikan hasil dari *sprint* yang telah diselesaikan dan mendapatkan umpan balik dari tim lain atau dari klien. Dalam sesi ini, pemagang belajar bahwa kolaborasi dengan tim dari berbagai fungsi, seperti tim desain, pengujian, dan pengembangan, sangat penting untuk mencapai hasil akhir yang sesuai dengan harapan klien. Kolaborasi ini memberikan

kesempatan bagi setiap fungsi untuk berkontribusi dalam evaluasi proyek dan memastikan bahwa hasil akhir memenuhi kebutuhan dan standar kualitas.

Pengalaman ini mengajarkan pemagang bahwa kerja lintas fungsi sangat penting dalam mencapai keberhasilan proyek. *Sprint review* memungkinkan setiap anggota tim untuk mendapatkan perspektif yang lebih luas dan belajar dari sudut pandang rekan dari tim lain. Pemagang menyadari bahwa kerja lintas fungsi bukan hanya tentang mengintegrasikan pekerjaan dari berbagai departemen, tetapi juga tentang membangun komunikasi yang kuat, saling menghormati peran masing-masing, dan bekerja secara kolaboratif untuk mencapai tujuan bersama.

5. Penyesuaian Proses dalam *Sprint Retrospective*

Dalam setiap *sprint retrospective*, anggota tim mengevaluasi apa yang berjalan baik dan apa yang perlu diperbaiki. Evaluasi ini membantu tim untuk memahami kekuatan dan kelemahan proses yang telah diterapkan dan mencari cara untuk meningkatkan kinerja di *sprint* berikutnya. Pemagang belajar bahwa di lapangan, *retrospective* sering kali dilakukan secara informal atau dipersingkat agar dapat berfokus pada isu-isu yang mendesak.

Penyesuaian ini mengajarkan pemagang bahwa Agile menuntut adaptasi yang konstan dalam mencapai efisiensi. Pemagang menyadari bahwa evaluasi yang berkelanjutan membantu tim menjadi lebih proaktif dalam menyelesaikan masalah dan beradaptasi dengan perubahan. Selain itu, *sprint retrospective* memberi tim ruang untuk mengevaluasi produktivitas dan menemukan cara agar proses kerja menjadi lebih baik di masa depan.

6. Pentingnya Umpan Balik Cepat dari Pengguna

Dalam metodologi Agile, salah satu prinsip penting adalah memberikan nilai tambah secara terus-menerus kepada pengguna. Melalui umpan balik cepat dari pengguna, tim dapat menyesuaikan pengembangan fitur dengan kebutuhan pengguna secara aktual. Pemagang belajar bahwa *feedback* dari pengguna ini berfungsi sebagai panduan dalam menentukan prioritas fitur dan memastikan bahwa produk akhir relevan dengan kebutuhan pengguna.

Pengalaman ini mengajarkan pemagang bahwa keputusan dalam Agile harus berpusat pada pengguna, dan setiap fitur atau perubahan yang dibuat harus berdampak

positif bagi pengalaman pengguna. Umpan balik yang cepat memungkinkan tim untuk melakukan perubahan dengan cepat jika diperlukan dan meminimalkan risiko pengembangan yang tidak sesuai dengan ekspektasi pengguna.

7. Pengambilan Keputusan Berdasarkan Prioritas Bisnis

   Dalam proyek berbasis Agile, keputusan tidak hanya dibuat berdasarkan keinginan teknis atau estetika, tetapi juga prioritas bisnis dan kebutuhan klien. Pemagang belajar bahwa keputusan harus diprioritaskan sesuai dengan urgensi kebutuhan pengguna, tidak semua fitur bisa diimplementasikan dalam satu waktu. Setiap fitur yang dikembangkan harus disesuaikan dengan strategi bisnis perusahaan.

   Pemagang belajar bahwa keputusan yang fleksibel dan responsif dapat membantu tim dalam memenuhi kebutuhan yang lebih mendesak dan berdampak langsung pada pengguna. Dengan memahami prioritas bisnis, pemagang menyadari bahwa keberhasilan proyek tidak hanya dilihat dari kualitas teknis, tetapi juga dari kontribusi fitur terhadap pencapaian tujuan bisnis perusahaan.

8. Penyelesaian Konflik dalam Tim dengan Efektif

   Tim Agile terdiri dari anggota yang memiliki latar belakang, keahlian, dan sudut pandang yang berbeda. Hal ini membuat potensi konflik di dalam tim menjadi hal yang wajar, terutama ketika ada perbedaan pendapat tentang pendekatan terbaik dalam menyelesaikan masalah atau ketika menghadapi kendala teknis. Pemagang menyadari bahwa dalam lingkungan yang dinamis ini, penyelesaian konflik secara efektif sangat penting untuk menjaga produktivitas dan keharmonisan tim.

   Pemagang belajar bahwa komunikasi terbuka dan kesediaan untuk mendengarkan pendapat yang berbeda menjadi kunci utama dalam menyelesaikan perbedaan pandangan. Dengan saling menghargai, tim dapat mencapai konsensus yang bermanfaat bagi proyek. Pengalaman ini mengajarkan pemagang pentingnya sikap terbuka dan toleransi terhadap perbedaan, serta kesadaran bahwa kolaborasi yang efektif membutuhkan kerja sama dan pengertian bersama.

9. Pengalaman Memfasilitasi Perubahan *Scope* Proyek

   Salah satu keunggulan metodologi Agile adalah kemampuannya untuk memfasilitasi perubahan *scope* proyek dengan fleksibilitas tinggi. Pemagang belajar bahwa Agile memungkinkan tim untuk menyesuaikan rencana proyek tanpa

mengganggu keseluruhan jadwal. Dalam lingkungan kerja yang dinamis, perubahan *scope* sering kali menjadi kebutuhan yang tidak terhindarkan, baik karena permintaan pengguna atau perubahan strategi bisnis perusahaan.

Pengalaman ini memberikan pemahaman bahwa keberhasilan proyek tidak selalu ditentukan oleh ketepatan jadwal, tetapi juga oleh kemampuan tim dalam menyesuaikan proyek dengan kebutuhan yang berkembang. Agile membantu tim untuk beradaptasi dengan perubahan dengan tetap menjaga produktivitas dan komitmen terhadap hasil yang diharapkan.

10. Adaptasi Metodologi untuk Kebutuhan Organisasi

Pemagang menyadari bahwa Agile dan Scrum tidak selalu dapat diterapkan secara penuh dan kaku di setiap organisasi. Setiap organisasi memiliki kebutuhan, budaya kerja, dan struktur yang berbeda, sehingga beberapa aspek dari metodologi ini sering kali harus disesuaikan dengan kondisi dan kebutuhan organisasi tertentu. Sebagai contoh, beberapa organisasi mungkin lebih menyukai *sprint* yang lebih singkat, sementara yang lain mungkin menyesuaikan *daily stand-up* dengan kebutuhan pertemuan mingguan.

Pengalaman ini mengajarkan pemagang bahwa metodologi Agile adalah pendekatan yang fleksibel, yang dapat diadaptasi sesuai konteks untuk mencapai tujuan yang diinginkan. Fleksibilitas dalam penerapan metodologi menjadi aspek penting dalam mencapai kesuksesan proyek, dan pemagang menyadari bahwa Agile harus digunakan sebagai panduan, bukan sebagai aturan yang kaku.

BAB V
PENUTUP

5.1 Kesimpulan

Setelah menjalani program magang di Departemen Solution, Delivery, and Assurance (SDA) Regional II di PT Telkom Indonesia, pemagang merasakan banyak capaian dan pembelajaran yang signifikan dalam pengembangan diri dan profesionalisme. Selama masa magang, pemagang terlibat dalam pengembangan aplikasi berbasis web, yang mencakup berbagai aspek dari front-end hingga back-end. Melalui pengalaman ini, pemagang memperoleh pemahaman yang lebih dalam tentang siklus hidup pengembangan perangkat lunak, mulai dari analisis kebutuhan, perancangan, pengembangan, hingga pengujian dan deployment. Pengalaman langsung ini memberikan pemagang wawasan tentang bagaimana teori yang dipelajari di bangku kuliah dapat diterapkan dalam situasi nyata di industri.

Salah satu capaian utama adalah peningkatan keterampilan teknis yang terkait dengan pengembangan perangkat lunak. Dengan menggunakan teknologi terkini seperti Vue.js untuk pengembangan antarmuka pengguna dan Express.js untuk back-end, pemagang tidak hanya belajar teori tetapi juga secara aktif menerapkannya dalam proyek nyata. Pemahaman tentang konsep-konsep seperti REST API, pengelolaan basis data dengan Prisma ORM, serta penerapan praktik CI/CD menggunakan GitLab semakin memperkaya pengalaman teknis pemagang. Selain itu, pemagang juga memperoleh keterampilan dalam kolaborasi tim yang sangat penting, mengingat proyek yang dikerjakan melibatkan banyak anggota tim dengan latar belakang dan keahlian yang berbeda. Pengalaman bekerja dalam lingkungan yang dinamis ini membantu pemagang mengembangkan kemampuan komunikasi yang efektif dan kemampuan untuk bekerja sama dalam tim, yang merupakan keterampilan vital di dunia kerja.

Di samping capaian teknis, program magang ini juga memberikan pelajaran berharga tentang manajemen waktu dan penyelesaian masalah. Pemagang belajar untuk mengatur waktu dengan baik, memprioritaskan tugas-tugas yang ada, serta menyelesaikan tantangan yang muncul dalam proses pengembangan. Salah satu tantangan yang dihadapi adalah memahami dan menguasai berbagai teknologi baru dalam waktu yang singkat, yang mendorong pemagang untuk belajar mandiri dan mencari referensi tambahan. Pengalaman ini mengasah kemampuan

analitis pemagang dalam mengidentifikasi dan memecahkan masalah, yang sangat berharga dalam konteks pengembangan perangkat lunak.

Rencana untuk pekerjaan ke depan hingga akhir magang dan setelahnya mencakup beberapa langkah strategis. Pemagang berkomitmen untuk terus meningkatkan pengetahuan dan keterampilan dalam teknologi dan framework terbaru yang relevan dengan industri. Selain itu, pemagang berencana untuk aktif berpartisipasi dalam proyek-proyek tambahan yang mungkin tersedia di Departemen SDA, guna mendapatkan pengalaman yang lebih luas dan mendalam. Dalam hal networking, pemagang berencana untuk memperluas jaringan profesional dengan berinteraksi lebih banyak dengan rekan kerja, mentor, dan pemangku kepentingan di perusahaan. Hal ini diharapkan dapat membuka peluang karier yang lebih baik di masa depan.

Pemagang juga memiliki keinginan untuk memberikan kontribusi positif terhadap proses kerja tim dengan mengusulkan ide-ide inovatif yang dapat meningkatkan efisiensi dan efektivitas kerja. Dengan demikian, pemagang berharap dapat membantu tim mencapai tujuan bersama. Terakhir, pemagang berencana untuk memberikan umpan balik yang konstruktif tentang pengalaman magang ini untuk meningkatkan program bagi pemagang di periode selanjutnya, sehingga lebih banyak mahasiswa dapat merasakan manfaat yang sama.

Dengan semua pengalaman dan pembelajaran yang diperoleh, pemagang merasa lebih siap untuk menghadapi tantangan di dunia kerja dan mengembangkan keterampilan yang akan berguna dalam karier profesional di masa depan. Harapan ke depan adalah agar program magang ini dapat terus berkembang dan memberikan manfaat bagi generasi berikutnya, sehingga lebih banyak pemagang dapat memperoleh pengalaman yang berharga dan memperkaya kompetensi mereka. Program ini diharapkan dapat menciptakan hubungan yang lebih baik antara dunia pendidikan dan industri, sehingga lulusan dapat lebih siap menghadapi tuntutan pasar kerja yang terus berubah.

5.2 Saran

Berdasarkan pengalaman magang di Departemen Solution, Delivery, and Assurance (SDA) Regional II PT Telkom Indonesia, berikut adalah beberapa saran yang dapat meningkatkan efektivitas program magang ini agar lebih mendukung pemagang dalam pencapaian tujuan pembelajaran dan persiapan karier. Saran-saran ini disusun berdasarkan capaian yang telah dirasakan pemagang, serta identifikasi area yang belum sepenuhnya

tercapai, guna memastikan program magang ke depan dapat lebih selaras dengan harapan dan kebutuhan industri.

Pertama, disarankan agar perusahaan mengadakan modul pelatihan awal yang terstruktur dan komprehensif pada minggu pertama program magang. Mengingat banyaknya teknologi baru yang harus dikuasai dalam waktu singkat, seperti Vue.js, Express.js, dan CI/CD, modul ini akan mempermudah pemagang dalam memahami dasar-dasar teknologi yang akan digunakan selama proyek berlangsung. Dalam modul ini, pelatihan dapat mencakup penggunaan framework dasar, pengenalan REST API, pengelolaan basis data, dan praktik CI/CD, sehingga pemagang dapat langsung memahami proses dan alat yang relevan. Pembekalan awal ini akan sangat membantu, mengingat keterbatasan waktu magang yang tidak selalu cukup untuk mempelajari teknologi secara mandiri tanpa panduan.

Kedua, pemagang menyarankan adanya sistem bimbingan teknis yang lebih terarah melalui sesi mentoring rutin yang dipandu oleh mentor berpengalaman di departemen. Dalam praktiknya, tantangan utama pemagang adalah menguasai teknologi baru yang belum sepenuhnya dijelaskan di bangku perkuliahan. Sesi mentoring ini dapat diadakan secara mingguan, di mana pemagang bisa berdiskusi langsung dengan mentor mengenai permasalahan teknis yang dihadapi, sekaligus mendapatkan panduan untuk meningkatkan efisiensi pekerjaan. Dengan bimbingan yang lebih intensif, pemagang akan merasa lebih didukung dan memiliki akses untuk memperoleh jawaban dari berbagai pertanyaan teknis yang mungkin tidak ditemukan dalam pembelajaran mandiri.

Ketiga, disarankan agar perusahaan menyediakan sumber daya pembelajaran tambahan yang mencakup dokumentasi teknis dan panduan tutorial yang dapat diakses oleh pemagang kapan pun dibutuhkan. Meskipun banyak referensi pembelajaran daring yang dapat diakses, beberapa teknologi yang spesifik untuk proyek tertentu mungkin memerlukan dokumentasi internal atau panduan khusus. Dengan akses ke dokumentasi ini, pemagang akan lebih siap untuk menyelesaikan tantangan dalam pengembangan proyek dan mengurangi kesalahan dalam penerapan teknologi baru. Sumber daya tambahan ini, jika dipadukan dengan modul pembekalan awal, akan mempercepat kurva belajar pemagang dan meningkatkan kualitas kontribusi mereka dalam proyek.

Keempat, untuk meningkatkan keterampilan komunikasi dan kerja sama tim, disarankan agar pemagang dilibatkan dalam pertemuan lintas departemen atau sesi kolaborasi bersama tim dari departemen lain. Mengingat keberagaman tugas dan peran dalam

pengembangan aplikasi berbasis web, pemagang akan memperoleh manfaat besar dari memahami cara kerja dan peran tim lain, seperti desain UI/UX, analisis bisnis, atau manajemen produk. Dengan berinteraksi secara langsung dengan tim-tim ini, pemagang dapat melihat bagaimana peran mereka di SDA berkontribusi pada keseluruhan proses pengembangan, meningkatkan kemampuan kolaborasi, dan mengasah keterampilan komunikasi lintas disiplin yang akan sangat berguna dalam karier di masa mendatang.

Kelima, disarankan agar perusahaan mengadakan sesi networking formal di akhir program magang, yang melibatkan pemagang, rekan kerja, mentor, dan manajer dari berbagai departemen. Sesi ini akan memberikan kesempatan bagi pemagang untuk membangun jaringan profesional yang kuat dan mendapatkan pemahaman lebih mendalam mengenai peluang karier yang ada di PT Telkom Indonesia dan industri telekomunikasi secara umum. Dengan adanya kesempatan networking ini, pemagang akan lebih mudah menjalin hubungan kerja yang mungkin bisa membuka peluang karier di masa depan, baik di perusahaan ini maupun di industri yang sama. Hubungan profesional yang dibangun melalui program magang akan memperkuat kesiapan pemagang dalam menghadapi dunia kerja setelah lulus kuliah.

Terakhir, disarankan agar perusahaan mengadakan sesi umpan balik terstruktur di akhir masa magang, di mana pemagang dapat memberikan masukan terkait pengalaman magang mereka, kendala yang dihadapi, dan saran untuk perbaikan program. Sesi ini tidak hanya akan membantu perusahaan memahami kebutuhan dan harapan pemagang, tetapi juga menjadi langkah awal dalam proses evaluasi berkelanjutan untuk menyempurnakan program magang. Melalui masukan ini, PT Telkom Indonesia dapat terus mengembangkan program magang agar semakin relevan dan memberikan pengalaman yang berharga bagi pemagang generasi berikutnya. Umpan balik ini juga akan membantu perusahaan menciptakan lingkungan magang yang mendukung proses pembelajaran dengan lebih baik, sehingga setiap pemagang dapat mencapai tujuan pembelajaran mereka secara maksimal.

Dengan saran-saran ini, diharapkan program magang di PT Telkom Indonesia dapat menjadi sarana pengembangan keterampilan yang lebih komprehensif, berdaya guna, dan memberikan manfaat yang optimal bagi para pemagang. Implementasi saran ini akan semakin mempersiapkan pemagang untuk menghadapi tantangan dunia kerja dan mendukung PT Telkom Indonesia dalam mencetak sumber daya manusia yang berkompeten dan siap bersaing dalam industri telekomunikasi.

DAFTAR PUSTAKA

Adair, J. (2009). *Effective Teamwork: How to Influence Your Team for Peak Performance*. Pan Macmillan.

Anderson, J., & Smith, R. (2023). *Advanced Web Development with Prisma and Node.js*. Journal of Web Development, 29(3), 120-135.

Anderson, T., Rivera, L., & Zhang, M. (2022). *Building Responsive Web Applications Using Modern JavaScript Frameworks*. Web Technologies Journal, 25(4), 310-322.

Arief, M., & Nugraha, E. (2020). "Penerapan Metodologi Agile pada Pengembangan Sistem Informasi Akademik di Perguruan Tinggi." *Jurnal Teknologi Informasi dan Ilmu Komputer*, 7(1), 15-25.

Atlassian. (2023). The benefits of project management software.

Bauer, C., & King, G. (2021). Java persistence with Hibernate. Manning Publications.

Beck, K., et al. (2001). *Manifesto for Agile Software Development*. Agile Alliance. Retrieved from https://agilemanifesto.org/

Brown, M., Lin, X., & Wang, Z. (2022). *Efficient Database Management with Prisma ORM*. Software Engineering Journal, 34(2), 201-215.

Burns, K., & Beath, S. (2021). *Express.js Guide: The Comprehensive Book on Express.js*. Fullstack.io.

Cahyono, A. S., & Wahyudi, S. (2019). "Analisis Penggunaan Framework Vue.js dalam Pengembangan Aplikasi Web di Indonesia." *Jurnal Informatika Universitas Pamulang*, 4(2), 112-119.

Choi, L. (2021). *The Impact of RESTful API on Real-Time Project Management Applications*. International Journal of IT Systems, 40(5), 100-112.

Covey, S. R. (1989). *The 7 Habits of Highly Effective People: Powerful Lessons in Personal Change*. Free Press.

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine).

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral Dissertation, University of California, Irvine.

Gomez, H. (2020). *SQL vs ORM: Enhancing Data Management for Web Applications*. Data Systems Journal, 18(1), 75-89.

Handayani, P. W., Hidayanto, A. N., & Budi, I. (2017). "Analisis Faktor-Faktor yang Mempengaruhi Penggunaan Continuous Integration dan Continuous Delivery pada Industri Teknologi Informasi." *Jurnal Sistem Informasi*, 13(2), 89-97.

Hill, S., Jones, D., & Stevens, E. (2022). *Optimizing RESTful API Performance in Web Applications*. Computer Networks Journal, 38(6), 215-228.

Johnson, L. (2021). *The Evolution of Static to Dynamic Websites: A Comprehensive Study*. Web Development Research, 32(7), 405-420.

Jones, D. (2023). *Using Prisma for Simplified Data Access in Web Applications*. Journal of Software Engineering, 37(4), 240-255.

Kerzner, H. (2017). Project management: A systems approach to planning, scheduling, and controlling. John Wiley & Sons.

Kumar, S., Williams, J., & Martin, J. (2020). *Web-Based Project Management Solutions for Efficient Resource Allocation*. Project Management Journal, 45(3), 85-98.

Kurniawan, I., & Rahardjo, M. (2018). "Penerapan RESTful API dalam Pengembangan Sistem Informasi untuk Meningkatkan Keterhubungan Data." *Jurnal Informatika dan Sistem Informasi*, 5(3), 221-230.

Lazarus, R. S., & Folkman, S. (1984). *Stress, Appraisal, and Coping*. Springer Publishing Company.

Lee, Y., & Park, J. (2022). *Improving Project Efficiency through Advanced Management Software*. Journal of Modern Project Management, 20(5), 132-145.

Lin, X., & Lee, H. (2023). *Database Management Techniques in RESTful API Development*. International Journal of Databases, 50(1), 195-208.

Liu, W., Nguyen, T., & Shin, H. (2022). *Preventing SQL Injection through Input Validation in Web Applications*. Security in Computing, 22(3), 122-135.

Mansfield-Devine, S. (2013). The cost of insecure software. Computer Fraud & Security, 2013(6), 5-9.

Martin, J., Stevens, E., & Wang, Z. (2021). *Authorization Models for Securing RESTful APIs in Modern Applications*. Security & Privacy Journal, 29(5), 150-162.

Maulana, M., & Pratama, B. (2021). "Implementasi Agile Scrum dalam Pengembangan Aplikasi Start-up Berbasis Web." *Jurnal Manajemen dan Teknologi Informasi*, 10(1), 45-56.

Nguyen, T., Shin, H., & Kim, J. (2022). *Enhancing Security in API-Based Applications Using JSON Web Tokens (JWT)*. International Journal of Information Security, 35(4), 210-226.

Nugroho, Y., & Susanto, H. (2019). "Efektivitas Penggunaan Continuous Integration/Continuous Deployment (CI/CD) di Lingkungan Perusahaan Teknologi Indonesia." *Jurnal Teknologi Informasi dan Komputer*, 8(2), 134-142.

Pernici, B., Tofanelli, R., & Vassiliadis, P. (2018). From ORM to NoSQL: A systematic mapping study. Journal of Systems and Software, 136, 141-156.

Reddy, M., Kiran, R., & Jain, S. (2019). A study on RESTful web services and its performance. International Journal of Computer Applications, 182(24), 5-9. Schwaber, K., & Sutherland, J. (2020). The Scrum Guide.

Rivera, P. (2021). *The Role of SEO in Modern Web Architecture*. Journal of Web Technologies, 21(8), 315-329.

Saputra, R. A., & Sari, M. (2021). "Analisis Penggunaan GitLab sebagai Alat CI/CD dalam Pengembangan Perangkat Lunak." *Jurnal Teknik Informatika*, 15(1), 71-78.

Sarcar, M., Masood, S., & Srivastava, H. (2019). *Developing Vue.js Applications: An Introduction to Building and Deploying SPAs with Vue.js*. Apress.

Schwaber, K., & Sutherland, J. (2017). *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*.

Sutrisno, T., & Setiawan, A. (2020). "Peran Komunikasi Efektif dalam Tim Agile untuk Meningkatkan Produktivitas Proyek." *Jurnal Teknologi dan Sistem Informasi*, 11(4), 101-113.

Turner, R., Zhang, M., & Lee, H. (2021). *Object-Relational Mapping as a Tool for Efficient Data Handling*. Computing Journal, 19(9), 115-128.

Wang, Z., Lin, X., & Stevens, E. (2022). *Reducing Development Time with ORM in Web Applications*. Journal of Database Management, 44(2), 145-160.

Wibowo, R., & Putra, A. (2019). "Studi Kasus Penerapan Scrum pada Pengembangan Aplikasi di Perusahaan Teknologi Indonesia." *Jurnal Informatika Indonesia*, 13(2), 200-210.

Yulianto, R., & Fitriani, R. (2018). "Pengaruh Penerapan Agile Methodology Terhadap Keberhasilan Pengembangan Sistem di Lingkungan Perusahaan IT di Indonesia." *Jurnal Sistem Informasi Bisnis*, 9(3), 187-195.

Zhang, M. (2023). *Best Practices in Modern Web Development Using JavaScript Frameworks*. Frontiers in Web Development, 26(3), 65-78.

LAMPIRAN

Lampiran tidak perlu diberi nomor halaman. Dokumen apa saja yang dimasukkan dalam lampiran cukup diberi judul dengan kata ‘LAMPIRAN’ yang dilanjutkan dengan huruf abjad besar untuk penomoran. Cukup judul ‘LAMPIRAN’ saja yang dimasukkan dalam daftar isi. Judul-judul lampiran, seperti Lampiran A, Lampiran B dan seterusnya, tidak perlu dimasukkan dalam daftar isi.