

Tutorial Implementasi REST API pada E-Library dengan Gin Framework Golang, ReactJS, dan Deployment: Panduan Langkah demi Langkah



Golang



React

AHMAD FATHONI R
RONI ANDARSYAH



Tutorial Implementasi REST API pada E-Library dengan Gin Framework Golang, ReactJS, dan Deployment: Panduan Langkah demi Langkah

Ahmad Fathoni R
Roni Andarsyah



Tutorial Implementasi REST API pada E-Library dengan Gin Framework Golang, ReactJS, dan Deployment: Panduan Langkah demi Langkah

Penulis:

Ahmad Fathoni R

Roni Andarsyah

ISBN:

Editor:

Penyunting:

Desain sampul dan Tata letak:

Ahmad Fathoni R

Penerbit:

Penerbit Buku Pedia

Redaksi:

Athena Residence Blok. E No. 1, Desa Ciwaruga,

Kec. Parongpong, Kab. Bandung Barat 40559

Tel. 628-775-2000-300

Email : penerbit@bukupedia.co.id

Distributor:

Informatics Research Center

Jl. Sariasih No. 54

Bandung 40151

Email : irc@ulbi.ac.id

Cetakan Pertama, 2023

Hak Cipta dilindungi undang-undang

Dilarang memperbanyak karya tulis ini dalam bentuk dan

Dengan cara apapun tanpa ijin tertulis dari penerbit

PRAKATA

Representational State Transfer merupakan salah satu metode pendekatan yang populer dalam membangun aplikasi. REST API akan memungkinkan komunikasi antara klien dan server melalui protokol HTTP dan menggunakan HTTP method yang telah ditentukan, aplikasi yang menggunakan REST API dapat berkomunikasi secara terstruktur dan konsisten.

Buku ini hadir sebagai panduan praktis para pengembang perangkat lunak yang tertarik untuk mempelajari cara membangun aplikasi web modern dengan menggunakan teknologi REST API, salah satunya adalah Gin Framework yang menggunakan Bahasa pemrograman Golang. Dalam buku tutorial ini akan membahas bagaimana cara penggunaan REST API dalam konteks pembangunan sebuah E-Library, selain itu anda juga akan dipandu bagaimana cara untuk mengintegrasikan REST API dengan framework JavaScript populer yaitu ReactJS. Penulis berharap buku ini dapat memberikan panduan yang berharga dalam mempelajari dan menguasai penggunaan REST API dalam membangun aplikasi web modern.

Link Github: <https://github.com/ahmadfr20/Penggunaan-Rest-Api-Pada-E-Library-Dengan-Gin-Framework-Golang-dan-ReactJS>

Daftar Isi

PRAKATA	i
Daftar Isi.....	ii
BAB 1 Pendahuluan	5
1.1. Golang.....	7
1.2. <i>Application Programming Interface</i>	7
1.3. ReactJS.....	8
1.4. Website.....	8
1.5. <i>Framework</i>	8
1.6. <i>Microservice</i>	9
1.7. <i>Gin Framework</i>	9
BAB 2 Pengenalan Tools.....	10
2.1. Visual Studio Code.....	10
2.2. Web Browser.....	10
2.3. Postman	10
2.4. PostgreSQL	11
BAB 3 Instalasi Kebutuhan Perangkat.....	12
3.1. Node JS dan React	12
1. Download Node JS.....	12
2. Instalasi Node JS	13
3. Instalasi ReactJS.....	15
3.2. Instalasi Postman	17
3.3. Instalasi PostgreSQL	19
3.4. Instalasi dan Setup Golang.....	26
3.5. Instalasi Gin.....	30
BAB 4 Tahapan Pembuatan Front-end.....	33
4.1. Pembuatan Front-End Buku	33
1. Menyiapkan Struktur Folder dan File.....	33

2.	Menyiapkan <i>dependency</i>	34
3.	Menyiapkan Koneksi API.....	34
4.	Menyiapkan Rute API	35
5.	Membuat Kode Untuk Tampilan Book List	36
6.	Set Up Index dan App JS.....	41
7.	Mencoba Pada Browser	43
8.	Pembuatan Halaman UpdateBook	43
4.2.	Pembuatan Front-End Peminjaman Buku	47
1.	Import dan Pembuatan State	47
2.	Membuat Tampilan dan Assign Function	49
BAB 5 Tahapan Pembuatan Back-end.....		55
5.1.	Perancangan Endpoint Autentifikasi User	55
1.	Koneksi Database	55
2.	Membuat Initializer Environment.....	56
3.	Membuat File Initializer	57
4.	Pembuatan Model	59
5.	Pembuatan Controller User.....	60
BAB 6 Hasil dan Kesimpulan.....		62
6.	Pembuatan Middleware	65
7.	Pembuatan URL Endpoint Autentifikasi	67
8.	Uji Coba Endpoint User	68
5.2.	Perancangan Endpoint Buku	69
1.	Membuat Model Buku	69
2.	Membuat Controller Buku	70
3.	Pembuatan URL Endpoint	74
5.3.	Pembuatan Endpoint Pinjam Buku	75
1.	Membuat Model Pinjam	75
BAB 6 Penggunaan Git dan Deploy Aplikasi		80
6.1	Penggunaan Git SCM	80

6.2	Tahapan Deploy Database	88
6.3	Tahapan CI/CD.....	97
6.4	Tahapan Deploy Backend.....	101
6.5	Tahapan Deploy Frontend.....	105
BAB 7 Kesimpulan.....		109
Daftar Pustaka		110
TENTANG PENULIS.....		111

BAB 1

Pendahuluan

Dalam buku panduan ini, akan membahas bagaimana cara membuat aplikasi menggunakan gaya arsitektur tertentu yaitu REST API dan penggunaan kerangka kerja ReactJS untuk tampilan dari aplikasi. REST API sendiri adalah merupakan arsitektur yang menyediakan jaringan untuk melakukan distribusi antar sistem guna memfasilitasi komunikasi antar sistem, penerapan arsitektur ini akan diterapkan dalam pembuatan manajemen perpustakaan menggunakan website.

Distribusi antar sistem merupakan hal yang penting dalam mengirimkan suatu informasi agar sistem dapat berjalan dengan semestinya. Dengan kemajuan teknologi saat ini, distribusi antar sistem menjadi lebih terstruktur dan efisien karena penggunaan arsitektur API dalam pengiriman informasi. Berbagai macam arsitektur API seperti RPC, SOAP, dan REST banyak digunakan pada oleh banyak sistem sebagai sarana untuk mengirimkan informasi. Dari banyaknya arsitektur tersebut menyediakan fitur-fitur yang berbeda dalam pengiriman data sebagai contoh perbedaan format untuk mengirimkan data, RPC dapat digunakan untuk pemindahan data dengan format khusus XML-RPC dan JSON-RPC, SOAP menggunakan XML sehingga memungkinkan untuk menyimpan data dalam bentuk dokumen, serta REST akan menyimpan data dalam JSON dan memungkinkan untuk membuat aplikasi menjadi ringan karena tidak perlu menyatukan *back-end* dan *front-end* dari sistem.

Layanan REST API dapat menyediakan layanan yang simpel dan sintaks yang mudah untuk dipakai untuk pengaksesan data, penggunaan REST server

dapat diterapkan pada perangkat manapun tanpa mempedulikan bahasa pemrograman yang dipakai. hal tersebut merupakan keuntungan pada saat menggunakan REST API (Serrano & Stroulia, 2017). Penggunaan REST API akan membuat pengembangan aplikasi menjadi lebih mudah karena tidak perlu memikirkan lagi bagaimana kita akan mengakses atau menyimpan data pada basis data karena sudah disediakan sarana distribusi dari REST tersebut. Pada REST API pun terjamin dari segi keamanannya karena dapat menerapkan beberapa sistem autentifikasi seperti OAuth2, JWT, dll. Adapun penggunaan ReactJS sebagai kerangka kerja dari tampilan aplikasi, ReactJS merupakan kerangka kerja atau *framework* yang mendukung dalam pembuatan tampilan aplikasi berbasis website. Penggunaan ReactJS akan mempermudah kita dalam membuat tampilan yang responsif serta integrasi antar API serta penerapan autentifikasi yang diambil dari API itu sendiri.

Dalam media pengarsipan, penggunaan website untuk sarana pengelolaan arsip perpustakaan sudah sering digunakan karena kemudahan dan tenaga yang digunakan menjadi lebih sedikit sehingga dapat meningkatkan efisiensi dalam kerja. Penggunaan website pada perpustakaan biasanya aplikasi berbentuk sistem informasi, karena dalam sistem informasi kita dapat mengelola dan menyimpan berbagai data seperti data keanggotaan perpustakaan, buku-buku yang tersedia, buku yang belum dikembalikan, dll. Oleh karena itu dibutuhkan kinerja sistem yang mendukung, ringan, dan bisa dibuka dalam *platform* manapun.

1.1. Golang

Golang merupakan salah satu bahasa pemrograman yang diciptakan oleh Karyawan Google yaitu Robert Griesemer, Rob Pike, dan Ken Thompson. Tujuan dari dibuatnya golang adalah menciptakan bahasa pemrograman yang ekspresif, cepat, efisien, dan mudah untuk ditulis. Bahasa pemrograman seperti C atau C++ merupakan bahasa pemrograman yang cepat saat dieksekusi namun dalam cara penulisan terlalu sulit untuk diterapkan, adapun bahasa pemrograman yang dalam penulisan cukup singkat dan simpel tetapi dalam segi pembuatan program dan eksekusi yang tidak begitu efisien seperti Java dan Python (McGrath, 2020). Golang adalah salah satu bahasa pemrograman *open-source* atau dapat digunakan oleh siapa saja, bentuk dari bahasa pemrograman ini mempunyai struktur yang mirip dengan bahasa pemrograman C namun dengan penulisan yang lebih singkat dengan tujuan menghindari kompleksitas dan kesulitan dalam penerapan.

Golang mempunyai kelebihan lainnya seperti dapat menggunakan kinerja CPU dengan mengefisienkan kinerja dari bagian multi-core agar mempermudah komputer dalam mengeksekusi banyak *task* secara bersamaan namun tidak memberatkan perangkat itu sendiri.

1.2. Application Programming Interface

Application Program Interface (API) merupakan sebuah service yang menyediakan ketersediaan data yang nanti akan direquest oleh aplikasi client respon dari request tersebut biasanya memiliki format json (Kurniawan, 2020). Bentuk respon dari API dibagi menjadi beberapa bagian yaitu ada GET, POST, PUT, dan Delete. Dari respon tersebut masing-masing memiliki fungsinya, salah satunya GET merupakan perintah untuk melakukan request pengambilan data

dari sebuah server ke client.

1.3. ReactJS

ReactJS merupakan library dari JavaScript open-source yang digunakan pada segi front-end untuk membangun User Interface berdasarkan komponen dari UI yang digunakan. ReactJS diciptakan oleh Facebook dan komunitas developer. React dapat digunakan pada pengembangan sebuah sistem berbasis website, mobile, atau kerangka kerja yang berasal dari server-side (Freeman, 2019). Jadi secara singkat ReactJS merupakan kerangka kerja yang dikhususkan untuk membuat sebuah tampilan website berdasarkan komponen yang ada pada library ReactJS itu sendiri ataupun menggunakan library tambahan untuk mempercantik tampilan pada website.

1.4. Website

Website merupakan kumpulan dari halaman situs, biasanya dirangkum dalam sebuah domain yang bertempat dalam World Wide Web (WWW) di internet. Halaman web biasanya ditulis dalam format HTML, yaitu protokol yang digunakan untuk menampilkan informasi yang diambil dari web server kepada pengguna melalui browser (Pamungkas, 2018).

1.5. Framework

Framework merupakan komponen pemrograman yang berfungsi untuk menyediakan berbagai skrip dalam bahasa pemrograman tertentu. Penggunaan framework akan mempermudah dalam penulisan program dan membantu dalam menjalankan program agar dapat berjalan dengan semestinya (Sari & Wijanarko, 2019).

1.6. *Microservice*

Microservice merupakan gaya arsitektur yang dibangun di atas konsep modularisasi yang dibuat dengan baik, yaitu dengan menggunakan alamat dari sebuah aplikasi yang berbeda. Layanan microservice menawarkan kapabilitas bisnis yang menawarkan akses internal terhadap sebuah data (Mendonca, Jamshidi, Garlan, & Pahl, 2019). Contoh dari microservice sendiri adalah API, karena dalam sebuah API akan menawarkan akses kedalam internal dengan melakukan request dari client menuju server yang dimana akan direspon dengan data.

1.7. *Gin Framework*

Gin merupakan *Framework* yang diprogram dengan bahasa pemrograman Golang. Gin akan memungkinkan Golang untuk berjalan lebih cepat karena penggunaan memori yang cukup ringan dan performa API yang dapat diprediksi. Gin juga mendukung penggunaan Middleware yang membuat penggunaan otorisasi menjadi lebih mudah (Gin, n.d.). Kelebihan dalam menggunakan Gin sendiri adalah mempunyai *Error Management* yang akan memungkinkan untuk mempermudah mencari *Error* pada syntax ataupun pada kode yang tersembunyi yang sulit ditemukan sekalipun.

BAB 2

Pengenalan Tools

2.1. Visual Studio Code

Visual Studio Code atau biasa disebut dengan VSCode adalah aplikasi editor teks yang digunakan untuk membuat atau mengubah teks yang berguna pada pembuatan sistem. VSCode ini dibuat oleh perusahaan besar yaitu Microsoft untuk platform Windows, Linux, dan juga MacOS. VSCode dapat mengenali berbagai jenis bahasa pemrograman seperti Python, Java, C++, Ruby, PHP dan GO. VSCode juga terdapat fitur agar dapat mempermudah menulis serta menjalankan program dalam bentuk ekstensi yang bisa diunduh dalam aplikasinya sendiri (Microsoft, n.d.).

2.2. Web Browser

Web browser merupakan perangkat lunak yang digunakan untuk mengakses informasi yang tersedia dalam sebuah aplikasi web. Informasi yang tersedia biasanya diberikan dalam format teks, gambar, maupun video. Saat ini terdapat banyak Web Browser populer seperti Mozilla Firefox, Google Chrome, Opera, Microsoft Edge, dan Safari. Setiap web browser memiliki kelebihan dan kekurangannya masing masing, web browser bisa digunakan sesuai dengan keinginan preferensi pengguna.

2.3. Postman

Postman adalah aplikasi yang berfungsi sebagai REST *Client* untuk menguji coba API yang telah dibuat dengan mengirimkan HTTP *Request* kedalam

aplikasi tersebut (Edy, Ferdiansyah, Pramusinto, & Waluyo, 2019). Pada Postman terdapat banyak fitur untuk membantu pengembangan REST API seperti pembuatan dokumentasi, publikasi API, dan penggunaan *request* yang beragam seperti GET, POST, PUT, Delete, dll.

2.4. PostgreSQL

PostgreSQL adalah RDBMS yang dapat diakses oleh semua orang karena aplikasinya sendiri merupakan open-source. PostgreSQL menyediakan layanan RDBMS yang stabil yang didukung oleh banyak pengembangan fitur dari komunitas, RDBMS ini sangat diminati oleh banyak pengembang aplikasi web, mobile, dan aplikasi analitik (Makris, Tserpes, Spiliopoulos, & Anagnostopoulos, 2019).

PostgreSQL ini akan digunakan sebagai perangkat lunak untuk menunjang manajemen *database* dalam aplikasi ini agar mempermudah proses pengembangan aplikasi kali ini.

BAB 3

Instalasi Kebutuhan Perangkat

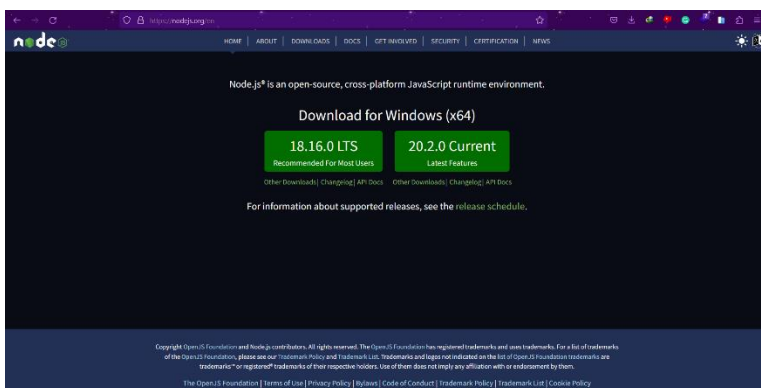
Sebelum memasuki tahap pengembangan aplikasi, sebelumnya ada yang harus anda persiapkan terlebih dahulu. Yaitu instalasi kebutuhan perangkat yang akan digunakan untuk membuat dan menjalankan pada saat mengembangkan aplikasi yang akan dijelaskan pada tahapan berikut.

3.1. Node JS dan React

Pada bagian ini akan dilakukan instalasi terlebih dahulu Node JS agar dapat menjalankan *project-project* yang membutuhkan *dependency* dari Node. Tahapan-tahapan akan dijelaskan dalam berikut:

1. Download Node JS

Instalasi Node JS dapat dilakukan dengan mengakses terlebih dahulu laman situs official Node JS dengan url <https://nodejs.org/en>, Lalu pilih sesuai dengan perangkat yang digunakan.



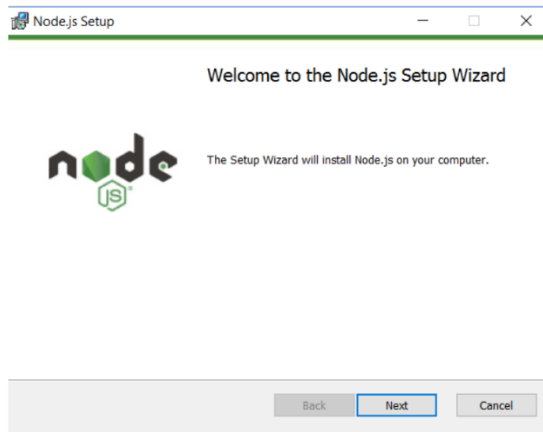
Gambar 1 Halaman Official Node JS

Pada Gambar 1 pilihlah versi 18.16.0 yang merupakan versi yang paling

stabil dan dipilih oleh banyak pengguna tunggu hingga proses unduh selesai.

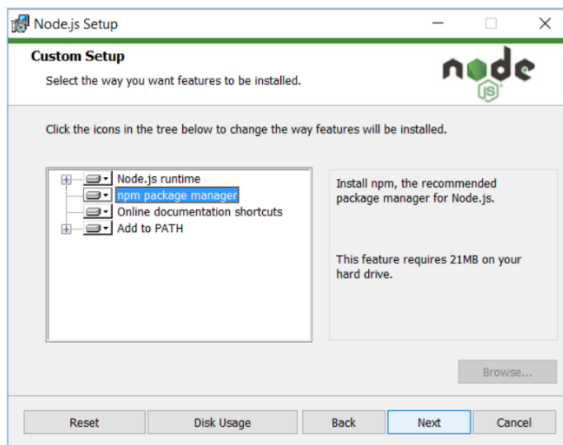
2. Instalasi Node JS

Setelah proses pengunduhan selesai, selanjutnya adalah membuka file yang sudah diunduh pada laman web official dari node.



Gambar 2 Tampilan awal instalasi

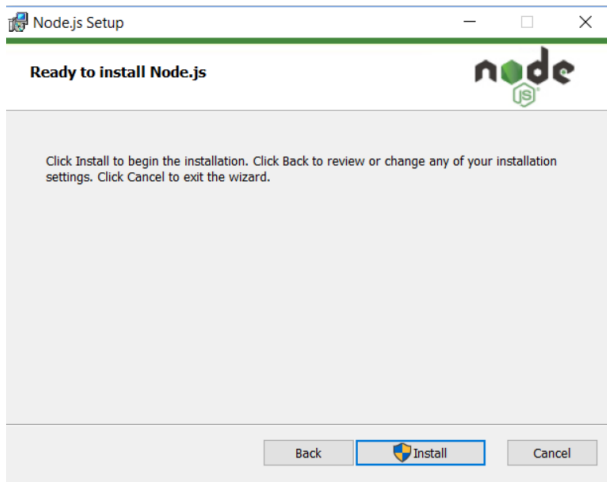
Pada gambar 2 merupakan tampilan awal dari proses instalasi Node JS, disini Anda hanya perlu menekan next pada tampilan.



Gambar 3 Pengaturan Instalasi Node JS

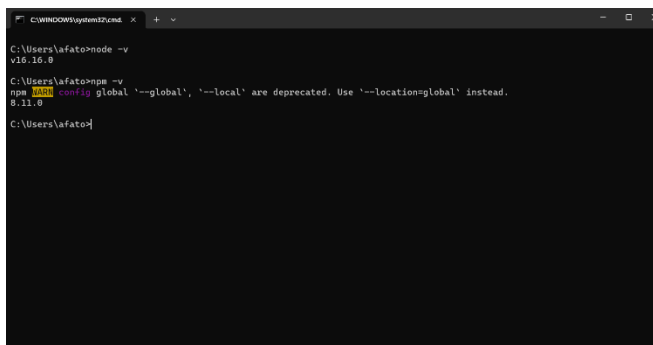
Pada Gambar 3 akan dilakukan pengaturan untuk Node untuk mengatur

PATH dan runtime dari npm agar dapat digunakan untuk mengatur atau menambah dependency dari suatu aplikasi, disini digunakan saja pengaturan secara default dan tekan pada next.



Gambar 4 Tahap Terakhir Instalasi

Pada Gambar 4 adalah tahapan terakhir dari instalasi Node, karena semua pengaturan sudah dilakukan. Tekan pada tombol install apabila muncul pop up administrator setting anda hanya perlu menekan pada tombol “yes” dan instalasi akan berjalan secara otomatis tunggu hingga proses instalasi selesai.



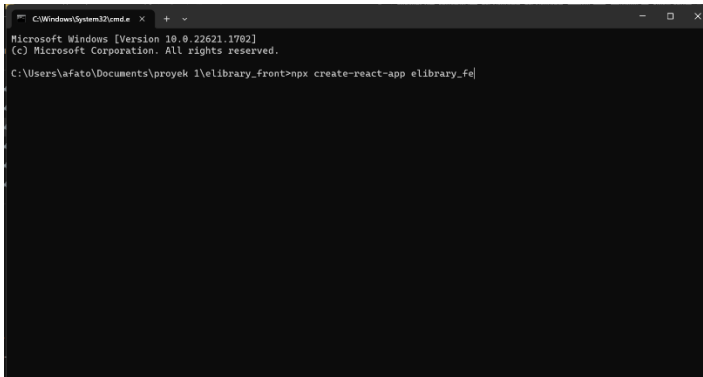
Gambar 5 Pengecekan Instalasi Node

Setelah proses instalasi selesai, selanjutnya adalah melakukan

pengecekan node menggunakan terminal pada perangkat masing-masing dengan cara mengetikkan “node -v” dan “npm -v” apabila output yang dikeluarkan sesuai dengan Gambar 5, maka instalasi node dan NPM sudah berhasil dilakukan.

3. Instalasi ReactJS

Apabila proses instalasi dari Node dan modul NPM sudah terpasang dengan baik, maka proses selanjutnya adalah melakukan persiapan ReactJS agar dapat membuat bagian dari *front-end* aplikasi. Berikut adalah tahapan-tahapannya.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22621.1782]
(c) Microsoft Corporation. All rights reserved.

C:\Users\afato\Documents\proyek 1\elibrary_front>npx create-react-app eLibrary_fe|
```

Gambar 6 Command Prompt React

Pertama buka terminal pada platform anda lalu, ketikkan “npx create-react-app <nama aplikasi>” lalu tekan enter dan tunggu proses pengunduhan ReactJS hingga selesai.

```
C:\Windows\System32\cmd.exe
Inside that directory, you can run several commands:

npm start
  Starts the development server.

npm run build
  Bundles the app into static files for production.

npm test
  Starts the test runner.

npm run eject
  Removes this tool and copies build dependencies, configuration files
  and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

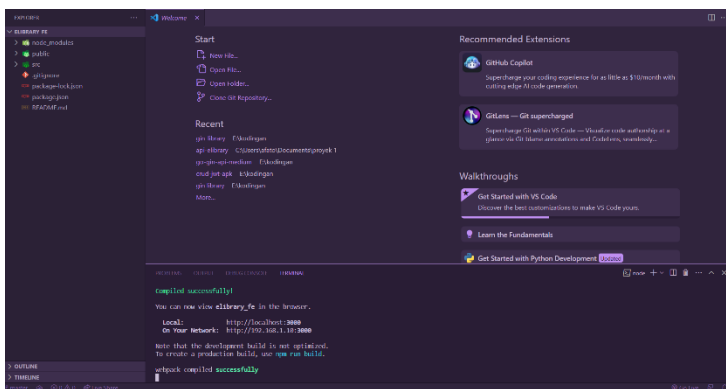
cd elibrary_fe
npm start

Happy hacking!
npm notice New major version of npm available! 8.11.0 -> 9.6.6
npm notice Changelog: https://github.com/npm/cli/releases/tag/v9.6.6
npm notice Run npm install -g npm@9.6.6 to update!
npm notice

C:\Users\afato\Documents\proyek 1\elibrary_front>cd elibrary_fe
C:\Users\afato\Documents\proyek 1\elibrary_front\elibrary_fe>code .\
```

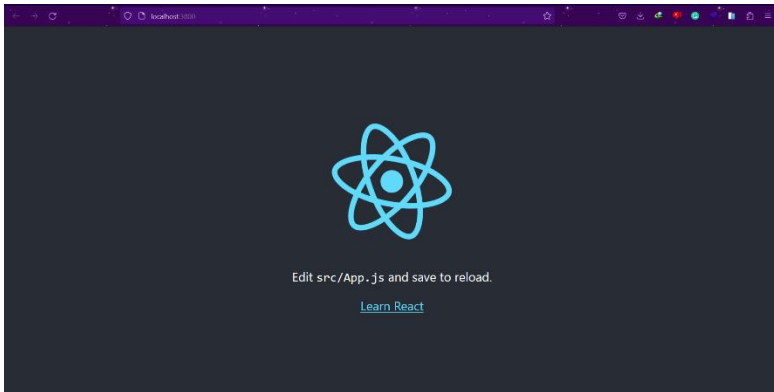
Gambar 7 Command Prompt setelah proses unduh

Setelah proses pengunduhan React telah selesai, selanjutnya masuk ke direktori folder yang terbuat secara otomatis pada saat pemasangan modul React dengan menggunakan command “cd <nama_aplikasi>” lalu tekan enter. Selanjutnya ketikkan “code .” agar dapat membuka text editor Visual Studio Code untuk melakukan perubahan atau penambahan pada kode.



Gambar 8 Tampilan Struktur React JS pada VSCode

Apabila Visual Studio Code telah terbuka, selanjutnya adalah membuka terminal baru pada Visual Studio Code dan mengetikkan perintah berupa “npm start”.

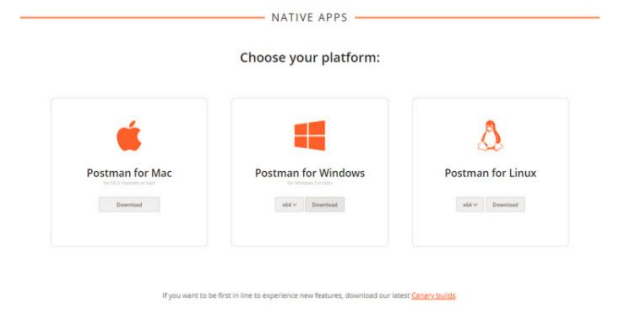


Gambar 9 Tampilan Awal React JS

Setelah mengetikkan “npm start”, browser akan terbuka secara otomatis. Apabila tampilan sudah sesuai dengan Gambar 9 maka proses pemasangan React JS sudah berhasil dan siap untuk digunakan.

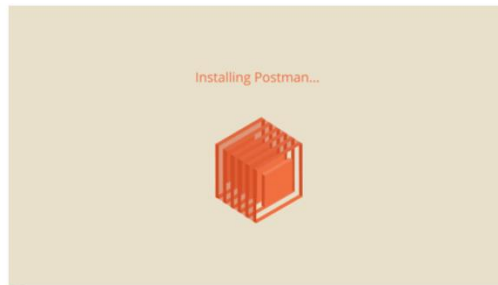
3.2. Instalasi Postman

Postman adalah perangkat lunak untuk melakukan pengujian terhadap suatu API atau pembuatan dokumentasinya. Pada bagian ini akan dilakukan pemasangan terlebih dahulu Postman untuk melakukan pengujian API. Berikut adalah tahapannya.



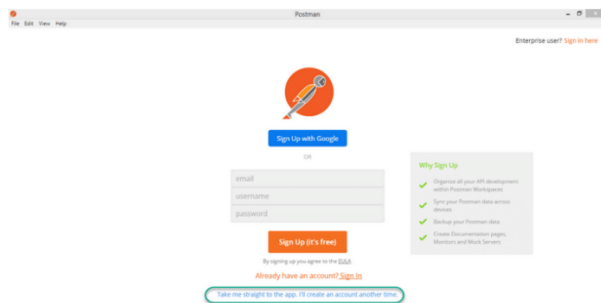
Gambar 10 Halaman Official Postman

Pada Gambar 10 merupakan tampilan *Landing Page* dari web official Postman yang bisa diakses dengan url berikut <https://www.postman.com/downloads/> dan unduh sesuai dengan perangkat yang digunakan.



Gambar 11 Proses Instalasi Postman

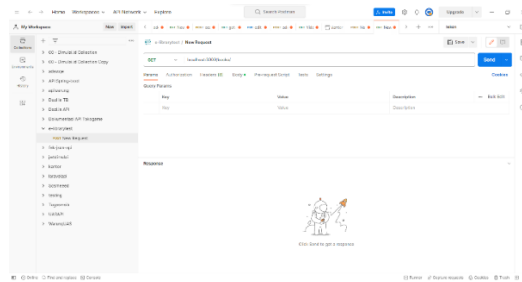
Pada Gambar 11 adalah saat proses instalasi Postman berjalan, instalasi akan otomatis dilakukan dengan pengaturan *default* pada saat membuka file .exe dari instalasi Postman yang sudah diunduh pada tahap sebelumnya.



Gambar 12 Tahap Awal Postman

Setelah instalasi Postman selesai, selanjutnya akan dihadapkan pada halaman *Landing Page* postman. Pada tahap ini anda akan diminta untuk melakukan *Login* terlebih dahulu atau *Sign Up* apabila belum memiliki akun.

Anda bisa untuk melewati tahap tersebut dengan menekan tombol *“Take me straight to the app”* agar dapat langsung menggunakan aplikasi tanpa tahap pendaftaran atau login.



Gambar 13 Halaman Utama Postman

Pada Gambar 13 adalah penampilan dari halaman utama Postman. Halaman ini bisa digunakan untuk melakukan tes terhadap API yang sudah dibuat atau mencoba API publik menggunakan Postman.

3.3. Instalasi PostgreSQL

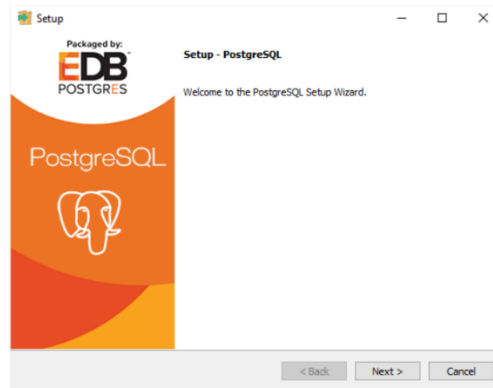
Pada tahapan ini akan dilakukan installasi RDBMS yaitu PostgreSQL agar dapat mengatur data-data yang akan diinputkan dan menunjang berjalannya aplikasi agar berjalan dengan semestinya. Berikut adalah tahapan-tahapannya.

PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
15.3	postgresql.org	postgresql.org			Not supported
14.8	postgresql.org	postgresql.org			Not supported
13.11	postgresql.org	postgresql.org			Not supported
12.15	postgresql.org	postgresql.org			Not supported
11.28	postgresql.org	postgresql.org			Not supported
10.23+					

Gambar 14 Kumpulan Link Download Postgres

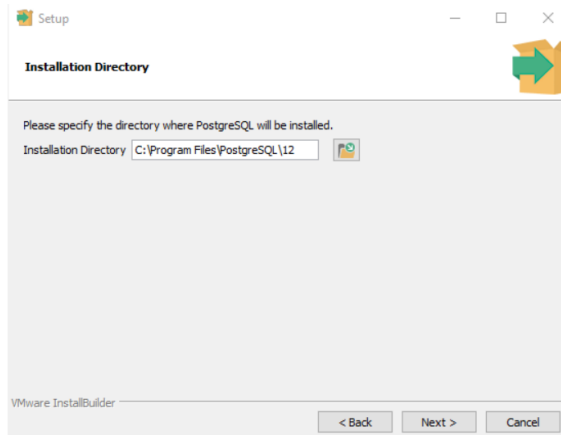
Tahap pertama yang harus dilakukan saat melakukan instalasi Postgres, adalah mengakses dahulu website official dari Postgres dengan menggunakan

link berikut <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>. Pilih sesuai dengan perangkat yang digunakan dan arsitektur dari sistem operasi yang digunakan.



Gambar 15 Tampilan Awal Setup

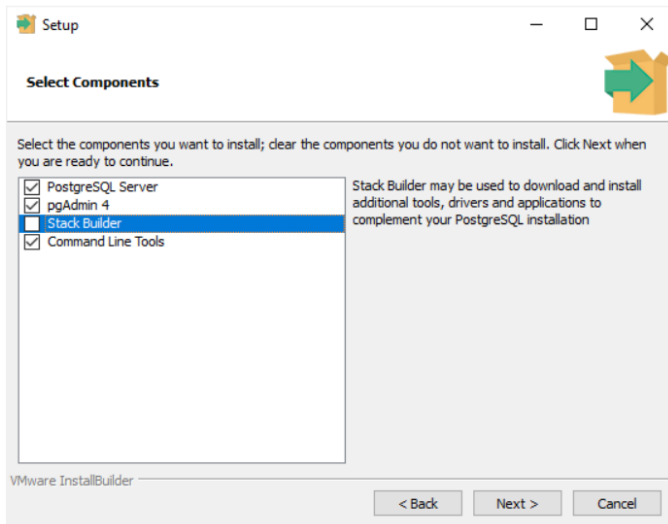
Setelah proses unduh selesai, buka file instalasi yang berformat .exe dan tampilan akan terlihat seperti pada gambar 15. Tekan pada next untuk menuju tahap selanjutnya.



Gambar 16 Pengaturan Lokasi Direktori Postgres

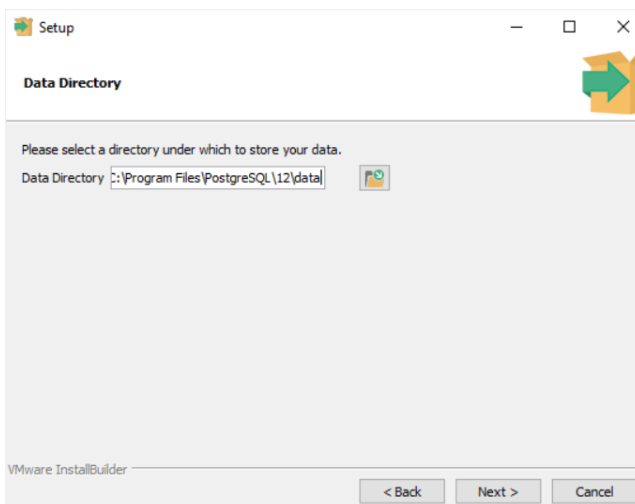
Aturlah direktori dari Postgres sesuai dengan yang diinginkan dengan menekan ikon dari folder. Anda dapat menyimpan direktori dari Postgres

dimanapun.



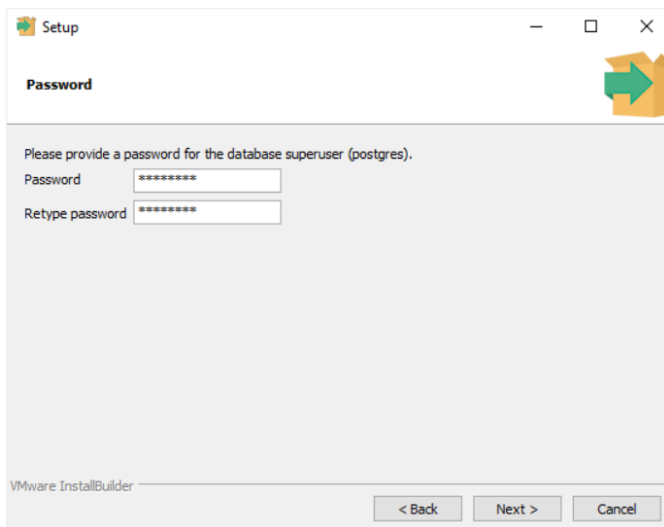
Gambar 17 Pengaturan Komponen Postgres

Pada tahapan ini akan ditampilkan pemilihan komponen seperti pada Gambar 17. Bagian ini dibiarkan saja secara default dan lanjut ke proses selanjutnya dengan menekan tombol next.



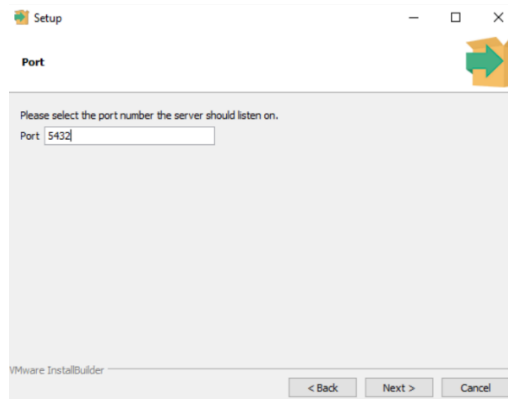
Gambar 18 Pengaturan Direktori Untuk Data

Pada Gambar 18 digunakan untuk mengganti direktori untuk data utama pada Postgres, pengaturan tersebut dapat diganti dengan menekan ikon folder dan menyesuaikannya dengan lokasi yang diinginkan. Apabila sudah menentukan lokasi yang sesuai lanjut dengan menekan tombol next.



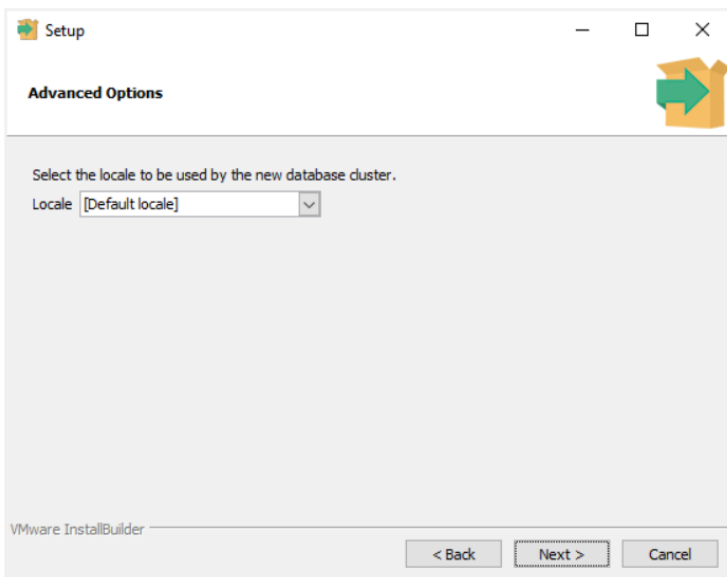
Gambar 19 Pembuatan Password Superuser

Pada Gambar 19 merupakan tahapan pembuatan Password untuk Superuser. Gunakanlah password yang mudah diingat agar tidak mempersulit pada saat membuka aplikasi dari Postgres.



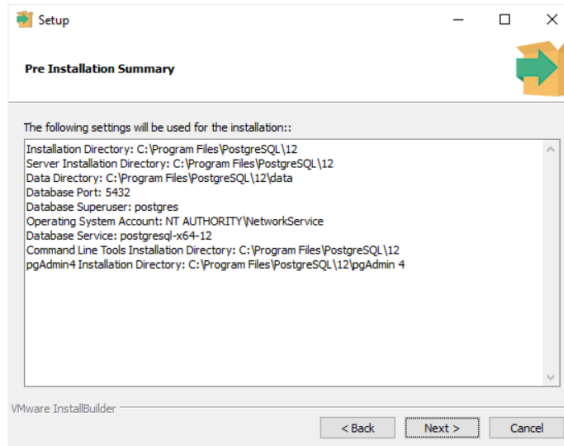
Gambar 20 Pengaturan Port Postgres

Pada tahap ini digunakan untuk mengatur Port pada postgres yang ditampilkan pada Gambar 20. Port diatur sedemikian rupa agar tidak bertabrakan dengan Port lainnya.



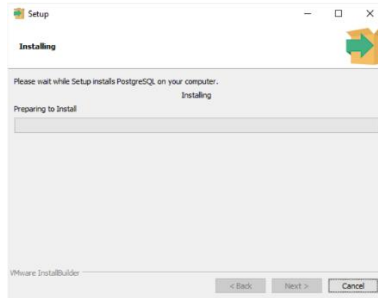
Gambar 21 Pengaturan lokasi cluster postgres

Pada gambar 21 merupakan proses pengaturan cluster database. Pilih pengaturan *default* disini tanpa mengganti apapun.



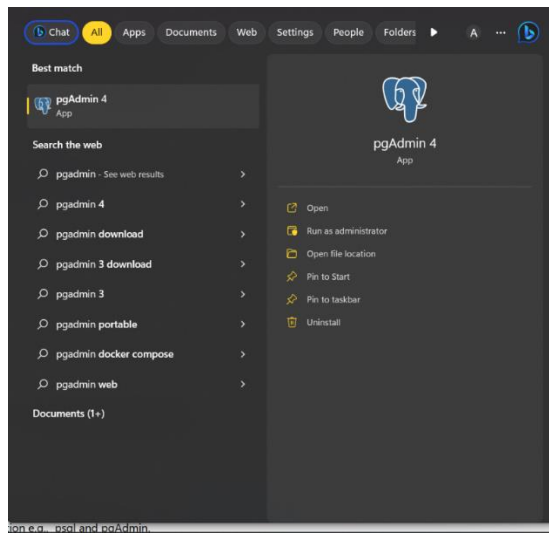
Gambar 22 Peninjauan Komponen Postgres

Pada Gambar 22 menunjukkan halaman instalasi untuk meninjau kembali komponen yang akan dimasukkan pada saat proses installasi. Apabila komponen yang dibutuhkan sudah sesuai dengan keinginan maka tekan tombol next sedangkan apabila belum memenuhi keinginan bisa dikembalikan dengan menggunakan tombol back.



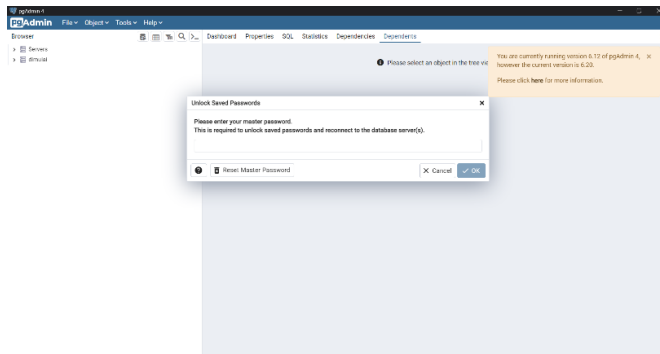
Gambar 23 Proses Instalasi Postgres

Pada Gambar 23 menunjukkan proses instalasi sudah dimulai. Tunggu hingga berakhir sehingga menampilkan tampilan dengan tombol finish untuk tahap akhir.



Gambar 24 pgAdmin pada Start Menu

Selanjutnya untuk memastikan aplikasi terpasang dengan benar, pergi ke Start Menu lalu ketikkan pgAdmin untuk membuka interface dari Postgres itu sendiri.

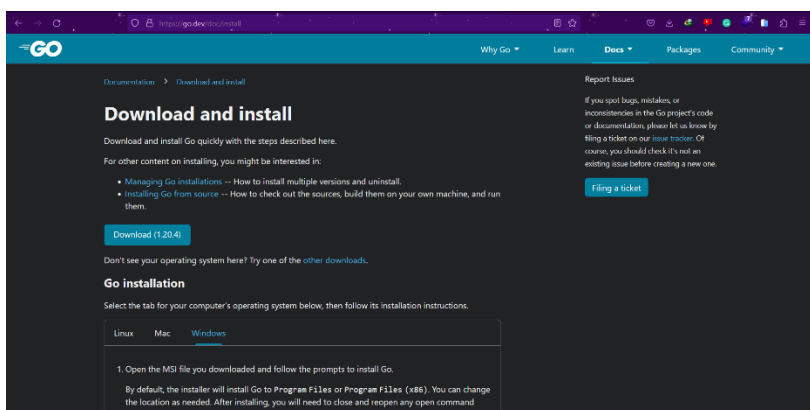


Gambar 25 Tampilan Postgres

Apabila instalasi berjalan tanpa kendala, maka tampilan dari postgres akan terlihat seperti pada Gambar 25. Disini pada saat ingin memasuki aplikasi akan diperintahkan terlebih dahulu untuk menginputkan password yang sudah dibuat pada saat proses instalasi sebelumnya.

3.4. Instalasi dan Setup Golang

Pada tahap ini akan dilakukan instalasi *runtime* untuk Golang agar dapat menjalankan program-program dengan Bahasa Pemrograman Go. Berikut adalah tahapannya.



Gambar 26 Halaman Official Golang

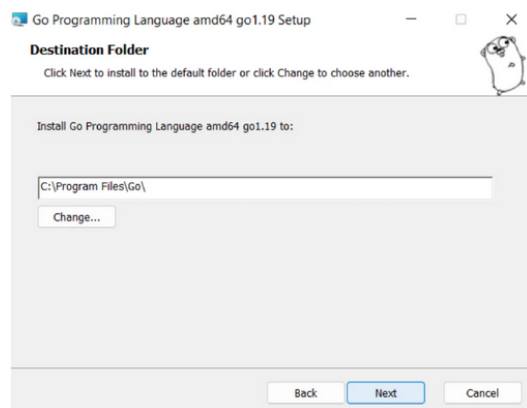
Gambar 26 menunjukkan tampilan pada halaman web resmi Golang

yang dapat diakses pada url <https://go.dev/doc/install> pada perangkat masing-masing. Tekan tombol “Download” untuk mengunduh file instalasi golang.



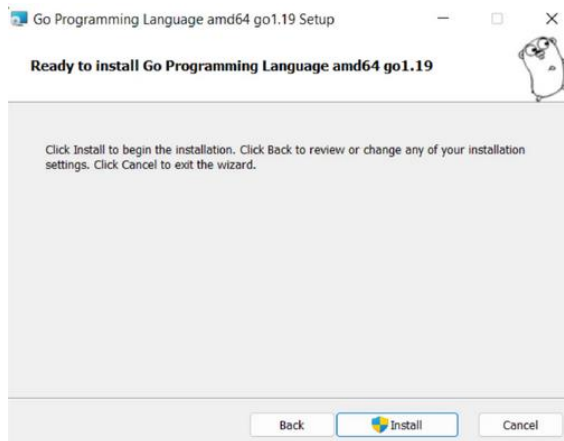
Gambar 27 Set up Golang

Selanjutnya apabila file instalasi selesai diunduh, tekan pada ikon file dan tampilan akan terlihat seperti pada Gambar 27. Klik next untuk melanjutkan ke tahap selanjutnya.



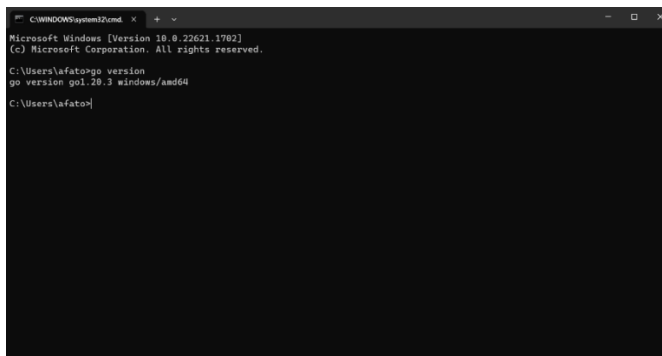
Gambar 28 Pemilihan Direktori Golang

Proses selanjutnya adalah memilih direktori untuk Golang. Pilihlah sesuai dengan direktori yang diinginkan dengan menekan tombol change.



Gambar 29 Tahap Terakhir Instalasi Golang

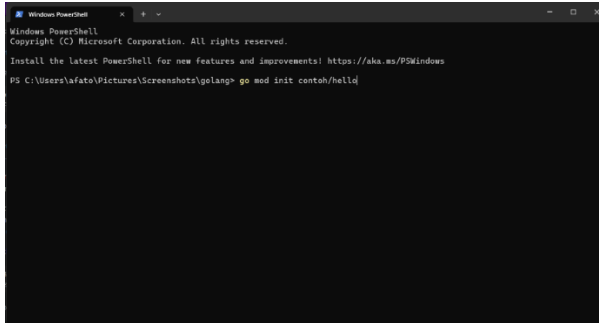
Pada Gambar 29 merupakan tahap terakhir dari instalasi golang. Klik pada tombol install untuk mulai memasang Golang pada perangkat, apabila muncul Pop up administrator klik yes untuk memulai.



Gambar 30 Pengecekan Instalasi Golang

Setelah semua tahapan instalasi selesai, buka CMD untuk mengecek Golang sudah terpasang dengan baik dengan memasukan command “go

version”. Apabila tampilan sudah sesuai dengan Gambar 30 maka instalasi sudah selesai.



Gambar 31 Command untuk Membuat go init

Pada tahap selanjutnya adalah mempersiapkan project go untuk membuat kode-kode dari program yang akan dibuat. Pertama yang harus dilakukan adalah membuka cmd lalu mengetikkan perintah “go mod init <nama_package/nama>” sesuai dengan Gambar 31.



Gambar 32 Struktur Awal File Go

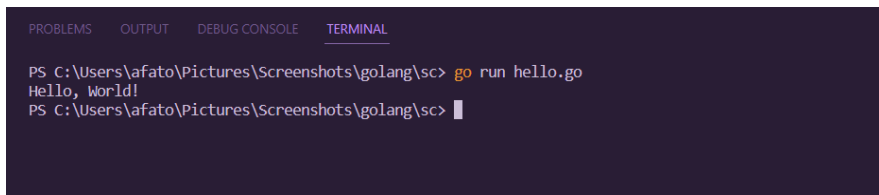
Tambahkan file dengan nama “helo.go” lalu copy code seperti pada kolom 1 dibawah.


```
package main

import "fmt"

func hello() {
    fmt.Println("Hello, World!")
}
```

Kolom 1 Hello World Go

A screenshot of a terminal window within a code editor. The terminal has tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL', with 'TERMINAL' being the active tab. The terminal shows the command 'go run hello.go' being executed in a PowerShell prompt at the path 'C:\Users\afato\Pictures\Screenshots\golang\sc>'. The output of the program is 'Hello, World!' displayed on the next line. The prompt is now 'PS C:\Users\afato\Pictures\Screenshots\golang\sc>' with a cursor.

Gambar 33 Terminal Go

Terakhir adalah melakukan percobaan dengan cara membuka terminal pada Visual Studio Code lalu, mengetikan command “go run <<nama_file.go>>” dan apabila output sudah sesuai dengan Gambar 33 maka, Golang sudah siap digunakan.

3.5. Instalasi Gin

Gin adalah kerangka kerja yang dibutuhkan untuk pembuatan API pada project ini, maka dibutuhkan instalasi Gin terlebih dahulu. Pada proses ini dibutuhkan terlebih dahulu runtime dari Golang yang sudah dijelaskan pada tahap sebelumnya. Berikut adalah cara untuk melakukan instalasi Gin dalam *project* Golang yang sudah dibuat.

```
PS C:\Users\afato\Pictures\Screenshots\golang\sc> go get -u github.com/gin-gonic/gin
go: added github.com/bytedance/sonic v1.8.8
go: added github.com/chenzhuoyi/basexdx v0.0.0-20221115062448-fe3a3abad311
go: added github.com/gin-contrib/sse v0.1.0
go: added github.com/gin-gonic/gin v1.9.0
go: added github.com/go-playground/locales v0.14.1
go: added github.com/go-playground/universal-translator v0.18.1
go: added github.com/go-playground/validator/v10 v10.13.0
go: added github.com/goccy/go-json v0.10.2
go: added github.com/json-iterator/go v1.1.12
go: added github.com/klauspost/cpuid/v2 v2.2.4
go: added github.com/leodido/go-urn v1.2.4
```

Gambar 34 Command Instalasi Gin

Tahap pertama adalah membuka terminal pada Visual Studio Code, lalu ketikkan “go get -u github.com/gin-gonic/gin” dan tekan enter. Tunggu hingga proses selesai.

```
package main

import "github.com/gin-gonic/gin"

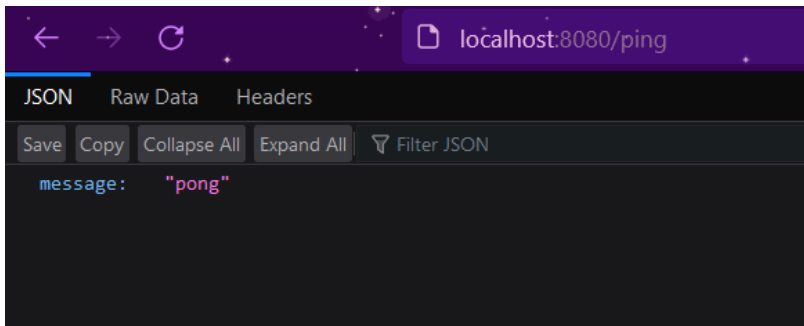
func main() {
    r := gin.Default()
    r.GET("/ping", func(c *gin.Context) {
        c.JSON(200, gin.H{
            "message": "pong",
        })
    })
    r.Run()
}
```

Kolom 2 Penerapan Gin pada code

```
PS C:\Users\afato\Pictures\Screenshots\golang\sc> go run hello.go
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.
[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
- using env: export GIN_MODE=release
- using code: gin.SetMode(gin.ReleaseMode)
[GIN-debug] GET    /ping          --> main.main.func1 (3 handlers)
[GIN-debug] [WARNING] You trusted all proxies, this is NOT safe. We recommend you to set a value.
Please check https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-all-proxies for details.
[GIN-debug] Environment variable PORT is undefined. Using port :8080 by default
[GIN-debug] Listening and serving HTTP on :8080
```

Gambar 35 Running Gin

Ubah kode pada “hello.go” seperti pada kolom 2, lalu running pada terminal seperti pada Gambar 35 dengan perintah “go run hello.go”.



Gambar 36 Test Response Gin pada Browser

Tahap terakhir adalah mencoba membuka URL local yang disediakan pada terminal dalam browser. Apabila output sesuai dengan Gambar 36 maka Gin sudah siap untuk digunakan.

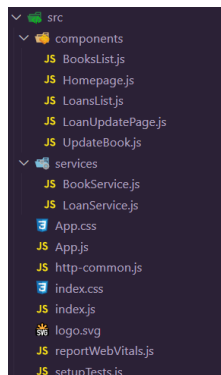
BAB 4

Tahapan Pembuatan Front-end

Front-end merupakan bagian dari aplikasi yang memuat sektor tampilan dari sebuah aplikasi. *Front-end* bisa juga disebut sebagai *client* atau penerima layanan dari *back-end* atau *server-side*, pada bagian ini kumpulan data-data akan ditampilkan berdasarkan perangkat yang digunakan (Liu & Gupta, 2019). Pada bagian ini akan diarahkan bagaimana cara kita untuk membuat bagian dari penampilan aplikasi agar data yang ditampilkan dapat mudah dimengerti dan mempermudah manajemen data melewati integrasi antara *server-side* dan *client*. Berikut adalah tahapan-tahapan yang dilewati untuk membuat *client* pada buku tutorial ini.

4.1. Pembuatan Front-End Buku

1. Menyiapkan Struktur Folder dan File



Gambar 37 Struktur Front-end

Hal yang harus dipersiapkan pertama kali adalah menyiapkan terlebih dahulu Folder dan File dalam project React yang sudah dibuat pada tahap

sebelumnya. Hal ini dilakukan untuk mempermudah pada saat akan memasukan kode agar lebih terurut.

2. Menyiapkan *dependency*

```
npm install axios  
npm install bootstrap  
npm install react-router-dom@5.2.0
```

Kolom 3 Install Dependency

Sebelum memasuki pembuatan kode lakukan instalasi *dependency* seperti pada Kolom 3 terlebih dahulu, agar dapat memenuhi kebutuhan komponen yang dibutuhkan dalam membuat aplikasi. Kegunaan dari *dependency* tersebut adalah yang pertama Axios digunakan untuk berkomunikasi dengan API berdasarkan URL yang diberikan, lalu bootstrap digunakan untuk mempercantik tampilan dari website, dan react-router-dom digunakan untuk memberikan rute terhadap aplikasi kepada modul-modul yang sudah dibuat agar dapat diakses.

3. Menyiapkan Koneksi API

```
import axios from "axios";  
  
export default axios.create({  
  baseURL: "sesuaikan_dengan_alamat_api",  
  headers: {  
    "Content-type": "application/json"  
  }  
});
```

Kolom 4 Persiapan http-common.js

Proses selanjutnya adalah menyiapkan file “http-common.js” seperti pada Kolom 4. *Import* axios dibutuhkan karena akan berkomunikasi dengan API

dengan mengambil BaseURL dan header untuk menentukan tipe header dari output API.

4. Menyiapkan Rute API

```
import http from "../http-common";

const getAll = () => {
  return http.get("/books");
};

const get = (id) => {
  return http.get(`/books/${id}`);
};

const create = (data) => {
  return http.post("/books", data);
};

const update = (id, data) => {
  return http.put(`/books/${id}`, data);
};

const remove = (id) => {
  return http.delete(`/books/${id}`);
};

const removeAll = () => {
  return http.delete(`/books`);
};

const findByTitle = (title) => {
  return http.get(`/books?title=${title}`);
};
```

Kolom 5 Rute Untuk HTTP Method

Pada Kolom 5 merupakan sekumpulan *function* yang digunakan untuk memberikan rute tambahan pada Base URL yang ada pada “http-common.js”. Beberapa rute digunakan untuk mengambil data dengan GET method, PUT method untuk mengubah data, POST method untuk menambahkan data, dan DELETE method untuk menghapus data.

```
const BookService = {
  getAll,
  get,
  create,
  update,
  remove,
  removeAll,
  findByTitle,
};

export default BookService;
```

Kolom 6 constraint BookService

Pada Kolom 6 *constraint* BookService digunakan untuk menampung semua rute yang sudah dibuat dengan mendefinisikannya kembali.

5. Membuat Kode Untuk Tampilan Book List

```
import React, { useState, useEffect } from "react";
import BookService from "../services/BookService";
import { Link } from "react-router-dom";

const BooksList = () => {
  const [books, setBooks] = useState([]);
  const [currentBook, setCurrentBook] = useState(null);
  const [judul, setJudul] = useState("");
  const [penulis, setPenulis] = useState("");
  const [penerbit, setPenerbit] = useState("");
  const [tanggalRilis, setTanggalRilis] = useState("");

  useEffect(() => {
    retrieveBooks();
  }, []);
```

Kolom 7 Import dan State untuk Books

Pada Kolom 7 digunakan untuk *import* kebutuhan komponen yang dibutuhkan untuk melakukan manajemen *state* pada halaman tertentu. *useEffect* digunakan untuk memberikan efek terhadap komponen tertentu jika mengalami aksi tertentu seperti pada saat mengalami klik atau kondisi apapun, dan *useState* untuk manajemen kondisi dalam halaman tertentu.

```

const retrieveBooks = () => {
  BookService.getAll()
    .then((response) => {
      setBooks(response.data);
    })
    .catch((error) => {
      console.log(error);
    });
};

const saveBook = () => {
  const book = {
    Judul: judul,
    Penulis: penulis,
    Penerbit: penerbit,
    Tanggal_rilis: tanggalRilis,
  };

  BookService.create(book)
    .then((response) => {
      console.log(response.data);
      retrieveBooks();
    })
    .catch((error) => {
      console.log(error);
    });
};

const updateBook = () => {
  const updatedBook = {
    id: currentBook.ID,
    Judul: judul,
    Penulis: penulis,
    Penerbit: penerbit,
    Tanggal_rilis: tanggalRilis,
  };

  BookService.update(currentBook.ID, updatedBook)
    .then((response) => {
      console.log(response.data);
      retrieveBooks();
      setCurrentBook(null);
      setJudul("");
      setPenulis("");
      setPenerbit("");
      setTanggalRilis("");
    })
    .catch((error) => {
      console.log(error);
    });
};

```


Pada Kolom 8 Merupakan sekumpulan *function* yang masing-masing digunakan untuk melakukan pengambilan atau manipulasi terhadap data. *Function* “retrieveBook” digunakan untuk mengambil semua data buku yang diambil oleh API, “saveBook” digunakan untuk menyimpan data buku dengan method POST, dan “updateBook” digunakan untuk mengubah data dari buku dengan mengambil ID dari buku tersebut terlebih dahulu dalam *state* management.

```
const handleInputChange = (event) => {
  const { name, value } = event.target;
  if (name === "judul") {
    setJudul(value);
  } else if (name === "penulis") {
    setPenulis(value);
  } else if (name === "penerbit") {
    setPenerbit(value);
  } else if (name === "tanggal_rilis") {
    setTanggalRilis(value);
  }
};
```

Kolom 9 State HandleInputChange

Pada Kolom 9 digunakan untuk manajemen kondisi pada saat kolom tertentu diubah.

```
return (
  <div className="container">
    <div className="row">
      <div className="col-md-6">
        <div className="mb-3">
          <label className="form-label">Judul:</label>
          <input
            type="text"
            className="form-control"
            name="judul"
            value={judul}
            onChange={handleInputChange}
          />
        </div>
      </div>
    </div>
  )
```

```

<div className="mb-3">
  <label className="form-label">Penulis:</label>
  <input
    type="text"
    className="form-control"
    name="penulis"
    value={penulis}
    onChange={handleInputChange}
  />
</div>

<div className="mb-3">
  <label className="form-label">Penerbit:</label>
  <input
    type="text"
    className="form-control"
    name="penerbit"
    value={penerbit}
    onChange={handleInputChange}
  />
</div>

<div className="mb-3">
  <label className="form-label">Tanggal Rilis:</label>
  <input
    type="date"
    className="form-control"
    name="tanggal_rilis"
    value={tanggalRilis}
    onChange={handleInputChange}
  />
</div>

<div className="mb-3">
  <button className="btn btn-primary me-2" onClick={saveBook}>
    Create
  </button>
</div>
</div>
</div>

```

Kolom 10 Form Code

Pada Kolom 10 digunakan untuk membuat tampilan form dari beberapa *field* berdasarkan model dari API. Manajemen *state* digunakan disini dengan melakukan pemasangan function dengan mendefinisikan properti pada *value* dan kondisi *onChange*

parameter dari buku menggunakan *function* “books” dijadikan alias lalu dipanggil parameternya pada masing-masing kolom.

6. Set Up Index dan App JS

```
import React from "react";
import ReactDOM from "react-dom";
import { BrowserRouter } from "react-router-dom";

import App from "./App";
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById("root")
);

reportWebVitals();
```

Kolom 12 Pengaturan Index.js

Setelah pembuatan modul untuk tampilan dan integrasi sudah selesai, selanjutnya melakukan pengaturan pada Index JS seperti pada Kolom 12. Tambahkan *import* untuk “BrowserRouter” dan sisipkan pada render dan disimpan dengan “<App />” didalamnya.

```
import React from "react";
import { Switch, Route, Link } from "react-router-dom";
import "bootstrap/dist/css/bootstrap.min.css";
import "@fortawesome/fontawesome-free/css/all.css";
import "@fortawesome/fontawesome-free/js/all.js";
import "./App.css";

import BooksList from "./components/BooksList";
```

Kolom 13 Import Kebutuhan

Lakukan import terlebih dahulu seperti pada Kolom 13 untuk memanggil modul BookList dan memenuhi kebutuhan komponen agar dapat menjalankan modul

tertentu.

```
import UpdateBook from "../components/UpdateBook";
import LoansList from "../components/LoansList";
import LoanUpdatePage from "../components/LoanUpdatePage";
import Homepage from "../components/Homepage";

function App() {
  return (
    <div>
      <nav className="navbar navbar-expand navbar-dark bg-dark">
        <a href="/books" className="navbar-brand">
          Test
        </a>
        <div className="navbar-nav mr-auto">
          <li className="nav-item">
            <Link to={"/homepage"} className="nav-link">
              Home
            </Link>
          </li>
          <li className="nav-item">
            <Link to={"/books"} className="nav-link">
              Books
            </Link>
          </li>
          <li className="nav-item">
            <Link to={"/Loans"} className="nav-link">
              Loans
            </Link>
          </li>
        </div>
      </nav>

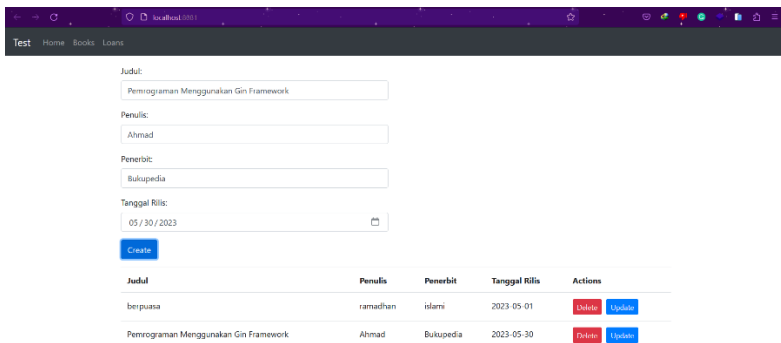
      <div className="container mt-3">
        <Switch>
          <Route exact path={["/", "/books"]} component={BooksList} />
          <Route exact path= "/loans" component={LoansList} />
          <Route exact path="/books/update/:id" component={UpdateBook} />
          <Route exact path="/loans/update/:id" component={LoanUpdatePage} />
        </Switch>
        <Route exact path="/add" component={AddBooks} />
        <Route path="/books/:id" component={Book} />
        <Route path="/homepage" component={Homepage} />
      </div>
    </div>
  );
}

export default App;
```

Kolom 14 Import Modul Lainnya dan Pemberian Rute

Lakukan *import* untuk modul lainnya dan buatlah rute menggunakan tag “react-router-dom” yaitu “Route path” dan masukkan URL pada setiap route, setelah pembuatan rute buat navbar agar dapat melakukan navigasi dengan mudah dalam aplikasi dengan menggunakan class navbar dan nav-item untuk kumpulan link dari rute yang sudah disiapkan.

7. Mencoba Pada Browser



Gambar 38 Tampilan BooksList

Setelah kode tampilan dibuat buka URL Localhost:8081, maka tampilan akan terlihat seperti pada Gambar 38 yaitu halaman manajemen buku yang dibuat dalam file “BooksList.js”.

8. Pembuatan Halaman UpdateBook

```
import React, { useState, useEffect } from "react";
import { useParams } from "react-router-dom";
import BookService from "../services/BookService";

const UpdateBook = () => {
  const { id } = useParams();
  const [judul, setJudul] = useState("");
  const [penulis, setPenulis] = useState("");
  const [penerbit, setPenerbit] = useState("");
  const [tanggalRilis, setTanggalRilis] = useState("");
```

Kolom 15 Import Dependency dan Pembuatan State

Pada Kolom 15 merupakan *import* untuk *dependency* yang dibutuhkan untuk menjalankan modul “UpdateBook.js”.

```
const retrieveBook = () => {
  BookService.get(id)
    .then((response) => {
      const { Judul, Penulis, Penerbit, Tanggal_rilis } = response.data;
      setJudul(Judul);
      setPenulis(Penulis);
      setPenerbit(Penerbit);
      setTanggalRilis(Tanggal_rilis);
    })
    .catch((error) => {
      console.log(error);
    });
};

const updateBook = () => {
  const updatedBook = {
    id: id,
    Judul: judul,
    Penulis: penulis,
    Penerbit: penerbit,
    Tanggal_rilis: tanggalRilis,
  };

  BookService.update(id, updatedBook)
    .then((response) => {
      console.log(response.data);
      // Redirect to the book list page after successful update
      window.Location.href = "/books";
    })
    .catch((error) => {
      console.log(error);
    });
};

const handleInputChange = (event) => {
  const { name, value } = event.target;
  if (name === "judul") {
    setJudul(value);
  } else if (name === "penulis") {
    setPenulis(value);
  } else if (name === "penerbit") {
    setPenerbit(value);
  } else if (name === "tanggal_rilis") {
    setTanggalRilis(value);
  }
};
```

Kolom 16 Pembuatan Function State

Selanjutnya pada Kolom 16 merupakan pembuatan *function* untuk

mendeklarasikan state dan menampung buku berdasarkan ID menggunakan *function* “retrieveBook”. Lalu pembuatan *function* untuk menampung *field* pada buku, dan terakhir ada *handler* untuk perubahan pada input yang didefinisikan dengan “handleInputChange”.

```
return (
  <div className="container">
    <h1>Update Book</h1>
    <div className="mb-3">
      <label className="form-label">Judul:</label>
      <input
        type="text"
        className="form-control"
        name="judul"
        value={judul}
        onChange={handleInputChange}
      />
    </div>

    <div className="mb-3">
      <label className="form-label">Penulis:</label>
      <input
        type="text"
        className="form-control"
        name="penulis"
        value={penulis}
        onChange={handleInputChange}
      />
    </div>

    <div className="mb-3">
      <label className="form-label">Penerbit:</label>
      <input
        type="text"
        className="form-control"
        name="penerbit"
        value={penerbit}
        onChange={handleInputChange}
      />
    </div>
  </div>
)
```



```

<div className="mb-3">
  <label className="form-label">Tanggal Rilis:</label>
  <input
    type="text"
    className="form-control"
    name="tanggal_rilis"
    value={tanggalRilis}
    onChange={handleInputChange}
  />
</div>

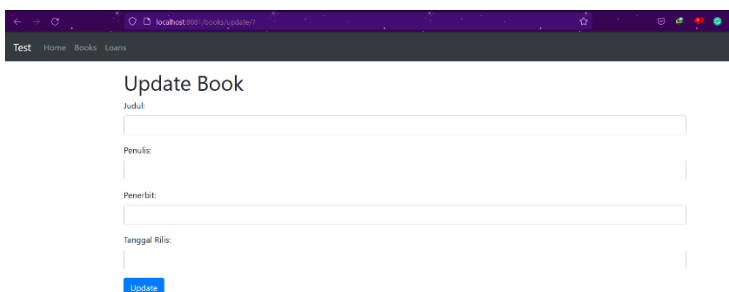
<div>
  <button className="btn btn-primary" onClick={updateBook}>
    Update
  </button>
</div>
</div>
);
};

export default UpdateBook;

```

Kolom 17 Code Interface UpdateBook

Pada Kolom 17 merupakan kode untuk tampilan Update Book dengan menggunakan bootstrap dan pendeklarasian HandlerInputChange pada tiap kolom dan pemberian function dalam kondisi “onClick” maka akan melakukan update terhadap buku.



The screenshot shows a web browser window with the URL 'localhost:3000/update'. The page title is 'Test'. The main content is a form titled 'Update Book'. The form contains four input fields: 'Judul', 'Penulis', 'Penerbit', and 'Tanggal Rilis'. Below these fields is a blue button labeled 'Update'.

Gambar 39 Tampilan Update Book

Pada Gambar 39 merupakan tampilan dari UpdateBook, yang berfungsi untuk mengubah data buku berdasarkan ID yang diambil dari URL

4.2. Pembuatan Front-End Peminjaman Buku

1. Import dan Pembuatan State

```
import React, { useState, useEffect } from "react";
import LoanService from "../services/LoanService";
import { Link } from "react-router-dom";

const LoansList = () => {
  const [loans, setLoans] = useState([]);
  const [currentLoan, setCurrentLoan] = useState(null);
  const [userID, setUserID] = useState("");
  const [bookID, setBookID] = useState("");
  const [tanggalPinjam, setTanggalPinjam] = useState("");
  const [dikembalikan, setDikembalikan] = useState(false);

  useEffect(() => {
    retrieveLoans();
  }, []);

  const retrieveLoans = () => {
    LoanService.getAll()
      .then((response) => {
        setLoans(response.data);
      })
      .catch((error) => {
        console.log(error);
      });
  };
};
```

Kolom 18 Import Untuk LoansList

Import *dependency* terlebih dahulu seperti pada Kolom 18 agar dapat menjalankan program. Lalu buat State dengan menggunakan function `useState` untuk menyimpan kondisi dan `useEffect` untuk memberikan efek dalam memberikan aksi dan function `retrieveLoans` digunakan untuk mengambil data-data pinjaman.

```

const saveLoan = () => {
  const loan = {
    UserID: userID,
    BooksID: bookID,
    Tanggal_pinjam: tanggalPinjam,
    Dikembalikan: dikembalikan
  };

  LoanService.create(loan)
    .then((response) => {
      console.log(response.data);
      retrieveLoans();
    })
    .catch((error) => {
      console.log(error);
    });
};

const deleteLoan = (id) => {
  LoanService.remove(id)
    .then((response) => {
      console.log(response.data);
      retrieveLoans();
    })
    .catch((error) => {
      console.log(error);
    });
};

const removeAllLoans = () => {
  LoanService.removeAll()
    .then((response) => {
      console.log(response.data);
      retrieveLoans();
    })
    .catch((error) => {
      console.log(error);
    });
};

const handleInputChange = (event) => {
  const { name, value } = event.target;
  if (name === "userID") {
    setUserID(parseInt(value));
  } else if (name === "bookID") {
    setBookID(parseInt(value));
  } else if (name === "tanggalPinjam") {
    setTanggalPinjam(value);
  } else if (name === "dikembalikan") {
    setDikembalikan(event.target.checked);
  }
};

```

Kolom 19 Kumpulan Function

2. Membuat Tampilan dan Assign Function

Selanjutnya adalah menambahkan *function* seperti pada Kolom 19, *function* tersebut berfungsi untuk menyimpan buku pada saveLoan dengan mengambil input dari form yang dideklarasikan pada tahapan selanjutnya. Lalu Remove All Loans digunakan untuk menghapus semua data pinjaman buku dan handleInputChange berfungsi untuk merekam kondisi apabila ada perubahan dalam sebuah form.

```
return (  
  <div className="container">  
    <div className="row">  
      <div className="col-md-6">  
        <div className="mb-3">  
          <label className="form-label">User ID:</label>  
          <input  
            type="number"  
            className="form-control"  
            name="userID"  
            value={userID}  
            onChange={handleInputChange}  
          />  
        </div>  
  
        <div className="mb-3">  
          <label className="form-label">Book ID:</label>  
          <input  
            type="number"  
            className="form-control"  
            name="bookID"  
            value={bookID}  
            onChange={handleInputChange}  
          />  
        </div>  
  
        <div className="mb-3">  
          <label className="form-label">Tanggal Pinjam:</label>  
          <input  
            type="date"  
            className="form-control"  
            name="tanggalPinjam"  
            value={tanggalPinjam}  
            onChange={handleInputChange}  
          />  
        </div>  
      </div>  
    </div>  
  )
```

```
export default LoansList;
```

Terakhir tambahkan bagian struktur HTML untuk menampilkan data-data dari pinjaman. Data akan ditampilkan dalam bentuk tabel dan kondisi akan disimpan dalam masing-masing label, untuk tampilan dari Update pinjaman akan memiliki halaman tersendiri yang akan dijelaskan dalam tahap selanjutnya.

```
import React, { useState, useEffect } from "react";
import LoanService from "../services/LoanService";

const LoanUpdatePage = (props) => {
  const [loan, setLoan] = useState(null);
  const [userID, setUserID] = useState("");
  const [bookID, setBookID] = useState("");
  const [tanggalPinjam, setTanggalPinjam] = useState("");
  const [dikembalikan, setDikembalikan] = useState(false);

  useEffect(() => {
    retrieveLoan();
  }, []);

  const retrieveLoan = () => {
    const loanId = props.match.params.id;
    LoanService.get(loanId)
      .then((response) => {
        setLoan(response.data);
        setUserID(response.data.UserID);
        setBookID(response.data.BooksID);
        setTanggalPinjam(response.data.Tanggal_pinjam);
        setDikembalikan(response.data.Dikembalikan);
      })
      .catch((error) => {
        console.log(error);
      });
  };

  const updateLoan = () => {
    const updatedLoan = {
      id: loan.id,
      UserID: userID,
      BooksID: bookID,
      Tanggal_pinjam: tanggalPinjam,
      Dikembalikan: dikembalikan,
    };

    LoanService.update(loan.id, updatedLoan)
      .then((response) => {
        console.log(response.data);
        props.history.push("/loans");
      })
      .catch((error) => {
        console.log(error);
      });
  };
};
```

Kolom 21 Pendeklarasian Function Tambah Pinjaman

Tahap selanjutnya adalah pembuatan tampilan untuk UpdateLoans yang diawali dengan import dan pembuatan function untuk menyimpan *state* dari halaman dan pendeklarasian *field* pada tiap *function*.

```
const handleInputChange = (event) => {
  const { name, value } = event.target;
  if (name === "userID") {
    setUserID(value);
  } else if (name === "bookID") {
    setBookID(value);
  } else if (name === "tanggalPinjam") {
    setTanggalPinjam(value);
  } else if (name === "dikembalikan") {
    setDikembalikan(event.target.checked);
  }
};

return (
  <div className="container">
    <h1>Update Loan</h1>
    {loan ? (
      <div>
        <div className="mb-3">
          <label className="form-label">User ID:</label>
          <input
            type="text"
            className="form-control"
            name="userID"
            value={userID}
            onChange={handleInputChange}
          />
        </div>
        <div className="mb-3">
          <label className="form-label">Book ID:</label>
          <input
            type="text"
            className="form-control"
            name="bookID"
            value={bookID}
            onChange={handleInputChange}
          />
        </div>
        <div className="mb-3">
          <label className="form-label">Tanggal Pinjam:</label>
          <input
            type="date"
            className="form-control"
            name="tanggalPinjam"
            value={tanggalPinjam}
            onChange={handleInputChange}
          />
        </div>
      </div>
    )}
  </div>
);
```

[illegible]

Kolom 22 Pembuatan Interface UpdateLoan

Tahap terakhir dalam pembuatan *Interface* seperti pada Kolom 22 untuk halaman UpdateLoan data akan ditampilkan sama seperti pada halaman Books, namun pada halaman Loans yang ditampilkan adalah data pinjaman berdasarkan *field* yang tersedia pada API. Pembuatan sisa *function* yaitu untuk menampung aksi yang dibuat oleh pengguna dibuat disini dalam *function* `handlerInputChange` yang berguna untuk merekam perubahan pada form agar data dapat diinputkan kedalam database. Lalu *function* akan di-*assign* kedalam tag HTML yang terdapat pada setiap struktur form masing-masing.

User ID:

Book ID:

Tanggal Pinjam:

☐ Dikembalikan

[Create](#)

User ID	Book ID	Tanggal Pinjam	Dikembalikan	Actions
1	7	2023-05-01	Yes	Delete Update

Gambar 40 Tampilan Tambah Pinjaman

Pada Gambar 40 merupakan tampilan dari halaman peminjaman buku yang dapat diakses dengan menekan tombol Loans pada Navbar atau pada URL “localhost:8081/loans”. Field UserID dan BookID akan ditampilkan dalam bentuk tabel dan dalam tabel akan ada tombol aksi yang digunakan untuk menghapus data pinjaman buku.

Update Loan

User ID:

Book ID:

Tanggal Pinjam:

☒ Dikembalikan

[Update](#)

Gambar 41 Update Loan

Gambar 41 adalah tampilan dari UpdateLoan yang digunakan untuk mengubah data pinjaman. Halaman ini dapat diakses dengan menekan tombol “update” dalam halaman list pinjaman dan pinjaman akan ditampilkan berdasarkan id yang dimuat dalam URL.

BAB 5

Tahapan Pembuatan Back-end

Pada BAB ini akan membahas tentang perancangan bagian *Back-end* pada aplikasi ini. Hal ini bertujuan agar aplikasi mempunyai alur untuk bagian logikanya dengan cara membuat API. API bisa menghubungkan sistem internal dengan mudah, berkomunikasi antar platform, membuat API akan mempermudah pengembangan aplikasi karena akan melancarkan proses automasi dan pengembangan secara cepat (Iyengar, Khanna, Ramadath, & Stephens, 2017). Berikut adalah tahapan-tahapan dalam membuat API untuk E-Library.

5.1. Perancangan Endpoint Autentifikasi User

1. Koneksi Database

```
"gorm.io/driver/postgres"  
"gorm.io/gorm"
```

Gambar 42 Instalasi Gorm

Hal yang pertama kali dilakukan adalah melakukan koneksi database dengan memasang *dependency* terlebih dahulu. Pemasangan *dependency* dapat dilakukan dengan cara mengikuti intruksi seperti pada Gambar 37 pada terminal di VSCode yang terpasang pada project, hasil dari pemasangan *dependency* bisa dilihat pada file yang bernama "go.mod".

```

42  gorm.io/driver/postgres v1.5.2 // indirect
43  gorm.io/gorm v1.25.1 // indirect
44  )
45

```

Gambar 43 Hasil Instalasi Gorm

Pada Gambar 38 merupakan keterangan Gorm sudah terpasang pada *dependency list* dalam file yang bernama “go.mod”. Gorm merupakan *library* yang mengimplementasikan teknik dan menyediakan layer yang berbasis *object-oriented* diantara *database* dan bahasa pemrograman OOP Go (Tomilov, 2020).

2. Membuat Initializer Environment

Tahapan selanjutnya adalah membuat initializer yang berfungsi untuk membuat koneksi antara API dan database. Hal tersebut dapat dilakukan dengan cara berikut.



Gambar 44 File .env

Buat file baru yang bernama .env yang digunakan untuk menampung *Environment Variable*. Hal tersebut berfungsi untuk menampung alamat dari database, JWT *Secret Key*, dan port API untuk dijalankan.

```

PORT=3000

SECRET=asd112j3ihshu192

DB="host=localhost user=postgres password=ahmad dbname=elibrary_app
port=5432 sslmode=disable"

```

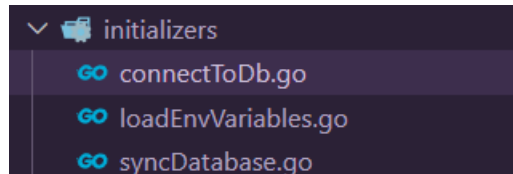
Kolom 23 Isi .env

Lalu masukkan *source code* yang ada pada Kolom 3 yang berisi *PORT*, *Secret Key*, dan *DB Address*. Isi dari port dapat disesuaikan dengan pengaturan pada device

Anda.

3. Membuat File Initializer

Proses selanjutnya adalah membuat file dari *initializer* itu sendiri. Disini initializer terdapat tiga file seperti pada Gambar 40 yaitu “connectToDB.go”, “loadEnvVariables.go”, dan “syncDatabase.go”. File tersebut masing-masing mempunyai kegunaanya yaitu “connectToDB” berfungsi untuk membuat koneksi ke database, “loadEnvVariables” digunakan untuk memuat file .env supaya bisa terbaca oleh program, dan terakhir “syncDatabase” digunakan untuk migrasi model ke database agar tabel bisa dibuat secara otomatis.



Gambar 45 File Initializers

Selanjutnya adalah memasukkan *source* code dari setiap file dengan mengikuti kolom berikut.

```
package initializers

import (
    "os"

    "gorm.io/driver/postgres"
    "gorm.io/gorm"
)

var DB *gorm.DB
```

```
func ConnectToDb() {
    var err error
    dsn := os.Getenv("DB")
    DB, err = gorm.Open(postgres.Open(dsn), &gorm.Config{})

    if err != nil {
        panic("failed to connect to database!")
    }
}
```

Kolom 24 ConnectToDb Initializer

Pada kolom 4 adalah source code untuk melakukan koneksi ke database. Pertama akan dilakukan import dahulu library untuk gorm dan gorm/postgres yang berfungsi untuk melakukan komunikasi antar database lalu ada function untuk menampung method dan variabel yang berfungsi untuk memanggil .env agar alamat dapat terdeteksi oleh gorm.

```
package initializers

import (
    "log"

    "github.com/joho/godotenv"
)

func LoadEnvVariables() {
    err := godotenv.Load()

    if err != nil {
        log.Fatal("error loading .env file")
    }
}
```

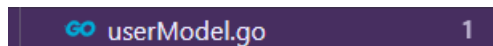
Kolom 25 Source Code LoadEnvVariables

Pada Kolom 5 merupakan *code* yang digunakan untuk mencari variabel yang terkandung dalam file .env. Namun sebelum melakukan akses terhadap

variabel kita harus melakukan instalasi terlebih dahulu untuk mendeteksi file `.env` yang sudah dibuat dengan cara mengetikkan “go install github.com/joho/godotenv”. Apabila sudah melakukan instalasi maka pembuatan *function* bisa dilakukan dan library “godotenv” akan dipanggil dalam function tersebut.

4. Pembuatan Model

Apabila struktur dari initializer sudah dibuat, selanjutnya adalah membuat model dari User. Model ini akan digunakan untuk membuat tabel secara otomatis menggunakan *library* Gorm dari golang. Berikut adalah cara membuat model tersebut.



Gambar 46 File User Model

Buat folder yang bernama Models lalu buat file dengan nama “userModel.go”.

```
package models

import "gorm.io/gorm"

type User struct {
    gorm.Model
    Email    string `gorm:"unique"`
    Password string
    Name     string
}
```

Kolom 26 Model User

Pada Kolom 6 merupakan struktur dari model dari User, pertama akan dilakukan terlebih dahulu *import library* dari gorm yang digunakan membuat

tabel. Selanjutnya ada properti dari model itu sendiri yang diberi nama “User” yang berisi “gorm.Model” yang berguna untuk mencetak kolom id, date created, dan deleted secara otomatis oleh *library* gorm. Lalu ada Email, Password, dan Name yang berfungsi untuk mencetak kolom pada database.

```
package initializers

import "elib_v2/models"

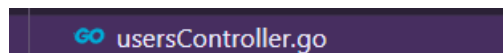
func SyncDatabase() {
    DB.AutoMigrate(&models.User{})
}
```

Kolom 27 Source SyncDatabase

Kembali ke folder initializers, disini akan dibuat file yang bernama “syncDatabase.go” yang digunakan untuk melakukan *automigrate*. *Automigrate* berfungsi untuk membuat tabel secara otomatis kedalam RDBMS berdasarkan model yang telah dibuat. Model User akan dipanggil dengan cara melakukan *import* terlebih dahulu *package* models, lalu model user akan dipanggil dalam function yang bernama *SyncDatabase* seperti pada Kolom 7.

5. Pembuatan Controller User

Setelah tahap pembuatan Initializer dan Model sudah dibuat maka tahapan selanjutnya adalah membuat controller yang berfungsi untuk membuat logika untuk endpoint yang akan dipanggil nantinya.



Gambar 47 File usersController

Buatlah folder yang bernama models, lalu buat file yang bernama “usersController”.

```
package controllers

import (
    "elib_v2/initializers"
    "elib_v2/models"
    "net/http"
    "os"
    "time"

    "github.com/gin-gonic/gin"
    "github.com/golang-jwt/jwt/v4"
    "golang.org/x/crypto/bcrypt"
)
```

Kolom 28 Import Library

Setelah membuat file selanjutnya adalah mengimport *library* seperti pada Kolom 8 yang akan digunakan untuk membuat function dari controller.

```
func Signup(c *gin.Context) {

    var body struct {
        Email    string
        Password string
        Name     string
    }

    if c.Bind(&body) != nil {
        c.JSON(http.StatusBadRequest, gin.H{
            "error": "Failed to read body",
        })

        return
    }
}
```

Kolom 29 Function Signup Pembuatan Property

Lalu membuat variabel *body* yang nantinya akan dipanggil untuk membuat method dari pembuatan user.


```

//hash password disini
hash, err := bcrypt.GenerateFromPassword([]byte(body.Password),
10)

if err != nil {
    c.JSON(http.StatusBadRequest, gin.H{
        "error": "failed to hash password",
    })

    return
}

// membuat user
user := models.User{Email: body.Email, Password: string(hash),
Name: body.Name}
result := initializers.DB.Create(&user)

```

Kolom 30 Hashing dan penampungan body

Selanjutnya adalah pembuatan variabel untuk melakukan enkripsi pada password menggunakan bcrypt seperti pada kolom 10. Juga pembuatan variabel user untuk mendeklarasikan model dari user dan menampung *value* dari model user.

```

if result.Error != nil {
    c.JSON(http.StatusBadRequest, gin.H{
        "error": "Failed to create user",
    })

    return
}

c.JSON(http.StatusOK, gin.H{})
}

```

Kolom 31 Method error untuk signup

Pada Kolom 11 adalah method untuk menunjukkan error apabila request yang dilakukan adalah salah. Apabila request dilakukan dengan benar maka akan direspon dengan respon StatusOK HTTP 200.

```
func Login(c *gin.Context) {
    var body struct {
        Email    string
        Password string
    }

    // failed read body
    if c.Bind(&body) != nil {
        c.JSON(http.StatusBadRequest, gin.H{
            "error": "Failed to read body",
        })

        return
    }
}
```

Kolom 32 Function Login

Selanjutnya setelah membuat function Signup untuk melakukan registrasi, maka tahapan selanjutnya adalah membuat function login. Pertama adalah membuat variabel *constructor* seperti pada Kolom 12. Selanjutnya membuat respon apabila ada error pada saat membaca body dengan menggunakan method if dan http StatusBadRequest.

```
var user models.User
initializers.DB.First(&user, "email = ?", body.Email)

if user.ID == 0 {
    c.JSON(http.StatusBadRequest, gin.H{
        "error": "invalid email or password",
    })

    return
}
```

Kolom 33 Comparator Email

Setelah membuat constructor selanjutnya adalah membuat comparator untuk kolom Email yang terdapat pada Kolom 13. Variabel user akan memanggil

kolom tertentu dengan memanggil initializer, apabila email salah akan direspon dengan output http *Bad Request*.

```
err := bcrypt.CompareHashAndPassword([]byte(user.Password),
[]byte(body.Password))

if err != nil {
    c.JSON(http.StatusBadRequest, gin.H{
        "error": "invalid email or password",
    })

    return
}

//generate token
token := jwt.NewWithClaims(jwt.SigningMethodHS256,
jwt.MapClaims{
    "foo": user.ID,
    "exp": time.Now().Add(time.Hour * 24 * 30).Unix(),
})
if err != nil {
    c.JSON(http.StatusBadRequest, gin.H{
        "error": "generate token gagal!",
    })

    return
}
```

Kolom 34 Komparasi password

Berikutnya adalah melakukan komparasi password yang telah melalui tahap enkripsi seperti pada Kolom 14. Pada variabel “err” akan dipanggil kolom Password yang sudah dienkripsi dan akan dikomparasikan salah atau benarnya. Apabila password salah akan direspon dengan *Bad Request*, sedangkan apabila password benar token akan ter-*generate* dengan batas waktu tertentu, serta apabila proses *generate* token gagal akan direspon dengan *Bad request* dengan pesan “generate token gagal”.

```

    c.SetSameSite(http.SameSiteLaxMode)
    c.SetCookie("Authorization", tokenString, 3600*24*30, "", "",
false, true)

    c.JSON(http.StatusOK, gin.H{})
}

```

Kolom 35 Penyimpanan Token

Terakhir adalah tahap penyimpanan token pada Kolom 15. Token akan dicetak dalam bentuk string yaitu tokenString dan akan kadaluarsa dalam waktu tertentu. Token akan disimpan dalam cookie pada browser atau pada cookie management aplikasi pengujian API seperti Postman.

6. Pembuatan Middleware

Middleware adalah perangkat yang terdapat dalam aplikasi yang mempunyai peran kunci yaitu mengintegrasikan data dengan berbagai perangkat, memungkinkan kita untuk berkomunikasi antar perangkat, serta mengatur keputusan apabila aplikasi tersebut membutuhkan perizinan (Cruz, Rodriguez, Al-Muhtadi, Korotaev, & Alberquerque, 2018). Berikut adalah tahapan yang akan dilewati pada saat membuat middleware.

```

package middleware

import (
    "elib_v2/initializers"
    "elib_v2/models"
    "fmt"
    "net/http"
    "os"
    "time"

    "github.com/gin-gonic/gin"
    "github.com/golang-jwt/jwt/v4"
)

```

Kolom 36 Import Untuk Middleware

Pertama adalah lakukan import terlebih dahulu kebutuhan *library* untuk Middleware. Import ini mencakup *package* dan *library* yang terdiri dari initializers, model, gin, jwt, dll.

```
func RequireAuth(c *gin.Context) {
    //get cookie
    tokenString, err := c.Cookie("Authorization")

    if err != nil {
        c.AbortWithStatus(http.StatusUnauthorized)
    }

    //validasi
    token, err := jwt.Parse(tokenString, func(token *jwt.Token)
(interface{}, error) {
        if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
            return nil, fmt.Errorf("Unexpected signing method: %v",
token.Header["alg"])
        }

        // hmacSampleSecret is a []byte containing your secret, e.g.
[]byte("my_secret_key")
        return []byte(os.Getenv("SECRET")), nil
    })

    if claims, ok := token.Claims.(jwt.MapClaims); ok && token.Valid {
        //cek waktu expire
        if float64(time.Now().Unix()) > claims["exp"].(float64) {
            c.AbortWithStatus(http.StatusUnauthorized)
        }

        //mencari user dengan token
        var user models.User
        initializers.DB.First(&user, claims["sub"])

        if user.ID == 0 {
            c.AbortWithStatus(http.StatusUnauthorized)
        }

        //memasangkan dengan request
        c.Set("user", user)
        //Lanjut
        c.Next()
    } else {
        c.AbortWithStatus(http.StatusUnauthorized)
    }

    c.Next()
}
```

Kolom 37 Function RequireAuth

Function diatas digunakan untuk menampung method-method yang berhubungan dengan autentifikasi user. Pertama adalah variabel tokenString yang digunakan untuk mengambil *cookie* yang di *generate* oleh usersController. Apabila tidak ditemukan cookie maka proses tidak akan dilanjutkan dan akan mengeluarkan output berupa HTTP Response yaitu StatusUnauthorized, sedangkan apabila token ditemukan di *cookie* maka akan dilanjutkan dengan melakukan set user sesuai dengan data dari tabel user tersebut.

7. Pembuatan URL Endpoint Autentifikasi

Proses terakhir dalam pembuatan *Endpoint* untuk autentifikasi pengguna adalah membuat URL dari *controller* yang sudah dibuat. Berikut adalah tahapannya.

```
package main

import (
    "elib_v2/controllers"
    "elib_v2/initializers"
    "elib_v2/middleware"

    "github.com/gin-gonic/gin"
)

func init() {
    initializers.LoadEnvVariables()
    initializers.ConnectToDb()
    initializers.SyncDatabase()
}

func main() {
    r := gin.Default()

    api := r.Group("/api")
    {
        //auth
        api.POST("/signup", controllers.Signup)
        api.POST("/login", controllers.Login)
    }
    r.Run()
}
```

Kolom 38 main.go

Buka kembali file `main.go` yang berada dalam direktori utama API, lalu import semua dependency yang dibutuhkan seperti package-package yang sudah dibuat seperti pada tahap sebelumnya. Setelah mengimport *library* yang dibutuhkan selanjutnya adalah membuat function untuk menampung initializer. Terakhir adalah membuat function untuk rute URL dengan cara membuat variabel “r” untuk rute dan “api” untuk menyatukan api dalam satu rute URL yaitu “/api”.

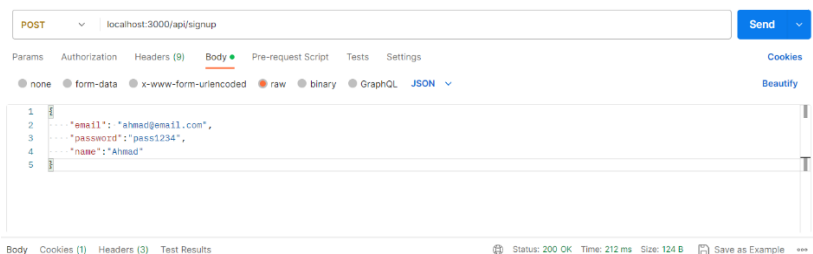
8. Uji Coba Endpoint User

Setelah semua tahapan dilalui selanjutnya adalah melakukan ujicoba terhadap Endpoint yang sudah dibuat. Pertama jalankan terlebih dahulu API dengan cara mengetikkan input “go run main.go” pada terminal dalam text editor atau terminal “cmd”.

```
2023/05/25 06:54:26 stdout: Please check https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-all-proxies for details.  
2023/05/25 06:54:26 stdout: [GIN-debug] Environment variable PORT="3000"  
2023/05/25 06:54:26 stdout: [GIN-debug] Listening and serving HTTP on :3000  
2023/05/25 07:57:52 Running build command!
```

Gambar 48 Status Running API

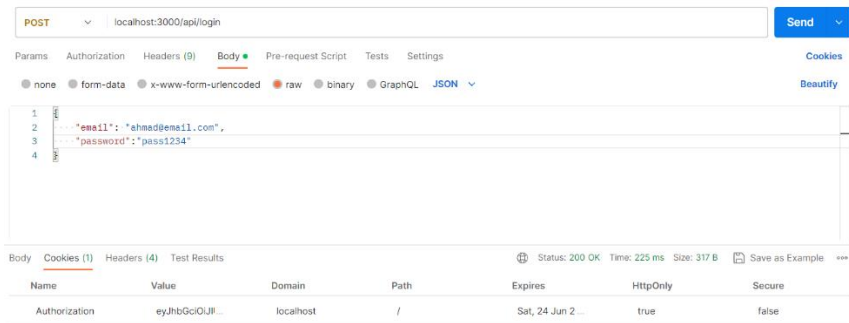
Pada Gambar 43 merupakan keterangan apabila API sudah berjalan dengan menggunakan PORT 3000 dan siap untuk dipakai.



Gambar 49 Register User

Akses URL “localhost:3000/api/signup” untuk melakukan register seperti pada Gambar 44. Apabila pendaftaran berhasil maka respon akan

dikembalikan dengan kode 200 Status OK.

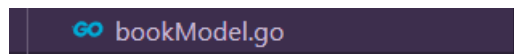


Gambar 50 Login

Setelah proses registrasi berhasil, selanjutnya adalah mencoba untuk Login menggunakan API dengan cara memasukkan input dengan bentuk JSON pada Postman. Apabila User terdaftar maka API akan merespon dengan kode 200 dan token akan langsung disimpan dalam cookie.

5.2. Perancangan Endpoint Buku

1. Membuat Model Buku



Gambar 51 File bookModel

Hal yang pertama harus dilakukan pada saat akan merancang endpoint untuk buku adalah membuat file nya terlebih dahulu. Buat file dengan nama `bookModel.go` seperti pada Gambar 46.

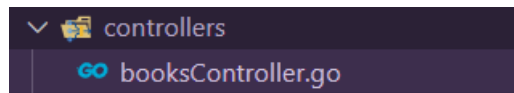

```
package models

type Books struct {
    ID          int `gorm:"primaryKey"`
    Judul       string
    Penulis     string
    Penerbit    string
    Tanggal_rilis string
}
```

Kolom 39 Code Untuk Model Buku

Selanjutnya adalah memasukkan kode seperti pada Kolom 19 pada file “bookModel.go”. Pada kode tersebut dilakukan terlebih dahulu pendeklarasian *package*, lalu dibuat struct untuk membuat struktur tabel sesuai dengan *field* dari buku yang akan diinputkan.

2. Membuat Controller Buku



Gambar 52 Controller Buku

Buatlah file seperti pada Gambar 47 yaitu file “booksController.go”, file ini digunakan untuk menampung berbagai *function* untuk *endpoint* yang akan dibuat nanti. Simpan file pada Controllers agar lebih mudah untuk manajemen filenya.

```

package controllers

import (
    "elib_v2/initializers"
    "elib_v2/models"

    "github.com/gin-gonic/gin"
)

func Books(c *gin.Context) {
    var books []models.Books
    initializers.DB.Find(&books)

    c.JSON(200, books)
}

```

Kolom 40 Code untuk Controller Buku

Selanjutnya adalah memasukan kode pada file “booksController.go” agar dapat membuat komunikasi dengan *database*. Pertama adalah melakukan import terhadap *library* yang dibutuhkan. Lalu membuat *function* pertama untuk mengambil semua data buku seperti pada Kolom 20. Output dari buku akan diambil dengan tipe data *array* agar sejumlah tipe data dapat ditampung dalam satu tempat, apabila pengambilan data berhasil akan direspon dengan HTTP response kode 200 untuk sukses dan output buku akan dikeluarkan.

```

func Viewbookid(c *gin.Context) {
    id := c.Param("id")

    var books []models.Books
    initializers.DB.Find(&books, id)

    c.JSON(200, books)
}

```

Kolom 41 Get Buku By ID

Selanjutnya membuat *function* untuk mengambil buku berdasarkan ID

seperti pada Kolom 21. Kolom ID akan ditampung dengan variabel `id` dan parameter `"id"`. Variabel buku akan diambil dengan bentuk array dan hanya akan mengambil *field* `"id"` nya saja. Memanggil data buku hanya dengan `id` ini dibutuhkan untuk mengambil item secara spesifik untuk kebutuhan tertentu.

```
func AddBooks(c *gin.Context) {
    // ambil data
    var body struct {
        Judul      string
        Penulis    string
        Penerbit    string
        Tanggal_rilis string
    }

    c.Bind(&body)

    //tambah buku
    post := models.Books{Judul: body.Judul, Penulis: body.Penulis,
        Penerbit: body.Penerbit, Tanggal_rilis: body.Tanggal_rilis}
    result := initializers.DB.Create(&post)

    if result.Error != nil {
        c.Status(400)
        return
    }

    c.JSON(200, gin.H{
        "book": post,
    })
}
```

Kolom 42 Function AddBooks

Pada Kolom 22 merupakan function yang berfungsi untuk menambahkan data buku. *Field* buku akan ditampung terlebih dahulu ditampung dan di *bind* untuk mengubah data menjadi respon JSON, lalu field dari buku akan dideklarasikan ulang dalam variabel `post` dan variabel `result` akan memanggil initializer untuk memanggil method POST agar data bisa dimasukkan kedalam tabel, terakhir akan direspon dengan output JSON kode 200 apabila input buku berhasil.

```

func UpdateBook(c *gin.Context) {
    // ambil dulu id buku
    id := c.Param("id")

    // ambil data request
    var body struct {
        Judul      string
        Penulis     string
        Penerbit    string
        Tanggal_rilis string
    }

    c.Bind(&body)

    //ambil post update
    var books models.Books
    initializers.DB.First(&books, id)

    //update
    initializers.DB.Model(&books).Updates(models.Books{
        Judul:      body.Judul,
        Penulis:     body.Penulis,
        Penerbit:    body.Penerbit,
        Tanggal_rilis: body.Tanggal_rilis,
    })

    //respon
    c.JSON(200, gin.H{
        "book": books,
    })
}

```

Kolom 43 Function Update Buku

Pada Kolom 23 adalah *function* yang digunakan untuk melakukan perubahan terhadap data buku. Parameter buku akan dideklarasikan terlebih dahulu pada body, lalu buku akan diambil berdasarkan ID dan dipanggil oleh initialized menggunakan DB.first untuk memanggil *primary key* dari tabel buku tersebut, dan dideklarasikan kembali dalam function model agar dapat melakukan perubahan. Selanjutnya apabila perubahan berhasil akan direspon dengan HTTP response kode 200.

```
func DeleteBook(c *gin.Context) {
    id := c.Param("id")

    initializers.DB.Delete(&models.Books{}, id)

    //respon
    c.JSON(200, gin.H{
        "message": "Buku terhapus!",
    })
}
```

Kolom 44 Function Delete Book

Tahap selanjutnya adalah membuat function untuk menghapus buku seperti pada Kolom 24. Buku akan dipanggil berdasarkan input ID lalu dipanggil dalam function DB.Delete dan diisi oleh ID dari buku itu sendiri. Apabila proses menghapus buku berhasil maka akan direspon dengan HTTP Response dan message “buku terhapus”.

3. Pembuatan URL Endpoint

```
func main() {
    r := gin.Default()
    api := r.Group("/api")
    {
        //crud buku
        api.GET("/books", controllers.Books)
        api.GET("/books/:id", controllers.Viewbookid)
        api.POST("/books", controllers.AddBooks)
        api.PUT("/books/:id", controllers.UpdateBook)
        api.DELETE("/books/:id", controllers.DeleteBook)
    }
    r.Run()
}
```

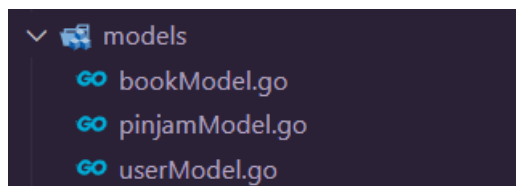
Kolom 45 Pembuatan Endpoint URL

Pada file “main.go” tambahkan kode untuk membuat rute URL *endpoint* agar dapat memanggil *function* yang telah dibuat. URL akan dibuat secara grup dalam “/api” dan sisanya URL spesifik hanya untuk buku. Terdapat beberapa HTTP *method* yang terdapat dalam URL yang pertama ada GET yang digunakan

untuk memanggil data, selanjutnya ada POST untuk menyimpan data, lalu ada PUT untuk mengubah data, dan terakhir ada DELETE untuk menghapus data. Masing-masing URL mempunyai rute tersendiri, URL yang mempunyai “:id” digunakan untuk memanggil data spesifik menggunakan *primary key* dari tabel buku.

5.3. Pembuatan Endpoint Pinjam Buku

1. Membuat Model Pinjam



Gambar 53

Hal pertama yang harus dilakukan adalah membuat model untuk peminjaman buku seperti pada Gambar 48. Model dari pinjaman buku disimpan dalam folder models bersama dengan model-model lainnya.

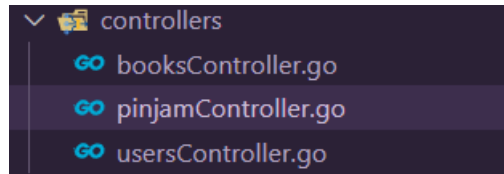
```
package models

type Pinjam struct {
    ID          int `gorm:"primaryKey"`
    UserID      int
    User        User
    BooksID     int
    Books       Books
    Tanggal_pinjam string
    Dikembalikan bool
}
```

Kolom 46 Model pinjam

Pada kolom 26 merupakan struktur dari model pinjam buku. Pada model tersebut terdapat beberapa *field* yang berhubungan dengan beberapa

tabel yaitu UserID dan BooksID yang digunakan untuk mengambil *value* primary key dari masing-masing tabel yaitu tabel User dan Books.



Gambar 54 File Pinjam Controller

Pada Gambar 49 merupakan nama dan lokasi dari file “pinjamController.go” yang disimpan bersamaan dengan model-model lainnya.

```
package controllers

import (
    "elib_v2/initializers"
    "elib_v2/models"

    "github.com/gin-gonic/gin"
)

func Pinjam(c *gin.Context) {
    var pinjam []models.Pinjam
    initializers.DB.Find(&pinjam)

    c.JSON(200, pinjam)
}
```

Kolom 47 Import dan Function Get All Pinjam

Pada Kolom 27 adalah struktur awal dari controller untuk pinjaman buku. Pertama akan dilakukan *import* terlebih dahulu untuk library yang digunakan, lalu pembuatan *function* untuk pengambilan semua data pinjaman buku yang dideklarasikan dalam bentuk *array*. Lalu function tersebut akan memanggil initializer untuk berkomunikasi dengan database, apabila komunikasi berhasil dilakukan akan direspon dengan kode 200 yaitu StatusOK.

```
func GetPinjamid(c *gin.Context) {
    id := c.Param("id")

    var pinjam models.Pinjam
    initializers.DB.Find(&pinjam, id)

    c.JSON(200, pinjam)
}
```

Kolom 48 Get Pinjam By id

Selanjutnya pembuatan *function* untuk mengambil data pinjaman berdasarkan ID seperti pada Kolom 28. Pada *function* ini data akan dipanggil menggunakan initializer dan diambil field id nya saja.

```
func AddPinjam(c *gin.Context) {
    // ambil data
    var body struct {
        UserID      int
        BooksID      int
        Tanggal_pinjam string
        Dikembalikan bool
    }

    c.Bind(&body)

    //tambah buku
    pinjam := models.Pinjam{UserID: body.UserID, BooksID: body.BooksID,
    Tanggal_pinjam: body.Tanggal_pinjam, Dikembalikan: body.Dikembalikan}
    result := initializers.DB.Create(&pinjam)

    if result.Error != nil {
        c.Status(400)
        return
    }

    c.JSON(200, gin.H{
        "pinjaman": pinjam,
    })
}
```

Kolom 49 AddPinjam

Pada Kolom 29 merupakan kolom untuk menambahkan data pinjaman buku pada tabel pinjam. *Field* terisi dengan beberapa value *foreign key* yang diambil dari *primary key* yang terdapat pada tabel User dan Books. Lalu

dideklarasikan pada variabel pinjam dan di-assign dalam array. apabila terdapat error pada saat melakukan input, maka akan menampilkan kode 400 Status Bad Request, sedangkan apabila input berhasil akan menampilkan kode 200 StatusOK.

```
func UpdatePinjam(c *gin.Context) {
    id := c.Param("id")

    // ambil data request
    var body struct {
        UserID      int
        BooksID     int
        Tanggal_pinjam string
        Dikembalikan bool
    }

    c.Bind(&body)

    //ambil post update
    var pinjam models.Pinjam
    initializers.DB.First(&pinjam, id)

    //update
    initializers.DB.Model(&pinjam).Updates(models.Pinjam{
        UserID:      body.UserID,
        BooksID:     body.BooksID,
        Tanggal_pinjam: body.Tanggal_pinjam,
        Dikembalikan: body.Dikembalikan,
    })

    //respon
    c.JSON(200, gin.H{
        "message": pinjam,
    })
}
```

Kolom 50 UpdatePinjam

Pada Kolom 30 merupakan *function* yang digunakan untuk mengubah data peminjaman buku, *field* yang digunakan untuk mengubah data peminjaman buku sama seperti yang digunakan pada penambahan buku yaitu beberapa *foreign key* dll. *Field* id pada tabel pinjam akan dipanggil untuk mengubah data spesifik pada kolom tertentu.

```
func DeletePinjam(c *gin.Context) {
    id := c.Param("id")

    initializers.DB.Delete(&models.Pinjam{}, id)

    //respon
    c.JSON(200, gin.H{
        "message": "Pinjaman terhapus!",
    })
}
```

Kolom 51 Function DeletePinjam

Pada Kolom 31 merupakan *function* yang digunakan untuk menghapus data pinjaman buku. ID pinjaman akan dipanggil terlebih dahulu untuk memasukan data tertentu kemudian apabila proses penghapusan berhasil maka akan direspon dengan kode 200 dengan statusOK dan message dalam format JSON yaitu “pinjaman terhapus!”.

BAB 6

Penggunaan Git dan Deploy Aplikasi

Sesudah tahapan pembuatan Front-end dan Back-end selesai tahap selanjutnya adalah penggunaan GIT dan proses *deploy* aplikasi agar dapat digunakan oleh publik, Berikut adalah tahapan-tahapannya.

6.1 Penggunaan Git SCM

Git SCM adalah sistem kontrol terdistribusi yang dapat digunakan oleh semua orang karena merupakan salah satu sistem open-source. Git biasanya digunakan untuk distribusi aplikasi antar pengguna maupun perusahaan, berikut adalah tata cara dalam menggunakan Git SCM.

1. Git SCM

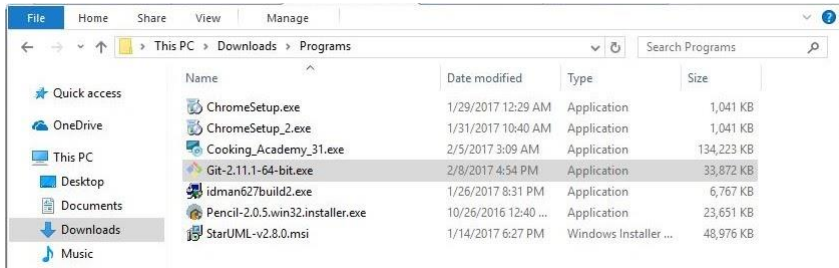
Akses halaman Git SCM official dengan menggunakan url <https://git-scm.com/> lalu tekan tombol “Unduh untuk Windows” seperti pada Gambar 55.



Gambar 55 Halaman Official Git

Apabila proses unduh telah selesai tahapan selanjutnya adalah

melakukan pemasangan terhadap Git tersebut dengan cara membuka file yang telah didownload seperti pada Gambar 56.



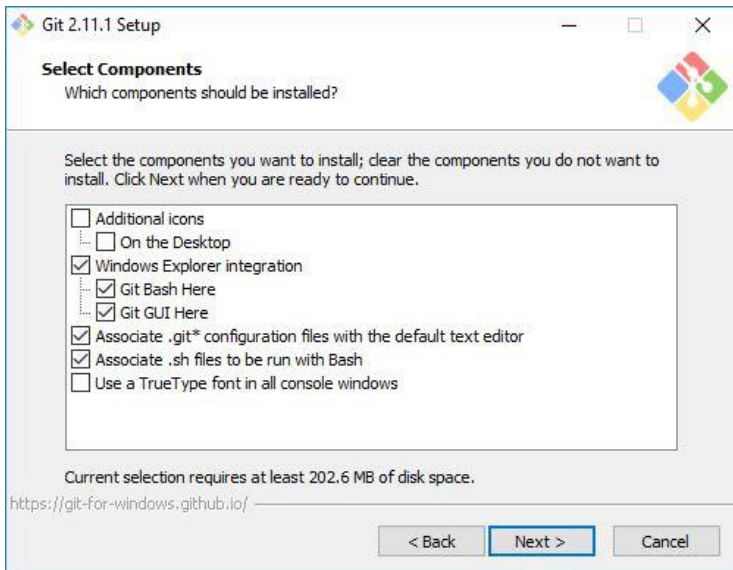
Gambar 56 File Instalasi Git

Setelah File dibuka maka tampilan dari halaman instalasi akan terlihat seperti pada Gambar 57.



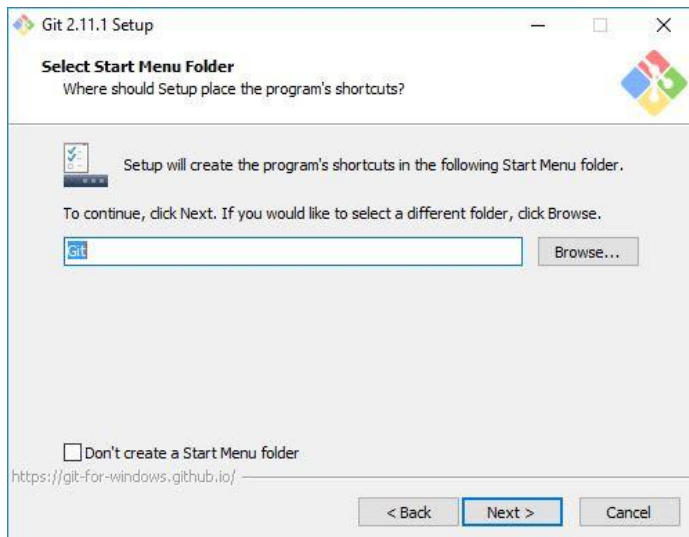
Gambar 57 Halaman Awal Instalasi Git

Tahap selanjutnya adalah memilih kebutuhan apa saja saat menginstall git, disini biarkan saja pengaturan seperti pada Gambar 58.



Gambar 58 Halaman Pemilihan Komponen Git

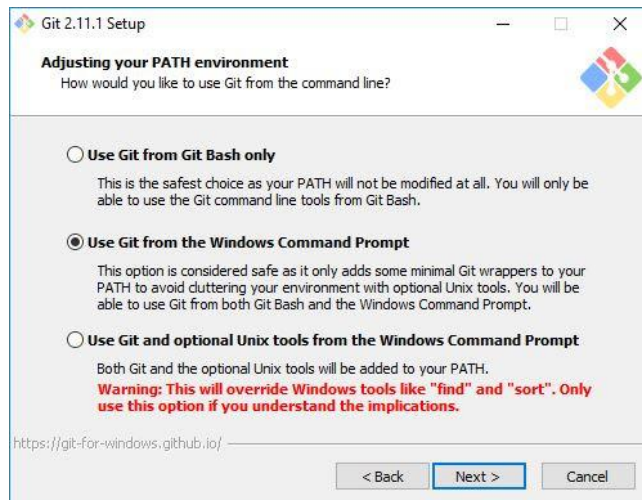
Selanjutnya adalah pemilihan direktori pada git seperti pada Gambar 59, pada tahap ini langsung saja klik next.



Gambar 59 Pemilihan direktori git

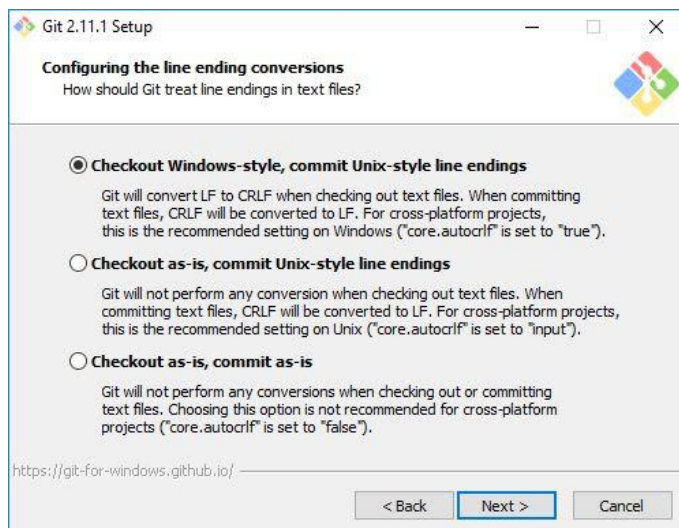
Pada gambar 60 merupakan pemilihan pengaturan untuk PATH dari GIT

tersebut, pilih opsi kedua agar GIT bisa dijalankan dalam command prompt pada Windows.



Gambar 60 Pengaturan PATH untuk GIT

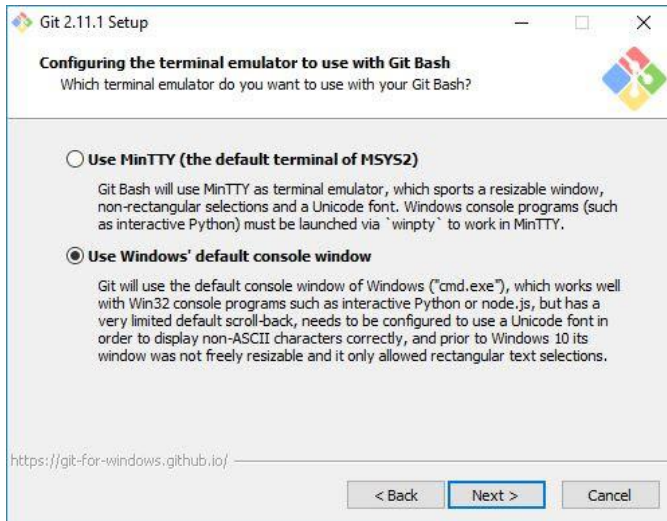
Pada Gambar 61 adalah konfigurasi untuk line ending pada git, biarkan saja default lalu klik next.



Gambar 61 Konfigurasi Line ending

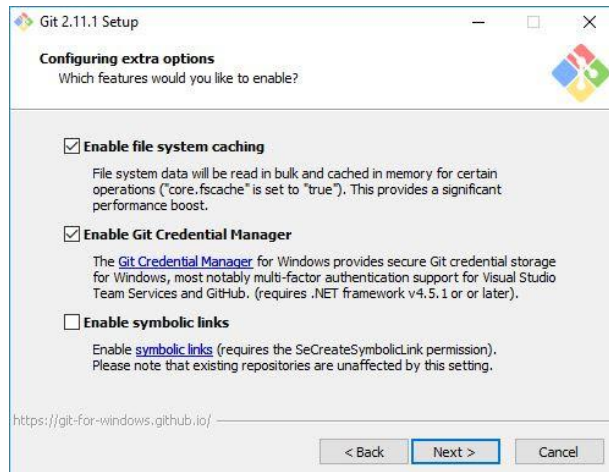
Pada Gambar 62 adalah pengaturan dari emulator terminal pada git

gunakan saja konfigurasi secara *default* lalu klik next.



Gambar 62 Pemilihan terminal emulator git

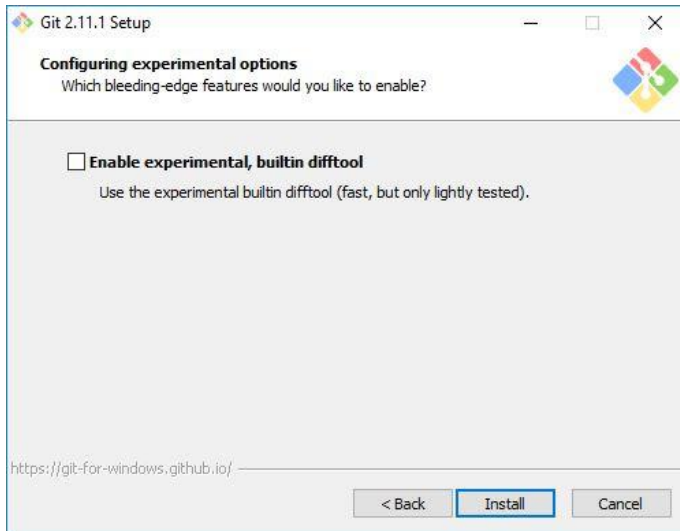
Pada Gambar 63 adalah opsi untuk pengaturan ekstra, disini kita tidak membutuhkan pengaturan ekstra jadi langsung saja untuk menekan tombol next.



Gambar 63 Pemilihan Opsi Ekstra

Pada Gambar 64 merupakan tahap terakhir dari instalasi Git, disini

centang tidak perlu ditekan dan langsung saja tekan tombol install agar instalasi segera dilaksanakan.



Gambar 64 Tahap Akhir

Setelah instalasi selesai maka akan ditampilkan halaman seperti pada Gambar 65.



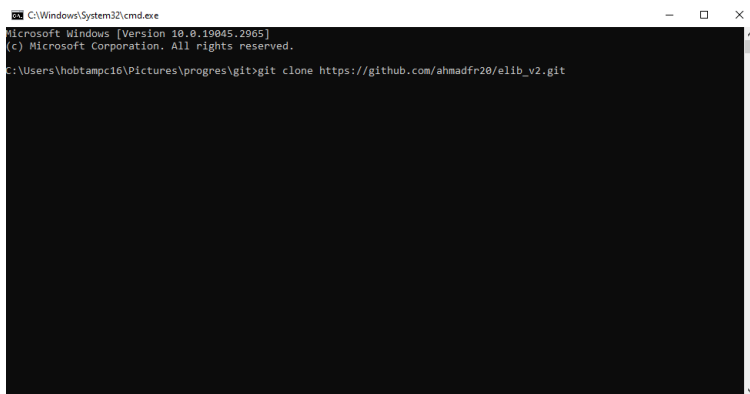
Gambar 65 Tampilan Terakhir Instalasi Git

Selanjutnya melakukan konfigurasi Git pada Git Bash/command prompt dengan cara membuka terminal ataupun git bash, lalu mengetikan command seperti pada Kolom dibawah.

```
git config --global user.name
git config --global user.email
```

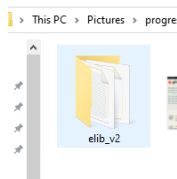
Kolom 52 Konfigurasi Git

Setelah Konfigurasi Git selesai, maka Git SCM sudah dapat digunakan untuk melakukan proses kloning, ataupun upload dokumen dari kode yang sudah dibuat.



Gambar 66 Percobaan Kloning Repository

Pada Gambar 66 adalah percobaan kloning untuk repository pada Backend untuk E-library.



Gambar 67 Hasil clone dari repository

Apabila proses klon berhasil, maka file akan langsung terlihat pada direktori yang diinginkan seperti pada gambar 67.

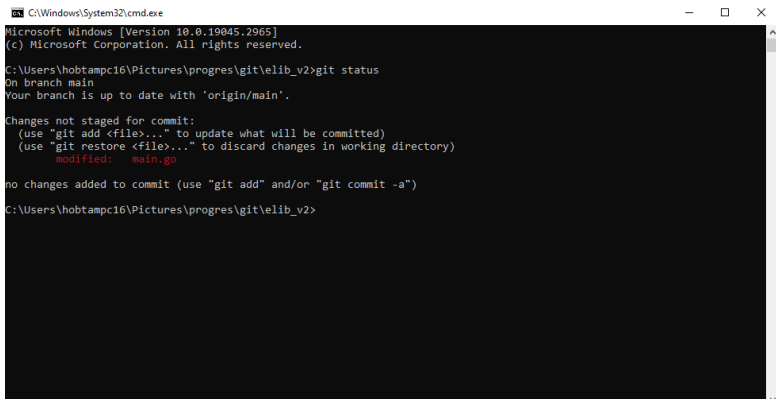
```

49 //crud untuk pinjam
50 api.GET("/pinjam", controllers.Pinjam)
51 api.GET("/pinjam/:id", controllers.GetPinjamid)
52 api.POST("/pinjam", controllers.AddPinjam)
53 api.PUT("/pinjam/:id", controllers.UpdatePinjam)
54 api.DELETE("/pinjam/:id", controllers.DeletePinjam)
55
56 api.GET("/test", controllers.Test)
57

```

Gambar 68 Penambahan Comment

Selanjutnya adalah mencoba untuk melakukan update pada repository dengan menambahkan *comment* pada line 49 pada kode Back-end seperti pada Gambar 68.



```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hobtampc16\Pictures\progres\git\elib_v2>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.go

no changes added to commit (use "git add" and/or "git commit -a")

C:\Users\hobtampc16\Pictures\progres\git\elib_v2>

```

Gambar 69 Pengecekan Status Perubahan

Pada Gambar 69 merupakan cara untuk mengecek status perubahan pada direktori hasil dari clone.

```
C:\Windows\System32\cmd.exe
C:\Users\hobtampc16\Pictures\progres\git\elib_v2>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.go

no changes added to commit (use "git add" and/or "git commit -a")
C:\Users\hobtampc16\Pictures\progres\git\elib_v2>git add .
C:\Users\hobtampc16\Pictures\progres\git\elib_v2>git commit -m "sedikit edit"
[main 2a13ff9] sedikit edit
1 file changed, 1 insertion(+), 1 deletion(-)
C:\Users\hobtampc16\Pictures\progres\git\elib_v2>git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (2/2), 291 bytes | 291.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/ahmadfr20/elib_v2.git
   6a0b55..2a13ff  main -> main
C:\Users\hobtampc16\Pictures\progres\git\elib_v2>
```

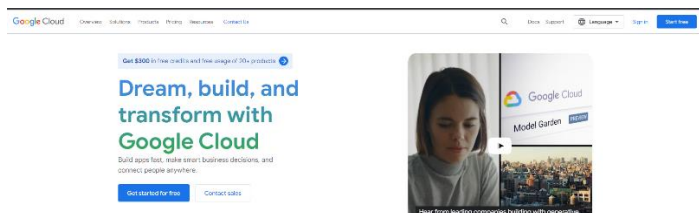
Gambar 70 Update pada Repository

Apabila ada perubahan pada saat pengecekan status perubahan, maka repository siap untuk diupdate seperti pada Gambar 70. Pertama masukkan command “git add .” untuk menginput semua file yang terkena perubahan, lalu ketik command untuk memberikan pesan perubahan apa yang telah diinputkan dengan “git commit -m “pesan””, dan terakhir melakukan push terhadap repository dengan command “git push”.

6.2 Tahapan Deploy Database

1. Google Cloud Platform

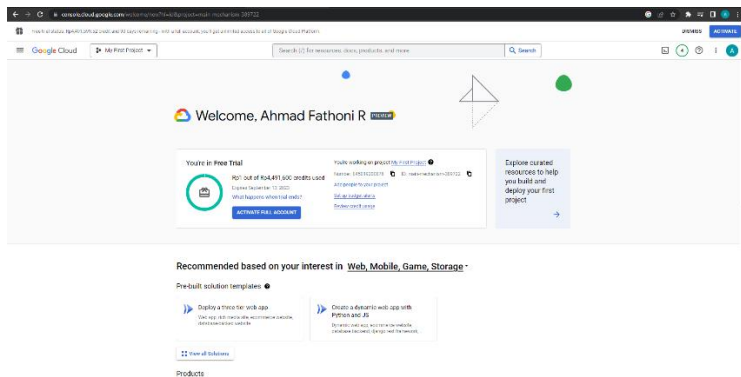
Tahap selanjutnya adalah untuk men-*deploy* database agar dapat diakses pada API saat melakukan publikasi.



Gambar 71 Halaman Google Cloud

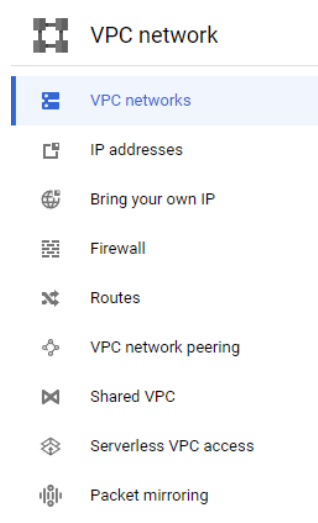
Pertama yang harus dilakukan adalah membuat akun Google Cloud Platform terlebih dahulu dengan mengakses link “cloud.google.com” seperti

pada Gambar 71, hal tersebut bertujuan untuk membuat instansi agar database dapat dipublikasikan klik pada tombol “Get Started For Free”.



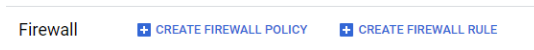
Gambar 72 Halaman Utama GCP

Apabila proses register untuk GCP sudah berhasil maka halaman akan terlihat seperti pada Gambar 72.



Gambar 73 VPC Network

Selanjutnya adalah melakukan konfigurasi firewall untuk *Virtual Machine* GCP dengan mengetikkan VPC pada searchbar. Hasil akan terlihat seperti pada gambar 73, pada halaman ini pilih Firewall.



Gambar 74 Menu Firewall

Pada menu navigasi *Firewall* pilih “CREATE FIREWALL RULE” seperti pada Gambar 74 untuk membuat pengaturan terhadap firewall.

allow-postgres

Description
Allow postgres port 5432

Logs ?
Off
[view in Logs Explorer](#)

Network
default

Priority
1000

Direction
Ingress

Action on match
Allow

Targets

Target tags
allow-tcp-5432

Source filters

IP ranges
0.0.0.0/0

Protocols and ports
tcp:5432

Gambar 75 Pengaturan Firewall

Sesuaikan pengaturan seperti pada gambar 75, dan tag untuk “allow-firewall” akan otomatis terkonfigurasi.

Networking ^

Hostname and network interfaces

Network tags

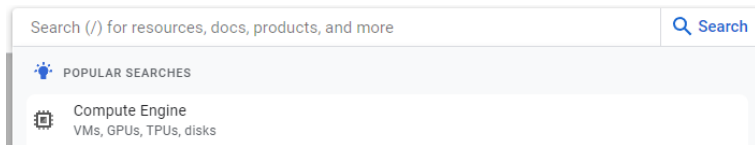
allow-tcp-5432

Hostname

Set a custom hostname for this instance or leave it default. Choice is permanent

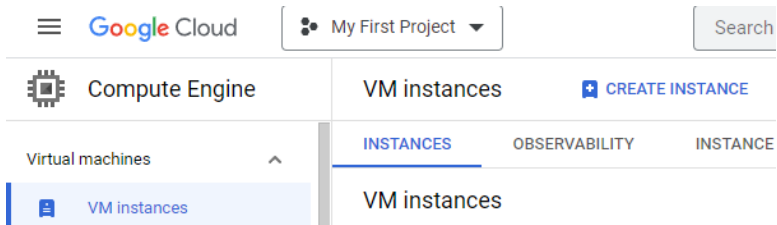
Gambar 76 Hasil Pengaturan Firewall

Pada gambar 76 merupakan hasil dari pengaturan firewall yang dikonfigurasi pada tahap sebelumnya.



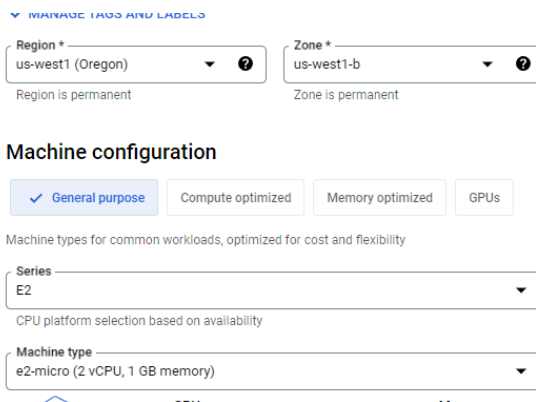
Gambar 77 Pembuatan Compute Engine

Setelah konfigurasi VPC dilakukan tahapan selanjutnya adalah membuat Compute Engine untuk membuat *Virtual Machine* agar dapat memulai deploy database. Compute Engine dapat dicari dengan mengetikkan Compute Engine pada search bar seperti pada Gambar 77.



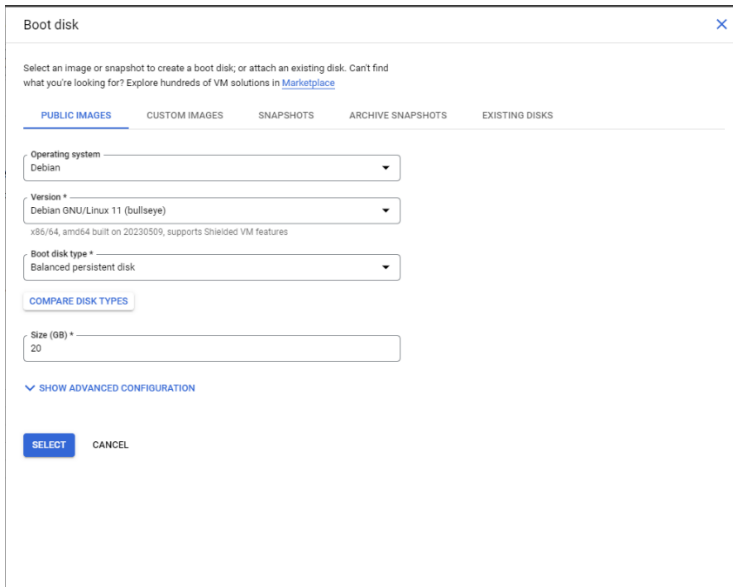
Gambar 78 Pembuatan Instance

Klik pada tombol Create Instance untuk memulai pembuatan *Virtual Machine* seperti pada gambar 78.



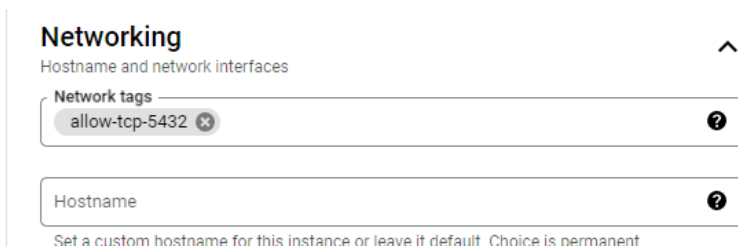
Gambar 79 Pengaturan Instance

Sesuaikan pengaturan seperti pada gambar 79, untuk machine type kita memakai e2-micro agar tarif dari GCP nya tidak menjadi mahal.



Gambar 80 Pengaturan Boot Disk

Pada pengaturan Boot Disk ukuran diubah menjadi 20 GB seperti pada Gambar 80, agar *Virtual Machine* dapat berjalan dengan mulus.



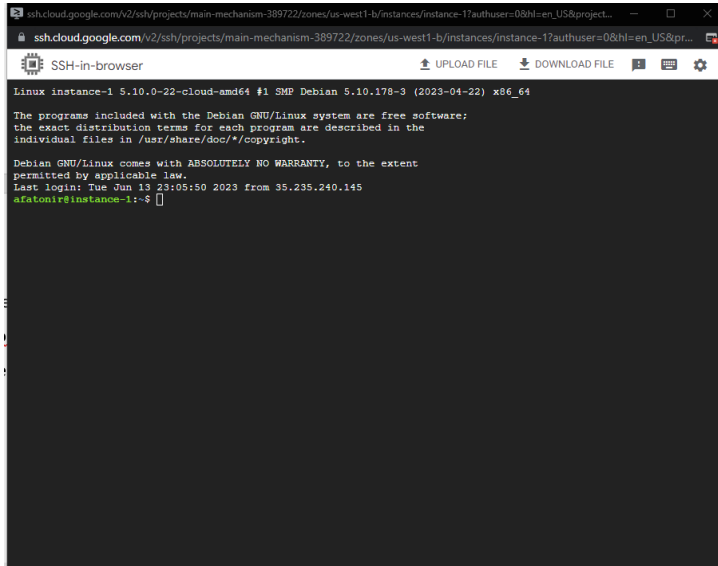
Gambar 81 Pengaturan TCP

Masuk ke Advance Settings lalu pilih networking dan masukkan “allow-tcp-5432” seperti pada Gambar 81.

<input type="checkbox"/>	Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
<input type="checkbox"/>		instance-1	us-west1-b			10.138.0.2 (nic0)	35.233.180.60 (nic0)	SSH

Gambar 82 Hasil Pembuatan VM

Hasil pembuatan VM akan terlihat pada dasbor menu Compute Machine seperti pada gambar 82. Untuk masuk ke tahap selanjutnya tekan tombol SSH pada kolom instance yang sudah dibuat agar dapat membuka console VM.



Gambar 83 Konsol Virtual Machine

Gambar 83 merupakan hasil dari pembuatan instansi dengan tampilan konsol.

```
-sudo apt-get update  
  
-sudo apt-get install ca-certificates curl gnupg  
  
-sudo install -m 0755 -d /etc/apt/keyrings  
  
-curl -fsSL https://download.docker.com/linux/debian/gpg | sudo  
gpg --dearmor -o /etc/apt/keyrings/docker.gpg  
  
-sudo chmod a+r /etc/apt/keyrings/docker.gpg
```



```
-echo \

"deb [arch= "$(dpkg --print-architecture)" signed-
by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian \

"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" |
\

-sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

-sudo apt-get update

-sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin
```

Kolom 53 Konfigurasi Docker Pada VM

Tahapan selanjutnya adalah instalasi docker dan komponen lainnya pada terminal dengan menggunakan command seperti pada Kolom 53. Command dimasukkan secara bertahap tanpa tanda strip.

```
ssh.cloud.google.com: v2/ssh/projects/muan-mechanum-389722/zones/us-west1-b/instances/instance-1?authuser=0&hl=en_US&project=...
SSH-in-browser
UPLOAD FILE
DOWNLOAD FILE

Linux instance-1 5.10.0-22-cloud-amd64 #1 SMP Debian 5.10.178-3 (2023-04-22) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jun 13 23:05:50 2023 from 35.235.240.145
afatonir@instance-1:~$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

afatonir@instance-1:~$
```

Gambar 84 Percobaan Docker

Apabila semua tahap instalasi selesai, selanjutnya untuk mencoba melakukan *running* pertama pada docker dengan mengetikan “sudo docker run hello-world” seperti pada gambar 84. Apabila output sesuai seperti gambar, maka proses instalasi sudah selesai.

```
docker run --name some-postgres -e  
POSTGRES_PASSWORD=mysecretpassword -d postgres
```

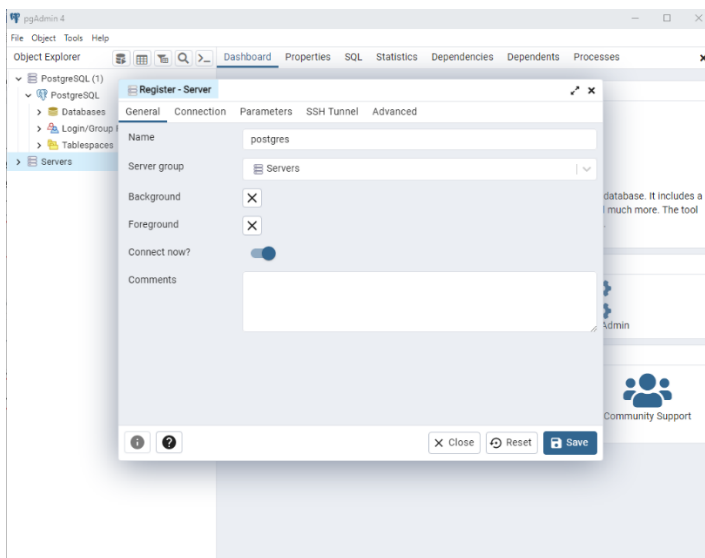
Kolom 54 Konfigurasi Postgres Docker

Tahap selanjutnya adalah konfigurasi postgres pada docker seperti pada kolom 54. Masukkan command pada CLI lalu tekan enter.

<input type="checkbox"/>	Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
<input type="checkbox"/>	●	instance-1	us-west1-b			10.138.0.2 (info)	35.233.180.80 (info)	SSH

Gambar 85 External IP

Salin External IP yang terdapat pada kolom instance dengan menekan tombol yang ada pada samping kanan External IP.



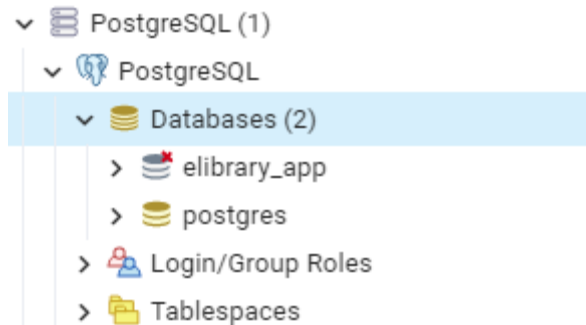
Gambar 86 Konfigurasi pada pgAdmin

Setelah tahap persiapan pada GCP selesai selanjutnya adalah untuk memanggil alamat dari database itu sendiri seperti pada gambar 86. Nama dari server dapat dibuat sesuai keinginan.

Register - Server	
General Connection Parameters SSH Tunnel Advanced	
Host name/address	externalipmasukdisini
Port	5432
Maintenance database	postgres
Username	postgres
Kerberos authentication?	<input type="checkbox"/>
Password
Save password?	<input type="checkbox"/>
Role	
Service	
<input type="button" value="Close"/> <input type="button" value="Reset"/> <input type="button" value="Save"/>	

Gambar 87 Konfigurasi Koneksi

Selanjutnya pengaturan pada koneksi seperti pada Gambar 87. Sesuaikan hostname dengan External IP yang sudah disalin pada proses sebelumnya, dan password dengan yang sudah dibuat pada Virtual Machine.



Gambar 88 Hasil Database GCP

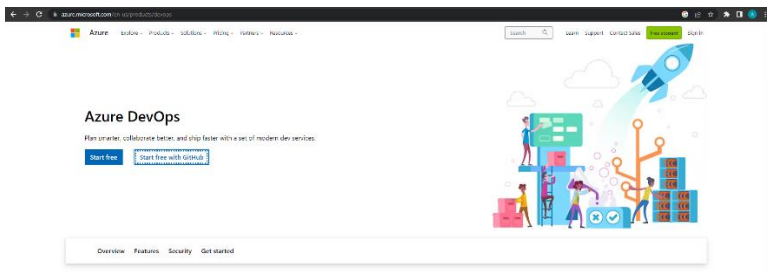
Hasil dari konfigurasi dapat dilihat seperti pada Gambar 88 apabila tidak ada kendala. Tambahkan database elibrary_app agar dapat melakukan automigrate saat deploy API.

6.3 Tahapan CI/CD

CI/CD merupakan proses integrasi kode kedalam repositori, kemudian dijalankan secara otomatis dengan tujuan untuk menjalankan pengujian secara otomatis, cepat, dan dilakukan secara berulang kali. Proses tersebut akan dijelaskan pada bagian ini pada tahapan-tahapan berikut.

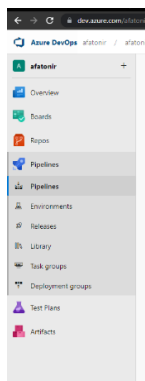
1. Penggunaan Azure DevOps

Azure DevOps merupakan perangkat lunak yang dikhususkan untuk bidang DevOps, Azure DevOps ini dibuat oleh Microsoft. Berikut adalah tahapan penggunaannya.



Gambar 89 Landing Page Azure DevOps

Buka halaman utama Azure dengan mengakses link <https://dev.azure.com/> dan lalu tekan tombol “start free with github” seperti pada Gambar 89.



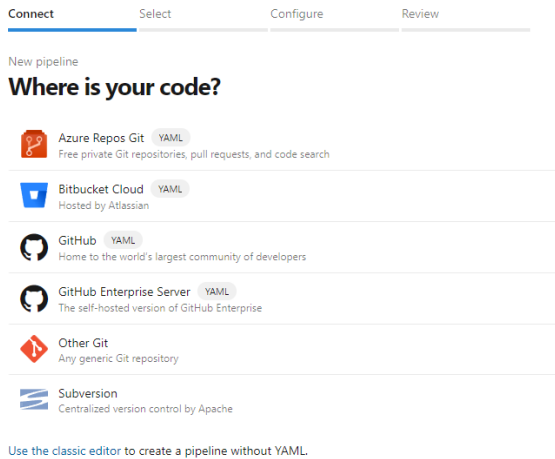
Gambar 90 Sidebar Azure

Pada Gambar 90 merupakan sidebar dari Azure, pilih “pipelines” untuk memulai proses CI/CD.



Gambar 91 Tambahkan Pipeline

Tambahkan pipeline dengan menekan tombol New Pipeline seperti pada Gambar 91.



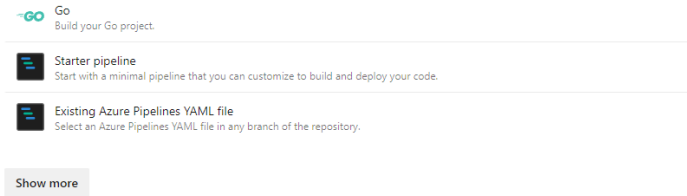
Gambar 92 Pemilihan Repository

Selanjutnya adalah tampilan pemilihan repository seperti pada Gambar 92, pilih Github untuk memilih repository.



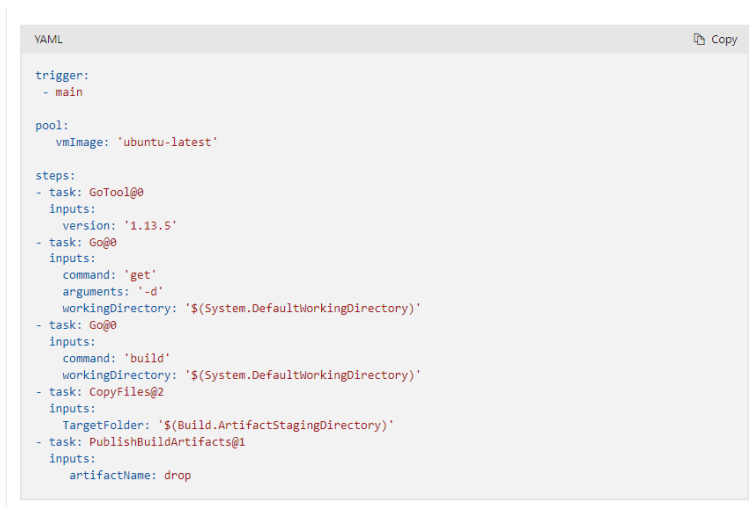
Gambar 93 Pilih Repository

Pilih repository elib_v2 seperti pada gambar 93 untuk memulai pengetesan.



Gambar 94 Pemilihan Opsi Build

Pada Gambar 94 merupakan menu pemilihan opsi build. Pilih Go lalu lanjutkan.



Gambar 95 Script Build

Gambar 95 merupakan kode untuk melakukan build script, ubahlah version pada inputs sesuai dengan versi Go yang digunakan.

Save and run

Saving will commit azure-pipelines-1.yml to the repository.

Commit message

Set up CI with Azure Pipelines

Optional extended description

Add an optional description...

☒ Commit directly to the main branch

☐ Create a new branch for this commit

Save and run

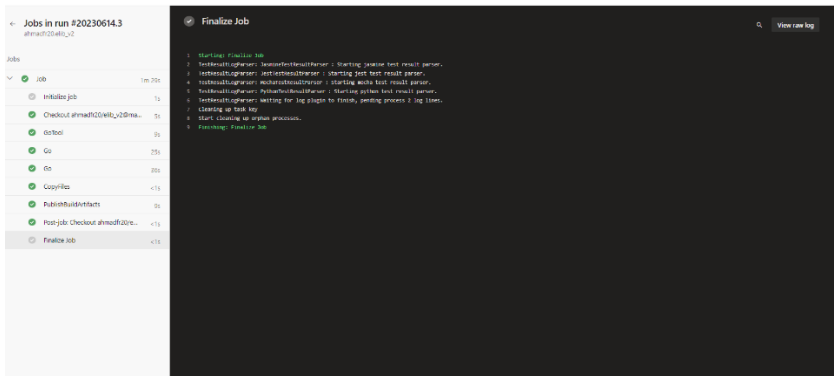
Gambar 96 Opsi Commit

Gambar 96 merupakan tahap terakhir konfigurasi dari CI/CD, yaitu memilih aksi apabila pipeline sudah dijalankan. Pilih commit directly to main branch.

Manually run by Ahmad Retno			
Repository and version	Time started and stopped	Results	Tests and coverage
07a5a6202946122	07 Jun 2024	14/0 tests items	0 - Not started
07 main - 0 build	-	10/0 artifacts	
Jobs			
Name	Status	Duration	
0 jobs		0:00:00	

Gambar 97 Proses CI/CD

Gambar 97 adalah proses dari CI/CD tersebut apabila konfigurasi sudah sesuai.



Gambar 98 Hasil CI/CD

Pada Gambar 98 merupakan hasil dari CI/CD, apabila tidak mengalami error atau terdapat warning maka API siap untuk dideploy.

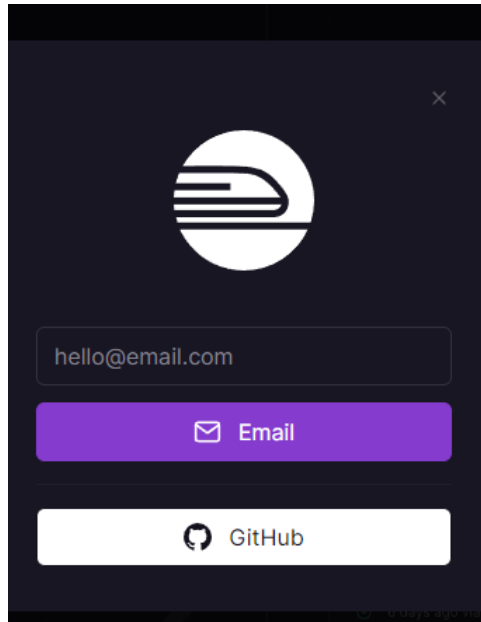
6.4 Tahapan Deploy Backend

Selanjutnya adalah tahapan publikasi terhadap API agar dapat diakses oleh bagian Frontend. Berikut adalah tahapan-tahapannya.



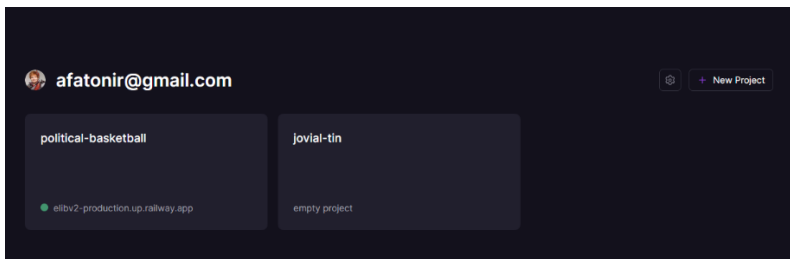
Gambar 99 Railway

Pertama yang harus dilakukan adalah mengakses link “railway.app” seperti pada gambar 99. Railway merupakan website yang menyediakan jasa untuk Hosting API secara gratis.



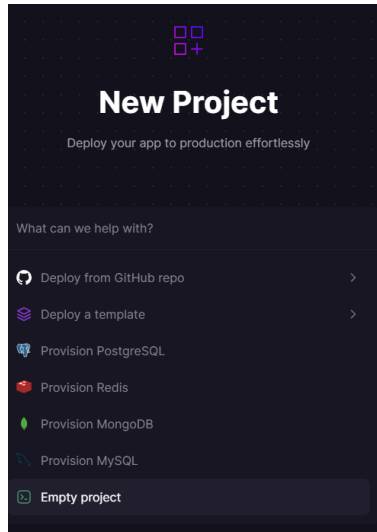
Gambar 100 Login Railway

Gambar 100 merupakan tahap autentifikasi pengguna, Login dengan menggunakan akun github agar dapat mengakses repositori dari API.



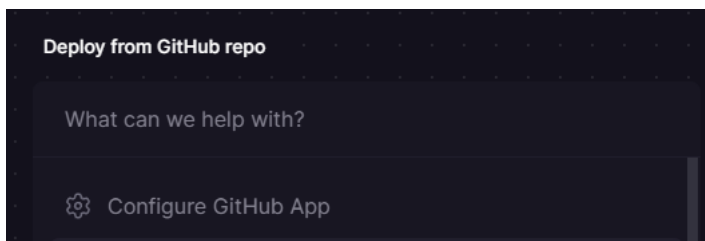
Gambar 101 Pembuatan Project Baru Railway

Pada Gambar 101 Merupakan proses pembuatan project baru pada Railway.



Gambar 102 Pemilihan Repository

Gambar 102 merupakan opsi untuk memilih repository yang akan dideploy pada Railway, pilih Deploy from Github repo.



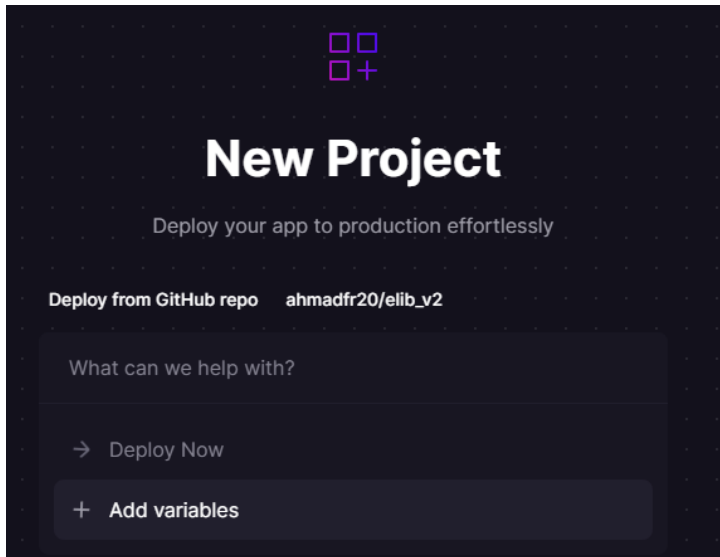
Gambar 103 Konfigurasi Github

Gambar 103 digunakan untuk melakukan konfigurasi Github terlebih dahulu, hal tersebut berguna agar Railway dapat mendeteksi repository apa saja yang terdapat di github.



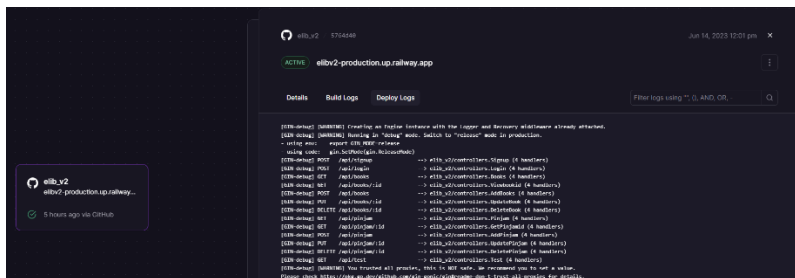
Gambar 104 Pemilihan Repo API

Gambar 104 adalah repository yang akan digunakan untuk dideploy nantinya.



Gambar 105 Proses Akhir

Gambar 105 adalah proses terakhir dalam deploy API, klik pada deploy now agar API segera dipublikasikan.



Gambar 106 Output Hasil Deploy

Gambar 106 merupakan output akhir dari deploy API, apabila tidak menemui masalah apapun maka output akan terlihat seperti pada gambar.

6.5 Tahapan Deploy Frontend

Setelah proses deploy pada bagian Database dan API sudah dilakukan maka tahap terakhir adalah melakukan deploy terhadap bagian Frontend aplikasi, berikut adalah tahapannya.

```
1 import axios from "axios";
2
3 export default axios.create({
4   baseURL: "http://elibv2-production.up.railway.app/api",
5   headers: {
6     "Content-type": "application/json"
7   }
8 });
9 });
```

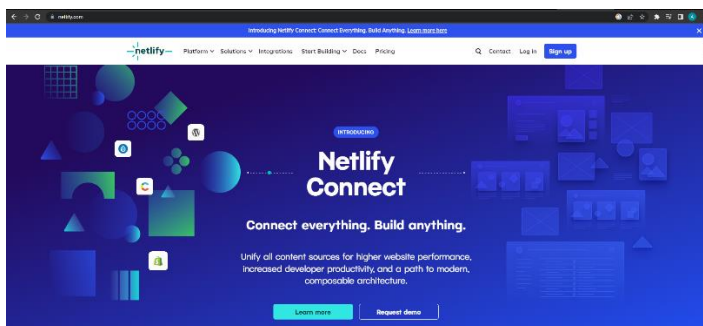
Gambar 107 Penggantian URL Localhost

Pertama adalah mengganti URL pada file http-common.js menjadi seperti pada Gambar 107 agar dapat tersambung dengan API.

```
PS D:\Proyek 1\elibrary_fullstack\elib_fe> yarn build
```

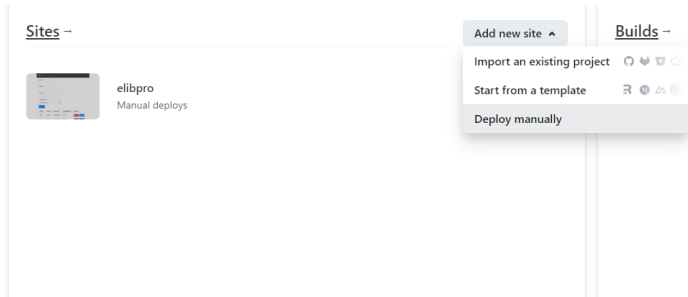
Gambar 108 yarn build

Selanjutnya ketikkan command “yarn build” seperti pada gambar 108 pada terminal agar dapat membuat aplikasi dalam tahap produksi.



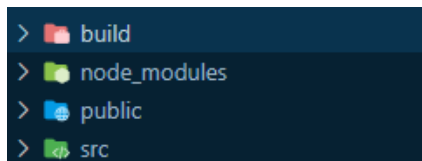
Gambar 109 Landing Page Netlify

Selanjutnya akses website netlify untuk melakukan deployment aplikasi React secara gratis dengan mengakses “netlify.com”. Gambar 109 merupakan tampilan dari website official dari netlify. Klik pada tombol sign up dan pilih github.



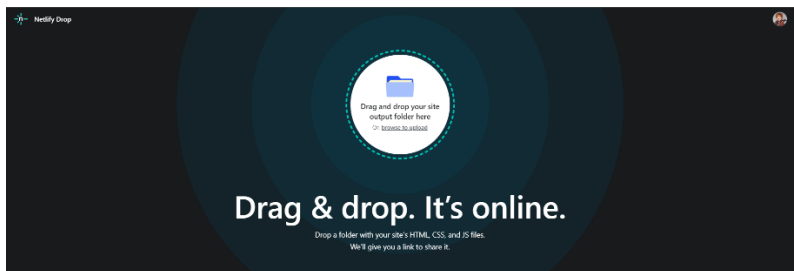
Gambar 110 halaman awal netlify

Gambar 110 merupakan section dari halaman awal Netlify, scroll pada bagian paling bawah dan pilih add new site lalu pilih deploy manually.



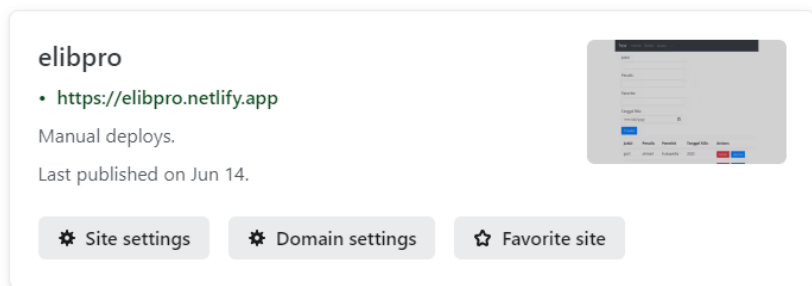
Gambar 111 Hasil yarn build

Gambar 111 merupakan hasil dari proses penginputan command “yarn build”. Kembali ke direktori project react dan lihat folder build.



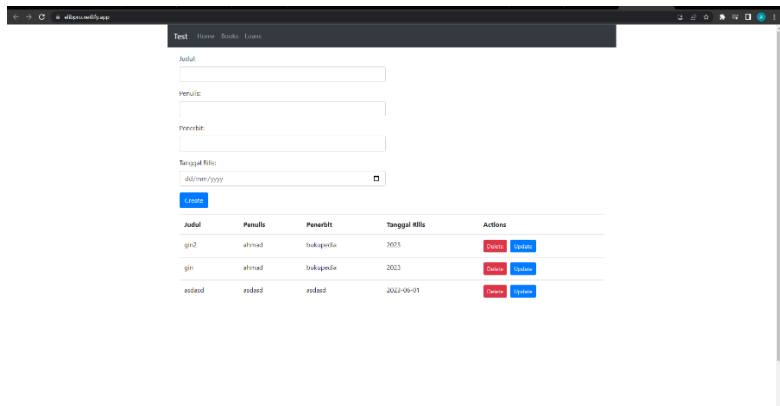
Gambar 112 Drag Drop File

Gambar 112 merupakan form untuk drag drop file yang akan dideploy nantinya. Masukkan file build yang sudah dibuat pada form tersebut.



Gambar 113 URL Hasil Deploy

Gambar 113 merupakan hasil dari URL setelah melakukan deploy. Alamat dari website bisa diganti dengan nama yang kita sukai, tetapi untuk .netlify belum bisa dihilangkan karena itu merupakan fitur berbayar dari netlify.



Gambar 114 Tampilan pada Browser

Gambar 114 merupakan tampilan saat mengakses URL yang sudah selesai dideploy pada Netlify.

BAB 7

Kesimpulan

6.1. Kesimpulan

Dalam buku ini Anda telah mempelajari tentang Langkah-langkah dalam membangun sebuah aplikasi modern menggunakan teknologi REST API menggunakan Gin Framework dan React JS. Pembahasan dalam buku ini diawali dengan pengenalan apa saja Tools yang digunakan, lalu cara pemasangan Tools, kemudian perancangan Front-end, dan terakhir Perancangan Back-end.

Dengan mengikuti langkah-langkah praktis dan penjelasan teoritis dalam buku ini. Anda telah memperoleh pemahaman yang cukup kuat dalam pembangunan aplikasi menggunakan REST API dan mengintegrasikannya kedalam Front-end. Semoga buku ini dapat memberikan panduan yang berharga dan mendorong Anda untuk terus mengembangkan kemampuan dalam mengembangkan perangkat lunak.

Daftar Pustaka

- Freeman, A. (2019). *Understanding React*. London, UK: apress.
- Gin. (n.d.). *Gin Web Framework*. (Gin) Retrieved 04 07, 2023 from <https://gin-gonic.com/>
- Iyengar, K., Khanna, S., Ramadath, S., & Stephens, D. (2017). What it really takes to capture the value of APIs. 1-9.
- Kurniawan, N. (2020, 01 02). *Apa Itu API*. (Medium) Retrieved 04 07, 2023 from <https://medium.com/@novancimol12/apa-itu-api-b7ec679a9e24>
- Liu, Z., & Gupta, B. (2019). Study of Secured Full-Stack Web Development. *EPiC Series in Computing*, 317-324.
- McGrath, M. (2020). Go Programming in Easy Steps: Discover Google's Go Language. In M. McGrath, *Go Programming in Easy Steps: Discover Google's Go Language* (pp. 2-4). Warwickshire: In Easy Steps Limited.
- Mendonca, N. C., Jamshidi, P., Garlan, D., & Pahl, C. (2019). Developing Self-Adaptive Microservice Systems: Challenges and Directions. *IEEE Software*, 38(2), 70-79.
- Microsoft. (n.d.). *Why did we build Visual Studio Code?* From Visual Studio Code: <https://code.visualstudio.com/docs/editor/whyvscode>
- Pamungkas, R. (2018). *Teori dan Implementasi Pemrograman Web*. Madiun: Unipma Press.
- ReactJS. (2020). *A JavaScript library for building user interfaces*. From ReactJS: <https://reactjs.org/>
- Sari, D. P., & Wijanarko, R. (2019). Implementasi Framework Laravel pada Sistem Informasi Penyewaan Kamera (Studi Kasus Di Rumah Kamera Semarang). *Informatika dan RPL*, 2(1), 32-36.
- Serrano, D., & Stroulia, E. (2017). Linked Rest APIs: A Middleware for Semantic Rest API Integration. *IEEE 24th International Conference on Web Services*, 138-145.

TENTANG PENULIS



Ahmad Fathoni R, Lahir di Kota Bandung pada tanggal 20 September 2000. Pendidikan tingkat dasar hingga menengah ditempuh di Kota Sumedang, sedangkan pendidikan tingkat atas di Kota Bandung. Sedang menempuh pendidikan di program D4 TI Program Studi Teknik Infomratika Politeknik Pos Indonesia (Sekarang Universitas Logistik dan Bisnis Internasional).



Roni Andarsyah, S.T., M.Kom., SFPC. Telah menyelesaikan pendidikan Diploma di Politeknik Pos Indonesia di program D3 Teknik Informatika, dan Magister di STMIK LIKMI di Bandung, dan saat ini sedang menjabat sebagai ketua program studi D4 Teknik Informatika di Universitas Logistik Bisnis Internasional.

Buku "Tutorial Implementasi REST API pada E-Library dengan Gin Framework Golang, ReactJS, dan Deployment: Panduan Langkah demi Langkah" adalah panduan praktis yang menawarkan pembaca langkah-demi-langkah dalam membangun aplikasi web modern untuk E-Library. Dalam buku ini, Anda akan belajar tentang penggunaan REST API, Gin Framework Golang, dan ReactJS untuk menciptakan sistem perpustakaan digital dengan integrasi antara Client dan Server.

