IMPLEMENTASI REST DALAM MEMBANGUN WEB SERVICE MENGGUNAKAN GOLANG (STUDI KASUS: APLIKASI SATUDIKTI)

Disusun Oleh:

Nama : Sigit Priadi
NIM : 18523125

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA
2022

HALAMAN PENGESAHAN DOSEN PEMBIMBING

IMPLEMENTASI REST DALAM MEMBANGUN WEB SERVICE MENGGUNAKAN GOLANG (STUDI KASUS: APLIKASI SATUDIKTI)

TUGAS AKHIR JALUR MAGANG

Disusun Oleh:

Nama : Sigit Priadi
NIM : 18523125

Yogyakarta, 14 November 2022

Pembimbing,

(Moh. Idris, S.Kom, M.Kom.)

HALAMAN PENGESAHAN DOSEN PENGUJI

IMPLEMENTASI REST DALAM MEMBANGUN WEB SERVICE MENGGUNAKAN GOLANG (STUDI KASUS: APLIKASI SATUDIKTI)

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, 14 November 2022

Tim Penguji

Ketua Penguji

Moh. Idris, S.Kom., M.Kom.

Anggota 1

Affan Mahtarami, S.Kom., M.T.

Anggota 2

Septia Rani, S.T., M.Cs.

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 : Sigit Priadi

NIM : 18523125

Tugas akhir dengan judul:

IMPLEMENTASI REST DALAM MEMBANGUN WEB SERVICE MENGGUNAKAN GOLANG (STUDI KASUS: APLIKASI SATUDIKTI)

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 resiko dan konsekuensi apapun.

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

Yogyakarta, 14 November 2022

(Sigit Priadi)

HALAMAN PERSEMBAHAN

Segala puji bagi Allah SWT atas nikmat dan karunia-Nya saya sebagai penulis mampu menyelesaikan laporan ini dengan lancar. Laporan tugas akhir ini saya persembahkan kepada kedua orang tua saya, Ibu Sumiati dan Bapak Sudarli, serta kakak saya yang terus memberikan semangat dan dukungan yang menjadi motivasi untuk diri saya. Kepada seluruh keluarga, saudara, dan sahabat yang selalu mendukung saya untuk menyelesaikan laporan tugas akhir ini. Kepada Bapak Idris sebagai dosen pembimbing yang telah membimbing saya untuk menyelesaikan tugas akhir ini. Kepada seluruh anggota tim Satudikti yang telah bekerja sama untuk menyelesaikan proyek Satudikti.

Kepada semua pihak yang telah mendukung penyelesaian laporan tugas akhir ini, saya ucapkan terima kasih, semoga Allah SWT selalu memberikan kesehatan dan perlindungan kepada kita semua.

HALAMAN MOTO

“Pengalaman tidak bisa dipelajari tetapi harus dilalui.”

— BJ. Habibie

“Akan ada solusi untuk setiap masalah. Hidup terlalu singkat jika hanya untuk mengeluh. Berusaha, percaya diri, dan berdoa.”

— Mario Teguh

“Akan ada solusi untuk setiap masalah. Hidup terlalu singkat jika hanya untuk mengeluh. Berusaha, percaya diri, dan berdoa.”

— Mario Teguh

“Maka, nikmat Tuhanmu manakah yang kamu dustakan ?”

— QS. Ar-Rahman:13

KATA PENGANTAR

Assalamu'alaikum Wr. Wb.

Alhamdulillah penulis haturkan kepada Allah Swt., yang telah melimpahkan rahmat dan taufik serta hidayah-Nya. Laporan ini disusun sebagai bukti pelaksanaan magang dan untuk memenuhi salah satu syarat kelulusan pada jalur magang jurusan Informatika Universitas Islam Indonesia. Shalawat serta salam tak lupa kita saya panjatkan kepada junjungan kita Nabi Muhammad SAW yang telah membawa kita dari zaman kebodohan menuju zaman terang benderang.

Program magang di program studi Informatika Universitas Islam Indonesia merupakan salah satu dari lima jalur yang dapat ditempuh untuk memperoleh gelar sarjana. Program magang ini, mengharuskan mahasiswa untuk terjun langsung ke dunia kerja selama 6 bulan. Penulis, menjalankan program magang ini di Direktorat Jenderal Pendidikan Tinggi, Riset, dan Teknologi.

Penyusunan laporan ini tidak terlepas dari bimbingan, dukungan dan semangat dari beberapa pihak. Oleh karena itu, penulis menyampaikan rasa terima kasih dan rasa syukur dalam bentuk ucapan kepada:

1. Allah SWT yang telah memberikan rahmat dan hidayahnya.
2. Kedua orang tua penulis yang senantiasa memberikan do'a serta dorongan semangat saat pelaksanaan magang dilakukan.
3. Bapak Moh. Idris, S.Kom, M.Kom, selaku dosen pembimbing.
4. Bapak Prof. Ir. Nizam, M.Sc., DIC, Ph.D., selaku Direktur Jenderal Pendidikan Tinggi yang telah memberikan kesempatan penulis untuk magang di Direktorat Jenderal Pendidikan Tinggi, Riset, dan Teknologi.
5. Bapak Franova Herdiyanto, S.Kom., M.T.I., selaku Project Manager pada Proyek Dikti Super App (Satudikti).
6. Saudara Muhammad Andry Mahdison dan Aldy Rivaldi, selaku pembimbing lapangan di Direktorat Jenderal Pendidikan Tinggi, Riset, dan Teknologi.
7. Saudara Muhammad Alghifari R., selaku trainer di Direktorat Jenderal Pendidikan Tinggi, Riset, dan Teknologi.
8. Segenap staf dan karyawan di Direktorat Jenderal Pendidikan Tinggi, Riset dan Teknologi yang telah membantu dan membimbing penulis selama pelaksanaan magang.

9. Para dosen Program Studi Informatika Universitas Islam Indonesia yang telah memberikan ilmunya kepada penulis selama perkuliahan sebagai bekal pelaksanaan magang.
10. Teman-teman, baik di luar maupun di dalam lingkungan Program Studi Informatika Universitas Islam Indonesia, yang selalu memberikan dukungan dan semangat kepada penulis hingga laporan ini dibuat.

Atas bantuan dari berbagai pihak tersebut, penulis dapat menyelesaikan laporan ini dengan baik. Dengan laporan ini harapannya dapat memberikan gambaran selama kegiatan magang di Direktorat Jenderal Pendidikan Tinggi, Riset, dan Teknologi. Namun, penulis menyadari bahwa laporan ini tidak bisa dikatakan sempurna.Oleh karena itu, penulis mengharapkan kritik dan saran dari pembaca agar penulis bisa intropeksi kembali untuk menjadi lebih baik. Akhir kata, semoga laporan ini bermanfaat untuk orang banyak.

Wassalamu'alaikum Wr. Wb.

Yogyakarta, 14 November 2022

(Sigit Priadi)

SARI

Direktorat Jenderal Pendidikan Tinggi, Riset, dan Teknologi (Ditjen Diktiristek) merupakan lembaga pemerintahan bergerak pada perumusan dan pelaksanaan kebijakan di bidang pendidikan tinggi akademik. Ditjen Diktiristek memiliki berbagai layanan. Banyaknya layanan yang dikembangkan berpotensi menjadi aplikasi yang bersifat silo (sistem terpisah). Untuk mengatasi hal tersebut Ditjen Diktiristek mengadakan proyek yang akan dilaksanakan oleh mahasiswa magang dan keluaran dari proyek tersebut berupa aplikasi untuk menggabungkan seluruh layanan yaitu aplikasi Satudikti. Aplikasi Satudikti adalah aplikasi yang menampilkan data dan informasi dari seluruh layanan yang ada pada Ditjen Diktiristek. Aplikasi tersebut berbasis *mobile* dan *website*. Setiap layanan memiliki sumber data terpisah sehingga aplikasi Satudikti memerlukan sebuah *platform* untuk mengakses kebutuhan data setiap layanan. Pada penelitian ini, sebagai *Backend Developer* diberikan tugas untuk membangun Web *Service*. Web *Service* tersebut sebagai wadah yang menyediakan kebutuhan dan pengolahan data menjadi informasi yang dibutuhkan terhadap masing-masing layanan pada aplikasi Satudikti. *Output* dari Web *Service* yaitu berupa API. Selain itu, REST API dipilih sebagai arsitektur yang diimplementasikan dalam membangun Web *Service*. Dalam pengembangannya, teknologi yang digunakan dalam membangun Web *Service* menggunakan bahasa pemrograman Golang serta menggunakan *framework* Echo yang mendukung implementasi arsitektur REST. Pada penelitian ini terdapat beberapa aktivitas yang akan dilakukan dalam membangun Web *Service* mulai dari perancangan, implementasi, dan pengujian. Dari hasil penelitian yang dilakukan, Web *Service* mempermudah *client* seperti aplikasi Satudikti dalam mendapatkan kebutuhan data. *Client* tidak perlu berinteraksi secara langsung dari seluruh layanan dan *client* tidak terbebani dalam pengolahan data sesuai kebutuhan yang diinginkan karena aktivitas tersebut sudah ditangani oleh Web *Service* yang sudah dibangun.

Kata kunci: Aplikasi Satudikti, *Backend Developer*, Web *Service*, REST API, Golang.

GLOSARIUM

Baseurl	alamat utama dalam mengakses suatu sumber.
Constructor	method yang akan dieksekusi saat membuat objek.
Database	kumpulan data terorganisir yang umumnya disimpan dan diakses secara elektronik oleh sistem komputer.
Endpoint	sub alamat dalam mengakses suatu sumber.
Field	kolom atau atribut suatu tabel.
Forwarding API	meneruskan data dari *Third-party* API atau *service* lain.
Framework	kerangka kerja untuk mengembangkan suatu sistem aplikasi berbasis *mobile* atau *website*.
Function	satu blok perintah yang akan dieksekusi ketika dipanggil dari bagian lain dalam suatu program.
Interface	tipe data yang menampung segala jenis data.
Method	suatu *function* dengan penambahan parameter *receiver_argument type*.
Platform	suatu wadah perangkat keras (komputer) dan perangkat lunak (sistem operasi) tempat aplikasi perangkat lunak dapat dijalankan.
Request	permintaan pada suatu sumber yang dilakukan oleh klien.
Response	proses layanan yang diberikan oleh server terhadap klien yang melakukan *request*.
Server	perangkat atau sistem yang menyediakan layanan untuk klien.
Sintaks	blok kode program.
Struct	tipe data yang menampung kumpulan definisi variabel/*property* atau *function*/*method*.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

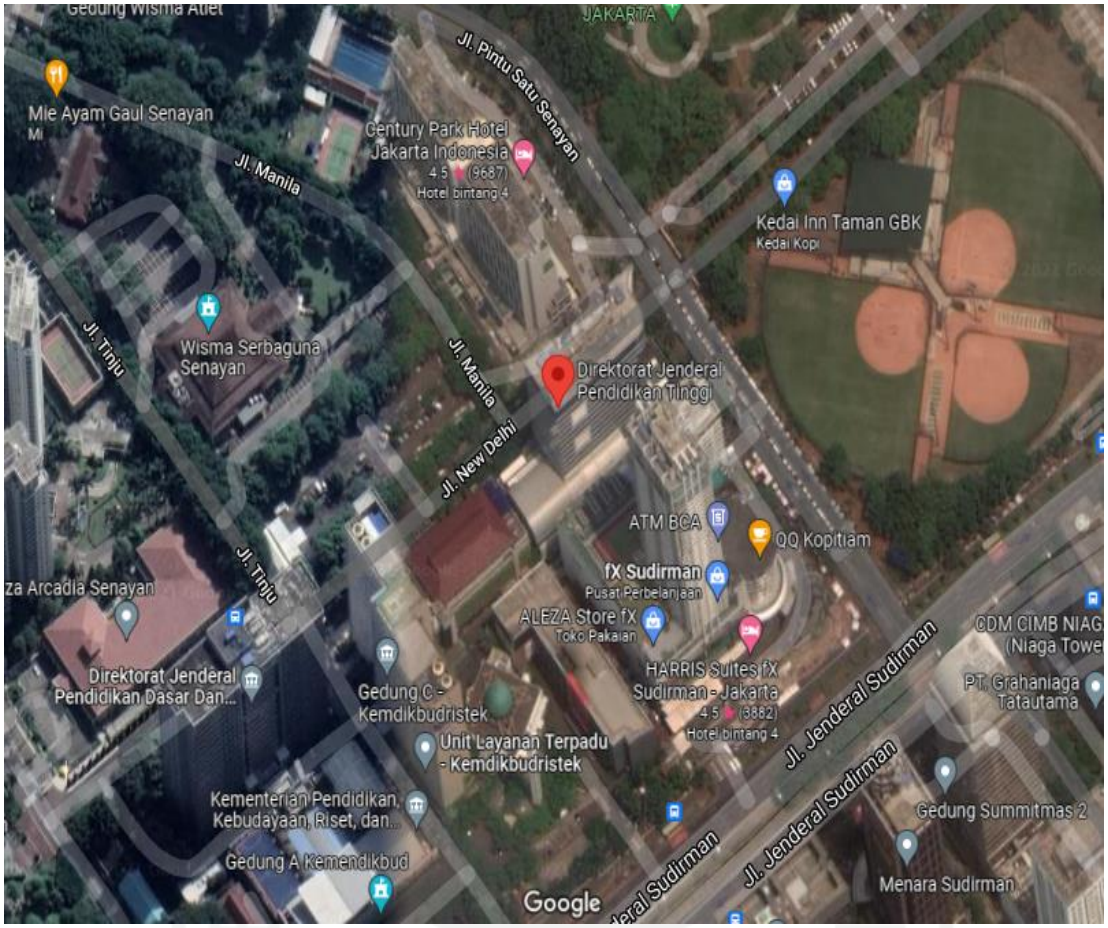
Aplikasi Satudikti merupakan solusi yang diberikan terhadap masalah yang ada pada Direktorat Jenderal Pendidikan Tinggi, Riset, dan Teknologi (Ditjen Diktiristek) yaitu banyaknya aplikasi layanan yang dikembangkan dan berpotensi menjadi aplikasi yang bersifat silo dan media penyebaran informasi yang tidak terpusat. Dengan adanya aplikasi Satudikti, pengguna layanan bisa memanfaatkan satu aplikasi untuk mengetahui layanan dan informasi, media penyebaran informasi yang terpadu, dan pengembangan aplikasi bisa dilakukan terpusat, terintegrasi, dan terpadu dalam satu aplikasi.

Setiap layanan yang ada pada Ditjen Diktiristek mempunyai tujuan yang berbeda sehingga informasi yang dihasilkan dari masing-masing layanan juga berbeda. Kemudian masing-masing layanan juga melakukan pengolahan dan penampungan data yang berdiri sendiri tanpa berkaitan dengan layanan lainnya. Dilihat dari kondisi tersebut, aplikasi Satudikti harapannya dapat mengakses sumber data dari masing-masing layanan untuk mendapatkan data yang dibutuhkan terhadap semua layanan yang akan diintegrasikan.

Namun, agar aplikasi Satudikti tidak terbebani dalam melakukan pengambilan dan pengolahan data dari masing-masing layanan dibutuhkan sebuah layanan atau *platform* yang melakukan hal tersebut yaitu berupa Web *Service*. Web *Service* akan menyediakan kebutuhan aplikasi Satudikti terhadap pengambilan dan pengolahan data. Dengan adanya Web *Service* aplikasi Satudikti tidak perlu mengakses seluruh layanan yang ada melainkan cukup mengakses Web *Service* yang menyediakan data dari seluruh layanan yang akan diintegrasikan.

1.1.1 Gambaran Umum

Ditjen Diktiristek merupakan unit eselon I pada Kementerian Pendidikan, Kebudayaan, Riset, dan Teknologi. Institusi ini memiliki banyak layanan yang berasal dari sekretariat dan tiga direktorat yang bergerak dalam menyelenggarakan perumusan dan pelaksanaan kebijakan di bidang pendidikan tinggi akademik. Alamat kantor di Jalan Pintu Satu Senayan No.1, RW.3, Senayan, Kecamatan Tanah Abang, Kota Jakarta Pusat, Daerah Khusus Ibukota Jakarta 10270. Gambar 1.1 menunjukkan Lokasi Ditjen Diktiristek.



Gambar 1.1 Lokasi Ditjen Diktiristek

1.1.2 Struktur

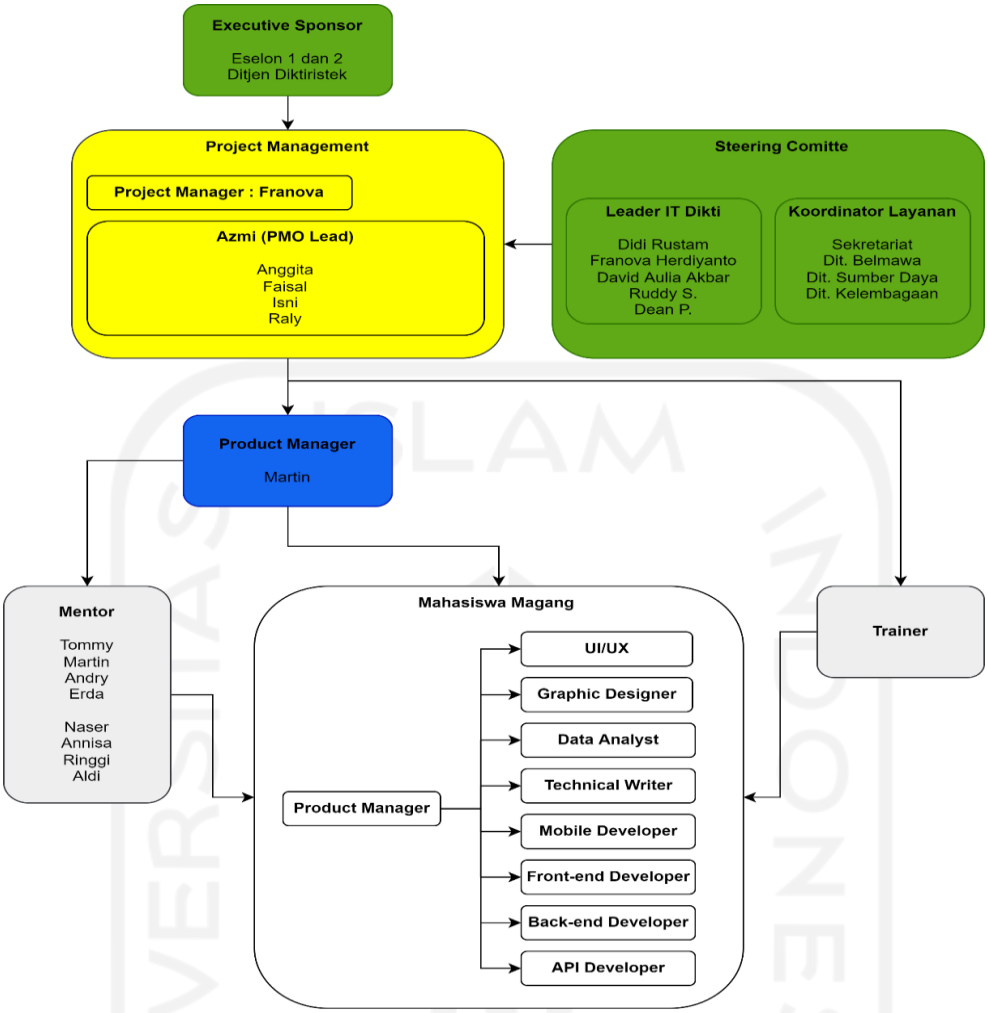
Struktur Organisasi

Ditjen Diktiristek memiliki struktur organisasi yang terbagi menjadi beberapa bagian, diantaranya terdapat Direktur Jenderal Pendidikan Tinggi, Riset, dan Teknologi, Sekretaris Direktorat Jenderal Pendidikan Tinggi, Riset, dan Teknologi, Direktur Pembelajaran dan Kemahasiswaan, Direktur Riset, Teknologi, dan Pengabdian Kepada Masyarakat, Direktur Sumber Daya, dan Direktur Kelembagaan. Gambar 1.2 menunjukkan Struktur Organisasi Ditjen Diktiristek.

Gambar 1.2 Struktur Organisasi Ditjen Diktiristek

Struktur Proyek

Setiap pengerjaan proyek di Ditjen Diktiristek, instansi tersebut akan membentuk tim atau struktur organisasi dalam pengembangannya. Pengerjaan proyek pada kegiatan magang memiliki struktur organisasi sendiri yang terbagi menjadi beberapa bagian. Gambar 1.3 menunjukkan Struktur Organisasi Proyek Satudikti.



Gambar 1.3 Struktur Organisasi Proyek Satudikti

1.1.3 Sejarah Institusi

Ditjen Diktiristek salah satu unit utama di Kementerian Pendidikan, Kebudayaan, Riset, dan Teknologi. Sesuai perannya dalam Peraturan Menteri Pendidikan dan Kebudayaan Nomor 45 Tahun 2019, Direktorat Jenderal Pendidikan Tinggi, Riset, dan Teknologi menyelenggarakan fungsinya sebagai berikut (Diktiristek, 2022):

a. Perumusan kebijakan di bidang pendidikan tinggi akademik.
b. Pelaksanaan kebijakan di bidang pembelajaran, kemahasiswaan, kelembagaan, dan sumber daya pendidikan tinggi akademik.
c. Perumusan pemberian izin penyelenggara perguruan tinggi swasta yang diselenggarakan oleh masyarakat.
d. Pelaksanaan evaluasi dan pelaporan di bidang pendidikan tinggi akademik.
e. Pelaksanaan administrasi Direktorat Jenderal.

f. Pelaksanaan fungsi lain yang diberikan Menteri.

Pendidikan tinggi akan selalu menjadi modal penting bagi lahirnya generasi penerus unggul di berbagai bidang. Untuk itu, Ditjen Diktiristek akan terus memberikan pelayanan terbaik di bidang pendidikan tinggi agar cita-cita pembinaan Sumber Daya Manusia yang unggul dapat terwujud.

Sesuai dengan perannya, Ditjen Diktiristek telah menghasilkan beberapa layanan diantaranya :

a. PDDikti (Pangkalan Data Pendidikan Tinggi) merupakan program Ditjen Diktiristek untuk memperoleh informasi Perguruan Tinggi beserta civitas akademika.
b. Kampus Merdeka merupakan program Merdeka Belajar di luar kampus dengan memiliki pengalaman yang relevan dengan kebutuhan industri.
c. Kedaireka (Kerja Sama Dunia Usaha dan Kreasi Reka) merupakan program Kementerian Pendidikan, Kebudayaan, Riset, dan Teknologi untuk mewujudkan sinergi antara Perguruan Tinggi dan Industri pada program *Matching Found*.
d. Penyetaraan Ijazah Luar Negeri merupakan Kementerian Pendidikan, Kebudayaan, Riset, dan Teknologi untuk menyetarakan Ijazah Luar Negeri dengan pendidikan yang berlaku di Indonesia.
e. SIVIL (Sistem Verifikasi Ijazah Elektronik) merupakan program Ditjen Diktiristek sebagai sistem untuk memverifikasi nomor ijazah dan memeriksa keaslian ijazah.
f. Selancar PAK (Sistem Pelacakan Secara Mandiri Penilaian Angka Kredit) merupakan program Ditjen Diktiristek sebagai sistem pelacakan angka kredit dalam mendapatkan informasi pengusulan kenaikan pangkat atau jabatan akademik.
g. Silemkerma (Sistem Informasi Layanan Perizinan Kelembagaan Perguruan Tinggi) merupakan program Ditjen Diktiristek sebagai sistem pengusulan yang berkaitan dengan pendidikan Perguruan Tinggi Swasta (PTS), perubahan, dan pembukaan PTS.

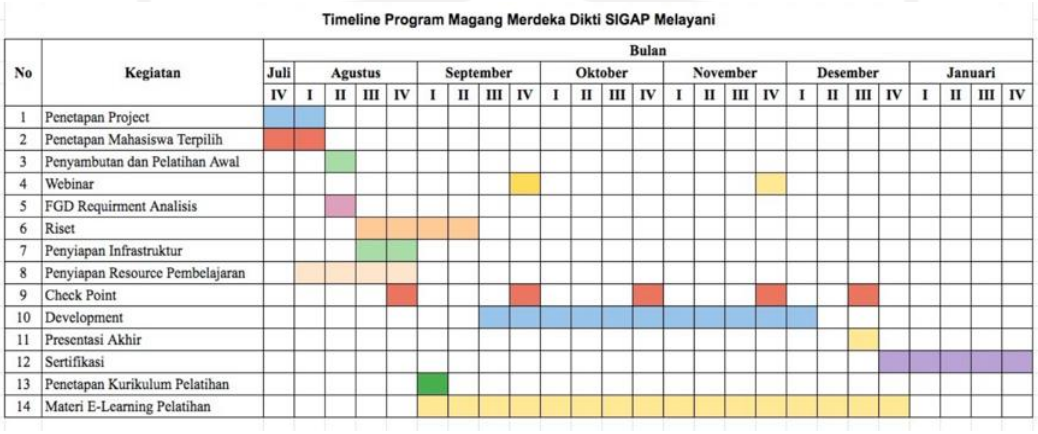
1.2 Ruang Lingkup

Terdapat beberapa posisi yang berperan dalam pengembangan proyek Satudikti diantaranya *Executive Sponsor*, *Steering Committe*, *Project Manager*, *Senior Product Manager*, Mentor, *Trainer*, *Product Manager*, *UI/UX*, *Graphic Designer*, *Data Analyst*, *Technical Writer*, *Mobile Developer*, *Front-end Developer*, *Back-end Developer*, dan *API Developer*. Setiap posisi memiliki peran dan tanggung jawab yang berbeda. Saat ini, penulis sebagai *Back-end Developer* dengan tanggung jawab yang diberikan yaitu pengerjaan di

belakang sistem dan berfokus pada pengerjaan *database*, membuat maupun mengkonsumsi API (*Application Programming Interface*), dan mengimplementasikannya ke dalam aplikasi Satudikti.

Dalam pengembangan proyek Satudikti, semua kegiatan yang dilakukan berada dibawah naungan *Project Manager*. Terdapat beberapa posisi yang berperan sebagai pemberi arahan dalam pengembangan proyek Satudikti yaitu *Senior Product Manager* yang berperan sebagai orang yang bertanggung jawab atas pemberi arahan dalam pengembangan suatu produk yaitu aplikasi Satudikti dan melaporkan progres pengembangan produk kepada *Project Manager*, *Trainer* yang berperan sebagai orang yang bertanggung jawab dalam pemberian pelatihan maupun materi sesuai dengan apa yang dibutuhkan oleh mahasiswa magang, dan Mentor yang berperan sebagai orang yang bertanggung jawab sebagai pemberi arah dalam pengerjaan *task* masing-masing posisi mahasiswa magang. Mentor hanya sebatas memberikan referensi dalam pengerjaan *task*, memeriksa pekerjaan mahasiswa magang, membantu memberikan solusi dalam memecahkan suatu masalah, dan mentor tidak ikut serta dalam pengembangannya. Pada pengembangan proyek Satudikti, penulis mendapat peran sebagai *Back-end Developer*.

Selama kurang lebih 6 bulan, kegiatan magang dilakukan pada periode bulan Agustus 2021 hingga Januari 2022. Dalam rentang waktu tersebut, penulis telah mendapatkan berbagai aktivitas selama magang di Ditjen Diktiristek. Aktivitas yang dilakukan diantaranya Penyambutan dan Pelatihan Awal, Webinar, FGD *Requirement Analysis*, Riset, Penyiapan Infrastruktur, Penyiapan *Resource* Pembelajaran, *Checkpoint*, *Development*, dan Presentasi Akhir. Pada Gambar 1.4 menunjukkan *Timeline* Kegiatan Magang.



Gambar 1.4 *Timeline* Kegiatan Magang

Proyek Satudikti dikembangkan bertujuan untuk menciptakan aplikasi yang menyatukan semua layanan yang ada pada Ditjen Diktiristek. Untuk memenuhi tujuan tersebut, sebagai *Back-end Developer* diberikan tanggung jawab dalam membangun Web *Service*. Web *Service* dibangun agar aplikasi Satudikti tidak berkomunikasi secara langsung terhadap data atau informasi yang disediakan oleh masing-masing layanan maupun pengolahan terhadap datanya. Sehingga Web *Service* yang dibangun sebagai penyedia data atau informasi dari masing-masing layanan dan pengolahan datanya. *Output* dari Web *Service* yang dibuat yaitu berupa API yang menyediakan kebutuhan data yang akan dikonsumsi oleh aplikasi Satudikti.

Adapun lingkup bahasan yang akan disusun pada laporan ini sebagai berikut:

a. Membuat API *Contract.*
b. Membuat *Database*.
c. Membuat REST API.

1.3 Tujuan

Tujuan dari dilaksanakannya kegiatan magang adalah sebagai berikut:

a. Tujuan Umum
    1. Pengguna layanan menggunakan satu aplikasi untuk mengetahui informasi dan dapat mengakses seluruh layanan yang ada pada Ditjen Diktiristek.
    2. Pengelolaan dan pengembangan layanan yang dilakukan secara terpadu, terpusat, dan terintegrasi dalam satu aplikasi.

b. Tujuan Khusus
    1. Membangun *database* sebagai wadah dalam menyimpan data yang dibutuhkan pada aplikasi Satudikti.
    2. Membangun Web *Service* yang menyediakan API sebagai sarana pertukaran data maupun informasi yang akan dikonsumsi oleh aplikasi Satudikti.
    3. Mengimplementasikan arsitektur REST API dalam membangun Web *Service*.
    4. Mendokumentasikan API agar mudah dibaca dan dimengerti oleh pemangku kepentingan yang akan menggunakan API yang dibuat.

1.4 Manfaat

Manfaat yang didapatkan dari kegiatan magang yang dilakukan dalam membangun Web *Service* yang dibuat dengan menerapkan arsitektur REST API dan menggunakan bahasa pemrograman Golang dalam pengembangannya adalah sebagai berikut:

a. Mempermudah interaksi antar platform terutama dalam pertukaran data maupun informasi dengan adanya API.
b. API yang dikembangkan bisa dikonsumsi oleh platform lain yang membutuhkan.
c. REST API yang diimplementasikan dapat membantu pembuatan Web *Service* sehingga API yang dikembangkan sesuai dengan standarisasi dari arsitektur REST.
d. Mampu menampung data-data apa saja yang dibutuhkan oleh aplikasi Satudikti dengan adanya *database*.

1.5 Sistematika Penulisan

Dalam penulisan Tugas Akhir ini, digunakan sistematika penulisan sebagai berikut:

a. BAB I PENDAHULUAN

   Bab ini berisi latar belakang, ruang lingkup, tujuan, manfaat, dan sistematika penulisan pada Laporan Tugas Akhir.

b. BAB II LANDASAN TEORI DAN TINJAUAN PUSTAKA

   Bab ini berisi teori-teori pembahasan yang mendukung membantu proses pembuatan Laporan Tugas Akhir.

c. BAB III PELAKSANAAN MAGANG

   Bab ini berisi tentang tahapan pelaksanaan dalam pengembangan proyek Satudikti yang dilakukan dalam mengimplementasikan REST API dalam membangun Web *Service* menggunakan bahasa pemrograman Golang.

d. BAB IV REFLEKSI PELAKSANAAN MAGANG

   Bab ini berisi tentang hal-hal yang didapatkan selama magang di Ditjen Diktiristek.

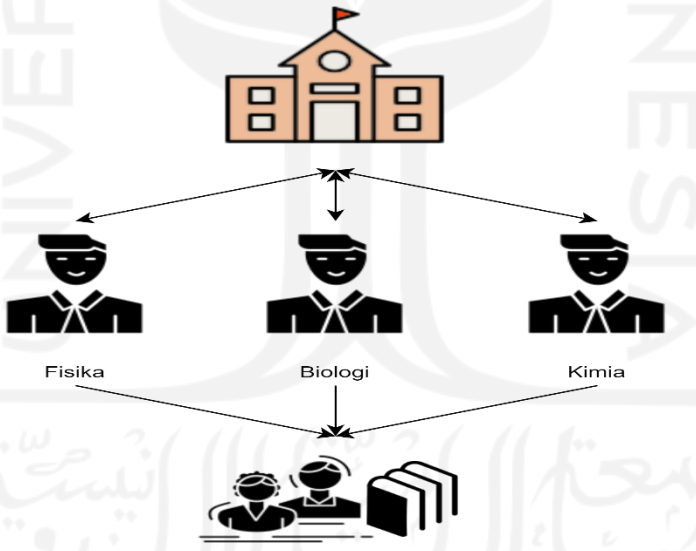
e. BAB V PENUTUP

   Bab ini berisi tentang kesimpulan dari penulisan Laporan Tugas Akhir yang telah dilakukan dan saran yang disampaikan oleh penulis.

BAB II
LANDASAN TEORI DAN TINJAUAN PUSTAKA

2.1 Application Programming Interface (API)

Application Programming Interface (API) adalah suatu antarmuka aplikasi yang memiliki sekumpulan instruksi yang tersimpan pada *library* dan menjelaskan bagaimana antar aplikasi dapat berkomunikasi dalam berbagi data dengan aplikasi lainnya (Firdaus et al., 2019). API bisa dianalogikan sebagai sekolah, guru, dan siswa. Interaksi yang terjadi yaitu sekolah menuntut guru untuk mengajarkan mata pelajaran kepada siswa. Sekolah tidak perlu mengetahui seperti apa cara guru dalam mendidik siswanya. Dari analogi ini dapat dijelaskan bahwa siswa sebagai perangkat lunak yang akan dikembangkan dan guru sebagai API yang mengerjakan bagian-bagian tertentu terhadap perangkat lunak tanpa mengetahui bagaimana cara melakukannya. Pada Gambar 2.1 menjelaskan Analogi API pada Sekolah.



Gambar 2.1 Analogi API Pada Sekolah

2.2 REST API

REST API merupakan suatu pengembangan API yang mengimplementasikan arsitektur REST. REST kependekan dari *Representational State Transfer*. REST bertujuan untuk menyederhanakan metode dalam melakukan transmisi data pada suatu jaringan (Kristanto, 2020). Dalam mentransmisikan data, API ditransmisikan melalui antarmuka yang terstandarisasi yaitu HTTP (Akbar, 2018).

Arsitektur REST mempunyai beberapa komponen sebagai berikut (Perdana, 2018):

a. URI *design*

REST API menggunakan protokol HTTP dalam mengakses *resource*. Untuk mengakses *resource* pada API, diperlukan *Uniform Resource Identifier* (URI) atau pada API biasanya disebut *endpoint*. Contohnya dalam pemanggilan *endpoint* atau URI yaitu *users*, *users/125*, *users/125/orders*, dan lain sebagainya.

b. HTTP *verbs*

Komponen ini bisa disebut juga sebagai metode ketika *client* melakukan *request* pada suatu *endpoint* terhadap *server*. Beragam metode yang terdapat pada HTTP, namun ada beberapa metode yang sering digunakan yaitu GET, POST, PUT, dan DELETE.

c. HTTP *response code*

Komponen ini merupakan suatu kode yang sudah menjadi standar pada protokol HTTP ketika *client* melakukan *request*. Terdapat 3 jenis kode yang sering digunakan pada suatu REST API diantaranya sebagai berikut:

- 2XX, menandakan *request* yang diminta *client* berhasil.
- 4XX, menandakan terjadi kesalahan pada sisi *client* saat melakukan *request*.
- 5XX, menandakan terjadi kesalahan pada sisi *server* saat melakukan *request*.

d. Format *response*

Setiap *request* yang dilakukan oleh *client*, *server* akan mengembalikan *response* berupa data atau informasi. Ada 2 format *response* yang umum digunakan pada REST API yaitu *Extensible Markup Language* (XML) ataupun *JavaScript Object Notation* (JSON).

2.3 API Contract

API *Contract* merupakan suatu dokumen yang menjelaskan bagaimana API bekerja. API disebut juga sebagai dokumen yang berisi kesepakatan bersama antar pengembang API dengan pengguna API agar *output* yang dihasilkan sesuai dengan apa yang mereka katakan. API *Contract* sangat diperlukan untuk menghindari hal-hal yang tak terduga dari sisi *client* yang menggunakan API seperti perubahan *response*. Hal ini akan mengakibatkan kerusakan yang terjadi pada sisi *client*. Sebagai pengembang API, API *Contract* merupakan hal yang penting dipastikan agar tidak terjadi kesalahan saat menggunakan API tersebut (Jools.dev, 2021).

Ada beberapa hal yang perlu diperhatikan dalam membuat API *Contract* sebagai berikut (BeattieM, 2016):

a. *Title*
b. URI/*Endpoint*
c. URI *Method*
d. URI *Params*
e. Data *Params*
f. *Headers*
g. *Success Response*
h. *Error Response*
i. *Code Response*
j. *Content Response*

2.4 Web Service

Web *Service* merupakan suatu software yang dirancang untuk melakukan interoperabilitas pada sistem dalam suatu jaringan dengan menggunakan format pertukaran data dalam mengirim pesan atau informasi. Web *Service* dibuat untuk berinteraksi dengan aplikasi ataupun *platform*. Dengan demikian Web *Service* tidak digunakan oleh user karena pada Web *Service* tidak menyediakan antarmuka untuk *user* (Saputra & Fathoni Aji, 2018).

Dalam proses pertukaran data, terdapat 3 komponen Web *Service* yaitu sebagai berikut (Pratomo & Haryono, 2020):

a. *Service Provider* (penyedia layanan), sebagai pemilik dan *platform* yang menyimpan akses layanan yang ada.
b. *Service Requestor* (peminta layanan), sebagai *client* yang melakukan interaksi terhadap layanan tersebut. *Client* tersebut bisa memiliki antarmuka maupun tanpa antarmuka.
c. *Service Registry*, sebagai komponen yang menyediakan deskripsi suatu layanan dimana *Service Provider* mempublikasikan deskripsi pada layanannya.

2.5 Third-party API

Third-party API merupakan API yang dikembangkan oleh pihak ketiga. Dalam penerapannya, *Third-party* API merupakan API eksternal yang ditambahkan dalam pengembangan API internal. Banyak keuntungan yang didapat dalam mengintegrasikan *Third-*

party API, salah satunya dapat memperkaya fungsionalitas pada API internal yang dikembangkan dan juga sangat efisien dalam mengurangi biaya maupun waktu dalam pengembangan suatu API yang dibutuhkan (Nagaraj & N, 2020).

Berdasarkan cara kerjanya, terdapat beberapa jenis pada *Third-party* API yaitu sebagai berikut (iTech, 2021):

a. *Public or Free APIs*, merupakan API terbuka dan gratis diakses oleh bisnis maupun pengembang luar manapun.
b. *Partner APIs*, merupakan API yang memfasilitasi *business-to-business* dan hanya untuk pengguna resmi saja
c. Internal *APIs*, merupakan API yang digunakan secara pribadi maupun digunakan oleh perusahaan bisnis itu sendiri.

2.6 Golang

Golang (Google *Language*) adalah bahasa pemrograman yang dikembangkan oleh Google pada tahun 2009 bersama dengan Ken Thompson, Robert Griesemer dan Rob Pike. Tujuan pengembangannya adalah untuk menciptakan bahasa yang memiliki keunggulan dalam hal kecepatan, kehandalan, skalabilitas, dan kesederhanaan (Sari & Hidayat, 2022). Golang merupakan bahasa pemrograman yang menggabungkan keamanan dan efisiensi untuk mengembangkan sistem *open source*. Golang mendukung konkurensi dalam sistem pemrograman melalui implementasi yang sangat sederhana. Golang juga memiliki sistem *garbage collection* yang baik dengan memanfaatkan bantuan *built-in garbage collector process* (Goroutines). Golang termasuk bahasa pemrograman yang andal dan cepat dalam skala besar dan bersifat *clean code* agar tidak membebani sistem (Ari Kristanto et al., 2020).

2.7 Echo

Echo merupakan web *framework* pada bahasa pemrograman Golang. *Framework* Echo memiliki performa tinggi, *extensibility*, dan minimalis sehingga cocok untuk digunakan pada sistem dengan skala yang besar (LabStack, 2022). Selain itu, *framework* Echo memiliki beberapa keunggulan lainnya yaitu *optimized router*, didukung HTTP/2, *scalable*, data *binding*, data *rendering*, *middleware*, *automatic* TLS, dan error handling yang bisa dikustomisasi. Dengan demikian, *framework* Echo dapat mendukung dalam pembuatan Web *Service* (Subagio & Muttaqin, 2022).

2.8 Postman

Postman merupakan *tools* atau *platform* kolaboratif untuk melakukan pengembangan suatu API. Postman awalnya dikembangkan oleh Abhinav Astana dan dua mantan rekannya, Ankit Sobti dan Abhijit Kane, untuk menyederhanakan proses pengembangan dan pengujian API (Postman, 2022). Postman berfungsi sebagai REST *client* untuk melakukan pengujian REST API yang biasanya digunakan oleh *developer* sebagai *tools* pengujian terhadap API yang dibuat.

Terdapat berbagai fitur sederhana pada Postman yang dapat membantu proses pengembangan suatu API yaitu sebagai berikut (Ahmad, 2018):

a. *Collection*, pengelompokkan API yang disimpan dalam bentuk folder sehingga memudahkan pengelompokkan API yang dibuat sesuai dengan modul yang dikerjakan.
b. *Environment*, seperti konfigurasi untuk menyimpan suatu variabel atau atribut dan atribut tersebut bisa digunakan atau memanipulasi suatu proses *request* API.
c. *"Example" Response*, *developer* dapat merancang *mockup* atau sampel API sebelum mengimplementasikan API pada proyek.
d. *Mock Server*, mirip *fake server* yang bertugas sebagai *server* sementara. Fitur ini memungkinkan sampel API yang dibuat menggunakan fitur *"Example Response"* bisa diakses dengan internet seperti API yang sudah diimplementasikan dan disebarkan.
e. *Script Test*, fitur yang dapat melakukan validasi dan memanipulasi suatu *response* dan di dalamnya dituliskan *test* sesuai kebutuhannya.
f. *Automated Test*, melakukan proses *request* dalam satu *collection* secara otomatis dan umumnya digunakan bersamaan dengan *script test*.
g. API *Documentation*, dokumentasi API atau API *Contract* dapat di *generate* secara otomatis sesuai dengan *request* yang tersimpan dalam satu collection.

2.9 Scrum

Scrum adalah kerangka kerja yang mengelola proses pengembangan perangkat lunak. *Scrum* termasuk dalam metode *Agile Development* yang menggunakan pendekatan iterasi dan berkelanjutan. Ada 3 elemen organisasi utama dalam *Scrum* yaitu *Product Owner*, *Scrum Master* dan *The Scrum Team*. Seorang *Scrum Master* hadir untuk membantu anggota tim sepenuhnya memanfaatkan kerangka kerja *Scrum*. Pada elemen ini, *Scrum Master* digunakan

oleh tim sebagai pelatih. *Product Owner* berfungsi untuk mendukung tim untuk mengembangkan produk ke arah yang benar dengan melibatkan perusahaan dan pelanggan. Sedangkan *The Scrum Team* mencakup sekelompok orang yang bertugas mengembangkan perangkat lunak (Sunardi & Fadli, 2018).

Metode *Scrum* memiliki rangkaian aktivitas yaitu sebagai berikut (Suharno et al., 2020):

a. *Backlog*, merupakan *list* dari fitur-fitur atau kebutuhan yang diinginkan *client.*
b. *Sprints*, merupakan iterasi atau fase kerja yang dibutuhkan dalam menyelesaikan *list* pada *Backlog* sesuai dengan waktu yang ditentukan.
c. *Scrum Meetings*, merupakan pertemuan rutin yang dilakukan yang bertujuan untuk melakukan evaluasi terhadap apa yang dikerjakan, kendala yang dihadapi, dan juga target terhadap pekerjaan tersebut.
d. Demo, memberikan peningkatan atau pembaruan terhadap perangkat lunak kepada *client* yang telah ditampilkan, diterapkan, dan dievaluasi oleh *client*.

2.10 Tinjauan Pustaka

Selama penulisan laporan ini, dilihat dari beberapa penelitian yang menggunakan informasi kasus serupa sebagai dasar penulisan. Dimulai dari penelitian (Putra et al., 2018) yang memanfaatkan Web *Service* pada pengembangan aplikasi Sistem Inventaris Berbasis QR Code. Dengan adanya Web *Service*, dapat bermanfaat dalam melakukan pertukaran data sehingga mampu menjadi sebuah jembatan penghubung antara berbagai sistem yang ada dan menyelesaikan permasalahan berbagai bidang.

Dalam membangun Web *Service*, terdapat arsitektur yang bisa diterapkan pada pengembangannya. Penelitian (Firdaus et al., 2019) menjelaskan tentang perancangan Web *Service* pada Sistem Informasi Perpustakaan. Pada penelitiannya, menjelaskan bahwa arsitektur yang digunakan dalam membangun Web *Service* ada dua yaitu REST dan SOAP. Namun, pada penelitiannya menggunakan arsitektur RESTful. RESTful adalah nama lain dari Web *Service* yang menerapkan arsitektur REST pada pengembangannya. karena arsitektur tersebut merupakan salah satu standar arsitektur yang umum digunakan dan berjalan pada protokol HTTP dalam melakukan pertukaran data. Selain itu, penelitian (Rizal & Rahmatulloh, 2019) juga menerapkan RESTful dalam merancang Web *Service* untuk Integrasi Sistem Akademik dan Perpustakaan. Pada penelitiannya menggunakan JSON sebagai format pertukaran datanya. Menurutnya, JSON bersifat *stateless* sehingga dapat memudahkan pengaksesan melalui bahasa pemrograman, arsitektur, atau sistem operasi lainnya.

Berdasarkan tinjauan penelitian tersebut, pengerjaan proyek yang dikerjakan akan menerapkan metode REST sebagai arsitektur dalam membangun Web *Service* dan memilih JSON sebagai format pertukaran datanya. Untuk teknologi yang akan digunakan yaitu menggunakan bahasa pemrograman Golang dalam pengembangan Web *Service*. Seperti penelitian (Kristanto, 2020) yang menggunakan Golang dalam pengembangan Web *Service* karena Golang mendukung sistem konkurensi, bantuan *built-in garbage collector process*, dan pemrosesan kode yang cepat dan handal dalam skala yang besar. Penelitian ini bertujuan untuk membangun Web *Service* yang keluarannya berupa API yang akan dikonsumsi aplikasi Satudikti dalam mendapatkan data maupun informasi.

BAB III
PELAKSANAAN MAGANG

3.1 Manajemen Proyek

Dalam pengembangan proyek Satudikti, manajemen proyek yang diterapkan ada beberapa tahapan meliputi pendefinisian proyek, inisialisasi proyek, perencanaan proyek, dan pelaksanaan proyek. Tahapan tersebut dilakukan agar pengembangan proyek Satudikti menjadi terarah dan terorganisir.

3.1.1 Pendefinisian Proyek

Banyaknya layanan yang dikembangkan pada Ditjen Diktiristek berpotensi menjadi aplikasi yang bersifat silo. Harapannya proyek Satudikti yang dikembangkan dapat mengintegrasikan semua layanan yang ada pada Ditjen Diktiristek agar semua layanan menjadi terpusat dan terpadu dalam sebuah aplikasi. Produk yang dihasilkan pada proyek ini berupa aplikasi *mobile* dan *website*. Namun, aplikasi *mobile* menjadi prioritas utama pada keluaran proyek karena aplikasi *mobile* banyak digunakan oleh masyarakat dan bisa menjadi media yang baik untuk peningkatan layanan.

3.1.2 Inisialisasi Proyek

Pada tahap inisialisasi proyek Satudikti, telah ditentukan peran masing-masing anggota magang selama pengerjaan proyek Satudikti. Setiap peran memiliki tanggung jawab yang berbeda. Adapun peran yang dibutuhkan dalam pengembangan proyek Satudikti adalah sebagai berikut:

a. *Product Manager*, bertanggung jawab atas kebutuhan sumber daya dan mengontrol pengembangan produk secara keseluruhan untuk mencapai tujuan proyek.
b. *UI/UX Designer*, bertanggung jawab dalam merancang desain dan interaksi pada aplikasi yang akan dibuat.
c. *Graphic Designer*, bertanggung jawab atas penyedia elemen atau komponen visual terhadap desain yang dibutuhkan.
d. *Data Analyst*, bertanggung jawab atas pengumpulan, pengolahan, dan penyedia data untuk dijadikan informasi.

e. *Technical Writer*, bertanggung jawab atas penyusunan dokumentasi teknis yang dibutuhkan terhadap pemangku kepentingan.
f. *Mobile Developer*, bertanggung jawab atas pengembangan aplikasi berbasis *Mobile*.
g. *Front-end Developer*, bertanggung jawab atas pengembangan aplikasi berbasis Web.
h. *Back-end Developer*, bertanggung jawab atas pengembangan Web *Service*.
i. *API Developer*, bertanggung jawab atas membangun infrastruktur yang terotomatisasi dan terintegrasi dalam proses antara pengembangan aplikasi dan tim pengembang.

3.1.3 Perencanaan Proyek

Pada tahap perencanaan proyek Satudikti, telah ditentukan kebutuhan modul-modul apa saja yang dikembangkan pada proyek Satudikti. Modul-modul yang dikembangkan adalah sebagai berikut:

a. Modul Registrasi.
b. Modul Berita/Pengumuman.
c. Modul Penelusuran Status Usulan Penyetaraan Ijazah Luar Negeri.
d. Modul Penelusuran Selancar PAK.
e. Modul Penelusuran SIVIL.
f. Modul PDDikti.
g. Modul Kedaireka.
h. Modul Kampus Merdeka.
i. Modul Penelusuran Silemkerma.

Setelah kebutuhan modul-modul yang akan dikembangkan sudah ditentukan, kemudian menentukan spesifikasi teknologi yang akan digunakan dalam pengembangan proyek. Sebagai *Back-end Developer* yang bertugas dalam membangun Web *Service* pada proyek Satudikti, teknologi yang digunakan dapat dilihat pada Tabel 3.1.

Tabel 3.1 Teknologi yang Digunakan

Aspek	Teknologi
Bahasa Pemrograman	Golang
Framework	Echo
Code Editor	Goland/VSCode
Database	PostgreSQL
Database Platform	DBeaver
Version Control System	Git
Version Control System Platform	GitHub
API Testing	Postman
Virtual Private Network	OpenVPN
Software Management Project	Asana

Dalam pengembangannya, proyek Satudikti mengimplementasikan metode manajemen proyek yaitu *agile* dengan kerangka kerja *Scrum*. Pengembangan proyek ini akan berlangsung selama 4 bulan, terhitung dari bulan September 2021 sampai dengan bulan Desember 2021. Dalam penerapannya, ada beberapa rangkaian aktivitas yang akan dilakukan selama pengembangan proyek yaitu sebagai berikut:

a. *Sprint Refinement*, aktivitas ini dilakukan di awal *sprint* yang bertujuan untuk mendaftarkan *task* yang akan dikerjakan masing-masing tim.
b. *Daily Stand-Up Meeting*, aktivitas ini merupakan aktivitas harian di pagi hari yang bertujuan untuk memantau kerjaan setiap tim mulai dari progres, kebutuhan, dan kendala yang dihadapi.
c. *Technical Meeting*, aktivitas ini dilakukan di pertengahan *sprint* yang bertujuan untuk *sharing session* terhadap masalah teknis yang dihadapi masing-masing tim bersama mentornya.
d. *Sprint Review*, aktivitas ini dilakukan di akhir *sprint* yang bertujuan untuk meninjau kembali kerjaan masing-masing tim apakah sudah mencapai target yang diharapkan.
e. Koordinasi Proyek, aktivitas ini dilakukan setiap 2 *sprint* sekali yang bertujuan untuk melaporkan progres, kebutuhan, maupun kendala pengerjaan proyek kepada *Project Manager*.

3.1.4 Pelaksanaan Proyek

Dalam pelaksanaan proyek Satudikti, masing-masing anggota tim akan bekerja sesuai dengan tanggung jawabnya dalam mengembangkan modul-modul yang dibutuhkan. Kebutuhan utama dari modul-modul yang dikembangkan adalah penelusuran dan pencarian

data. Dalam pengembangan modul-modul yang sudah direncanakan, penulis diikut sertakan beberapa modul sebagai berikut:

a. Modul Berita/Pengumuman.
b. Modul Penelusuran Status Usulan Penyetaraan Ijazah Luar Negeri.
c. Modul PDDikti.
d. Modul Kedaireka.

Untuk memenuhi kebutuhan tersebut, penulis sebagai *Back-end Developer* yang bertanggung jawab dalam pengembangan Web *Service* berfokus pada pengembangan REST API. Terdapat 3 tahapan yang penulis lakukan dalam mengembangkan REST API adalah sebagai berikut:

a. Perancangan REST API

   Tahap perancangan berfungsi sebagai dasar dalam pengembangan. Saat merancang API menggunakan arsitektur REST berbasis web. Dengan demikian, pada tahap ini dibutuhkan pembuatan dokumentasi kebutuhan suatu API agar API yang dibuat oleh *Back-end Developer* sesuai dengan kesepakatan *Front-end Developer* dan *Mobile Developer*. Selain itu, tahap ini juga membahas rancangan database yang akan digunakan.

b. Implementasi REST API

   Tahap implementasi menjelaskan bagaimana membangun REST API dengan menggunakan bahasa pemrograman Golang pada Web *Service* yang akan dibuat dan implementasi *database* jika dibutuhkan penampungan data pada aplikasi Satudikti. Pada tahap ini, simulasi yang dilakukan dengan penyimpanan lokal, menggunakan alamat *localhost*, dan nomor *port* 8089.

c. Pengujian API

   Tahap pengujian bertujuan untuk menguji API dibuat sudah berjalan dengan baik. Dalam pengujiannya dapat dilakukan dengan menggunakan teknologi Postman. Pengujian juga dilakukan untuk melihat kesesuaian API yang dibuat dengan kontrak API yang sudah dirancang pada dokumentasi API *Contract*.

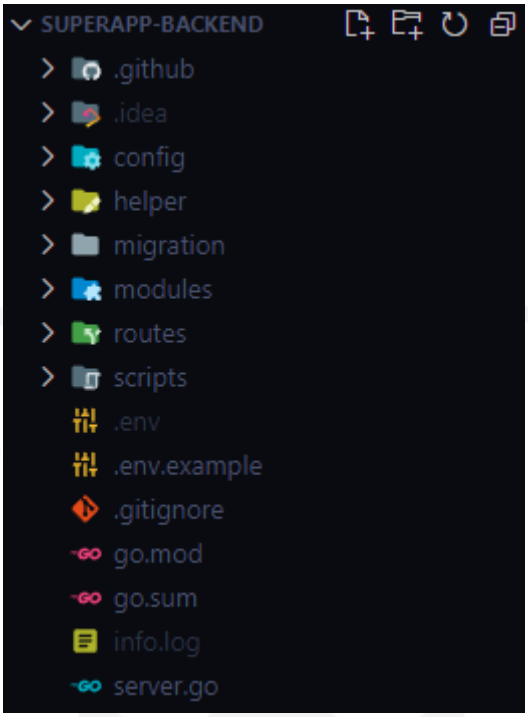
Dalam pelaksanaannya, setiap pengembangan modul tersebut memiliki durasi pada proses pengembangannya. Durasi tersebut telah ditentukan oleh tim yang mengontrol pengembangan produk yaitu *Product Manager*. Dari modul-modul yang akan dikembangkan oleh penulis, terdapat rincian kebutuhan yang telah ditentukan dan memiliki durasi dalam pengerjaannya yang dapat dilihat pada tabel 3.2. Selain itu terdapat aktivitas yang dilakukan

selama pengembangan proyek dan pelaksanaan magang yang sudah tercatat secara rinci yang dapat dilihat pada Lampiran C.

Tabel 3.2 Pelaksanaan Proyek

Modul	**Waktu Pengerjaan**	**Tugas**	**Durasi**
Berita/Pengumuman	Oktober (1 Bulan)	Membuat Rancangan Sistem	1 Minggu (Pekan Pertama)
		Membuat PDM	
		Membuat API *Contract*	
		Create API *List* Berita dan Pengumuman	2 Minggu (Pekan Kedua dan Ketiga)
		Create API *Detail* Berita dan Pengumuman *by* Id	
		Create API *Detail* Berita dan Pengumuman *by* Slug	1 Minggu (Pekan Keempat)
		Create API *List* Kategori Berita dan Pengumuman	
Penelusuran Status Usulan Penyetaraan Ijazah Luar Negeri	November (1 Bulan)	Membuat Rancangan Sistem	1 Minggu (Pekan Pertama)
		Membuat PDM	
		Membuat API *Contract*	
		Create API *List* Negara	1 Minggu (Pekan Kedua)
Kedaireka	November (1 Bulan)	Membuat Rancangan Sistem	1 Minggu (Pekan Ketiga)
		Membuat API *Contract*	
		Create API Kreasireka	1 Minggu (Pekan Keempat)
PDDikti	Desember – Januari (2 Bulan)	Membuat Rancangan Sistem	1 Minggu (Pekan Pertama)
		Membuat API *Contract*	
		Create API *List* Perguruan Tinggi *by* Keyword	1 Minggu (Pekan Kedua)
		Create API *Detail* Perguruan Tinggi *by* Id	1 Minggu (Pekan Ketiga)
		Create API Akreditasi Perguruan Tinggi *by* Id	
		Create API *List* Prodi *from* Perguruan Tinggi *by* Keyword	1 Minggu (Pekan Keempat)
		Create API *Detail* Prodi *from* Perguruan Tinggi *by* Id	1 Minggu (Pekan Kelima)
		Create API Akreditasi Prodi *from* Perguruan Tinggi *by* Id	1 Minggu (Pekan Keenam)
		Create API *List* Dosen *from* Prodi	1 Minggu (Pekan Ketujuh)
		Create API *Search All*	1 Minggu (Pekan Kedelapan)

Sebelum melaksanakan pengembangan Web *Service* yang akan digunakan aplikasi Satudikti, kerangka kerja yang akan digunakan dalam membuat sintaks sudah disediakan. Harapannya *Back-end Developer* tidak perlu membuat ulang dari nol, melainkan menyesuaikan dengan kerangka kerja yang sudah dibuat oleh mitra. Pada Gambar 3.1 menunjukkan kerangka kerja yang akan digunakan.



Gambar 3.1 Kerangka Kerja

Dilihat dari Gambar 3.1, terdapat beberapa folder dan file yang digunakan perlu diketahui sebagai berikut:

a. *config*, folder ini berisi file yang berhubungan dengan konfigurasi terhadap sesuatu seperti konfigurasi *database.*
b. *helper*, folder ini berisi file yang berhubungan fungsi atau *method* bantuan yang berguna ketika membuat sintaks nantinya seperti *error handling, validation, sql query, call http request,* dan lain-lain.
c. *migration*, folder ini berisi file yang berhubungan dengan *version control* pada *database.*
d. *modules*, folder ini berisi file yang berhubungan dengan sintaks pada semua modul yang akan dikembangkan. Pada saat pengerjaan API pada modul-modul yang akan dikembangkan, penulis akan banyak menulis sintaks pada folder ini.
e. *routes*, folder ini berisi file yang berhubungan dengan pembuatan alamat *baseurl* suatu *endpoint* yang akan direncanakan. Pada saat pengerjaan API pada modul-modul yang akan dikembangkan, penulis akan ada menulis sintaks pada folder ini.
f. *scripts*, folder ini berisi file yang berhubungan dengan perintah *Command Line Interface* (CLI) yang dibutuhkan dan dijalankan pada terminal.

g. *.env*, file ini berisi variabel-variabel yang menampung suatu data yang dibutuhkan.
h. *server.go*, file ini berisi fungsi atau method yang berhubungan dengan pertama kali suatu program akan dijalankan atau diproses. File ini juga berisi perintah dalam mendefinisikan suatu alamat URL yaitu *localhost* dan *port* yaitu 8089.

Dalam penelitian yang dilakukan, terdapat batasan yang harus dilakukan karena adanya perjanjian yang telah disepakati baik secara lisan dan tulisan yang terdapat pada Lampiran B. Batasan lain yang disepakati yaitu sebagai berikut:

a. Pada tahap Perancangan REST API, rancangan arsitektur sistem dan *Physical Data Model* (PDM) pada semua modul yang dikerjakan oleh penulis dan untuk dokumentasi API *Contract* hanya pada modul Berita/Pengumuman. Untuk dokumentasi API *Contract* tidak ditampilkan contoh *request* ataupun *response* terhadap API.
b. Pada tahap Implementasi REST API, hanya menjelaskan pengerjaan yang dilakukan pada modul Berita/Pengumuman.
c. Pada tahap Pengujian REST API, hanya menjelaskan pengujian yang dilakukan pada modul Berita/Pengumuman.

3.2 Proyek Satudikti

3.2.1 Perancangan REST API

Dalam merancang REST API, ada beberapa tahapan yang perlu dilakukan diantaranya sebagai berikut:

a. Merancang Arsitektur Sistem, bertujuan untuk memahami alur proses dan kebutuhan dari modul yang dikembangkan.
b. Merancang *Physical Data Model* (PDM), bertujuan untuk membuat konsep model *database* sebelum diimplementasikan terhadap modul yang membutuhkan penyimpanan data.
c. Membuat Dokumentasi API *Contract*, bertujuan untuk membuat aturan-aturan yang ditetapkan terhadap *request* dan *response* dari API yang dikembangkan pada modul tersebut.

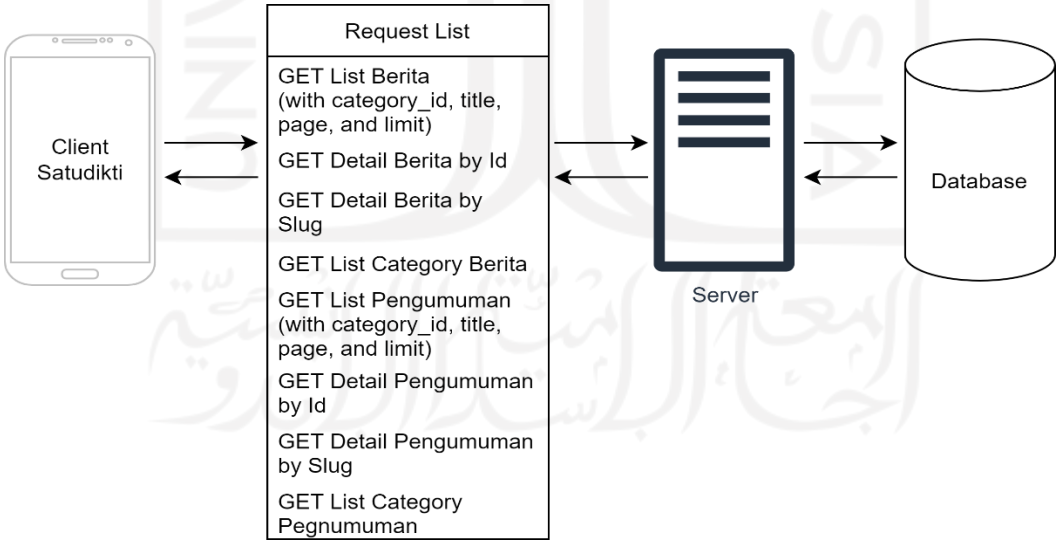
Dalam merancang REST API, saat membuat Dokumentasi API *Contract* ada beberapa hal yang perlu diperhatikan diantaranya sebagai berikut:

a. Menentukan HTTP *method* yang digunakan seperti GET, POST, PUT, atau DELETE.

b. Menentukan URI atau alamat identitas untuk mengakses *resource* tersebut yang biasanya disebut dengan *Endpoint*.
c. Memberikan *Parameter* jika dibutuhkan yang merupakan bagian dari *endpoint* ketika mengakses *endpoint*.
d. Menyediakan *Query Parameter* jika dibutuhkan yang merupakan bagian dari argumen permintaan ketika mengakses *endpoint*.
e. Menentukan struktur response untuk hasil response ketika mengakses *endpoint*.
f. Memberikan contoh request dalam mengakses *endpoint*.
g. Memberikan contoh response dalam mengakses *endpoint*.

Modul Berita/Pengumuman

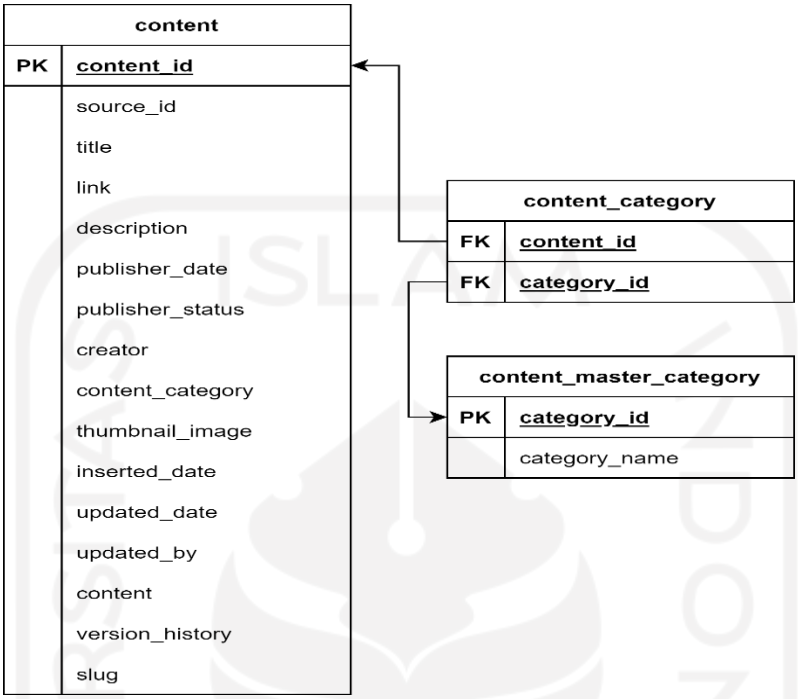
Kebutuhan yang diinginkan pada aplikasi Satudikti adalah untuk menampilkan informasi seputar berita/pengumuman di Ditjen Diktiristek. Harapannya data tersedia pada *database*. Untuk memenuhi kebutuhan tersebut, maka dibutuhkan API sebagai sarana untuk mendapatkan informasi tersebut. Seperti yang telah dijelaskan sebelumnya, selanjutnya akan dilakukan perancangan arsitektur sistem. Pada Gambar 3.2 menunjukkan Arsitektur Sistem pada Modul Berita/Pengumuman.



Gambar 3.2 Arsitektur Sistem Modul Berita/Pengumuman

Dilihat dari rancangan arsitektur yang telah dibuat, aplikasi Satudikti membutuhkan 6 *request endpoint*. *Request* yang diminta berupa aksi membaca data dari *database*. Harapannya, *server* dapat memenuhi permintaan tersebut.

Setelah merancang arsitektur sistem, selanjutnya membuat rancangan PDM yang akan dibuat. Pada Gambar 3.3 menunjukkan Rancangan PDM pada Modul Berita/Pengumuman.



Gambar 3.3 Rancangan PDM Modul Berita/Pengumuman

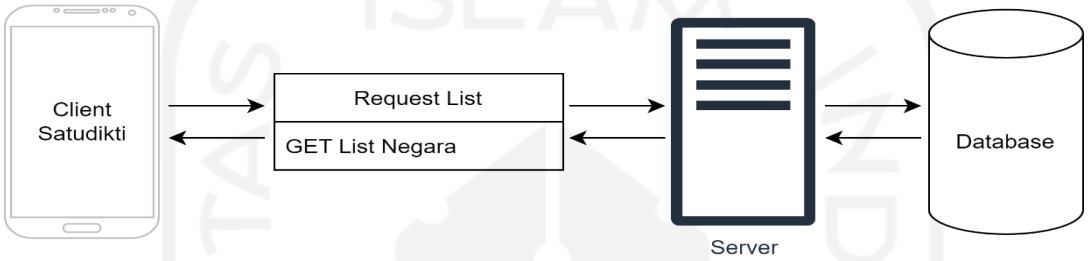
Modul Berita/Pengumuman membutuhkan *database* yang dapat menampung isi konten dan kategori konten. Dilihat dari kebutuhannya, konten dan kategori konten memiliki relasi *Many to Many* yang mana satu atau lebih konten dapat memiliki satu atau lebih kategori konten. Dari rancangan PDM yang dibuat, modul Berita/Pengumuman membutuhkan 3 tabel yaitu tabel *content* sebagai tabel yang menampung data konten dari suatu berita/pengumuman, tabel *content_master_category* sebagai tabel yang menampung data kategori konten dari suatu berita/pengumuman, dan tabel *content_category* sebagai *pivot table* relasi antara konten dan kategori konten.

Setelah merancang PDM, selanjutnya membuat dokumentasi API *Contract* terhadap API yang akan dibuat. Untuk dokumentasi API *Contract* modul Berita/Pengumuman dapat dilihat pada bagian Lampiran B.

Modul Penelusuran Status Usulan Penyetaraan Ijazah Luar Negeri

Kebutuhan yang diinginkan pada aplikasi Satudikti adalah untuk menampilkan informasi berupa status penyetaraan ijazah luar negeri yang diusulkan. Pada modul ini, penulis diberikan

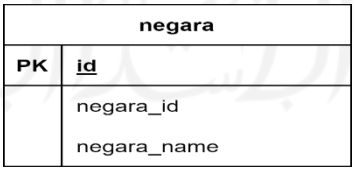
tugas untuk membuat API yang dapat memenuhi kebutuhan data berupa nama dan kode pada suatu negara yang terdaftar pada Ditjen Dikti. Harapannya data tersedia pada *database*. Untuk memenuhi kebutuhan tersebut, maka dibutuhkan API sebagai sarana untuk mendapatkan informasi status penyetaraan ijazah luar negeri yang diusulkan. Seperti yang telah dijelaskan sebelumnya, selanjutnya akan dilakukan perancangan arsitektur sistem. Pada Gambar 3.4 menunjukkan Arsitektur Sistem pada Modul Penelusuran Status Usulan Penyetaraan Ijazah Luar Negeri.



Gambar 3.4 Arsitektur Sistem Modul Penelusuran
Status Usulan Penyetaraan Ijazah Luar Negeri

Dilihat dari rancangan arsitektur yang telah dibuat, sesuai tugas yang diberikan kepada penulis aplikasi Satudikti membutuhkan 1 *request endpoint* untuk mendapatkan daftar data negara. *Request* yang diminta berupa aksi membaca data dari *database*. Harapannya, *server* dapat memenuhi permintaan tersebut.

Setelah merancang arsitektur sistem, selanjutnya membuat rancangan PDM yang akan dibuat. Pada Gambar 3.5 menunjukkan Rancangan PDM pada Modul Penelusuran Status Usulan Penyetaraan Ijazah Luar Negeri.



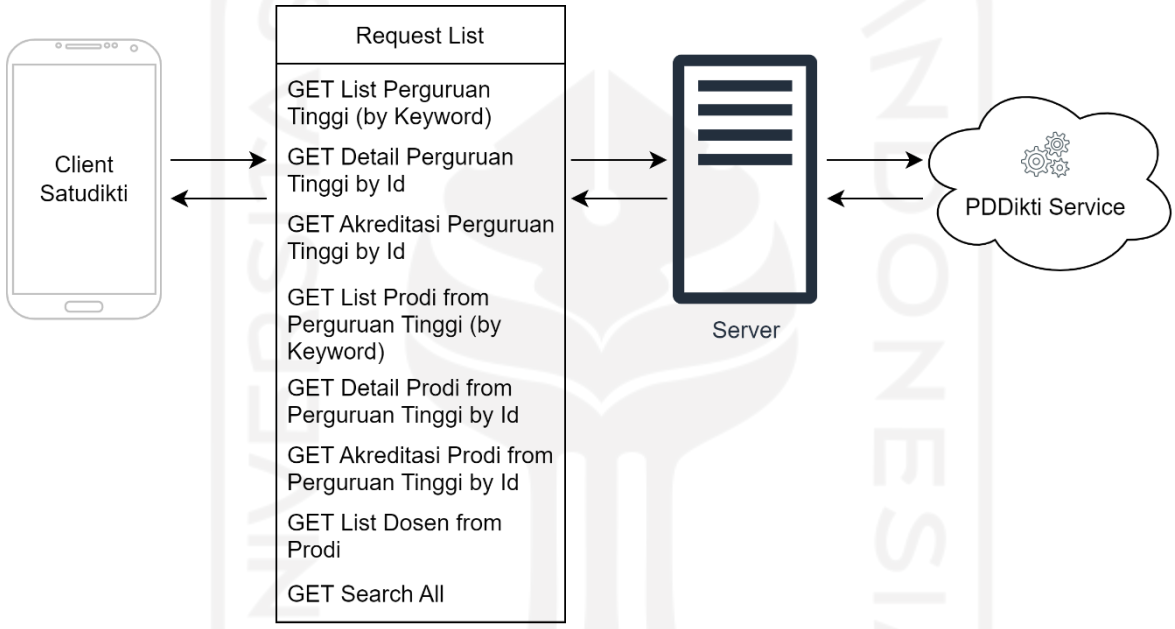
Gambar 3.5 Rancangan PDM Modul Penelusuran
Status Usulan Penyetaraan Ijazah Luar Negeri

Untuk menyimpan data negara, dibutuhkan 1 tabel untuk menyimpan kode negara dan nama negara. Sesuai dengan PDM yang telah dibuat, terdapat 3 *field* yaitu *id* sebagai *primary*

key, *negara_id* sebagai *field* yang menyimpan kode negara, dan *negara_name* sebagai *field* yang menyimpan data nama negara.

Modul PDDikti

Kebutuhan yang diinginkan pada aplikasi Satudikti adalah untuk mencari informasi seputar Perguruan Tinggi, Program Studi, Mahasiswa, dan Dosen. Untuk memenuhi kebutuhan tersebut, maka dibutuhkan API sebagai sarana untuk mendapatkan informasi yang dicari. Pada Gambar 3.6 menunjukkan Arsitektur Sistem pada Modul PDDikti.



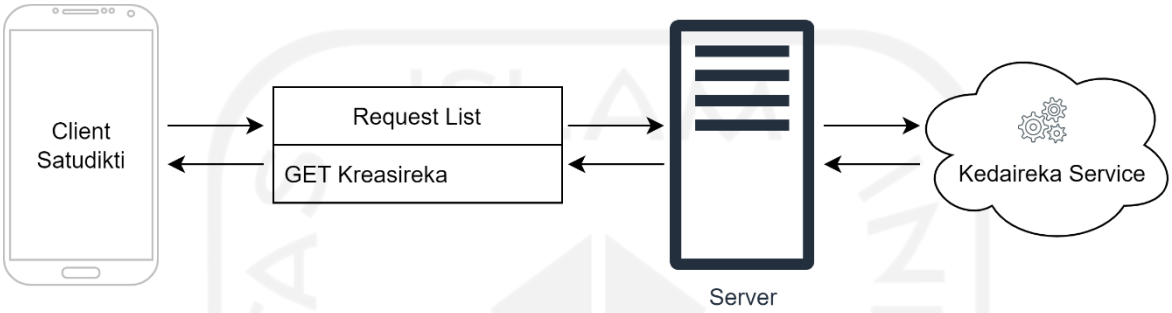
Gambar 3.6 Arsitektur Sistem Modul PDDikti

Dilihat dari rancangan arsitektur yang telah dibuat, sesuai tugas yang diberikan kepada penulis aplikasi Satudikti membutuhkan 8 *request endpoint* untuk mendapatkan informasi yang dibutuhkan. Request yang diminta tidak terdapat pada *database* yang disediakan, melainkan terdapat pada *Third-party APIs* yaitu PDDikti *Service*. Penulis dituntut untuk melakukan *forwarding* API terhadap *request* yang diminta oleh aplikasi Satudikti. Harapannya, *server* dapat memenuhi permintaan tersebut.

Modul PDDikti tidak membutuhkan *database* untuk menyimpan suatu data karena data yang dibutuhkan pada aplikasi Satudikti diperoleh dari PDDikti *Service*. Dengan demikian penulis diminta untuk melakukan *forwarding* API untuk memberikan data yang dibutuhkan.

Modul Kedaireka

Kebutuhan yang diinginkan pada aplikasi Satudikti adalah untuk menampilkan informasi seputar kreativitas dan inovasi antar perguruan tinggi dengan industri. Untuk memenuhi kebutuhan tersebut, maka dibutuhkan API sebagai sarana untuk menampilkan informasi yang dibutuhkan. Pada Gambar 3.7 menunjukkan Arsitektur Sistem pada Modul Kedaireka.



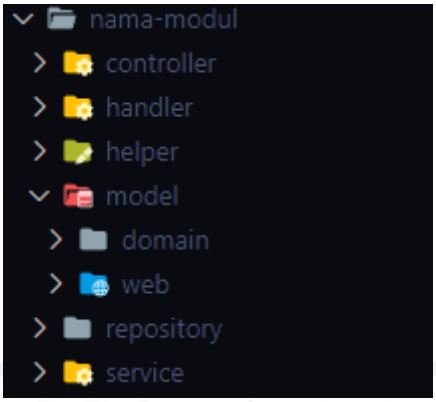
Gambar 3.7 Arsitektur Sistem pada Modul Kedaireka

Dilihat dari rancangan arsitektur yang telah dibuat, sesuai tugas yang diberikan kepada penulis aplikasi Satudikti membutuhkan 1 *request endpoint* untuk mendapatkan informasi yang dibutuhkan. Request yang diminta tidak terdapat pada *database* yang disediakan, melainkan terdapat pada *Third-party APIs* yaitu Kedaireka *Service*. Penulis dituntut untuk melakukan *forwarding* API terhadap *request* yang diminta oleh aplikasi Satudikti. Harapannya, *server* dapat memenuhi permintaan tersebut.

Modul Kedaireka tidak membutuhkan *database* untuk menyimpan suatu data karena data yang dibutuhkan pada aplikasi Satudikti diperoleh dari Kedaireka *Service*. Dengan demikian penulis diminta untuk melakukan *forwarding* API untuk memberikan data yang dibutuhkan.

3.2.2 Implementasi REST API

Dalam mengimplementasikan REST API terhadap modul yang dikembangkan pada aplikasi Satudikti, dengan kerangka kerja yang sudah disediakan penulis tidak perlu memikirkan konfigurasi-konfigurasi lainnya karena semua konfigurasi sudah dibuatkan dan diatur oleh pihak mitra. Penulis hanya perlu mengikuti alur penulisan sintaks. Pada tahap awal pengerjaan tugas yang diberikan, penulis dibantu oleh mentor dalam pembuatan sintaks yang akan dibuat. Sesuai pada bahasannya sebelumnya, penulis akan banyak menulis sintaks pada folder *modules*. Pada folder *modules*, terdapat struktur folder yang akan dibuat yang dapat dilihat pada Gambar 3.8.



Gambar 3.8 Struktur Folder *Modules*

Dilihat dari struktur folder tersebut, terdapat folder-folder yang perlu dipahami sebagai berikut:

a. *controller*, folder ini berisi *file* yang berhubungan dengan fungsi atau *method* yang berkaitan dengan penanganan respon suatu *request* oleh *client*.
b. *handler*, folder ini berisi *file* yang berhubungan dengan fungsi atau *method* yang berkaitan dengan pendefinisian alamat *endpoint*.
c. *helper*, folder ini berisi *file* yang berhubungan dengan fungsi atau *method* bantuan yang dibutuhkan pada modul tersebut.
d. *model*, folder ini berisi *file* yang berhubungan dengan pendeklarasian struktur data suatu *response* atau *request*. Pada folder *model*, terdapat 2 folder yaitu *domain* merupakan folder yang berisi pendeklarasian struktur data suatu *response* atau *request* yang dibutuhkan pada Web *Service* yang dibuat dan *web* merupakan folder yang berisi pendeklarasian struktur data suatu *response* atau *request* yang dibutuhkan oleh *client*.
e. *repository*, folder ini berisi *file* yang berhubungan dengan fungsi atau method yang berkaitan dengan *query logic* atau *call http request*.
f. *service*, folder ini berisi *file* yang berhubungan dengan fungsi atau *method* yang berkaitan dengan *business logic* atau semua operasional sintaks yang dibuat oleh penulis berada disini.

Dalam pengerjaannya, penulis memulai dari pembuatan *model* terlebih dahulu untuk mendeklarasikan struktur data suatu response atau request pada *domain* dan *web*. Kemudian, penulis membuat fungsi atau *method* yang akan digunakan pada modul tersebut pada folder *helper* agar meminimalisir duplikasi sintaks. Setelah pembuatan *helper*, penulis membuat *repository* sebagai *query logic*. Setelah itu, penulis membuat fungsi atau *method* pada *service* sebagai *business logic*. Setelah *service* sudah dibuat, selanjutnya membuat fungsi atau *method*

pada *controller* sebagai penanganan suatu *respon*se atau *request* dari *client*. Selanjutnya pendefinisian alamat *endpoint* pada folder *handler*. Setelah semua API yang dibutuhkan telah dibuat, selanjutnya mendefinisikan *baseurl* modul tersebut pada bagian *routes*.

Modul Berita/Pengumuman

Dilihat dari arsitektur sistem pada Gambar 3.2, *client* membutuhkan 8 *request* yang diinginkan pada modul Berita/Pengumuman. Dilihat dari *request* yang diinginkan, *method* yang akan digunakan pada protokol HTTP dalam arsitektur REST yaitu *method* GET. Pada Tabel 3.2 terdapat Daftar *Endpoint* yang akan dibuat.

Tabel 3.3 Daftar *Endpoint*

Endpoint	Method	Kegunaan
/berita	GET	Untuk membaca seluruh data berita
/berita/:id	GET	Untuk membaca data berita berdasarkan id
/berita/slug/:slug	GET	Untuk membaca data berita berdasarkan slug
/berita/category	GET	Untuk membaca data semua kategori berita
/pengumuman	GET	Untuk membaca seluruh data pengumuman
/pengumuman/:id	GET	Untuk membaca data pengumuman berdasarkan id
/pengumuman/slug/:slug	GET	Untuk membaca data pengumuman berdasarkan slug
/pengumuman/category	GET	Untuk membaca data semua kategori pengumuman

Setelah itu, dibutuhkan pendeklarasian struktur data *model* pada *domain* dan *web* untuk kebutuhan tersebut. Selanjutnya membuat 1 *file* pada *model domain* yaitu *content_sql.go* pada Gambar 3.9 dan 2 file pada *model web* yaitu *web_response.go* pada Gambar 3.10 dan *content_response.go* pada Gambar 3.11.

```
package domain
import "superapp_backend/modules/berita-pengumuman/model/helper"

type (
   ContentSQL struct {
      Id                    int64             `json:"id"`
      SourceId              int64             `json:"source_id"`
      Title                 helper.NullString `json:"title"`
      Slug                  helper.NullString `json:"slug"`
      Link                  helper.NullString `json:"link"`
      Description           helper.NullString `json:"description"`
      Content               helper.NullString `json:"content"`
      Publisher             helper.NullString `json:"publisher"`
      PublisherDate         string            `json:"publisher_date"`
      PublisherStatus       bool              `json:"publisher_status"`
      Creator               helper.NullString `json:"creator"`
      ContentCategory       helper.NullString `json:"content_category"`
```

```
    ThumbnailImage          helper.NullString `json:"thumbnail_image"`
    VersionHistory          helper.NullInt64  `json:"version_history"`
    InsertedDate            string            `json:"inserted_date"`
  }
  ContentCategorySQL struct {
    CategoryId   int64             `json:"category_id"`
    ContentId    int64             `json:"content_id"`
    CategoryName helper.NullString `json:"category_name"`
  }
  CategorySQL struct {
    CategoryId   int64             `json:"category_id"`
    CategoryName helper.NullString `json:"category_name"`
  }
)
```

Gambar 3.9 *Model Domain content_sql.go*

Pada *Model Domain content_sql.go* terdapat *struct* atau objek yang akan menampung data dari *database*. Terdapat 3 *struct* yang didefinisikan diantaranya sebagai berikut:

a. *ContentSQL*, sebagai *struct* yang akan menampung data dari tabel *content.*
b. *ContentCategorySQL*, sebagai *struct* yang akan menampung data dari tabel *content_category.*
c. *CategorySQL*, sebagai *struct* yang akan menampung data dari tabel *content_master_category.*

```
package web

type WebResponseListContent struct {
  CurrentPage      int         `json:"current_page"`
  Data             interface{} `json:"data"`
  LastPage         int         `json:"last_page"`
  TotalItem        int         `json:"total_item"`
  TotalItemPerPage int         `json:"total_item_per_page"`
}
```

Gambar 3.10 *Model Web web_response.go*

Pada *Model Web web_response.go* terdapat *struct* atau objek yang akan menampung data response API pada modul Berita/Pengumuman.

```
package web

import "superapp_backend/modules/berita-pengumuman/model/helper"

type ListContentJSON struct {
  Id              int64             `json:"id"`
  Slug            helper.NullString `json:"slug"`
  Title           helper.NullString `json:"title"`
  Link            helper.NullString `json:"link"`
  Description     helper.NullString `json:"description"`
  Publisher       helper.NullString `json:"publisher"`
  PublisherDate   string            `json:"publisher_date"`
  PublisherStatus helper.NullString `json:"publisher_status"`
  Creator         helper.NullString `json:"creator"`
  ThumbnailImage  helper.NullString `json:"thumbnail_image"`
```

```
   InsertedDate    string            `json:"inserted_date"`
   Categories      []CategoryJSON    `json:"categories"`
}

type ContentJSON struct {
   Id              int64             `json:"id"`
   Slug            helper.NullString `json:"slug"`
   Title           helper.NullString `json:"title"`
   Link            helper.NullString `json:"link"`
   Description     helper.NullString `json:"description"`
   Content         helper.NullString `json:"content"`
   Publisher       helper.NullString `json:"publisher"`
   PublisherDate   string            `json:"publisher_date"`
   PublisherStatus helper.NullString `json:"publisher_status"`
   Creator         helper.NullString `json:"creator"`
   ThumbnailImage  helper.NullString `json:"thumbnail_image"`
   InsertedDate    string            `json:"inserted_date"`
   Categories      []CategoryJSON    `json:"categories"`
}

type CategoryJSON struct {
   CategoryId   int64             `json:"category_id"`
   CategoryName helper.NullString `json:"category_name"`
}
```

Gambar 3.11 *Model Web content_response.go*

Pada *Model Web content_response.go* terdapat *struct* atau objek yang akan menampung data yang akan diberikan ke *client*. Terdapat 3 *struct* yang didefinisikan diantaranya sebagai berikut:

a. *ListContentJSON*, sebagai *struct* yang akan menampung *list* data dari *ContentSQL*.
b. *ContentJSON*, sebagai *struct* yang akan menampung data dari *ContentSQL*.
c. *CategoryJSON*, sebagai *struct* yang akan menampung data dari *CategoryContent*.

Setelah struktur data yang dibutuhkan telah didefinisikan pada *model*, selanjutnya membuat fungsi atau *method* bantuan pada *helper*. Terdapat 2 *file* pada *helper* yaitu *db.go* yang berisi fungsi atau *method* yang menjembatani interaksi ke *database* dapat dilihat pada Gambar 3.12 dan *model_db.go* yang berisi fungsi atau *method* konversi struktur data dapat dilihat pada Gambar 3.13.

```
package helper

import (
   "context"
   "database/sql"
)

func ToQueryContext(ctx context.Context, db *sql.DB, sqlQuery string, args
[]interface{}) (*sql.Rows, error) {
   stmt, err := db.PrepareContext(ctx, sqlQuery)
   if err != nil {
```

```
      return nil, err
   }
   defer stmt.Close()

   rows, err := stmt.QueryContext(ctx, args...)
   if err != nil {
      return nil, err
   }

   return rows, nil
}

func ToQueryRowContext(ctx context.Context, db *sql.DB, sqlQuery string,
args []interface{}) *sql.Row {
   stmt, err := db.PrepareContext(ctx, sqlQuery)
   if err != nil {
      return nil
   }
   defer stmt.Close()

   return stmt.QueryRowContext(ctx, args...)
}
```

Gambar 3.12 *Helper db.go*

Pada *Helper db.go* terdapat *function* digunakan untuk membaca data dari *database*. Terdapat 2 *function* yang dibuat diantaranya sebagai berikut:

a. *ToQueryContext*, sebagai fungsi yang membaca lebih dari satu baris data.

b. *ToQueryRowContext*, sebagai fungsi yang membaca hanya satu baris data.

```
package helper

import (
   "database/sql"
   "html"
   "math"
   "strconv"
   "superapp_backend/modules/berita-pengumuman/model/domain"
   "superapp_backend/modules/berita-pengumuman/model/helper"
   "superapp_backend/modules/berita-pengumuman/model/web"
)

func ToParseRowListContent(rows *sql.Rows, contents []domain.ContentSQL,
counter int, argId string, argsIds []interface{}) ([]domain.ContentSQL,
string, []interface{}, error) {
   for rows.Next() {
      var contentDB domain.ContentSQL
      err := rows.Scan(&contentDB.Id, &contentDB.Slug, &contentDB.Title,
&contentDB.Link, &contentDB.Description, &contentDB.Publisher,
&contentDB.PublisherDate,
         &contentDB.PublisherStatus, &contentDB.Creator,
&contentDB.ThumbnailImage, &contentDB.InsertedDate)
      if err != nil {
         return contents, "", argsIds, err
      }
      contentDB.Title =
helper.NullString(html.UnescapeString(string(contentDB.Title)))
      contents = append(contents, contentDB)
```

```
      counter += 1
      argId += "$" + strconv.Itoa(counter) + ","
      argsIds = append(argsIds, contentDB.Id)
   }
   return contents, argId, argsIds, nil
}

func ToParseRowsCategory(rows *sql.Rows, categories
[]domain.ContentCategorySQL) ([]domain.ContentCategorySQL, error) {
   for rows.Next() {
      var category domain.ContentCategorySQL
      err := rows.Scan(&category.Id, &category.CategoryId,
&category.CategoryName)
      if err != nil {
         return categories, err
      }
      categories = append(categories, category)
   }
   return categories, nil
}

func ToListContentResponses(contents []domain.ContentSQL, categories
[]domain.ContentCategorySQL) []web.ListContentJSON {
   var listContentResponses []web.ListContentJSON

   for _, content := range contents {
      listContentResponse := web.ListContentJSON{
         Id:              content.Id,
         Slug:            content.Slug,
         Title:           content.Title,
         Link:            content.Link,
         Description:     content.Description,
         Publisher:       content.Publisher,
         PublisherDate:   content.PublisherDate,
         PublisherStatus:
helper.NullString(strconv.FormatBool(content.PublisherStatus)),
         Creator:         content.Creator,
         ThumbnailImage:  content.ThumbnailImage,
         InsertedDate:    content.InsertedDate,
      }
      for _, category := range categories {
         if category.ContentId == content.Id {
            temp := web.CategoryJSON{
               CategoryId:   category.CategoryId,
               CategoryName: category.CategoryName,
            }
            listContentResponse.Categories =
append(listContentResponse.Categories, temp)
         }
      }
      listContentResponses = append(listContentResponses,
listContentResponse)
   }

   return listContentResponses
}

func ToDetailContentResponses(content domain.ContentSQL, categories
[]domain.ContentCategorySQL) web.ContentJSON {
   detailContentResponse := web.ContentJSON{
```

```
      Id:              content.Id,
      Slug:            content.Slug,
      Title:           content.Title,
      Link:            content.Link,
      Description:     content.Description,
      Content:         content.Content,
      Publisher:       content.Publisher,
      PublisherDate:   content.PublisherDate,
      PublisherStatus:
helper.NullString(strconv.FormatBool(content.PublisherStatus)),
      Creator:         content.Creator,
      ThumbnailImage:  content.ThumbnailImage,
      InsertedDate:    content.InsertedDate,
   }

   for _, category := range categories {
      temp := web.CategoryJSON{
         CategoryId:   category.CategoryId,
         CategoryName: category.CategoryName,
      }
      detailContentResponse.Categories =
append(detailContentResponse.Categories, temp)
   }

   return detailContentResponse
}

func ToWebResponseListContent(page, count, limit int, data
[]web.ListContentJSON) web.WebResponseListContent {
   return web.WebResponseListContent{
      CurrentPage:     page,
      Data:            data,
      LastPage:        int(math.Ceil(float64(count) / float64(limit))),
      TotalItem:       count,
      TotalItemPerPage: len(data),
   }
}

func ToCategoryResponses(categories []domain.CategorySQL)
[]web.CategoryJSON {
    var categoryResponses []web.CategoryJSON

    for _, category := range categories {
        categoryResponse := web.CategoryJSON{
            CategoryId:   category.CategoryId,
            CategoryName: category.CategoryName,
        }
        categoryResponses = append(categoryResponses, categoryResponse)
    }

    return categoryResponses
}
```

Gambar 3.13 *Helper model_db.go*

Pada *Helper model_db.go* terdapat *function* digunakan untuk konversi data dari *sctruct model domain* ke *struct model web*. Terdapat 6 *function* yang dibuat diantaranya sebagai berikut:

a. *ToParseRowListContent*, melakukan data *binding* dari database ke dalam *struct list ContentSQL*.
b. *ToParseRowsCategory*, melakukan data *binding* dari database ke dalam *struct list ContentCategorySQL*.
c. *ToListContentResponses*, melakukan konversi data dari *struct list ContentSQL* ke dalam *struct list ListContentJSON*.
d. *ToDetailContentResponses*, melakukan konversi data dari *struct ContentSQL* ke dalam *struct ContentJSON*.
e. *ToWebResponseListContent*, melakukan data *binding* terhadap kebutuhan data dalam membentuk *response* dari API pada modul Berita/Pengumuman.
f. *ToCategoryResponses*, melakukan konversi data dari *struct list CategorySQL* ke dalam *struct list CategoryJSON*.

Setelah fungsi atau *method helper* yang dibutuhkan telah dibuat, selanjutnya pembuatan fungsi atau *method* pada *repository*. Pada *repository* terdapat 2 file yaitu *content_repository.go* yang berisi *interface* dari fungsi atau *method* yang akan dibuat dapat dilihat pada Gambar 3.14 dan *content_repository_impl.go* yang berisi fungsi atau *method* yang akan dibuat dapat dilihat pada Gambar 3.15.

```
package repository

import (
    "context"
    "superapp_backend/modules/berita-pengumuman/model/domain"
)

type ContentRepository interface {
    // from db
    ListContentByCategoryTitle(ctx context.Context, modul, category, title
string, page, limit int) (int, []domain.ContentSQL, []interface{}, string,
error)
    ListContentByCategory(ctx context.Context, modul, category string,
page, limit int) (int, []domain.ContentSQL, []interface{}, string, error)
    ListContentByTitle(ctx context.Context, modul, title string, page,
limit int) (int, []domain.ContentSQL, []interface{}, string, error)
    ListContentByOrder(ctx context.Context, modul string, page, limit int)
(int, []domain.ContentSQL, []interface{}, string, error)
    DetailContentById(ctx context.Context, modul, contentId string)
(domain.ContentSQL, error)
    DetailContentBySlug(ctx context.Context, modul, slug string)
(domain.ContentSQL, error)
    CategoryContent(ctx context.Context, modul string)
([]domain.CategorySQL, error)

    // relation
```

```
    CategoryListContent(ctx context.Context, argsIds []interface{}, argId
string) ([]domain.ContentCategorySQL, error)
    CategoryDetailContent(ctx context.Context, contents domain.ContentSQL)
([]domain.ContentCategorySQL, error)
}
```

Gambar 3.14 *Repository content_repository.go*

Pada *Repository content_repository.go* terdapat sebuah *interface* yang didefinisikan yaitu *ContentRepository*. *Interface* tersebut digunakan untuk mendefinisikan *method* yang dibutuhkan untuk berinteraksi dengan data pada database.

```
package repository

import (
    "context"
    "database/sql"
    "errors"
    "strconv"
    "superapp_backend/modules/berita-pengumuman/helper"
    "superapp_backend/modules/berita-pengumuman/model/domain"
)

type ContentRepositoryImpl struct {
    DB *sql.DB
}

func NewContentRepository(DB *sql.DB) ContentRepository {
    return &ContentRepositoryImpl{DB: DB}
}

func (repository *ContentRepositoryImpl) ListContentByCategoryTitle(ctx
context.Context, modul, category, title string, page, limit int) (int,
[]domain.ContentSQL, []interface{}, string, error) {
    var (
        count                  = 0
        counter                = 0
        contents                  []domain.ContentSQL
        sqlQuery, sqlWithLimit string
        args, argsIds          []interface{}
        argId                  = ""
        err                    error
    )

    sqlQuery = ` SELECT
                    content.news_id, content.slug, content.title,
                    content.link, content.description,
                    content.publisher, content.publisher_date,
                    content.publisher_status, content.creator,
                    content.thumbnail_image, content.inserted_date
                FROM
                    content
                INNER JOIN content_category
                    ON content_category.content_id = content.id
                INNER JOIN content_master_category
                    ON content_category.category_id =
content_master_category.category_id
                WHERE
```

```
                    lower(content.content_category) = lower($1)
                AND
                    content_master_category.category_id = $2
                AND
                    content.title ILIKE '%' || $3 || '%'
                GROUP
                    BY content.id
                ORDER BY
                    content.publisher_date DESC `
    sqlWithLimit = sqlQuery + " LIMIT $4 OFFSET $5"

    args = []interface{}{modul, category, title}
    rowsCount, err := helper.ToQueryContext(ctx, repository.DB, sqlQuery,
args)
    if err != nil {
        return 0, nil, nil, "", err
    }
    defer rowsCount.Close()

    for rowsCount.Next() {
        count += 1
    }

    args = []interface{}{modul, category, title, limit, strconv.Itoa((page
- 1) * limit)}
    rows, err := helper.ToQueryContext(ctx, repository.DB, sqlWithLimit,
args)
    if err != nil {
        return 0, nil, nil, "", err
    }
    defer rows.Close()

    contents, argId, argsIds, err = helper.ToParseRowListContent(rows,
contents, counter, argId, argsIds)
    if err != nil {
        return 0, nil, nil, "", err
    }
    //helper.PanicIfArgsIds(argsIds)
    if len(argsIds) == 0 {
        return 0, nil, nil, "", err
    }

    return count, contents, argsIds, argId, nil
}

func (repository *ContentRepositoryImpl) ListContentByCategory(ctx
context.Context, modul, category string, page, limit int) (int,
[]domain.ContentSQL, []interface{}, string, error) {
    var (
        count                   = 0
        counter                 = 0
        contents                    []domain.ContentSQL
        sqlQuery, sqlWithLimit string
        args, argsIds           []interface{}
        argId                   = ""
        err                     error
    )

    sqlQuery = ` SELECT
                    content.id, content.slug, content.title,
```

```
                    content.link, content.description,
                    content.publisher, content.publisher_date,
                    content.publisher_status, content.creator,
                    content.thumbnail_image, content.inserted_date
                FROM
                    content
                INNER JOIN content_category
                    ON content_category.content_id = content.id
                INNER JOIN content_master_category
                    ON content_category.category_id =
content_master_category.category_id
                WHERE
                    lower(content.content_category) = lower($1)
                AND
                    content_master_category.category_id = $2
                GROUP
                    BY content.id
                ORDER BY
                    content.publisher_date DESC `
    sqlWithLimit = sqlQuery + " LIMIT $3 OFFSET $4"

    args = []interface{}{modul, category}
    rowsCount, err := helper.ToQueryContext(ctx, repository.DB, sqlQuery,
args)
    if err != nil {
        return 0, nil, nil, "", err
    }
    defer rowsCount.Close()

    for rowsCount.Next() {
        count += 1
    }

    args = []interface{}{modul, category, limit, strconv.Itoa((page - 1) *
limit)}
    rows, err := helper.ToQueryContext(ctx, repository.DB, sqlWithLimit,
args)
    if err != nil {
        return 0, nil, nil, "", err
    }
    defer rows.Close()

    contents, argId, argsIds, err = helper.ToParseRowListContent(rows,
contents, counter, argId, argsIds)
    if err != nil {
        return 0, nil, nil, "", err
    }
    if len(argsIds) == 0 {
        return 0, nil, nil, "", err
    }

    return count, contents, argsIds, argId, nil
}

func (repository *ContentRepositoryImpl) ListContentByTitle(ctx
context.Context, modul, title string, page, limit int) (int,
[]domain.ContentSQL, []interface{}, string, error) {
    var (
        count                   = 0
        counter                 = 0
```

```
        contents                    []domain.ContentSQL
        sqlQuery, sqlWithLimit string
        args, argsIds           []interface{}
        argId                   = ""
        err                     error
    )

    sqlQuery = ` SELECT
                    content.id, content.slug, content.title,
                    content.link, content.description,
                    content.publisher, content.publisher_date,
                    content.publisher_status, content.creator,
                    content.thumbnail_image, content.inserted_date
                FROM
                    content
                INNER JOIN content_category
                    ON content_category.content_id = content.id
                INNER JOIN content_master_category
                    ON content_category.category_id =
content_master_category.category_id
                WHERE
                    lower(content.content_category) = lower($1)
                AND
                    content.title ILIKE '%' || $2 || '%'
                GROUP
                    BY content.id
                ORDER BY
                    content.publisher_date DESC `
    sqlWithLimit = sqlQuery + " LIMIT $3 OFFSET $4"

    args = []interface{}{modul, title}
    rowsCount, err := helper.ToQueryContext(ctx, repository.DB, sqlQuery,
args)
    if err != nil {
        return 0, nil, nil, "", err
    }
    defer rowsCount.Close()

    for rowsCount.Next() {
        count += 1
    }

    args = []interface{}{modul, title, limit, strconv.Itoa((page - 1) *
limit)}
    rows, err := helper.ToQueryContext(ctx, repository.DB, sqlWithLimit,
args)
    if err != nil {
        return 0, nil, nil, "", err
    }
    defer rows.Close()

    contents, argId, argsIds, err = helper.ToParseRowListContent(rows,
contents, counter, argId, argsIds)
    if err != nil {
        return 0, nil, nil, "", err
    }

    if len(argsIds) == 0 {
        return 0, nil, nil, "", err
    }
```

```
    return count, contents, argsIds, argId, nil
}

func (repository *ContentRepositoryImpl) ListContentByOrder(ctx
context.Context, modul string, page, limit int) (int, []domain.ContentSQL,
[]interface{}, string, error) {
    var (
        count                  = 0
        counter                = 0
        contents                   []domain.ContentSQL
        sqlQuery, sqlWithLimit string
        args, argsIds          []interface{}
        argId                  = ""
        err                    error
    )

    sqlQuery = ` SELECT
                    content.id, content.slug, content.title,
                    content.link, content.description,
                    content.publisher, content.publisher_date,
                    content.publisher_status, content.creator,
                    content.thumbnail_image, content.inserted_date
                FROM
                    content
                INNER JOIN content_category
                    ON content_category.content_id = content.id
                INNER JOIN content_master_category
                    ON content_category.category_id =
content_master_category.category_id
                WHERE
                    lower(content.content_category) = lower($1)
                GROUP BY
                    content.id
                ORDER BY
                    content.publisher_date DESC `
    sqlWithLimit = sqlQuery + " LIMIT $2 OFFSET $3"

    args = []interface{}{modul}
    rowsCount, err := helper.ToQueryContext(ctx, repository.DB, sqlQuery,
args)
    if err != nil {
        return 0, nil, nil, "", err
    }
    defer rowsCount.Close()

    for rowsCount.Next() {
        count += 1
    }

    args = []interface{}{modul, limit, strconv.Itoa((page - 1) * limit)}
    rows, err := helper.ToQueryContext(ctx, repository.DB, sqlWithLimit,
args)
    if err != nil {
        return 0, nil, nil, "", err
    }
    defer rows.Close()

    contents, argId, argsIds, err = helper.ToParseRowListContent(rows,
contents, counter, argId, argsIds)
```

```
    if err != nil {
        return 0, nil, nil, "", err
    }

    if len(argsIds) == 0 {
        return 0, nil, nil, "", err
    }

    return count, contents, argsIds, argId, nil
}

func (repository *ContentRepositoryImpl) DetailContentById(ctx
context.Context, modul, contentId string) (domain.ContentSQL, error) {
    var (
        content     domain.ContentSQL
        sqlQuery string
        args     []interface{}
    )

    sqlQuery = ` SELECT
                content.id, content.slug, content.title,
                content.link, content.description, content.content,
                content.publisher, content.publisher_date,
                content.publisher_status, content.creator,
                content.thumbnail_image, content.inserted_date
            FROM
                content
            WHERE
                lower(content.content_category) = $1
            AND
                content.id = $2
            LIMIT 1 `

    args = []interface{}{modul, contentId}
    rows, err := helper.ToQueryContext(ctx, repository.DB, sqlQuery, args)
    if err != nil {
        return content, err
    }
    defer rows.Close()

    if rows.Next() {
        err := rows.Scan(
            &content.Id, &content.Slug, &content.Title,
            &content.Link, &content.Description, &content.Content,
            &content.Publisher, &content.PublisherDate,
            &content.PublisherStatus, &content.Creator,
            &content.ThumbnailImage, &content.InsertedDate,
        )
        if err != nil {
            return content, err
        }
    } else {
        return content, errors.New("Berita tidak ditemukan")
    }

    return content, nil
}

func (repository *ContentRepositoryImpl) DetailContentBySlug(ctx
context.Context, modul, slug string) (domain.ContentSQL, error) {
```

```
    var (
        content     domain.ContentSQL
        sqlQuery string
        args     []interface{}
    )

    sqlQuery = ` SELECT
                content.id, content.slug, content.title,
                content.link, content.description, content.content,
                content.publisher, content.publisher_date,
                content.publisher_status, content.creator,
                content.thumbnail_image, content.inserted_date
            FROM
                content
            WHERE
                lower(content.content_category) = $1
            AND
                content.slug = $2
            LIMIT 1 `

    args = []interface{}{modul, slug}
    rows, err := helper.ToQueryContext(ctx, repository.DB, sqlQuery, args)
    if err != nil {
        return content, err
    }
    defer rows.Close()

    if rows.Next() {
        err := rows.Scan(
            &content.Id, &content.Slug, &content.Title,
            &content.Link, &content.Description, &content.Content,
            &content.Publisher, &content.PublisherDate,
            &content.PublisherStatus, &content.Creator,
            &content.ThumbnailImage, &content.InsertedDate,
        )
        if err != nil {
            return content, err
        }
    } else {
        return content, errors.New("Berita tidak ditemukan")
    }

    return content, nil
}

func (repository *ContentRepositoryImpl) CategoryContent(ctx
context.Context, modul string) ([]domain.CategorySQL, error) {
    var (
        categories []domain.CategorySQL
        sqlQuery   string
        args       []interface{}
    )

    sqlQuery = ` SELECT
                     content_master_category.category_id,
content_master_category.category_name
                FROM
                    content
                INNER JOIN content_category
                    ON content_category.content_id = content.id
```

```
                INNER JOIN content_master_category
                    ON content_category.category_id =
content_master_category.category_id
                WHERE
                    lower(content.content_category) = lower($1)
                GROUP
                    by  content_master_category.category_id `

    args = []interface{}{modul}
    rows, err := helper.ToQueryContext(ctx, repository.DB, sqlQuery, args)
    if err != nil {
        return categories, err
    }
    defer rows.Close()

    for rows.Next() {
        var category domain.CategorySQL
        err := rows.Scan(
            &category.CategoryId, &category.CategoryName,
        )
        if err != nil {
            return categories, err
        }
        categories = append(categories, category)
    }

    return categories, nil
}

func (repository *ContentRepositoryImpl) CategoryListContent(ctx
context.Context, argsIds []interface{}, argId string)
([]domain.ContentCategorySQL, error) {
    var (
        categories []domain.ContentCategorySQL
        sqlQuery   string
        err        error
    )

    if len(argsIds) <= 0 {
        return categories, err
    }

    sqlQuery = ` SELECT
                    content_category.content_id,
content_master_category.category_id, content_master_category.category_name
                FROM
                    content_category
                JOIN
                    content_master_category
                ON
                    content_master_category.category_id =
content_category.category_id
                WHERE
                    content_category.news_id IN (` + argId[:len(argId)-1] +
`) `

    rows, err := helper.ToQueryContext(ctx, repository.DB, sqlQuery,
argsIds)
    if err != nil {
        return nil, err
```

```
    }
    defer rows.Close()

    for rows.Next() {
        var category domain.ContentCategorySQL
        err := rows.Scan(&category.ContentId, &category.CategoryId,
&category.CategoryName)
        if err != nil {
            return nil, err
        }
        categories = append(categories, category)
    }

    return categories, nil
}

func (repository *ContentRepositoryImpl) CategoryDetailContent(ctx
context.Context, content domain.ContentSQL) ([]domain.ContentCategorySQL,
error) {
    var (
        categories []domain.ContentCategorySQL
        sqlQuery   string
        args       []interface{}
        err        error
    )

    sqlQuery = ` SELECT
                    content_category.content_id,
content_master_category.category_id, content_master_category.category_name
                FROM
                    content_category
                JOIN
                    content_master_category
                ON
                    content_master_category.category_id =
content_category.category_id
                WHERE
                    content_category.content_id = $1 `

    args = []interface{}{content.Id}
    rows, err := helper.ToQueryContext(ctx, repository.DB, sqlQuery, args)
    if err != nil {
        return nil, err
    }
    defer rows.Close()

    categories, err = helper.ToParseRowsCategory(rows, categories)
    if err != nil {
        return nil, err
    }

    return categories, nil
}
```

Gambar 3.15 *Repository content_repository_impl.go*

Pada *Repository content_repository_impl.go* terdapat sebuah *struct* yang didefinisikan yaitu *ContentRepositoryImpl*. *Struct* tersebut digunakan untuk menampung atribut yang dibutuhkan pada modul Berita/Pengumuman yaitu koneksi pada *database* yang akan disimpan

pada atribut DB. Terdapat sebuah *method* juga yang dibuat untuk mendefinisikan *constructor* yang berguna untuk memberikan nilai pada objek yang dibuat yaitu *NewContentRepository*. Kemudian terdapat 9 *method* yang dibuat dan terdefinisi pada *interface* yaitu sebagai berikut:

a. *ListContentByCategoryTitle*, untuk mendapatkan *list* data dari *database* berdasarkan judul dan kategori pada tabel *content*.
b. *ListContentByCategory*, untuk mendapatkan *list* data dari *database* berdasarkan kategori pada tabel *content*.
c. *ListContentByTitle*, untuk mendapatkan *list* data dari *database* berdasarkan judul pada tabel *content*.
d. *ListContentByOrder*, untuk mendapatkan *list* data dari *database* berdasarkan konten terbaru pada tabel *content*.
e. *DetailContentById*, untuk mendapatkan data dari *database* berdasarkan *id* pada tabel *content*.
f. *DetailContentBySlug*, untuk mendapatkan data dari *database* berdasarkan *slug* pada tabel *content*.
g. *CategoryContent*, untuk mendapatkan *list* data dari *database* pada tabel *content_master_category*.
h. CategoryListContent, untuk mendapatkan *list* data dari *database* pada tabel *content_master_category* terhadap *list* data konten.
i. *CategoryDetailContent*, untuk mendapatkan *list* data dari *database* pada tabel *content_master_category* terhadap data konten.

Setelah fungsi atau *method repository* yang dibutuhkan telah dibuat, selanjutnya pembuatan fungsi atau *method* pada *service*. Pada *service* terdapat 2 file yaitu *content_service.go* yang berisi *interface* dari fungsi atau *method* yang akan dibuat dapat dilihat pada Gambar 3.16 dan *content_service_impl.go* yang berisi fungsi atau *method* yang akan dibuat dapat dilihat pada Gambar 3.17.

```
package service

import (
    "context"
    "superapp_backend/modules/berita-pengumuman/model/domain"
    "superapp_backend/modules/berita-pengumuman/model/web"
)

type NewsService interface {
    // from db
```

```
    GetListContentByCategoryTitle(ctx context.Context, modul, category,
title string, page, limit int) (int, []web.ListContentJSON, error)
    GetListContentByCategory(ctx context.Context, modul, category string,
page, limit int) (int, []web.ListContentJSON, error)
    GetListContentByTitle(ctx context.Context, modul, title string, page,
limit int) (int, []web.ListContentJSON, error)
    GetListContentByOrder(ctx context.Context, modul string, page, limit
int) (int, []web.ListContentJSON, error)
    GetDetailContentById(ctx context.Context, modul, newsId string)
(web.ContentJSON, error)
    GetDetailContentBySlug(ctx context.Context, modul, slug string)
(web.ContentJSON, error)
    GetListCategoryContent(ctx context.Context, modul string)
([]web.CategoryJSON, error)
}
```

Gambar 3.16 *Service content_service.go*

Pada *Service content_service.go* terdapat sebuah *interface* yang didefinisikan yaitu *ContentService*. *Interface* tersebut digunakan untuk mendefinisikan *method* yang dibutuhkan untuk mengolah data menjadi *response* yang akan diberikan.

```
package service

import (
    "context"
    "superapp_backend/modules/berita-pengumuman/helper"
    "superapp_backend/modules/berita-pengumuman/model/web"
    "superapp_backend/modules/berita-pengumuman/repository"
)

type ContentServiceImpl struct {
    ContentRepository repository.ContentRepository
}

func NewContentService(contentRepository repository.ContentRepository)
ContentService {
    return &ContentServiceImpl{ContentRepository: contentRepository}
}

func (service *ContentServiceImpl) GetListContentByCategoryTitle(ctx
context.Context, modul, category, title string, page, limit int) (int,
[]web.ListContentJSON, error) {
    count, contents, argIds, argId, err :=
service.ContentRepository.ListContentByCategoryTitle(ctx, modul, category,
title, page, limit)
    if err != nil {
        return 0, nil, err
    }

    categories, err := service.ContentRepository.CategoryListContent(ctx,
argIds, argId)
    if err != nil {
        return 0, nil, err
    }

    data := helper.ToListContentResponses(contents, categories)
```

```
    return count, data, nil
}

func (service *ContentServiceImpl) GetListContentByCategory(ctx
context.Context, modul, category string, page, limit int) (int,
[]web.ListContentJSON, error) {
    count, contents, argIds, argId, err :=
service.ContentRepository.ListContentByCategory(ctx, modul, category, page,
limit)
    if err != nil {
        return 0, nil, err
    }

    categories, err := service.ContentRepository.CategoryListContent(ctx,
argIds, argId)
    if err != nil {
        return 0, nil, err
    }

    data := helper.ToListContentResponses(contents, categories)

    return count, data, nil
}

func (service *ContentServiceImpl) GetListContentByTitle(ctx
context.Context, modul, title string, page, limit int) (int,
[]web.ListContentJSON, error) {
    count, contents, argIds, argId, err :=
service.ContentRepository.ListContentByTitle(ctx, modul, title, page,
limit)
    if err != nil {
        return 0, nil, err
    }

    categories, err := service.ContentRepository.CategoryListContent(ctx,
argIds, argId)
    if err != nil {
        return 0, nil, err
    }

    data := helper.ToListContentResponses(contents, categories)

    return count, data, nil
}

func (service *ContentServiceImpl) GetListContentByOrder(ctx
context.Context, modul string, page, limit int) (int,
[]web.ListContentJSON, error) {
    count, contents, argIds, argId, err :=
service.ContentRepository.ListContentByOrder(ctx, modul, page, limit)
    if err != nil {
        return 0, nil, err
    }

    categories, err := service.ContentRepository.CategoryListContent(ctx,
argIds, argId)
    if err != nil {
        return 0, nil, err
    }
```

```
    data := helper.ToListContentResponses(contents, categories)

    return count, data, nil
}

func (service *ContentServiceImpl) GetDetailContentById(ctx
context.Context, modul, contentId string) (web.ContentJSON, error) {
    content, err := service.ContentRepository.DetailContentById(ctx, modul,
contentId)
    if err != nil {
        return web.ContentJSON{}, err
    }

    categories, err := service.ContentRepository.CategoryDetailContent(ctx,
content)

    data := helper.ToDetailContentResponses(content, categories)

    return data, nil
}

func (service *ContentServiceImpl) GetDetailContentBySlug(ctx
context.Context, modul, slug string) (web.ContentJSON, error) {
    content, err := service.ContentRepository.DetailContentBySlug(ctx,
modul, slug)
    if err != nil {
        return web.ContentJSON{}, err
    }

    categories, err := service.ContentRepository.CategoryDetailContent(ctx,
content)
    if err != nil {
        return web.ContentJSON{}, err
    }

    data := helper.ToDetailContentResponses(content, categories)

    return data, nil
}

func (service *ContentServiceImpl) GetListCategoryContent(ctx
context.Context, modul string) ([]web.CategoryJSON, error) {
    categories, err := service.ContentRepository.CategoryContent(ctx,
modul)
    if err != nil {
        return nil, err
    }

    categoriesResponse := helper.ToCategoryResponses(categories)

    return categoriesResponse, nil
}
```

Gambar 3.17 *Service content_service_impl.go*

Pada *Service content_service_impl.go* terdapat sebuah *struct* yang didefinisikan yaitu *ContentServiceImpl*. *Struct* tersebut digunakan untuk mengolah data yang dibutuhkan pada modul Berita/Pengumuman yaitu menampung *interface* yang berisi *method* untuk

mendapatkan data pada *database* yang telah dibuat pada *content_repository*.go yang akan disimpan pada atribut *ContentRepository*. Terdapat sebuah *method* juga yang dibuat untuk mendefinisikan *constructor* yang berguna untuk memberikan nilai pada objek yang dibuat yaitu *NewContentService*. Kemudian terdapat 7 *method* yang dibuat dan terdefinisi pada *interface* yaitu sebagai berikut:

a. *GetListContentByCategoryTitle*, melakukan konversi *list* data dari *struct ContentSQl* ke dalam *struct ListContentJSON* dan mengolah data menjadi *response* yang akan diberikan berdasarkan judul dan kategori.
b. *GetListContentByCategory*, melakukan konversi *list* data dari *struct ContentSQl* ke dalam *struct ListContentJSON* dan mengolah data menjadi *response* yang akan diberikan berdasarkan kategori.
c. *GetListContentByTitle*, melakukan konversi *list* data dari *struct ContentSQl* ke dalam *struct ListContentJSON* dan mengolah data menjadi *response* yang akan diberikan berdasarkan judul.
d. *GetListContentByOrder*, melakukan konversi *list* data dari *struct ContentSQl* ke dalam *struct ListContentJSON* dan mengolah data menjadi *response* yang akan diberikan berdasarkan konten terbaru.
e. *GetDetailContentById*, melakukan konversi data dari *struct ContentSQl* ke dalam *struct ContentJSON* dan mengolah data menjadi *response* yang akan diberikan berdasarkan *id*.
f. *GetDetailContentBySlug*, melakukan konversi data dari *struct ContentSQl* ke dalam *struct ContentJSON* dan mengolah data menjadi *response* yang akan diberikan berdasarkan *slug*.
g. *GetListCategoryContent*, melakukan konversi *list* data dari *struct CategorySQl* ke dalam *struct CategoryJSON* dan mengolah data menjadi *response* yang akan diberikan.

Setelah fungsi atau *method service* yang dibutuhkan telah dibuat, selanjutnya pembuatan fungsi atau *method* pada *controller*. Pada *controller* terdapat 2 file yaitu *content_controller.go* yang berisi *interface* dari fungsi atau *method* yang akan dibuat dapat dilihat pada Gambar 3.18 dan *content_controller_impl.go* yang berisi fungsi atau *method* yang akan dibuat dapat dilihat pada Gambar 3.19.

```
package controller

import "github.com/labstack/echo/v4"

type ContentController interface {
   // @Tags News
   GetListBerita(c echo.Context) error
   GetDetailBeritaById(c echo.Context) error
   GetDetailBeritaBySlug(c echo.Context) error
   GetListCategoryBerita(c echo.Context) error

   // @Tags Pengumuman
   GetListPengumuman(c echo.Context) error
   GetDetailPengumumanById(c echo.Context) error
   GetDetailPengumumanBySlug(c echo.Context) error
   GetListCategoryPengumuman(c echo.Context) error
}
```

Gambar 3.18 *Controller content_controller.go*

Pada *Controller content_controller.go* terdapat sebuah *interface* yang didefinisikan yaitu *ContentController*. *Interface* tersebut digunakan untuk mendefinisikan *method* yang dibutuhkan dalam penanganan *request* yang diminta dan *response* yang akan diberikan terhadap *client*.

```
package controller

import (
   "net/http"
   "net/url"
   "strconv"
   "superapp_backend/modules/berita-pengumuman/helper"
   "superapp_backend/modules/berita-pengumuman/model/web"
   "superapp_backend/modules/berita-pengumuman/service"

   "github.com/labstack/echo/v4"
)

type ContentControllerImpl struct {
   ContentService service.ContentService
}

func NewContentController(contentService service.ContentService)
ContentController {
   return &ContentControllerImpl{ContentService: contentService}
}

func (controller *ContentControllerImpl) GetListBerita(c echo.Context)
error {
   var (
      count  int
      result []web.ListContentJSON
      modul  = "berita"
   )

   //result := make(map[string]interface{})
   category := c.QueryParam("category_id")
```

```
   title := c.QueryParam("title")
   page, err := strconv.Atoi(c.QueryParam("page"))
   if err != nil {
      return c.JSON(http.StatusBadRequest, map[string]string{
         "message": "Page harus angka.",
         "error":   err.Error(),
      })
   }
   limitParam := c.QueryParam("limit")
   if limitParam == "" {
      limitParam = "10"
   }
   limit, err := strconv.Atoi(limitParam)
   if err != nil {
      return c.JSON(http.StatusBadRequest, map[string]string{
         "message": "Limit harus angka.",
         "error":   err.Error(),
      })
   }

   if limit <= 0 {
      limit = 10
   }

   if page <= 0 {
      page = 1
   }

   switch {
   case len(category) > 0 && len(title) > 0:
      count, result, err =
controller.ContentService.GetListContentByCategoryTitle(c.Request().Context
(), modul, category, title, page, limit)
   case len(category) > 0:
      count, result, err =
controller.ContentService.GetListContentByCategory(c.Request().Context(),
modul, category, page, limit)
   case len(title) > 0:
      count, result, err =
controller.ContentService.GetListContentByTitle(c.Request().Context(),
modul, title, page, limit)
   default:
      count, result, err =
controller.ContentService.GetListContentByOrder(c.Request().Context(),
modul, page, limit)
   }

   if err != nil {
      return c.JSON(http.StatusInternalServerError, map[string]string{
         "message": "Terjadi Error",
         "error":   err.Error(),
      })
   }

   if count == 0 {
      return c.JSON(http.StatusNotFound, map[string]string{
         "message": "Berita tidak ditemukan",
      })
   }
```

```
   return c.JSON(http.StatusOK, helper.ToWebResponseListContent(page,
count, limit, result))
}

func (controller *ContentControllerImpl) GetDetailBeritaById(c
echo.Context) error {
   var (
      result web.ContentJSON
      modul  = "berita"
   )

   id := c.Param("id")

   result, err :=
controller.ContentService.GetDetailContentById(c.Request().Context(),
modul, id)
   if err != nil {
      return c.JSON(http.StatusInternalServerError, map[string]string{
         "message": "Terjadi Error",
         "error":   err.Error(),
      })
   }

   return c.JSON(http.StatusOK, result)
}

func (controller *ContentControllerImpl) GetDetailBeritaBySlug(c
echo.Context) error {
   var (
      result web.ContentJSON
      modul  = "berita"
   )

   slug := url.QueryEscape(c.Param("slug"))

   result, err :=
controller.ContentService.GetDetailContentBySlug(c.Request().Context(),
modul, slug)
   if err != nil {
      return c.JSON(http.StatusInternalServerError, map[string]string{
         "message": "Terjadi Error",
         "error":   err.Error(),
      })
   }

   return c.JSON(http.StatusOK, result)
}

func (controller *ContentControllerImpl) GetListCategoryBerita(c
echo.Context) error {
   var (
      result []web.CategoryJSON
      modul  = "berita"
   )

   result, err :=
controller.ContentService.GetListCategoryContent(c.Request().Context(),
modul)
   if err != nil {
      return c.JSON(http.StatusInternalServerError, map[string]string{
```

```
            "message": "Terjadi Error",
            "error":   err.Error(),
        })
    }

    if len(result) == 0 {
        return c.JSON(http.StatusNotFound, map[string]string{
            "message": "Kategori tidak ditemukan",
        })
    }

    return c.JSON(http.StatusOK, result)
}

func (controller *ContentControllerImpl) GetListPengumuman(c echo.Context)
error {
    var (
        count  int
        result []web.ListContentJSON
        modul  = "pengumuman"
    )

    //result := make(map[string]interface{})
    category := c.QueryParam("category_id")
    title := c.QueryParam("title")
    page, err := strconv.Atoi(c.QueryParam("page"))
    if err != nil {
        return c.JSON(http.StatusBadRequest, map[string]string{
            "message": "Page harus angka.",
            "error":   err.Error(),
        })
    }
    limitParam := c.QueryParam("limit")
    if limitParam == "" {
        limitParam = "10"
    }
    limit, err := strconv.Atoi(limitParam)
    if err != nil {
        return c.JSON(http.StatusBadRequest, map[string]string{
            "message": "Limit harus angka.",
            "error":   err.Error(),
        })
    }

    if limit <= 0 {
        limit = 10
    }

    if page <= 0 {
        page = 1
    }

    switch {
    case len(category) > 0 && len(title) > 0:
        count, result, err =
controller.ContentService.GetListContentByCategoryTitle(c.Request().Context
(), modul, category, title, page, limit)
    case len(category) > 0:
```

```
      count, result, err =
controller.ContentService.GetListContentByCategory(c.Request().Context(),
modul, category, page, limit)
   case len(title) > 0:
      count, result, err =
controller.ContentService.GetListContentByTitle(c.Request().Context(),
modul, title, page, limit)
   default:
      count, result, err =
controller.ContentService.GetListContentByOrder(c.Request().Context(),
modul, page, limit)
   }

   if err != nil {
      return c.JSON(http.StatusInternalServerError, map[string]string{
         "message": "Terjadi Error",
         "error":   err.Error(),
      })
   }

   if count == 0 {
      return c.JSON(http.StatusNotFound, map[string]string{
         "message": "Pengumuman tidak ditemukan",
      })
   }

   return c.JSON(http.StatusOK, helper.ToWebResponseListContent(page,
count, limit, result))
}

func (controller *ContentControllerImpl) GetDetailPengumumanById(c
echo.Context) error {
   var (
      result web.ContentJSON
      modul  = "pengumuman"
   )

   id := c.Param("id")

   result, err :=
controller.ContentService.GetDetailContentById(c.Request().Context(),
modul, id)
   if err != nil {
      return c.JSON(http.StatusInternalServerError, map[string]string{
         "message": "Terjadi Error",
         "error":   err.Error(),
      })
   }

   return c.JSON(http.StatusOK, result)
}

func (controller *ContentControllerImpl) GetDetailPengumumanBySlug(c
echo.Context) error {
   var (
      result web.ContentJSON
      modul  = "pengumuman"
   )

   slug := url.QueryEscape(c.Param("slug"))
```

```
   result, err :=
controller.ContentService.GetDetailContentBySlug(c.Request().Context(),
modul, slug)
   if err != nil {
      return c.JSON(http.StatusInternalServerError, map[string]string{
         "message": "Terjadi Error",
         "error":   err.Error(),
      })
   }

   return c.JSON(http.StatusOK, result)
}

func (controller *ContentControllerImpl) GetListCategoryPengumuman(c
echo.Context) error {
   var (
      result []web.CategoryJSON
      modul  = "pengumuman"
   )

   result, err :=
controller.ContentService.GetListCategoryContent(c.Request().Context(),
modul)
   if err != nil {
      return c.JSON(http.StatusInternalServerError, map[string]string{
         "message": "Terjadi Error",
         "error":   err.Error(),
      })
   }

   if len(result) == 0 {
      return c.JSON(http.StatusNotFound, map[string]string{
         "message": "Kategori tidak ditemukan",
      })
   }

   return c.JSON(http.StatusOK, result)
}
```

Gambar 3.19 *Controller content_controller_impl.go*

Pada *Controller content_controller_impl.go* terdapat sebuah *struct* yang didefinisikan yaitu *ContentControllerImpl*. *Struct* tersebut digunakan untuk penanganan *request* dan *response* yang dibutuhkan pada modul Berita/Pengumuman yaitu menampung *interface* yang berisi *method* untuk mendapatkan data sudah diolah yang dibuat pada *content_service*.go yang akan disimpan pada atribut *ContentService*. Terdapat sebuah *method* juga yang dibuat untuk mendefinisikan *constructor* yang berguna untuk memberikan nilai pada objek yang dibuat yaitu *NewContentController*. Kemudian terdapat 8 *method* yang dibuat dan terdefinisi pada *interface* yaitu sebagai berikut:

a. *GetListBerita*, menangani *request* dan *response* terhadap *client* terkait untuk mendapatkan *list* data berita dengan *Query Params* yang disediakan yaitu *category_id, title, page,* dan *limit*.

b. *GetDetailBeritaById*, menangani *request* dan *response* terhadap *client* terkait untuk mendapatkan data berita dengan URI *Params* yang disediakan yaitu *id*.
c. *GetDetailBeritaBySlug*, menangani *request* dan *response* terhadap *client* terkait untuk mendapatkan data berita dengan URI *Params* yang disediakan yaitu *slug*.
d. *GetListCategoryBerita*, menangani *request* dan *response* terhadap *client* terkait untuk mendapatkan *list* data kategori berita.
e. *GetListPengumuman*, menangani *request* dan *response* terhadap *client* terkait untuk mendapatkan *list* data pengumuman dengan *Query Params* yang disediakan yaitu *category_id, title, page,* dan *limit*.
f. *GetDetailPengumumanById*, menangani *request* dan *response* terhadap *client* terkait untuk mendapatkan data pengumuman dengan URI *Params* yang disediakan yaitu *id.*
g. *GetDetailPengumumanBySlug*, menangani *request* dan *response* terhadap *client* terkait untuk mendapatkan data pengumuman dengan URI *Params* yang disediakan yaitu *slug.*
h. *GetListCategoryPengumuman*, menangani *request* dan *response* terhadap *client* terkait untuk mendapatkan *list* data kategori pengumuman.

Setelah fungsi atau *method controller* yang dibutuhkan telah dibuat, selanjutnya pembuatan fungsi atau *method* pada *handler*. Pada *handler* berisi inisialisasi *repository, service, dan controller* serta berisi fungsi atau *method* untuk mendeklarasikan *endpoint* yang akan dibuat. Terdapat 2 file pada *handler* yaitu *berita_handler.go* dapat dilihat pada Gambar 3.20 dan *pengumuman_handler.go* dapat dilihat pada Gambar 3.21.

```
package handler

import (
   "database/sql"
   "github.com/labstack/echo/v4"
   "superapp_backend/modules/berita-pengumuman/controller"
   "superapp_backend/modules/berita-pengumuman/repository"
   "superapp_backend/modules/berita-pengumuman/service"
)

func BeritaHandler(g *echo.Group, db *sql.DB) {
   var (
      //repo
      contentRepo = repository.NewContentRepository(db)
      //service
      contentService = service.NewContentService(contentRepo)
      //controller
      contentController = controller.NewContentController(contentService)
   )
```

```
   g.GET("", contentController.GetListBerita)
   g.GET("/:id", contentController.GetDetailBeritaById)
   g.GET("/slug/:slug", contentController.GetDetailBeritaBySlug)
   g.GET("/category", contentController.GetListCategoryBerita)
}
```

Gambar 3.20 *Handler berita_handler.go*

Pada *Handler berita_handler.go* terdapat sebuah *method* yang dibuat yaitu *BeritaHandler*. *Method* tersebut digunakan untuk memanggil semua *constructor* yang dibuat pada modul Berita/Pengumuman serta memberikan *value* yang dibutuhkan pada *constructor* tersebut yaitu pada *NewContentReposistory*, *NewContentService*, dan *NewContentController*. Selain itu, sesuai dengan *method* yang dibuat *BeritaHandler* akan mendefinisikan *endpoint* API untuk kebutuhan data berita. Terdapat 4 *endpoint* yang didefinisikan yaitu sebagai berikut:

a. *“”*, alamat *endpoint* yang digunakan untuk mendapatkan *list* data berita dengan metode protokol HTTP yang digunakan yaitu GET.
b. *“/:id”*, alamat *endpoint* yang digunakan untuk mendapatkan data berita berdasarkan *id* dengan metode protokol HTTP yang digunakan yaitu GET.
c. *“/slug/:slug”*, alamat *endpoint* yang digunakan untuk mendapatkan data berita berdasarkan *slug* dengan metode protokol HTTP yang digunakan yaitu GET.
d. *“/category”*, alamat *endpoint* yang digunakan untuk mendapatkan *list* data kategori berita dengan metode protokol HTTP yang digunakan yaitu GET.

```
package handler

import (
   "database/sql"
   "github.com/labstack/echo/v4"
   "superapp_backend/modules/berita-pengumuman/controller"
   "superapp_backend/modules/berita-pengumuman/repository"
   "superapp_backend/modules/berita-pengumuman/service"
)

func PengumumanHandler(g *echo.Group, db *sql.DB) {
   var (
      //repo
      contentRepo = repository.NewContentRepository(db)
      //service
      contentService = service.NewContentService(contentRepo)
      //controller
      contentController = controller.NewContentController(contentService)
   )

   g.GET("", contentController.GetListPengumuman)
   g.GET("/:id", contentController.GetDetailPengumumanById)
   g.GET("/slug/:slug", contentController.GetDetailPengumumanBySlug)
   g.GET("/category", contentController.GetListCategoryPengumuman)
```

```
}
```

Gambar 3.21 *Handler pengumuman_handler.go*

Pada *Handler pengumuman_handler.go* terdapat sebuah *method* yang dibuat yaitu *PengumumanHandler*. *Method* tersebut digunakan untuk memanggil semua *constructor* yang dibuat pada modul Berita/Pengumuman serta memberikan *value* yang dibutuhkan pada *constructor* tersebut yaitu pada *NewContentReposistory*, *NewContentService*, dan *NewContentController*. Selain itu, sesuai dengan *method* yang dibuat *PengumumanHandler* akan mendefinisikan *endpoint* API untuk kebutuhan data pengumuman. Terdapat 4 *endpoint* yang didefinisikan yaitu sebagai berikut:

e. *“”*, alamat *endpoint* yang digunakan untuk mendapatkan *list* data pengumuman dengan metode protokol HTTP yang digunakan yaitu GET.
f. *“/:id”*, alamat *endpoint* yang digunakan untuk mendapatkan data pengumuman berdasarkan *id* dengan metode protokol HTTP yang digunakan yaitu GET.
g. *“/slug/:slug”*, alamat *endpoint* yang digunakan untuk mendapatkan data pengumuman berdasarkan *slug* dengan metode protokol HTTP yang digunakan yaitu GET.
h. *“/category”*, alamat *endpoint* yang digunakan untuk mendapatkan *list* data kategori pengumuman dengan metode protokol HTTP yang digunakan yaitu GET.

Setelah fungsi atau *method handler* yang dibutuhkan telah dibuat, selanjutnya fungsi atau *method* pada *handler* harus didaftarkan pada file *route.go* yang terdapat pada *routes*. Tujuannya adalah agar menggabungkan *endpoint* menjadi satu *baseurl* dan *endpoint* pada API yang dibuat dapat diakses oleh *client.* Pada *file route.go* sebelumnya sudah terdapat sintaks yang sudah dibuat. Penulis hanya perlu menambahkan baris sintaks terhadap *handler* yang sudah dibuat. Terdapat 2 *handler* yang perlu didaftarkan yaitu *BeritaHandler* dan *PengumumanHandler* dapat dilihat pada Gambar 3.22.

```
package routes

import (
    ...
    "database/sql"
    beritapengumuman "superapp_backend/modules/berita-pengumuman/handler"

    "github.com/labstack/echo/v4"
)

func InitRouter(e *echo.Echo, db *sql.DB) {
    ...

    // baseurl berita untuk BeritaHandler
    beritapengumuman.BeritaHandler(e.Group("/berita"), db)
```

```
    // baseurl pengumuman untuk PengumumanHandler
    beritapengumuman.PengumumanHandler(e.Group("/pengumuman"), db)
}
```

Gambar 3.22 *Routes route.go*

Pada *Routes route.go* terdapat sebuah *method* yaitu *InitRouter*. *Method* tersebut digunakan untuk mendaftarkan *handler* yang telah dibuat dan mendefinisikan alamat *baseurl* suatu *endpoint* yang didefinisikan. Terdapat 2 *baseurl* yang didefinisikan pada modul Berita/Pengumuman yaitu sebagai berikut:

a. *"/berita"*, *baseurl* yang didefinisikan pada *endpoint* yang telah dibuat oleh *BeritaHandler* untuk mendapatkan data berita.
b. *"/pengumuman" baseurl* yang didefinisikan pada *endpoint* yang telah dibuat oleh *PengumumanHandler* untuk mendapatkan data pengumuman.

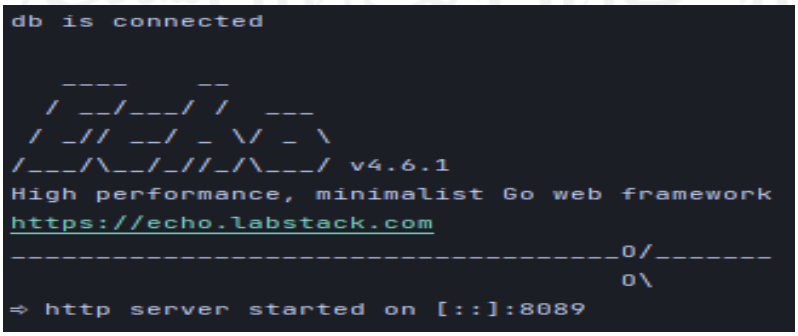
3.2.3 Pengujian API

Tahap terakhir yang perlu dilakukan adalah melakukan pengujian pada API. Tujuannya adalah menentukan kelayakan API yang telah dibuat. Kelayakan pada API dapat dikatakan layak apabila telah memenuhi kebutuhan *response* yang diinginkan dapat dilihat pada dokumentasi API *Contract* yang telah dibuat pada modul tersebut.

Untuk melakukan pengujian, Web *Service* yang telah dibuat perlu dijalankan terlebih dahulu. Cara menjalankannya dilakukan melalui terminal yang dapat dilihat pada Gambar 3.23. Sebelum menjalankannya, harapannya posisi kursor pada terminal sudah masuk pada direktori Web *Service* berada.

```
go run server.go
```

Gambar 3.23 Sintaks Menjalankan Web *Service* pada Terminal



Gambar 3.24 *Output* pada Terminal

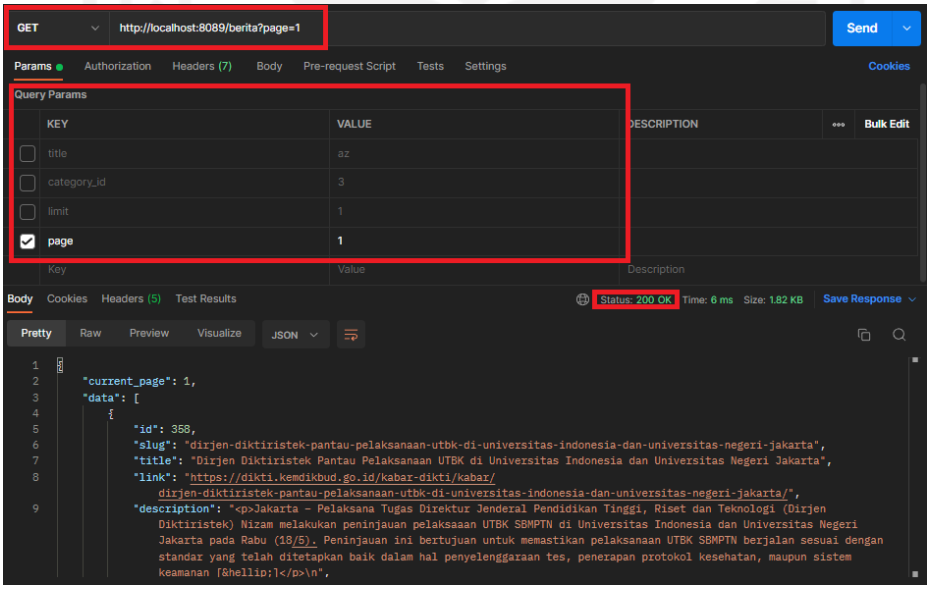
Perintah *go run server.go* adalah suatu perintah di terminal untuk menjalankan fungsi atau *method* utama pada *file .go*. Kemudian terminal akan menampilkan berupa *output* seperti pada Gambar 3.24 ketika berhasil dijalankan.

Setelah Web *Service* berhasil dijalankan, selanjutnya akan dilakukan pengujian API pada modul yang telah dibuat.

Modul Berita/Pengumuman

a. *Response* membaca seluruh data berita

Endpoint /berita digunakan untuk mengambil seluruh data berita yang disediakan oleh server. Pengujian dilakukan dengan cara mengirimkan *request* data ke *endpoint* tersebut dengan metode GET dan memasukkan URL *http://localhost:8089/berita*. Untuk membaca seluruh data berita, tersedia 4 *Query Params* yaitu *title* bertipe *string*, *category_id* bertipe *integer*, *limit* bertipe *integer*, dan *page* bertipe *integer*. *Query Params page* wajib diisi dan selain *page* bersifat opsional. Kemudian, *server* memberikan *response* seperti pada Gambar 3.25 yang menunjukkan *response* berhasil dengan status kode 200.

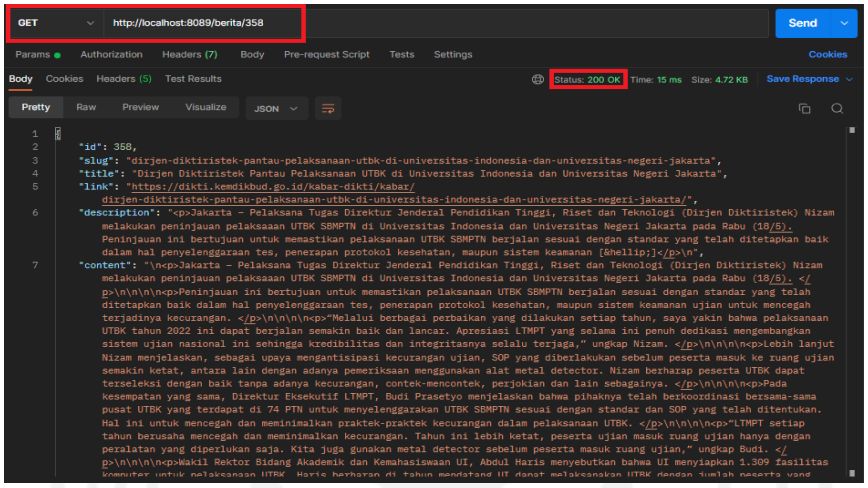


Gambar 3.25 Response Membaca Seluruh Data Berita

b. *Response* membaca data berita berdasarkan *id*

Endpoint /berita/:id digunakan untuk mengambil data berita berdasarkan *id* yang disediakan oleh server. Pengujian dilakukan dengan cara mengirimkan *request* data ke *endpoint* tersebut dengan metode GET dan memasukkan URL

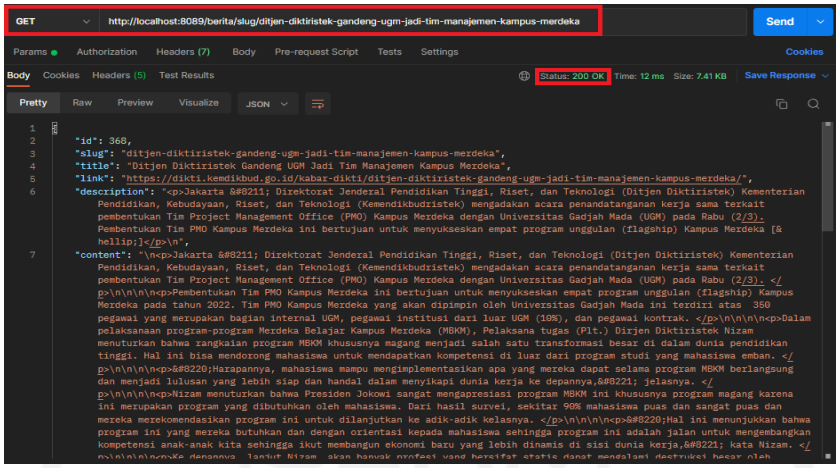
http://localhost:8089/berita/:id. Untuk membaca data berita berdasarkan *id* yang diinginkan, tidak perlu mengisi *Query Params* karena hanya menambahkan URI *Params* berdasarkan *id*. Contoh *id* yang akan digunakan adalah 358. Kemudian, *server* memberikan *response* seperti pada Gambar 3.26 yang menunjukkan *response* berhasil dengan status kode 200.



Gambar 3.26 Response Membaca Data Berita berdasarkan *Id*

c. *Response* membaca data berita berdasarkan *slug*

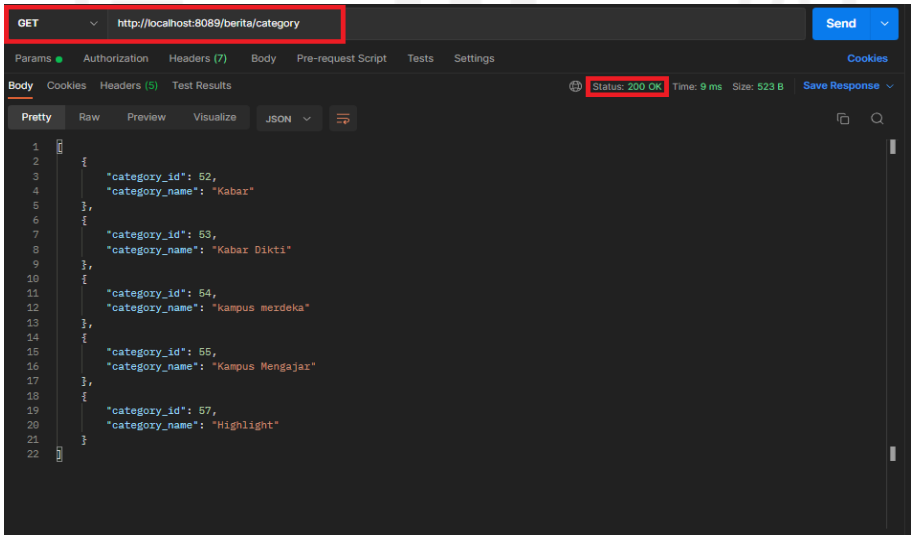
Endpoint /berita/slug/:slug digunakan untuk mengambil data berita berdasarkan *slug* yang disediakan oleh server. Pengujian dilakukan dengan cara mengirimkan *request* data ke *endpoint* tersebut dengan metode GET dan memasukkan URL *http://localhost:8089/berita/slug/:slug*. Untuk membaca data berita berdasarkan *slug* yang diinginkan, tidak perlu mengisi *Query Params* karena hanya menambahkan URI *Params* berdasarkan *slug*. Contoh *slug* yang akan digunakan adalah ditjen-diktiristek-gandeng-ugm-jadi-tim-manajemen-kampus-merdeka. Kemudian, *server* memberikan *response* seperti pada Gambar 3.27 yang menunjukkan *response* berhasil dengan status kode 200.



Gambar 3.27 Response Membaca Data Berita berdasarkan *Slug*

d. *Response* membaca data semua kategori berita

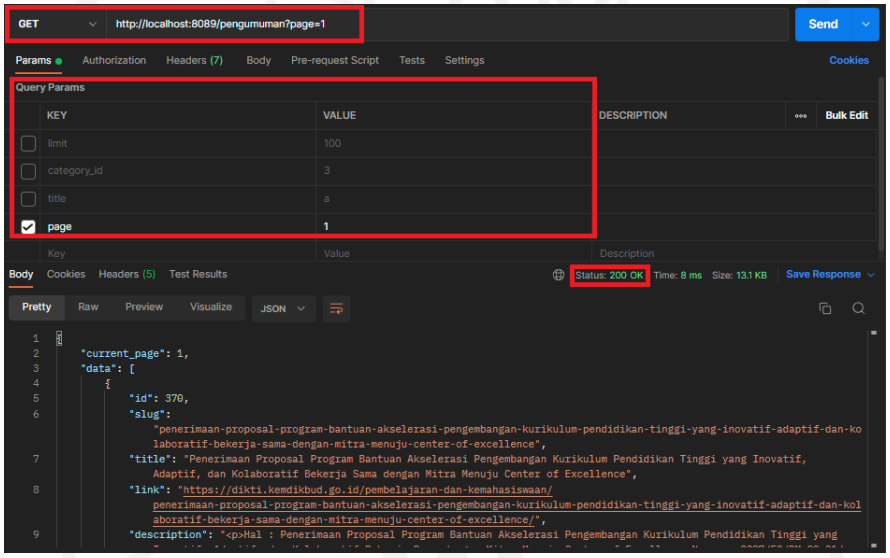
Endpoint /berita/category digunakan untuk mengambil data semua kategori berita yang disediakan oleh server. Pengujian dilakukan dengan cara mengirimkan *request* data ke *endpoint* tersebut dengan metode GET dan memasukkan URL *http://localhost:8089/berita/category*. Untuk membaca data semua kategori berita, tidak perlu mengisi *Query Params* ataupun URI *Params*. Kemudian, *server* memberikan *response* seperti pada Gambar yang menunjukkan *response* berhasil dengan status kode 200.



Gambar 3.28 Response Membaca Data Semua Kategori Berita

e. *Response* membaca seluruh data pengumuman

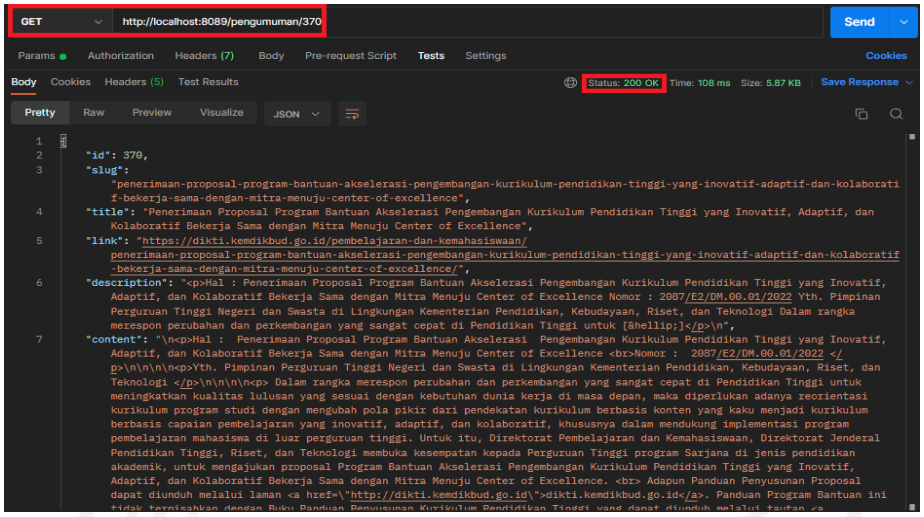
Endpoint /pengumuman digunakan untuk mengambil seluruh data pengumuman yang disediakan oleh server. Pengujian dilakukan dengan cara mengirimkan *request* data ke *endpoint* tersebut dengan metode GET dan memasukkan URL *http://localhost:8089/pengumuman*. Untuk membaca seluruh data pengumuman, tersedia 4 *Query Params* yaitu *title* bertipe *string*, *category_id* bertipe *integer*, *limit* bertipe *integer*, dan *page* bertipe *integer*. *Query Params page* wajib diisi dan selain *page* bersifat opsional. Kemudian, *server* memberikan *response* seperti pada Gambar 3.29 yang menunjukkan *response* berhasil dengan status kode 200.



Gambar 3.29 Response Membaca Seluruh Data Pengumuman

f. *Response* membaca data pengumuman berdasarkan *id*

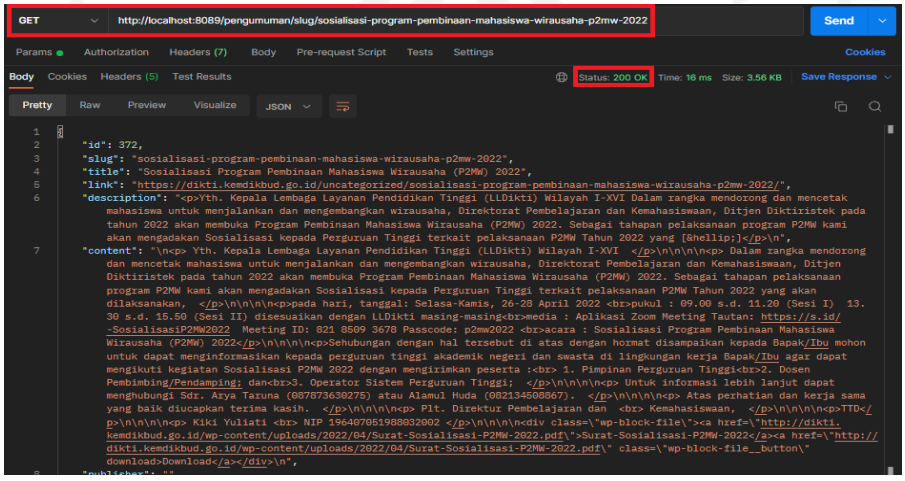
Endpoint /pengumuman/:id digunakan untuk mengambil data pengumuman berdasarkan *id* yang disediakan oleh server. Pengujian dilakukan dengan cara mengirimkan *request* data ke *endpoint* tersebut dengan *method* GET dan memasukkan URL *http://localhost:8089/pengumuman/:id*. Untuk membaca data pengumuman berdasarkan *id* yang diinginkan, tidak perlu mengisi *Query Params* karena hanya menambahkan URI *Params* berdasarkan *id*. Contoh *id* yang akan digunakan adalah 370. Kemudian, *server* memberikan *response* seperti pada Gambar 3.30 yang menunjukkan *response* berhasil dengan status kode 200.



Gambar 3.30 Response Membaca Data Pengumuman berdasarkan *Id*

g. *Response* membaca data pengumuman berdasarkan *slug*

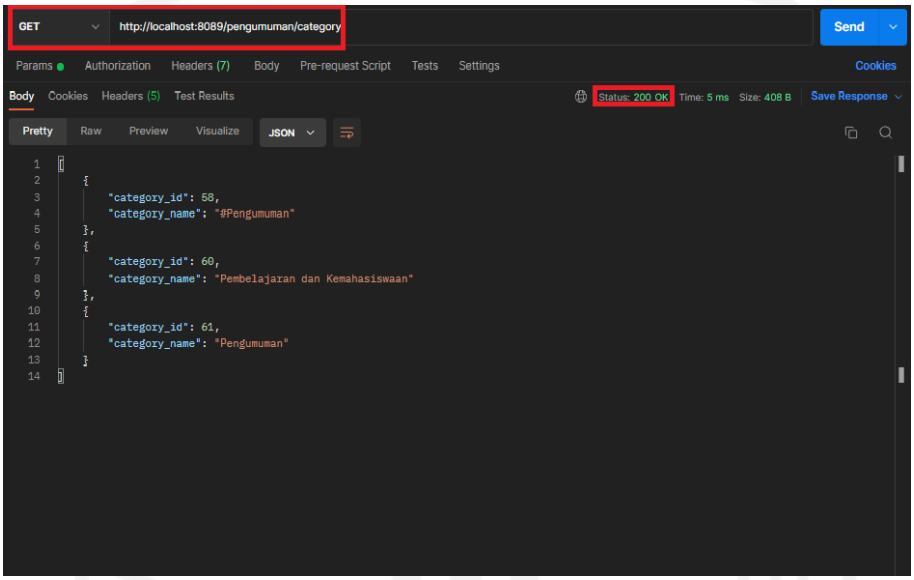
Endpoint /pengumuman/slug/:slug digunakan untuk mengambil data pengumuman berdasarkan *slug* yang disediakan oleh server. Pengujian dilakukan dengan cara mengirimkan *request* data ke *endpoint* tersebut dengan *method* GET dan memasukkan URL *http://localhost:8089/pengumuman/slug/:slug*. Untuk membaca data pengumuman berdasarkan *slug* yang diinginkan, tidak perlu mengisi *Query Params* karena hanya menambahkan URI *Params* berdasarkan *slug*. Contoh *slug* yang akan digunakan adalah sosialisasi-program-pembinaan-mahasiswa-wirausaha-p2mw-2022. Kemudian, *server* memberikan *response* seperti pada Gambar 3.31 yang menunjukkan *response* berhasil dengan status kode 200.



Gambar 3.31 Response Membaca Data Pengumuman berdasarkan *Slug*

h. *Response* membaca data semua kategori pengumuman

Endpoint /pengumuman/category digunakan untuk mengambil data semua kategori pengumuman yang disediakan oleh server. Pengujian dilakukan dengan cara mengirimkan *request* data ke *endpoint* tersebut dengan *method* GET dan memasukkan URL *http://localhost:8089/pengumuman/category*. Untuk membaca data semua kategori pengumuman, tidak perlu mengisi *Query Params* ataupun URI *Params*. Kemudian, *server* memberikan *response* seperti pada Gambar 3.32 yang menunjukkan *response* berhasil dengan status kode 200.



Gambar 3.32 Response Membaca Semua Data Kategori Pengumuman

BAB IV
REFLEKSI PELAKSANAAN MAGANG

4.1 Relevansi Akademik

Selama kurang lebih 6 bulan melaksanakan magang di Ditjen Diktiristek, penulis mendapatkan pengalaman serta pengetahuan yang berdampak dan bermanfaat. Diantaranya yaitu penerapan manajemen proyek *Scrum*, penggunaan Golang, dan implementasi *framework* Echo. Terdapat beberapa relevansi terkait pelaksanaan magang dengan teori yang telah diuraikan pada bab 2 yaitu *Scrum*, Golang, dan *Framework* Echo.

4.1.1 Relevansi Scrum

Scrum merupakan kerangka kerja yang mengelola proses pengembangan perangkat lunak. *Scrum* termasuk dalam metode *Agile Development* yang menggunakan pendekatan iterasi dan berkelanjutan. Ada 3 elemen organisasi utama dalam *Scrum* yaitu *Product Owner, Scrum Master* dan *The Scrum Team*. Pada penerapannya, terdapat rangkaian aktivitas yang dilakukan yaitu *Backlog, Sprints, Scrum Meetings,* dan Demo.

Pada teori dijelaskan bahwa terdapat elemen-elemen yang berpartisipasi dalam pelaksanaan pengembangan proyek dengan tugas yang berbeda. Pelaksanaan yang dilakukan selama magang, penulis sebagai *Backend Developer* termasuk pada elemen *The Scrum Team*. Elemen tersebut merupakan elemen yang mencakup sekelompok orang atau tim yang bertugas sebagai pengembang perangkat lunak.

Selama proses pengembangan perangkat lunak, penulis mengikuti beberapa rangkaian aktivitas yang dilakukan dalam menerapkan *Scrum*. Rangkaian aktivitas tersebut yaitu *Sprint Refinement, Daily Stand-Up Meeting, Technical Meeting, Sprint Review*, dan Koordinasi Proyek. Terdapat perbedaan teori yang dituliskan terhadap pelaksanaan yang terjadi di lapangan yaitu penulis tidak ikut serta dalam aktivitas *Backlog* yang mana aktivitas tersebut bertujuan untuk membuat daftar fitur atau kebutuhan yang akan dikembangkan dan aktivitas Demo yang mana aktivitas tersebut bertujuan untuk simulasi sementara dari hasil proyek yang dikembangkan.

Berdasarkan hal-hal yang telah dijelaskan, dapat disimpulkan bahwa penerapan metode *scrum* memiliki persamaan aktivitas yang dilakukan. Namun dalam penerapan aktivitas tersebut memiliki cara yang berbeda dalam pelaksanaannya. Setiap aktivitas tergantung pada

peran masing-masing di dalamnya untuk mengembangkan aplikasi yang tujuannya untuk mencapai *requirement* yang dibutuhkan.

4.1.2 Relevansi Golang

Golang merupakan bahasa pemrograman yang bersifat *open source*. Golang memiliki beberapa keunggulan terhadap kecepatan, kehandalan, skalabilitas, kesederhanaan dan keamanan. Golang mendukung konkurensi dalam sistem pemrograman melalui implementasi yang sederhana. Golang termasuk bahasa pemrograman yang andal dan cepat dalam skala besar dan bersifat *clean code* agar tidak membebani sistem.

Pada teori dijelaskan bahwa dengan keunggulan yang dimiliki oleh Golang dapat mempermudah *developer* dalam menulis sintaks yang dibuat untuk melakukan pengolahan data. Pelaksanaan yang dilakukan selama magang, dengan kesederhanaan penulisan sintaks pada Golang dan bersifat *clean code* dapat membantu penulis sebagai *Backend Developer* dalam membangun Web *Service*. Namun terdapat beberapa keunggulan yang belum dimanfaatkan ketika pelaksanaan magang yaitu konkurensi pada Golang. Dengan didukungnya konkurensi, proses eksekusi pada komputer tidak lagi bekerja secara sekuensial melainkan akan bekerja secara sinkronisasi atau secara paralel sehingga pengelolaan sumber daya lebih optimal dan kecepatan proses eksekusi program akan meningkat.

Berdasarkan hal-hal yang telah dijelaskan, dapat disimpulkan bahwa bahasa pemrograman Golang dapat diimplementasikan dalam membangun Web *Service*. Web *Service* merupakan layanan yang berisikan API yang berupa data sehingga dengan keunggulan yang dimiliki Golang sangat cocok diterapkan dalam membangun Web Service. Namun dalam penerapannya diharapkan dapat memanfaatkan fitur yang ada pada Golang sehingga keunggulan yang dimiliki pada Golang dapat tercapai.

4.1.3 Relevansi Framework Echo

Echo adalah sebuah web *framework* pada bahasa pemrograman Golang. *Framework* Echo memiliki beberapa keunggulan yaitu *optimized router*, didukung HTTP/2, *scalable*, data *binding*, data *rendering*, *middleware*, *automatic* TLS, dan *error handling* yang bisa dikustomisasi. Dengan demikian, *framework* Echo dapat mendukung dalam pembuatan Web *Service*.

Pada teori dijelaskan bahwa dengan keunggulan yang dimiliki oleh *framework* Echo pada Golang dapat mempermudah *developer* dalam pembuatan Web *Service* yang menerapkan

arsitektur REST API karena standar protokol yang digunakan yaitu HTTP. Pelaksanaan yang dilakukan selama magang, beberapa keunggulan yang terdapat pada *framework* Echo telah diimplementasikan yaitu data *binding*, data rendering, dan manajemen routing. Dengan adanya *framework* Echo dapat membantu penulis dalam penanganan *request* dan *response* terhadap API yang dibuat. *Framework* Echo juga membantu dalam mendefinisikan alamat *endpoint* karena penggunaannya yang minimalis dan sederhana. Namun terdapat beberapa keunggulan yang belum diajarkan ketika pelaksanaan magang yaitu *scalable* REST API, implementasi *middleware* dan kustomisasi *error handling*.

Berdasarkan hal-hal yang telah dijelaskan, dapat disimpulkan bahwa *framework* Echo pada Golang dapat diimplementasikan dalam membangun Web *Service* yang mengimplementasikan arsitektur REST API. REST API merupakan arsitektur dalam membangun Web *Service* yang terstandarisasi protokol HTTP sehingga *framework* Echo dapat diterapkan dalam membangun Web *Service* karena mendukung protokol HTTP. Namun dalam penerapannya diharapkan dapat memaksimalkan keunggulan yang dimiliki *framework* Echo sehingga Web *Service* yang dikembangkan lebih optimal.

4.2 Pembelajaran Magang

Selama kurang lebih 6 bulan magang di Ditjen Diktiristek, banyak pengalaman dan ilmu yang didapat. Magang merupakan bekal untuk merasakan langsung seperti apa dunia kerja. Dengan kegiatan yang diberikan selama magang tersebut terdapat manfaat, hambatan dan tantangan yang didapatkan terhadap hal-hal yang diajarkan terkait teori maupun pelaksanaan di lapangan.

4.2.1 Manfaat Magang

Selama kurang lebih 6 bulan magang di Ditjen Diktiristek, banyak manfaat yang didapat. Magang merupakan bekal untuk merasakan langsung seperti apa dunia kerja. Pada saat magang, penulis bergabung dalam satu proyek yaitu Satudikti. Kegiatan yang dilakukan tidak hanya kegiatan magang saja, tetapi juga mengikuti satu perkuliahan yang harus diikuti. Oleh karena itu harapannya dapat mengatur waktu dengan baik. Manfaat non teknis yang dirasakan yaitu manajemen waktu. Dimulai dari membagi tugas dalam mengerjakan task pada proyek tersebut dan juga menyelesaikan tugas-tugas yang ada di perkuliahan. Selain manajemen waktu, komunikasi sangat dibutuhkan karena pada kegiatan magang ini banyak pihak-pihak yang berpartisipasi dalam pengembangan proyek tersebut. Manfaat non teknis berikutnya yaitu

komunikasi. Komunikasi biasanya dilakukan secara lisan dan tulisan. Secara lisan, penulis biasanya berkomunikasi secara langsung maupun dengan video *conference call*. Secara tulisan, penulis biasanya berkomunikasi via *group chatting*. Komunikasi biasanya dilakukan ketika membahas pekerjaan yang akan dilakukan seperti brainstorming hingga membahas kendala yang sedang dialami. Untuk itu, cara berkomunikasi yang baik terus dilatih agar apa yang disampaikan bisa dimengerti oleh orang lain maupun tim. Manfaat lain dari komunikasi yang baik yaitu bisa akrab dengan tim dan tentunya juga mendapat sebuah relasi baru untuk mencari peluang pekerjaan setelah lulus nanti.

Manfaat teknis juga didapatkan ketika magang yaitu meningkatkan skill dan keterampilan diri. Membangun Web *Service* pada proyek Satudikti merupakan pengalaman pertama kalinya bagi penulis dan tidak didapatkan ketika di perkuliahan. Penulis diajarkan bagaimana membangun Web *Service* yang menyediakan data sesuai kebutuhan dengan menerapkan arsitektur REST API. Web *Service* dibangun menggunakan bahasa pemrograman Golang dan menggunakan *framework* Echo untuk membantu dalam penerapan arsitektur REST API. Dalam membangun REST API juga diajarkan membuat dokumentasi API *Contract*. API *Contract* berguna untuk mempermudah client untuk melihat ketentuan yang sudah ditetapkan oleh pengembang. Ketentuan tersebut juga mengikuti aturan yang ada pada REST API.

4.2.2 Hambatan Magang

Hambatan non teknis yang dirasakan yaitu ketika kegiatan magang dilakukan secara *Work From Home* (WFH). Penulis merasa kesulitan dalam mendiskusikan proyek yang dikerjakan secara online dengan tim karena penulis masih belum akrab dengan tim lainnya. Selama bergabung pada proyek Satudikti saat kegiatan dilakukan secara WFH, penulis merasa bingung mau menanggapi diskusi yang dilakukan. Namun, ini tidak berlangsung lama. Penulis harus ikut berpartisipasi terhadap diskusi yang dilakukan baik itu membahas pekerjaan maupun di luar pekerjaan. Kegiatan magang secara WFH berlangsung selama kurang lebih 1 bulan. Setelah itu, kegiatan dilakukan secara *Work From Office* (WFO). Ketika kegiatan dilakukan secara WFO, penulis juga mendapatkan hambatan lain yaitu terkait komunikasi dengan tim. Hambatannya seperti apa yang penulis maupun anggota tim sampaikan, terkadang penulis maupun anggota tim mengartikannya berbeda. Oleh karena itu hambatan ini membuat penulis terkadang sering terjadi *miss communication*. Untuk mengatasi hal ini, biasanya penulis akan menjelaskan secara detail maksud dari apa yang penulis sampaikan begitu juga sebaliknya.

Hambatan teknis yang dirasakan ketika magang yaitu saat membuat sintaks dan *output* yang diharapkan tidak sesuai. Ini akan berdampak pada sisi *client* karena informasi yang ditampilkan tidak sesuai. Hal tersebut terjadi karena tidak teliti dan kurangnya wawasan yang luas terkait penggunaan bahasa pemrograman Golang. Untuk mengatasi hal tersebut, mitra magang membuat aturan setiap sintaks yang dibuat anggota tim yang lain bisa meninjau sintaks terlebih dahulu dan untuk meminimalkan hal tersebut dibutuhkan *sharing session* sesama anggota tim terkait pengetahuan terhadap bahasa pemrograman Golang.

4.2.3 Tantangan Magang

Tantangan yang dihadapi secara non teknis yaitu terhadap beban *task* yang diberikan oleh tim *Product*. Pengerjaan *task* yang diberikan berkaitan dengan wawasan dan pengetahuan yang dimiliki karena itu sangat mempengaruhi waktu pengerjaannya. Dengan demikian penulis terus melatih diri mendalami bahasa pemrograman Golang agar dapat membantu ketika mengerjakan *task* tersebut dan bisa mengerjakan *task* dengan tepat waktu.

Tantangan yang dihadapi secara teknis ketika melaksanakan magang yaitu harus bisa menguasai bahasa pemrograman Golang yang mana bahasa pemrograman tersebut penulis tidak pernah menggunakannya dan juga tidak mendapatkan pembelajarannya di perkuliahan. Sulit untuk beradaptasi terhadap bahasa pemrograman yang baru karena sebelumnya penulis menguasai bahasa pemrograman PHP yang menggunakan *framework* Laravel di perkuliahan. Untuk menghadapi tantangan tersebut, penulis latihan mandiri di rumah agar bisa mengimplementasikan bahasa pemrograman Golang. Penulis juga merasa terbantu dengan adanya *training* pada kegiatan magang ini.

BAB V
PENUTUP

5.1 Kesimpulan

Tujuan dari penulisan tugas akhir ini adalah untuk memberikan gambaran tentang membangun Web *Service* selama magang di Ditjen Diktiristek. Selain itu, ini bertujuan untuk memberikan pengetahuan mendalam tentang arsitektur REST API yang digunakan dalam membangun Web *Service*. Diharapkan ilmu yang diperoleh dengan penulisan tugas akhir ini dapat menjadi dasar pengetahuan untuk membangun Web *Service* yang mengimplementasikan arsitektur REST API dan menggunakan bahasa pemrograman Golang.

Berdasarkan hasil penulisan tugas akhir yang telah dibuat, dapat disimpulkan sebagai berikut:

a. Aplikasi Satudikti menyediakan informasi dari seluruh layanan yang ada pada Ditjen Diktiristek. Selain itu, aplikasi sudah bisa digunakan karena aplikasi sudah rilis pada layanan distribusi digital seperti App Store dan Play Store sehingga pengguna hanya perlu mengakses aplikasi Satudikti untuk menggunakan seluruh layanan yang ada pada Ditjen Diktiristek..
b. Pengembangan Web *Service* yang dilakukan dapat menggambarkan proses pengerjaan secara keseluruhan karena melewati proses perencanaan hingga menyentuh proses pengujian dan keluaran yang diinginkan telah tercapai yaitu berupa API.
c. Kebutuhan data pada aplikasi Satudikti telah tersedia pada *database* dan *Third-party* API.
d. Dibuatnya Web *Service* adalah untuk menyediakan API dan mempermudah *client* seperti aplikasi Satudikti untuk mendapatkan data maupun informasi terhadap seluruh layanan yang ada pada Ditjen Diktiristek.
e. Arsitektur REST API diterapkan dalam membangun Web *Service* karena komponen yang terdapat pada REST API cukup untuk memenuhi kebutuhan dalam menyediakan suatu API.
f. Dalam pengembangannya, bahasa pemrograman Golang dipilih karena bersifat *open source, clean code,* dan pemrosesan kode yang cepat dan handal untuk skala besar.

g. Dokumentasi API telah menjelaskan aturan secara rinci dari API yang dibuat sehingga pemangku kepentingan yang menggunakan API dapat memahami bagaimana cara mengkonsumsi API tersebut.
h. Keuntungan yang didapat dalam membangun Web *Service* yaitu mempermudah mendapatkan data atau informasi yang diakses oleh *client* aplikasi atau platform yang memiliki antarmuka maupun tanpa antarmuka.

5.2 Saran

Ada beberapa saran yang ingin disampaikan penulis terkait pelaksanaan magang akan ditujukan kepada:

a. Mitra Magang

   Dari hasil pelaksanaan magang di Ditjen Diktiristek, perlu ditingkatkan komunikasi internal perusahaan agar tidak terjadi *miscommunication*. Kemudian, ketika tim pengembang meminta suatu kebutuhan yang berkaitan dengan proses pengembangan proyek diharapkan segera memproses permintaan tersebut agar target pengembangan dapat diselesaikan dengan tepat waktu. Selain itu *training* yang diberikan terkait pekerjaan apa yang dikerjakan harapannya materi yang diberikan sesuai dengan kebutuhan terhadap pengembangan proyek yang dikerjakan agar membantu tim pengembang untuk beradaptasi dengan cepat terhadap *tools* maupun teknologi yang digunakan.

b. Prodi Informatika UII

   Pada kegiatan magang di Ditjen Diktiristek, penulis melakukan magang melalui salah satu program yang diadakan oleh Kementrian Pendidikan Kebudayaan, Riset dan Teknologi yaitu Kampus Merdeka. Harapannya benefit yang diberikan oleh program Kampus Merdeka yaitu konversi 20 sks dapat terpenuhi secara penuh karena ini merupakan benefit yang seharusnya didapatkan mahasiswa yang mengikuti program tersebut. Terkait masalah komunikasi, sebaiknya jika ada pengumuman atau perubahan hal-hal penting disampaikan secara jelas dan tidak mendadak.

c. Mahasiswa Penjaluran Magang

   Untuk mahasiswa yang akan mengambil penjaluran magang, harapannya dapat memiliki bekal terlebih dahulu sebelum melaksanakan magang. Bekal tersebut berupa hal teknis dan non teknis. Hal teknis yang perlu dimiliki yaitu tidak harus sampai menguasai semua hal melainkan harus tau dan memiliki pengetahuan apa

yang akan dilakukan terhadap aktivitas yang akan dilakukan atau peran yang akan diambil. Hal non teknis yang perlu dimiliki diantaranya yaitu kedisiplinan, manajemen waktu, berpikir kritis, tanggung jawab, dan bisa bekerja sama dalam tim.

d. Diri Sendiri

Saran untuk diri sendiri yaitu lebih ditingkatkan terkait pembelajaran mengenai hal-hal teknis yang didapatkan selama magang dan juga untuk hal-hal non teknis dapat diterapkan dikehidupan sebagai bekal untuk memasuki dunia kerja. Selain itu juga harapannya penulis dapat mengeksplor lagi terkait memaksimalkan keunggulan dari teknologi yang digunakan selama pengembangan proyek.

e. Penelitian atau Pengembangan Lanjutan

Dalam pengembangannya, untuk mengatasi kesalahan pembuatan sintaks yang telah dibuat, sintaks perlu diuji terlebih dahulu. Salah satu pengujian yang bisa dilakukan adalah *Unit Testing*. Kemudian, agar penulisan sintaks menjadi efektif dan efisien dapat menerapkan arsitektur yang ada pada pengembangan perangkat lunak agar dapat membantu dalam perkembangan aplikasi maupun pemeliharaan sintaks.

DAFTAR PUSTAKA

Ahmad, F. (2018, September 12). *Test Driven Development (TDD) Restful API Using Mock Server Postman*. Medium. Dipetik Oktober 18, 2022, dari https://medium.com/@fatoniahmad.mail/test-driven-development-tdd-restful-api-using-mock-server-postman-1fd1cb271ed0

Akbar, M. (2018). *Pengembangan RESTful API Untuk Application Specific High Level Location Service*. Dipetik September 09, 2022, dari https://dspace.uii.ac.id/handle/123456789/9836

Ari Kristanto, A., Harjoseputro, Y., & Samodra, J. E. (2020). Implementasi Golang Dan New Simple Queue Pada Sistem Sandbox Pihak Ketiga Berbasis REST API. *J. RESTI (Rekayasa Sist. Dan Teknol. Informasi)*, *4*(4), 745–750.

BeattieM. (2016, February 24). *API Contract Example*. GitHub. Dipetik September 10, 2022, dari https://gist.github.com/BeattieM/324063512d166122ba5c

Diktiristek, D. (2022). *Profil Institusi Direktorat Jenderal Pendidikan Tinggi, Riset, dan Teknologi*. Direktorat Jenderal Pendidikan Tinggi. Dipetik Oktober 13, 2022, dari https://dikti.kemdikbud.go.id/profil-institusi-direktorat-jenderal-pendidikan-tinggi/

Firdaus, A., Widodo, S., Sutrisman, A., Gading, S., Nasution, F., Mardiana, R., Komputer, J. T., Negeri, P., & Palembang, S. (2019). Rancang Bangun Sistem Informasi Perpustakaan Menggunakan Web Service Pada Jurusan Teknik Komputer POLSRI. *Jurnal Informanika*, *5*(2).

iTech. (2021, October 1). *What Are Third-Party API Integrations And Why Your App Needs It*. Dipetik September 11, 2022, dari https://itechindia.co/blog/third-party-api-integrations/

Jools.dev. (2021, February 27). *What is an API contract?* Dipetik September 10, 2022, dari https://jools.dev/what-is-an-api-contract

Kristanto, A. A. (2020). *Pembangunan Sistem Sandbox Pada Pengujian Terhadap Pihak Ketiga Berbasis REST API Menggunakan Golang Dan NSQ*. http://e-journal.uajy.ac.id/id/eprint/23329

LabStack. (2022). *Echo - High performance, minimalist Go web framework*. Dipetik Oktober 18, 2022, dari https://echo.labstack.com/

Nagaraj, P., & N, S. C. (2020). Third-party API Integration to Applications. *International Research Journal of Engineering and Technology*, *7*(5). www.irjet.net

Perdana, M. A. K. (2018). *Pengembangan REST API Layanan Penyimpanan Menggunakan Metode Rapid Application Development (Studi Kasus: PT. XYZ)*.

Postman. (2022). *About Postman*. Dipetik Oktober 18, 2022, dari https://www.postman.com/company/about-postman/

Pratomo, B. D., & Haryono, K. (2020). Perancangan RESTful Web Service Satuan Kredit Partisipasi di Universitas Islam Indonesia. *Seri Prosiding Seminar Nasional Dinamika Informatika*, *4*(1).

Putra, I. N. T. A., Bisenna, I. K. A., & Kartini, K. S. (2018). Pengembangan Sistem Inventaris Berbasis QR Code Menggunakan Web Service Pada Bidang Sarana Dan Prasarana STMIK STIKOM Indonesia. *Jurnal Nasional Pendidikan Teknik Informatika: JANAPATI*, *7*(3), 315–323.

Rizal, R., & Rahmatulloh, A. (2019). RESTful Web Service Untuk Integrasi Sistem Akademik Dan Perpustakaan Universitas Perjuangan. *Jurnal Ilmiah Informatika*, *7*(01), 54–59.

Saputra, D., & Fathoni Aji, R. (2018). Analisis Perbandingan Performa Web Service REST Menggunakan Framework Laravel, Django Dan Ruby On Rails Untuk Akses Data Dengan Aplikasi Mobile (Studi Kasus: Portal E-Kampus STT Indonesia Tanjungpinang). *Bangkit Indonesia*, *2*.

Sari, A. S., & Hidayat, R. (2022). Designing Website Vaccine Booking System Using Golang Programming Language And Framework React JS. *Journal of Information System, Informatics and Computing Issue Period*, *6*(1), 22–39. https://doi.org/10.52362/jisicom.v6i1.760

Subagio, A. W., & Muttaqin, F. (2022). Penerapan Clean Architecture Pada Pengembangan Sistem Payment Point Online Bank. *Jurnal Teknologi Elektro Dan Kejuruan*, *32*(2), 324–333. https://doi.org/http://dx.doi.org/10.17977/um034v32i2p324-333

Suharno, H. R., Gunantara, N., & Sudarma, M. (2020). Analisis Penerapan Metode Scrum Pada Sistem Informasi Manajemen Proyek Dalam Industri & Organisasi Digital. *Majalah Ilmiah Teknologi Elektro*, *19*(2), 203–210. https://doi.org/10.24843/mite.2020.v19i02.p12

Sunardi, S., & Fadli, S. (2018). Identifikasi Masalah Penerapan Metode Agile (SCRUM) Pada Pengembangan Perangkat Lunak Di Perguruan Tinggi (Studi Kasus Universitas Nahdlatul Ulama Nusa Tenggara Barat). *Jurnal Manajemen Informatika Dan Sistem Informasi*, *1*(2), 14–18. http://e-journal.stmiklombok.ac.id/index.php/misi

LAMPIRAN

Lampiran A

KEMENTERIAN PENDIDIKAN, KEBUDAYAAN,
RISET, DAN TEKNOLOGI
DIREKTORAT JENDERAL PENDIDIKAN TINGGI,
RISET, DAN TEKNOLOGI
Jalan Jenderal Sudirman, Senayan, Jakarta 10270
Telepon (021) 57946104, Pusat Panggilan ULT Dikti 126
Laman www.dikti.kemdikbud.go.id

SURAT PERNYATAAN
PERJANJIAN KERAHASIAAN DATA NEGARA

Saya yang bertandatangan di bawah ini :

Nama	:	**Sigit Priadi**
NIM	:	18523125
Perguruan Tinggi	:	Universitas Islam Indonesia
Alamat	:	Jl. Nusantara Km. 18, RT 002 RW 005, Kelurahan Sungai Lekop, Kecamatan Bintan Timur, Kabupaten Bintan, Provinsi Kepulauan Riau
No. HP	:	085740120843

Sebagai mahasiswa pada Program Magang Merdeka Kampus Merdeka di Pangkalan Data Pendidikan Tinggi (PDDikti), Sekretariat Direktorat Jenderal Pendidikan Tinggi, Riset, dan Teknologi, Kementerian Pendidikan, Kebudayaan, Riset, dan Teknologi pada periode Agustus Tahun 2021 s.d. Januari Tahun 2022 dengan ini menyatakan bahwa saya :

1. Sanggup menjaga kerahasiaan informasi maupun data yang saya ketahui dan yang saya miliki berkaitan dengan data Negara dengan tidak memberikan dan/atau menyebarluaskannya kepada pihak-pihak yang tidak berkepentingan dan pihak-pihak lain yang dapat memanfaatkannya untuk kepentingan pribadi atau kelompok yang berpotensi merugikan.
2. Bersedia dikenakan sanksi sesuai peraturan perundang-undangan yang berlaku apabila saya lalai atau dengan sengaja berbuat sesuatu yang mengakibatkan tersebarnya data dan atau informasi yang saya ketahui/miliki.

Demikian surat pernyataan ini saya buat dengan sebenarnya tanpa ada paksaan dari pihak manapun.

Jakarta, 16 Agustus 2021
Yang membuat pernyataan,

Sigit Priadi

Lampiran B

1. **GET List Berita**
   - **Endpoint : /berita**
   - **Method : GET**
   - **Query Params :**

```
{
   category_id: “optional|integer”,
   title: “required|string”,
   page: “required|numeric”,
   limit: “required|numeric”
}
```

   - **URI Params : None**
   - **Data Params : None**
   - **Headers :**
     - **Content-Type : application/json**
   - **Success Response**
     - **Code : 200**
       Content Response

```
{
    current_page: integer,
    data: [
            {
                id: integer,
                slug: string,
                title: text,
                link: text,
                description: text,
                publisher: string,
                publisher_date: datetime,
                publisher_status: string,
                creator: string,
                thumbnail_image: string,
                inserted_date: datetime,
                categories: [
                        {
                          category_id: integer,
                          category_name: string,
                        }
                ],
            }
    ],
    last_page: integer,
    total_item: integer,
    total_item_per_page: integer,
}
```

   - **Error Response**
     - **Code : 400 (Bad Request)**
       Content Response

```
{
  message: string,
  error:   string
}
```

- **Code : 404 (Not Found)**
  Content Response
  ```
  {
    message: string
  }
  ```
- **Code : 500 (Server Error)**
  Content Response
  ```
  {
    message: string,
    error:   string
  }
  ```

2. GET Detail Berita by Id

- **Endpoint : /berita/:id**
- **Method : GET**
- **Query Params : None**
- **URI Params :**
  ```
  {
     id: “required|integer”
  }
  ```
- **Data Params : None**
- **Headers :**
  - **Content-Type : application/json**
- **Success Response**
  - **Code : 200**
    Content Response
    ```
    {
       id: integer,
       slug: string,
       title: text,
       link: text,
       description: text,
       publisher: string,
       publisher_date: datetime,
       publisher_status: string,
       creator: string,
       content: long_text,
       thumbnail_image: string,
       inserted_date: datetime,
       categories: [
              {
                category_id: integer,
                category_name: string,
              }
       ],
    }
    ```
- **Error Response**
  - **Code : 404 (Not Found)**
    Content Response
    ```
    {
      message: string
    }
    ```
  - **Code : 500 (Server Error)**

Content Response

```
{
  message: string,
  error:   string
}
```

3. **GET Detail Berita by Slug**
   - **Endpoint : /berita/slug/:slug**
   - **Method : GET**
   - **Query Params : None**
   - **URI Params :**

     ```
     {
        slug: "required|string"
     }
     ```
   - **Data Params : None**
   - **Headers :**
     - **Content-Type : application/json**
   - **Success Response**
     - **Code : 200**

       Content Response

       ```
       {
          id: integer,
          slug: string,
          title: text,
          link: text,
          description: text,
          publisher: string,
          publisher_date: datetime,
          publisher_status: string,
          creator: string,
          content: long_text,
          thumbnail_image: string,
          inserted_date: datetime,
          categories: [
                {
                  category_id: integer,
                  category_name: string,
                }
          ],
       }
       ```
   - **Error Response**
     - **Code : 404 (Not Found)**

       Content Response

       ```
       {
         message: string
       }
       ```
     - **Code : 500 (Server Error)**

       Content Response

       ```
       {
         message: string,
         error:   string
       }
       ```
4. **GET List Category Berita**

- **Endpoint : /berita/category**
- **Method : GET**
- **Query Params : None**
- **URI Params : None**
- **Data Params : None**
- **Headers :**
  - **Content-Type : application/json**
- **Success Response**
  - **Code : 200**
    Content Response

```
{
   category_id:   integer,
   category_name: string
}
```

- **Error Response**
  - **Code : 404 (Not Found)**
    Content Response

```
{
  message: string
}
```

  - **Code : 500 (Server Error)**
    Content Response

```
{
  message: string,
  error:   string
}
```

5. **GET List Pengumuman**
   - **Endpoint : /pengumuman**
   - **Method : GET**
   - **Query Params :**

```
{
   category_id: “optional|integer”,
   title: “required|string”,
   page: “required|numeric”,
   limit: “required|numeric”
}
```

   - **URI Params : None**
   - **Data Params : None**
   - **Headers :**
     - **Content-Type : application/json**
   - **Success Response**
     - **Code : 200**
       Content Response

```
{
   current_page: integer,
   data: [
            {
               id: integer,
               slug: string,
               title: text,
               link: text,
               description: text,
```

```
                    publisher: string,
                    publisher_date: datetime,
                    publisher_status: string,
                    creator: string,
                    thumbnail_image: string,
                    inserted_date: datetime,
                    categories: [
                           {
                             category_id: integer,
                             category_name: string,
                           }
                    ],
                }
        ],
        last_page: integer,
        total_item: integer,
        total_item_per_page: integer,
      }
```

- **Error Response**
  - **Code : 400 (Bad Request)**
    Content Response
    ```
    {
      message: string,
      error:   string
    }
    ```
  - **Code : 404 (Not Found)**
    Content Response
    ```
    {
      message: string
    }
    ```
  - **Code : 500 (Server Error)**
    Content Response
    ```
    {
      message: string,
      error:   string
    }
    ```

6. **GET Detail Pengumuman by Id**
   - **Endpoint : /pengumuman/:id**
   - **Method : GET**
   - **Query Params : None**
   - **URI Params :**
     ```
     {
        id: “required|integer”
     }
     ```
   - **Data Params : None**
   - **Headers :**
     - **Content-Type : application/json**
   - **Success Response**
     - **Code : 200**
       Content Response
       ```
           {
          id: integer,
       ```

```
            slug: string,
            title: text,
            link: text,
            description: text,
            publisher: string,
            publisher_date: datetime,
            publisher_status: string,
            creator: string,
            content: long_text,
            thumbnail_image: string,
            inserted_date: datetime,
            categories: [
                  {
                    category_id: integer,
                    category_name: string,
                  }
            ],
        }
```

- **Error Response**
  - **Code : 404 (Not Found)**
    Content Response

```
{
  message: string
}
```

  - **Code : 500 (Server Error)**
    Content Response

```
{
  message: string,
  error:   string
}
```

7. **GET Detail Pengumuman by Slug**
   - **Endpoint : /pengumuman/slug/:slug**
   - **Method : GET**
   - **Query Params : None**
   - **URI Params :**

```
{
   slug: “required|string”
}
```

   - **Data Params : None**
   - **Headers :**
     - **Content-Type : application/json**
   - **Success Response**
     - **Code : 200**
       Content Response

```
{
   id: integer,
   slug: string,
   title: text,
   link: text,
   description: text,
   publisher: string,
   publisher_date: datetime,
```

```
        publisher_status: string,
        creator: string,
        content: long_text,
        thumbnail_image: string,
        inserted_date: datetime,
        categories: [
               {
                 category_id: integer,
                 category_name: string,
               }
        ],
    }
```

- **Error Response**
  - **Code : 404 (Not Found)**
    Content Response

```
{
  message: string
}
```

  - **Code : 500 (Server Error)**
    Content Response

```
{
  message: string,
  error:   string
}
```

8. **GET List Category Pengumuman**
   - **Endpoint : /pengumuman/category**
   - **Method : GET**
   - **Query Params : None**
   - **URI Params : None**
   - **Data Params : None**
   - **Headers :**
     - **Content-Type : application/json**
   - **Success Response**
     - **Code : 200**
       Content Response

```
{
   category_id:   integer,
   category_name: string
}
```

   - **Error Response**
     - **Code : 404 (Not Found)**
       Content Response

```
{
  message: string
}
```

     - **Code : 500 (Server Error)**
       Content Response

```
{
  message: string,
  error:   string
}
```

Lampiran C

No.	Tanggal	Aktivitas
1.	16 Agu 2021	Bersama Mitra : Pembukaan Program Magang Merdeka Dikti SIGAP Melayani dan Sharing Session Jakarta Smart City (JAKI) Kepada Mahasiswa Magang Merdeka Dikti. Individu : Mencari informasi tentang bahasa pemrograman Go-Lang, seperti membaca dokumentasi di internet(golang.org)
2.	17 Agu 2021	Bersama Mitra : tidak ada kegiatan Individu : Mencari informasi tentang bahasa pemrograman Go-Lang, seperti membaca dokumentasi di internet(golang.org) serta mendengarkan penjelasan tentang Go-Lang di Youtube.
3.	18 Agu 2021	Bersama Mitra : Penjelasan terkait project magang, berkenalan dengan trainer dan Sosialisasi Layanan yang ada pada Sekretariat Ditjen Diktiristek. Individu : Masih memahami informasi tentang bahasa pemrograman Go-Lang, seperti membaca dokumentasi di internet(golang.org) serta mendengarkan penjelasan tentang Go-Lang di Youtube.
4.	19 Agu 2021	Bersama Mitra : tidak ada kegiatan Individu : Masih memahami informasi tentang bahasa pemrograman Go-Lang, seperti membaca dokumentasi di internet(golang.org) serta mendengarkan penjelasan tentang Go-Lang di Youtube.
5.	20 Agu 2021	Bersama Mitra : tidak ada kegiatan Individu : Masih memahami informasi tentang bahasa pemrograman Go-Lang, seperti membaca dokumentasi di internet(golang.org) serta mendengarkan penjelasan tentang Go-Lang di Youtube sekaligus mencoba menyiapkan starterpack yang dibutuhkan untuk menulis bahasa Go-Lang (editor, installer, dll)
6.	23 Agu 2021	National Onboarding: Pelepasan Mahasiswa program Magang dan Studi Independen Bersertifikat (MSIB)
7.	24 Agu 2021	Bersama Mitra : tidak ada kegiatan Individu : Belajar dasar pemrograman Go-Lang di canel youtube (pengenalan Go-Lang, program sederhana helloworld, tipe data number, tipe data boolean, tipe data string)
8.	25 Agu 2021	Bersama Mitra : tidak ada kegiatan Individu : Belajar dasar pemrograman Go-Lang di channel youtube (variabel, konstanta, konversi tipe data, type declaration)
9.	26 Agu 2021	Bersama Mitra : tidak ada kegiatan Individu : Belajar dasar pemrograman Go-Lang di channel youtube (operasi matematika, operasi perbandingan, operasi boolean, tipe data array)
10.	27 Agu 2021	Bersama Mitra : Diskusi terkait project Dikti Super Apps serta pengenalan bersama rekan-rekan divisi Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Tipe data slice, tipe data map, If expression, switch expression)
11.	30 Agu 2021	Bersama Mitra : Mitra menginvite ke software mangement project yaitu asana Individu : Mencari pengetahuan tentang asana, belajar menggunakan asana, belajar dasar pemrograman Go-Lang di channel youtube (For-Loops)
12.	31 Agu 2021	Bersama Mitra : Tidak ada kegiatan Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Break & Continue)
13.	1 Sep 2021	Bersama Mitra : Mengirim biodata dan foto untuk keperluan publikasi Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Break & Continue)

14.	2 Sep 2021	Bersama Mitra : Pelatihan Sesi 2 Back End Developer serta pemberian tugas yang berkaitan dengan materi, diberikan tugas oleh mentor untuk menganalisis terhadap modul aplikasi Ijazah LN & SIVIL agar dapat permasalahan yang dipertanyakan Individu : memahami kembali yang diajarkan oleh trainer, meet bersama rekan magang Back End untuk membahas tugas yang diberikan oleh mentor
15.	3 Sep 2021	Bersama Mitra : Audiensi Dikti Superapps Modul Aplikasi PIN SIVIL dan Ijazah LN, Arahan Pengembangan Aplikasi bersama mentor Back End Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Function, Function Parameter, Function Return Value)
16.	6 Sep 2021	Bersama Mitra : Audiensi Dikti SuperApp Modul Penilaian Angka Kredit Dosen Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Returning Multiple Values, Named Return Values)
17.	7 Sep 2021	Bersama Mitra : Tidak ada kegiatan Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Variadic Function, Function Value, Function as Parameter, Anonymous Function, Recursive Function)
18.	8 Sep 2021	Bersama Mitra : Audiensi Dikti SuperApp Modul Kelembagaan pada SIAGA Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Closure, Defer, Panic, & Recover, Komentar, Struct, Struct Method)
19.	9 Sep 2021	Bersama Mitra : Audiensi Dikti SuperApp Modul Informasi Rekacipta (Aplikasi Kedaireka) Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Interface, Interface Kosong, Nil, Error Interface, Type Assertions)
20.	10 Sep 2021	Bersama Mitra : Audiensi Dikti SuperApp Modul MBKM Individu : Bertukar pikiran bersama rekan-rekan magang dan mentor Back End terkait materi API, Belajar dasar pemrograman Go-Lang di channel youtube (Pointer, Pointer di Function, Pointer di Method, GOPATH, Package & Import) *Tambahan - 11 September 2021 Bersama Mitra : Audiensi Dikti SuperApp Modul News Individu : Melanjutkan tugas yang diberikan oleh trainer hingga tugas tersebut selesai
21.	13 Sep 2021	Bersama Mitra : Tidak ada kegiatan Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Access Modifier, Package Initialization, Package OS)
22.	14 Sep 2021	Bersama Mitra : Mentor memberikan task simple crud rest api dengan echo framework Individu : Mempelajari Go-Lang di echo labstack guide dan tutorial di youtube dalam pembuatan rest api dengan echo framework
23.	15 Sep 2021	Bersama Mitra : Tidak ada kegiatan Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Package Flag, Package Strings, Package Strconv)
24.	16 Sep 2021	Bersama Mitra : Sesi ke 3 Pelatihan Back End, Audiensi PDDikti Individu : Mempelajari kembali dasar-dasar pemrograman Go-Lang yang diajarkan oleh trainer saat pelatihan
25.	17 Sep 2021	Bersama Mitra : Tidak ada kegiatan Individu : Mempelajari kembali Go-Lang di echo labstack guide dan tutorial di youtube dalam pembuatan rest api dengan echo framework

26.	20 Sep 2021	Bersama Mitra : Koordinasi bersama semua rekan magang. Koordinasi terkait tugas yang diberikan mentor yang ada pada Asana (Setup Git - Sample Project CRUD Go with Echo). Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Package Math, Package container/list).
27.	21 Sep 2021	Bersama Mitra : Diskusi bersama divisi Back End di grup whatsapp terkait code yang error pada tugas yang diberikan. Individu : Mereview dan memahami kembali code yang dibuat terkait tugas yang diberikan oleh mentor (CRUD Go with Echo).
28.	22 Sep 2021	Bersama Mitra : Tidak ada kegiatan Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Package container/list, Package container/ring)
29.	23 Sep 2021	Bersama Mitra : Koordinasi bersama divisi PM + BE + API terkait modul MBKM. Diskusi bersama mentor terkait part-part yang dilakukan dalam membangun sebuah modul. Individu : Mempelajari dan memahami kembali Go-Lang di echo labstack guide dan tutorial di youtube dalam pembuatan rest api dengan echo framework.
30.	24 Sep 2021	Bersama Mitra : Koordinasi bersama semua rekan magang. Individu : Mempelajari dan memahami kembali materi dasar Go-Lang pada pertemuan sebelumnya bersama trainer (Tipe Data Array, Tipe Data Slice, Tipe Data Map, If Expression, Switch Expression, For Loops, Break & Continue, Function, Function Parameter).
31.	27 Sep 2021	Bersama Mitra : Diskusi divisi Back End terkait laporan harian dan informasi mengenai tugas Simple CRUD Go with Echo harus diselesaikan minggu ini. Individu : Mereview kembali tugas Simple CRUD Go with Echo
32.	28 Sep 2021	Bersama Mitra : Diskusi divisi Back End terkait revisi laporan harian Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Package Sort)
33.	29 Sep 2021	Bersama Mitra : Tidak ada kegiatan Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Package Time)
34.	30 Sep 2021	Bersama Mitra : Sesi ke 4 Pelatihan Back End, Progress Report Magang Merdeka Individu : Mereview kembali materi pelatihan yang diberikan (Defer, Panic, Recover, Struct, Error Interface, Pointer, Pointer di Function, Pointer di Method, GOPATH, Package, Import, Access Modifier, Package Initialization, Blank Identifier)
35.	1 Okt 2021	Bersama Mitra : Tidak ada kegiatan Individu : Belajar dasar pemrograman Go-Lang di channel youtube (Golang Modules)
36.	4 Okt 2021	Individu : Tidak ada kegiatan Bersama Mitra : Koordinasi Magang Merdeka, Sprint Planning dan Refinement
37.	5 Okt 2021	Bersama Mitra : Magang onsite (WFO), Stand Up Meeting Individu : Mempelajari Echo Framework terutama pada method GET dan memahami tipe data struct dan map
38.	6 Okt 2021	Bersama Mitra : Magang onsite (WFO), Stand Up Meeting, Membahas task sprint yang diberikan bersama rekan divisi, PM, dan Mentor, Mempelajari materi yang diberikan mentor sebagai referensi untuk mengerjakan task sprint yang diberikan. Individu : Memahami lebih lanjut create GET API pada Echo Framework dalam membuat call service yang akan dipanggil oleh frontend dan mobile.

39.	7 Okt 2021	Bersama Mitra : Stand Up Meeting, Membahas task news dan pengumuman bersama rekan back end dan mentor terkait cara read rss, flow dari task tersebut, serta endpoint yang akan dipanggil oleh front end dan mobile. Individu : Memahami lebih lanjut pembuatan contract API method GET pada task yang diberikan, memahami flow dari framework yang akan digunakan dalam mengerjakan project yang dibuat oleh mentor.
40.	8 Okt 2021	Bersama Mitra : Stand Up Meeting, Membahas task news dan pengumuman terkait category rss bersama rekan back end dan mentor, arahan penggunaan postgresql, desain db di local untuk sementara. Individu : Memahami kembali pembuatan contract API method GET pada task yang diberikan, memahami isi rss yang akan dikonsumsi, belajar menggunakan postgresql, implementasi postgresql pada task project yang diberikan.
41.	11 Okt 2021	Bersama Mitra : Stand Up Meeting, Review Sprint and Start Sprint, Membahas task news dan pengumuman bersama rekan back end dan mentor terkait task tambahan dan revisinya. Individu : Memahami lebih lanjut pembuatan contract API method GET pada task yang diberikan, memahami flow dari framework yang akan digunakan dalam mengerjakan project yang dibuat oleh mentor.
42.	12 Okt 2021	Bersama Mitra : Stand Up Meeting, Review Sprint and Start Sprint, Membahas contract API task news dan pengumuman bersama rekan back end dan mentor terkait task tambahan dan revisinya. Individu : Pembuatan service method GET pada task yang diberikan berdasarkan contract API.
43.	13 Okt 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting, Final Revisi contract API sesuai yang ada pada task asana, Diskusi terkait contract API task news dan pengumuman bersama rekan back end, product manager, dan technical writer, Take Video bersama tim Humas. Individu :Memahami materi GET List Data dari database postgresql dan implementasi ke project sesuai yang ada pada task asana.
44.	14 Okt 2021	Bersama Mitra : Stand Up Meeting, Membahas contract API task pengumuman bersama rekan back end terkait pembuatan API yang dikonsumsi client. Individu : Pembuatan service method GET pada task pengumuman berdasarkan contract API dan berkolaborasi bersama dengan tim back end lainnya.
45.	15 Okt 2021	Bersama Mitra : Magang On Site (WFO), Stand Up Meeting, Diskusi bersama mentor terkait pembuatan API pada task pengumuman. Individu : Perbaikan source code terhadap task GET API Pengumuman bersama rekan back end.
46.	18 Okt 2021	Individu : Memperbaiki API Contract serta script yang berantakan. Bersama Mitra : Magang Onsite (WFO), Koordinasi Magang Merdeka, Review Sprint and Start Sprint
47.	19 Okt 2021	Bersama Mitra : Stand Up Meeting, Review Sprint and Start Sprint. Individu : Mempelajari dan memahami cara consume API menggunakan authentication.
48.	20 Okt 2021	Individu : Mempelajari dan memahami cara consume API menggunakan authentication. Bersama Mitra : Libur Nasional
49.	21 Okt 2021	Individu : Riset terhadap response data yang dibutuhkan oleh client pada service API Store PDDikti sekaligus start pembuatan

		API PDDikti Perguruan Tinggi. Bersama Mitra : Magang onsite (WFO), Stand Up Meeting
50.	22 Okt 2021	Bersama Mitra : Stand Up Meeting, Review Sprint. Individu : Riset terhadap response data yang dibutuhkan oleh client pada service API Store PDDikti sekaligus start pembuatan API PDDikti Perguruan Tinggi dan Prodi.
51.	25 Okt 2021	Individu : Riset terhadap response data yang dibutuhkan oleh client pada service API Store PDDikti sekaligus pembuatan API PDDikti Perguruan Tinggi dan Prodi. Membuat API Contract modul PDDikti untuk Perguruan Tinggi dan Prodi. Bersama Mitra : Koordinasi Magang Merdeka, Review Sprint and Start Sprint.
52.	26 Okt 2021	Individu : Riset terhadap response data yang dibutuhkan oleh client pada service API Store PDDikti sekaligus masih pada tahap pengembangan API PDDikti Perguruan Tinggi dan Prodi. Bersama Mitra : Magang onsite (WFO), Stand Up Meeting, Review Sprint and Start Sprint
53.	27 Okt 2021	Individu : Riset terhadap response data yang dibutuhkan oleh client pada service API Store PDDikti sekaligus masih pada tahap pengembangan API PDDikti Perguruan Tinggi dan Prodi. Bersama Mitra : Magang onsite (WFO), Stand Up Meeting, Diskusi bersama tim PM dan BE terkait permasalahan service API Store PDDikti
54.	28 Okt 2021	Individu : Riset terhadap response data yang dibutuhkan oleh client pada service API Store PDDikti sekaligus masih pada tahap pengembangan API PDDikti Perguruan Tinggi dan Prodi. Bersama Mitra : Magang onsite (WFO), Stand Up Meeting, Weekly meeting bersama mentor
55.	29 Okt 2021	Individu : Riset terhadap response data yang dibutuhkan oleh client pada service API Store PDDikti sekaligus masih pada tahap pengembangan API PDDikti Perguruan Tinggi dan Prodi. Bersama Mitra : Magang onsite (WFO), Stand Up Meeting Sabtu : Meeting bersama tim PM terkait permasalahan service API Store PDDikti
56.	1 Nov 2021	Bersama Mitra : Magang Onsite (WFO), Koordinasi Magang Merdeka
57.	2 Nov 2021	Bersama Mitra : Magang Onsite (WFO), Koordinasi Magang Merdeka Individu : Tidak ada kegiatan
58.	3 Nov 2021	Bersama Mitra : Koordinasi Magang Merdeka Individu : Riset terhadap response data yang dibutuhkan oleh client pada service API Store PDDikti sekaligus masih pada tahap pengembangan API PDDikti.
59.	4 Nov 2021	Bersama Mitra: Magang Onsite (WFO), Standup Meeting, Weekly meeting bersama mentor Individu : masih pada tahap pengembangan API PDDikti.
60.	5 Nov 2021	Bersama Mitra: Magang Onsite (WFO), Standup Meeting Individu : masih pada tahap pengembangan API PDDikti sekaligus push pengerjaan yang sudah dilakukan.
61.	8 Nov 2021	Sabtu : pengerjaan ijazah ln terkait Mapping data negara ke DB Super App dan Get List API by Negara. Bersama Mitra : Koordinasi Magang Merdeka terkait start and review sprint. Individu : Riset terhadap response data yang dibutuhkan oleh client pada service API Elastic PDDikti.

62.	9 Nov 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting Individu : Riset terhadap response data yang dibutuhkan oleh client pada service API Elastic PDDikti sekaligus tahap pengembangan Get API Elastic List Perguruan Tinggi, Prodi, dan Provinsi
63.	10 Nov 2021	Bersama Mitra : Stand Up Meeting Individu : Riset terhadap response data yang dibutuhkan oleh client pada service API Elastic PDDikti sekaligus tahap pengembangan Get API Elastic Search All Query
64.	11 Nov 2021	Bersama Mitra: Standup Meeting, Koordinasi Tim BE bersama Tim PM terkait kendala kerjaan, Weekly meeting bersama mentor Individu : masih pada tahap pengembangan API Elastic PDDikti Search All.
65.	12 Nov 2021	Bersama Mitra: Magang Onsite (WFO), Standup Meeting, Koordinasi Tim BE bersama Tim PM terkait kendala pekerjaan Individu : tahap pengembangan API Elastic PDDikti Search All sekaligus penambahan API Contract Modul PDDikti dan push ke github.
66.	15 Nov 2021	Bersama Mitra : Koordinasi Magang Merdeka. Individu : Bersama mentor mempelajari dan memahami software architecture DDD (Domain Driven Design).
67.	16 Nov 2021	Bersama Mitra : Koordinasi Magang Merdeka Individu : Perbaikan script sql pada beberapa modul
68.	17 Nov 2021	Bersama Mitra : Start Sprint & Sprint Review Individu : Perbaikan script sql pada beberapa modul dan close db connection di beberapa modul.
69.	18 Nov 2021	Bersama Mitra : Standup Meeting, Training Backend terkait materi Goroutine, Weekly meeting bersama mentor Individu : Riset terhadap response data yang dibutuhkan oleh client pada service API Kedaireka pada bagian Kreasi Reka sekaligus pada tahap pengembangan API nya.
70.	19 Nov 2021	Bersama Mitra : Standup Meeting Individu : Membuat API Contract modul Kedaireka, melanjutkan perbaikan API pada modul PDDikti.
71.	22 Nov 2021	Bersama Mitra : Stand Up Meeting, Koordinasi Magang Merdeka terkait start and review sprint. Individu : Perbaikan parameter pada API Rasio Dosen/Mahasiswa.
72.	23 Nov 2021	Bersama Mitra : Stand Up Meeting. Individu : Review kembali response data yang dibutuhkan oleh client pada service API Elastic PDDikti dan API Store PDDikti.
73.	24 Nov 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting. Individu : Mempelajari dan memahami kembali materi Golang Goroutine yang diberikan oleh trainer.
74.	25 Nov 2021	Bersama Mitra : Stand Up Meeting. Individu : Review kembali response data yang dibutuhkan oleh client pada service API Elastic PDDikti dan API Store PDDikti.
75.	26 Nov 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting, Koordinasi magang bersama tim PM dan MD. Individu : Review kembali response data yang dibutuhkan oleh client pada service API Elastic PDDikti dan API Store PDDikti.
76.	29 Nov 2021	Bersama Mitra : Magang Onsite (WFO), Standup Meeting, Koordinasi Magang Merdeka. Individu : Mereview dan memperbaiki kembali API Spesifik Search Mahasiswa.

77.	30 Nov 2021	Bersama Mitra :Standup Meeting Individu : Screening script yang dibuat sebelumnya dan memperbaikinya pada modul FAQ.
78.	1 Des 2021	Bersama Mitra : Magang Onsite (WFO), Standup Meeting Individu : Melanjutkan screening script yang dibuat sebelumnya dan push ke server untuk modul FAQ.
79.	2 Des 2021	Bersama Mitra :Standup Meeting, Weekly Meeting bersama Mentor Backend, Koordinasi dengan tim FE terkait request Pembuatan API Slug pada modul news/pengumuman. Individu : Screening script yang dibuat sebelumnya, memperbaikinya, dan push ke server untuk modul Ijazah-Ln.
80.	3 Des 2021	Bersama Mitra :Magang Onsite (WFO), Standup Meeting, Koordinasi dengan tim FE terkait request Pembuatan API Slug pada modul news/pengumuman. Individu : Create API Slug pada modul news/pengumuman. Sabtu : Screening script yang dibuat sebelumnya dan memperbaikinya pada modul PDDikti dan Sivil.
81.	6 Des 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting, Koordinasi Magang Merdeka terkait start and review sprint. Individu : Perbaikan pada modul News/Pengumuman.
82.	7 Des 2021	Bersama Mitra : Stand Up Meeting Individu : Create API Get Detail Mahasiswa By Id Reg pada modul PDDikti sekaligus pembuatan API Contractnya, Membantu anggota BE pada task FCM Push Notification Berita dan Pengumuman.
83.	8 Des 2021	Bersama Mitra :Standup Meeting, Weekly Meeting bersama Mentor Backend terkait kendala pekerjaan. Individu : Screening script yang dibuat sebelumnya modul News.
84.	9 Des 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting. Individu : Perbaikan pada modul News dan push ke server. Membantu anggota Backend pada pengerjaan modul notification.
85.	10 Des 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting, Koordinasi bersama tim PM dan MD. Individu : Screening code pada modul PDDikti.
86.	13 Des 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting, Koordinasi Magang Merdeka terkait start and review sprint. Progress Report bersama Project Manager. Individu : Mereview kembali pada modul News terkait kategori berita/pengumuman dan koordinasi bersama mentor Backend terkait bug special char parameter pada echo framework yang dilaporkan oleh anggota Pentest.
87.	14 Des 2021	Bersama Mitra : Stand Up Meeting Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah sudah ready atau belum pada modul PDDikti.
88.	15 Des 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting. Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah sudah ready atau belum pada modul PDDikti.
89.	16 Des 2021	Bersama Mitra : Stand Up Meeting, Koordinasi bersama tim PMs dan MD. Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah sudah ready atau belum pada modul PDDikti.
90.	17 Des 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting, Koordinasi bersama tim PM dan MD. Individu : Membuat API Get Detail Prodi by Id Reg Prodi pada modul PDDikti sekaligus screening bug-bug pada modul pddikti.
91.	20 Des 2021	Bersama Mitra :Stand Up Meeting. Individu : Mempelajari materi yang diberikan oleh mentor yaitu unit testing .

92.	21 Des 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting. Koordinasi bersama semua tim magang terkait persiapan presentasi hasil project kepada bapak Dirjen. Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah terdapat bug atau issue.
93.	22 Des 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting. Melakukan presentasi project kepada bapak Dirjen lewat perwakilan dari tim magang memberikan paparan terkait aplikasi SatuDikti. Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah terdapat bug atau issue.
94.	23 Des 2021	Bersama Mitra : Stand Up Meeting. Koordinasi bersama tim PM. Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah terdapat bug atau issue.
95.	24 Des 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting. Koordinasi bersama tim PM. Individu : Mereview tugas yang diberikan oleh trainer terkait materi unit testing.
96.	27 Des 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting. Koordinasi bersama tim Magang terkait sprint planning refinement. Berdiskusi terkait rencana kedepannya untuk proses development. Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah terdapat bug atau issue.
97.	28 Des 2021	Bersama Mitra : Tidak ada kegiatan. Individu : Perbaikan bug struktur response API pada modul PDDikti.
98.	29 Des 2021	Bersama Mitra :Stand Up Meeting. Koordinasi bersama tim PM. Individu : Penambahan fitur untuk Get Data Tableau PDDikti yang disimpan pada DB.
99.	30 Des 2021	Bersama Mitra : Stand Up Meeting. Koordinasi bersama tim PM. Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah terdapat bug atau issue.
100.	31 Des 2021	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting. Koordinasi bersama tim PM. Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah terdapat bug atau issue. Membantu pengerjaan logic programming tim mobile.
101.	3 Jan 2022	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting. Koordinasi bersama tim PM. Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah terdapat bug atau issue.
102.	4 Jan 2022	Bersama Mitra : Stand Up Meeting. Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah terdapat bug atau issue.
103.	5 Jan 2022	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting. Koordinasi bersama tim PM. Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah terdapat bug atau issue.
104.	6 Jan 2022	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting. Koordinasi bersama tim PM. Individu : Mereview dan mengecek kembali service yang dibutuhkan apakah terdapat bug atau issue.
105.	7 Jan 2022	Bersama Mitra : Magang Onsite (WFO), Stand Up Meeting. Koordinasi bersama tim PM. Individu : Perubahan akses port Backend.

Lampiran D

Nama Penguji	Revisi	Halaman
Affan Mahtarami, S.Kom., M.T.	Penulisan daftar isi mohon diperbaiki.	xi
	Latar belakang belum menjelaskan permasalahan yang dipecahkan merujuk pada rumusan dan tujuan penelitian.	1
	Tujuan umum fokuskan pada project yang dikerjakan.	7
	Laporkan proses pengembangan web servicenya di Bab 3.	19
	Pembelajaran magang tolong diperbaiki.	68
	Kesimpulan dan Saran sesuaikan dengan tujuan penelitian dan pembelajaran magang.	71
	Book-log dilampirkan.	84 (Lampiran C)
Septia Rani, S.T., M.Cs	Perbaiki bagian konten latar belakang.	1