BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Studi independen di Binar Academy dilakukan dengan peran sebagai peserta dalam kelas yang mempelajari *fullstack website*. Seluruh peserta pada kelas tersebut dibimbing dan diajar oleh Kakak Alma Rahmawati selaku fasilitator atau pengajar di *fullstack website* kelas kesembilan. Sebagai pelajar Full Stack Website, peserta memiliki kewajiban untuk mengerjakan setiap tugas yang diberikan dan ikut serta mendengarkan materi yang disampaikan oleh fasilitator.

Selama pelaksanaan kegiatan studi independen, koordinasi dilakukan dengan menggunakan aplikasi WhatsApp, Discord, Telegram, dan GitHub. WhatsApp dan Discord digunakan sebagai alat komunikasi yang berhubungan dengan group kelas. Telegram digunakan sebagai alat komunikasi antara peserta Binar Academy dan fasilitator. Github digunakan untuk mengirimkan hasil pekerjaan peserta Binar Academy ke fasilitator dan anggota kelompok.

3.2 Tugas yang Dilakukan

Berikut adalah daftar pembelajaran mingguan dari pelaksanaan studi independen peserta Binar Academy.

Tabel 3.1. Pembelajaran yang dilakukan setiap minggu selama pelaksanaan studi independen

Minggu Ke -	Pembelajaran yang dilakukan
1-2	•Menguasai kamampuan dasar untuk membuat halaman *website* •Mempelajari dan memahami *HTML, CSS dan CSS Framework (Bootstrap)* •Mempresentasikan hasil tugas yang sudah dibuat •Mengerjakan *challenge* mengenai *website* statis sederhana
3-4	•Mempelajari dan memahami data *structure, operator* & *expressions*, dan *basic javascript* •Mengerjakan *challenge* mengenai pembuatan fitur *sorting* (*Bubble sort*) dan *filtering*
5-6	•Mempelajari dan memahami terminal & IDE, GIT, *web layouting* dan *responsive design* •Mengerjakan *challenge* mengenai implementasi gitlab
7-8	•Menerapkan OOP, DOM, Node.js dan HTTP *Server* dalam pengembangan *website* •Mempelajari dan memahami OOP, DOM, Node.js, dan HTTP *Server* •Mengerjakan challenge memmbuat fitur filter berdasarkan tipe driver, jumlah penumpang, dan waktu sewa.

Tabel 3.2. Pekerjaan yang dilakukan setiap minggu selama pelaksanaan studi independen (lanjutan)

Minggu Ke -	Pekerjaan yang dilakukan
9-10	•Menerapkan perancangan database •Mempelejari dan memahami Express.js, *restful* API, *database*, dan ORM •Mengerjakan *challenge* untuk membuat HTTP *Server* yang dapat digunakan untuk melakukan manajemen data mobil
11-12	•Merancang arsitektur dan dokumentasi API •Mempelajari dan memahami *desgin pattern*, *asynchronous*, *authentication* dan *swagger* •Mengerjakan *challenge* untuk membuat REST API yang dapat digunakan untuk melakukan manajemen data mobil dengan fitur *authentication*
13-14	•Mempelajari React.js, React.js data, dan OAuth. •Mengerjakan challenge untuk membuat tampilan *website* dengan menggunakan React.js
15-16	•Mempelajari *web socket*, SSR (*server side rendering*), *media handling*, eslint, unit testing & TDD, dan deployment & CI/CD •Mengerjakan challenge untuk melakukan *unit testing* dan *deployment* di *website* yang sudah dibuat

3.3 Uraian Pembelajaran Studi Independen

Proses pelaksanaan studi indenpenden dibagi menjadi tiga bagian yaitu proses pelaksanaan, kendala yang ditemukan, dan solusi atas kendala yang ditemukan.

3.3.1 Proses Pelaksanaan

Proses pelaksanaan mengerjakan challenge di Binar Academy dari minggu ke-1 sampai ke-8 membutuhkan perangkat lunak dan perangkat keras. Perangkat lunak yang digunakan untuk mengerjakan challenge di Binar Academy dari minggu ke-1 sampai ke-8 adalah sebagai berikut.

1. Visual Studio Code versi 1.67.2
2. Git versi 2.35.1.windows.2
3. Node versi v16.14.2
4. NPM versi 8.7.0
5. Postgre versi 14.2
6. Operating System Windows 10 Home Single Language 64-bit
7. Google Chrome Versi 102.0.5005.63 64-bit

A.Pembelajaran *HTML, CSS,* dan *Framework CSS*

Pada bab ini, fasilitator membagi pembelajaran menjadi 3 materi, yaitu *HTML, CSS* dan *Framework CSS*. Materi pertama adalah HTML, mengajarkan *tag HTML*, *attributes*, kegunaannya, dll. Pada materi *CSS*, mempelajari 3 cara menulis *CSS* (*inline*, *external, dan internal*), attributes, kegunaannya, dll. Selanjutnya, pada materi *Framework CSS*, mempelajari tentang *Bootstrap 4*, mengimplementasikan cara untuk membuat *web responsive* dengan *Bootstrap*, membuat *card*, *navigation bar*, dll.

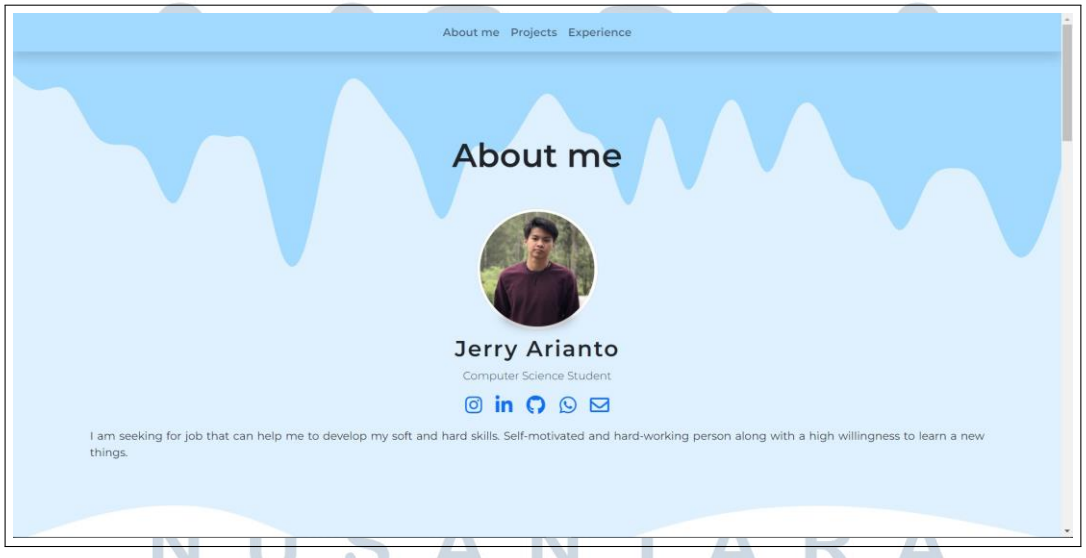
A.1Kriteria pembuatan *Website* Portofolio

Saat pembuatan *website* portoflio ini, peserta Binar Academy diperintahkan atas keputusan fasilitator untuk membuat *landing pages* sederhana dengan tema dan desain yang diberikan Binar Academy atau bisa juga menggunakan tema dan desain yang diinginkan peserta, dengan tujuan untuk meningkatkan kreativitas. Dengan kriteria yang diberikan, yaitu:

1. Memiliki fitur pada website umumnya seperti *navigation bar, card, accordion, dll.*
2. *Website* yang dibuat tidak diwajibkan untuk *responsive*.
3. Mengimplementasikan materi yang sudah diajarkan.

A.2Implementasi

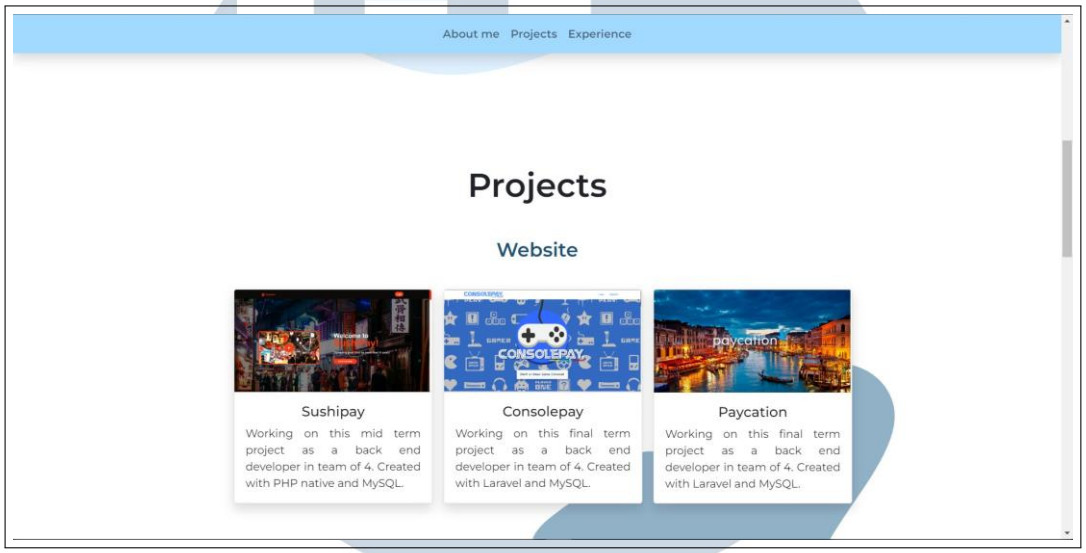
Pengerjaan *website* portofolio ini dikerjakan secara individu dengan tema dan desain yang dibuat unik. Konsep dari *website* yang dibuat adalah menerapkan *website* yang berisi seperti *portofolio* dan nantinya akan di*update* secara berkala serta *website* ini juga sudah di*hosting* menggunakan *GitHub*. Teknologi yang dipakai terdapat *HTML*, *CSS*, *Framework CSS*, *Animate on scroll library*, dan *GitHub*.



Gambar 3.1. Halaman *about me*

Pada gambar 3.1, menampilkan halaman *about me* berisi tentang foto profil, *social media*, dan deskripsi singkat tentang diri. Foto profil dibuat bundar menggu-

nakan *property border-radius* dan jika di*hover* akan menampilkan animasi seperti terangkat. Lalu, ada icons yang didapatkan dari fontawesome serta sudah ditambahkan library *Animate on Scroll* yang akan otomatis menampilkan animasi ketika situs web di*reload*. Begitu juga dengan deskripsi singkat, akan terjadi animasi seperti transisi setiap kali situs web di*reload*.

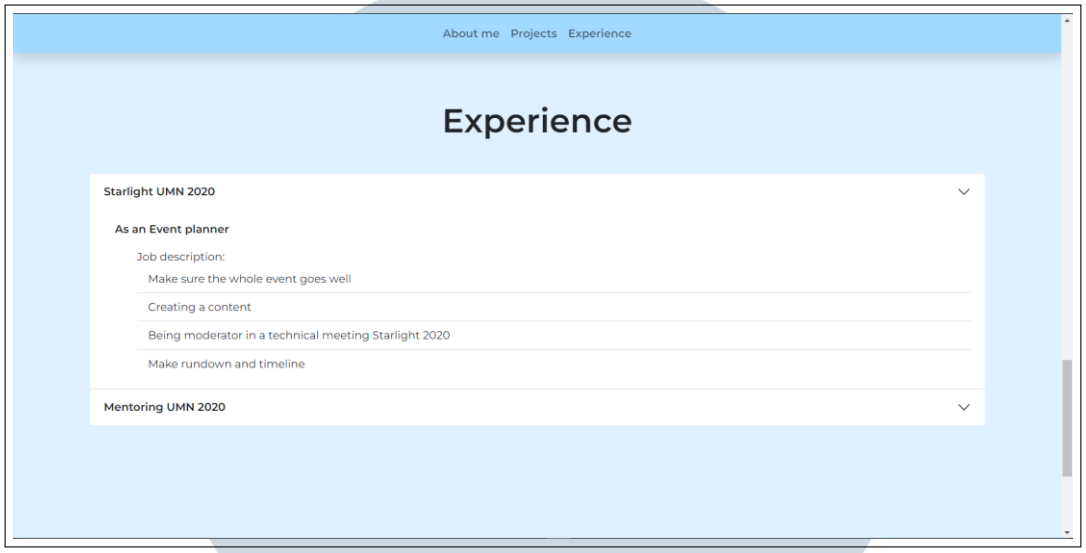


Gambar 3.2. Halaman *projects*

Gambar 3.3. Halaman *projects (lanjutan)*

Kemudian terdapat halaman *projects* pada gambar 3.2 dan 3.3. Halaman ini hanya menunjukkan *projects* yang dibuat beserta deskripsi singkat dan gambar tentang *project* tersebut. Deskripsi singkat dan gambar dari *projects* tersebut juga

ditampung dalam *card* yang dibuat dengan *Bootstrap* Lalu, juga terdapat hiasan *blob* di *background* dengan format *svg* yang dibuat menggunakan *blobmaker*.



Gambar 3.4. Halaman *experience*

Pada gambar 3.4, menampilkan halaman untuk halaman *experience*, berisi tentang pengalaman-pengalaman dari organisasi almamater. Teks tersebut berada pada sebuah *accordion* yang dibuat oleh *Bootstrap*. Ketika *dropdown* diklik akan menampilkan detail dari pengalaman organisasi almamater

B. Pembelajaran Javascript dasar

Memepelajari tentang dasar-dasar *Javascript*. Dalam pembelajaran ini terdapat 3 materi yang disampaikan, yaitu: *Introduction to Javascript*, *Operator & Expressions*, dan *Basic Javascript Algorithm*. Pada materi *Introduction to Javascript* diajarkan tentang 3 cara menambahkan (*inline*, *external*, *dan inline*), deklarasi, *basic syntax*, dll. Selanjutnya, di materi *Operators & Expressions* diajarkan tentang *binary operator*, *unary operator*, dan *conditional ternary*. Di materi terakhir, yaitu Algoritma Dasar *Javascript*, mempelajari tentang konsep algoritma, alur berpikir algoritma, serta alur untuk perulangan.

B.1 Kriteria pembuatan fungsi filtering dan sorting

Tugas ini dibuat untuk melatih pemahaman tentang *Javascript*. Pada proses pembuatan fungsi *filtering* dan *sorting*, peserta Binar Academy diwajibkan untuk

menyelesaikan tugas yang diberikan. Dengan kriteria yang diberikan, yaitu:

1. Membuat sebuah fungsi bernama *filterCarByAvailability*, yang berfungsi untuk menyaring daftar mobil yang diberikan, dan menyisakan daftar mobil yang atribut *available*-nya bernilai *true*.
2. Membuat sebuah fungsi bernama *sortCarByYearAscendingly*, yang berfungsi untuk menyortir daftar mobil yang sudah diurutkan dari tahun paling tua.
3. Membuat sebuah fungsi bernama *sortCarByYearDescendingly*, yang berfungsi untuk akan menyortir daftar mobil yang sudah diurutkan dari tahun paling muda.

B.2Implementatsi

Pada proses pengerjaan fungsi *filtering* dan *sorting* ini dikerjakan secara individu. Fungsi ini akan ditulis dalam *file Javascript* masing-masing yang sudah tersedia di dalam *template*.

```
function filterCarByAvailability(cars) {
  // Sangat dianjurkan untuk console.log semua hal
  console.log(cars);

  // Tempat penampungan hasil
  const result = [];

  // Tulis code-mu disini
  for (let i = 0; i < cars.length; i++) {
    if (cars[i]["available"]) {
      result.push(cars[i]);
    }
  }

  console.log(result);
  // Rubah code ini dengan array hasil filter berdasarkan availablity
  return result;
}
```

Gambar 3.5. Fungsi *filtering by availability*

Pada gambar 3.5, menampilkan sebuah fungsi *filtering* yang dibuat menggunakan perulangann *for* supaya semua mobil dapat dicek. Di dalam perulangan *for* terdapat *if* untuk mengecek apakah mobil tersebut tersedia untuk disewa atau tidak.

```
function sortCarByYearAscendingly(cars) {
  // Sangat dianjurkan untuk console.log semua hal
  // console.log(cars);

  // Clone array untuk menghindari side-effect
  // Apa itu side effect?
  const result = [...cars];

  // Tulis code-mu disini
  for (i = 0; i < result.length - 1; i++) {
    for (j = 0; j < result.length - 1; j++) {
      if (result[j]["year"] > result[j + 1]["year"]) {
        let temp = result[j]["year"];
        result[j]["year"] = result[j + 1]["year"];
        result[j + 1]["year"] = temp;
      }
    }
  }

  console.log(cars);
  // Rubah code ini dengan array hasil sorting secara ascending
  return result;
}
```

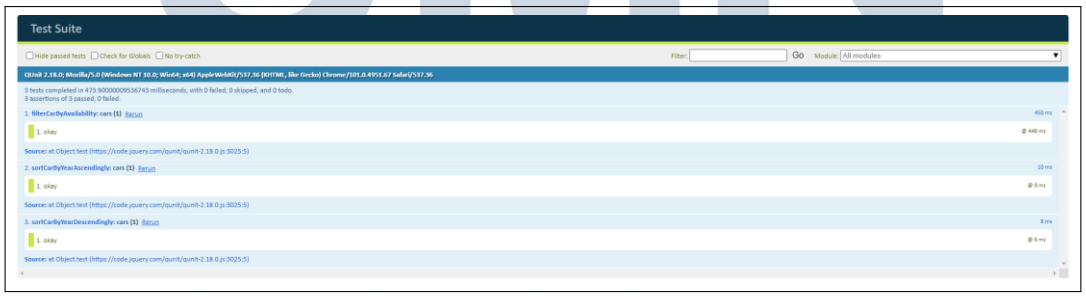
Gambar 3.6. Fungsi *sorting by year ascendingly*

Pada gambar 3.6, menampilkan sebuah fungsi untuk *sorting* mobil berdasarkan tahun terkecil sampai terbesar. Algoritma *sorting* yang dipakai adalah *bubble sort*.

```
function sortCarByYearDescendingly(cars) {
  // Sangat dianjurkan untuk console.log semua hal
  // console.log(cars);

  // Clone array untuk menghindari side-effect
  // Apa itu side effect?
  const result = [...cars];

  // Tulis code-mu disini
  for (i = 0; i < result.length - 1; i++) {
    for (j = 0; j < result.length - 1; j++) {
      if (result[j]["year"] < result[j + 1]["year"]) {
        let temp = result[j]["year"];
        result[j]["year"] = result[j + 1]["year"];
        result[j + 1]["year"] = temp;
      }
    }
  }

  console.log(cars);
  // Rubah code ini dengan array hasil sorting secara descending
  return result;
}
```

Gambar 3.7. Fungsi *filtering by year descendingly*

Pada gambar 3.7, menampilkan sebuah fungsi untuk *sorting* mobil berdasarkan tahun terbesar sampai terkecil. Sama seperti gambar 3.6, algoritma *sorting* yang dipakai adalah *bubble sort*. Perbedaan *code* terdapat pada simbol ”<” dan ”>”.



Gambar 3.8. Tampilan halaman ketika sudah mengimplementasi 3 fungsi tersebut dengan baik

Saat ketiga fungsi telah berhasil dikerjakan, ketika dijalankan akan menampilkan halaman berisi 3 centang seperti gambar 3.8.

C.Pembelajaran terminal & IDE, GIT, *web layouting* dan *responsive design*

Di bab ini, fasilitator membagi pembelajaran menjadi 4, yaitu cara menggunakan terminal & IDE, GIT, *web layouting* dan *responsive design*. Materi yang diajarkan pertama tentang terminal & IDE, fasilitator mengajari konsep dan fungsi terminal & IDE serta relevansi terminal & IDE pada *Fullstack Web Developer*. Pada materi yang GIT, materi yang diajarkan meliputi definisi, fungsi & sejarah GIT, tahap penyimpanan di GIT (*working directory, staging area, local repository, remote repository*), perintah dasar di GIT, serta praktek dasar penggunaan GIT. Selanjutnya, di materi *web layouting*, mempelajari konsep dari *web layout* dan teknik layouting. Seperti, *layouting* dapat dibuat dengan 3 cara yaitu, *flexbox, grid, old school layouting*. Pada materi terakhir mempelajari tentang *responsive design*, materi ini mempelajari tentang konsep *responsive design* (landscape, potrait, ratio), *media query*, unit, serta teknik dalam *responsive design* (*desktop first* dan *mobile first*).

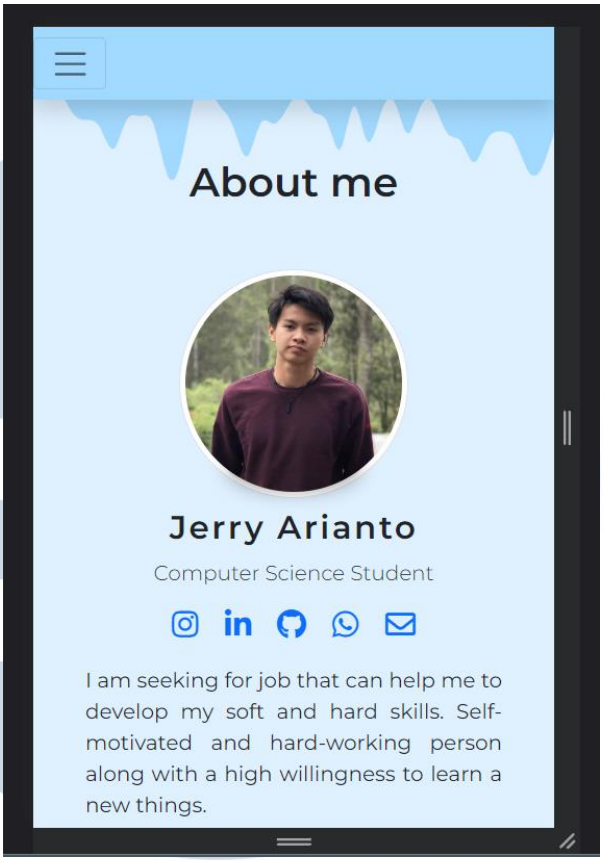
C.1Kriteria pembuatan responsive design dan implementasi GIT

Fasilitator memberi tugas untuk menambahkan *responsive design* pada *website* portofolio dan didokumentasikan dalam Gitlab. Dengan kriteria yang diberikan, yaitu:

1. Kode dari *website* yang dikerjakan di-*upload* ke Gitlab dan dibuat publik.
2. Pengumpulan tugas hanya mengirimkan *link* ke *repository* Gitlab.
3. *Website* yang dikerjakan dapat dilihat dengan baik sesuai dengan desain apabila dibuka menggunakan *smartphone*
4. *Website* yang dikerjakan dapat dilihat dengan baik sesuai dengan desain apabila dibuka menggunakan *desktop*.

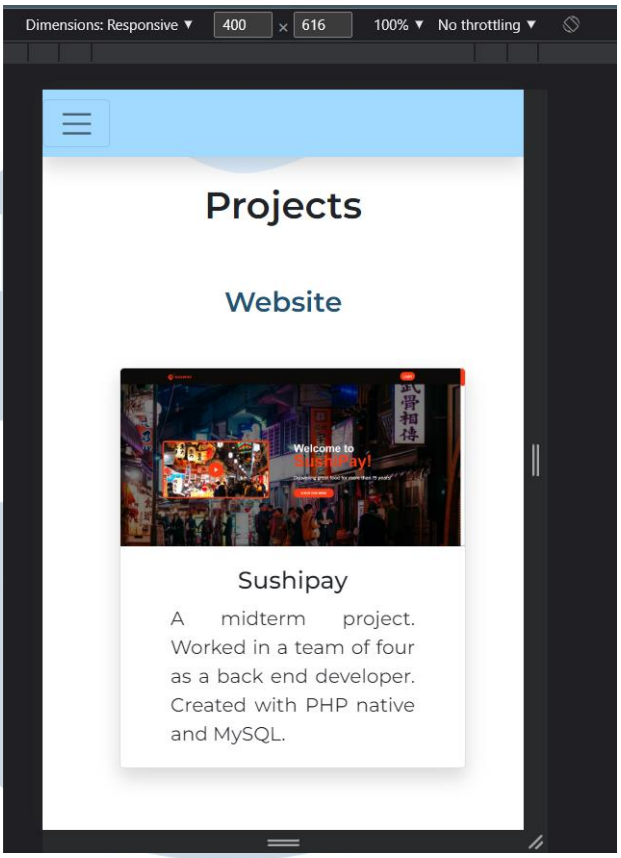
C.2Implementasi

Pengerjaan *responsive design* dan implementasi GIT dikerjakan secara individu.



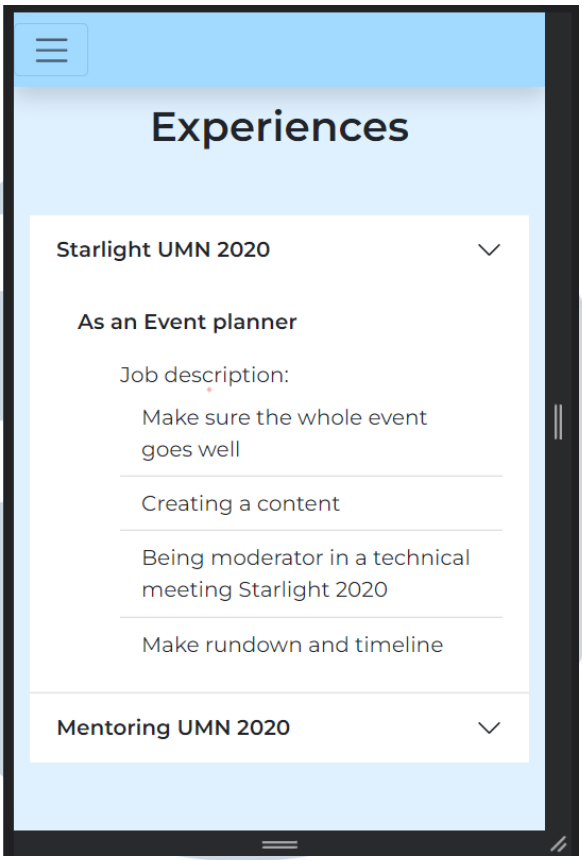
Gambar 3.9. Halaman *about me*

Pada gambar 3.9, merupakan tampilan *website* portofolio halaman *about me* di *platform mobile*. *Responsive design* dikerjakan secara manual menggunakan CSS, menggunakan *media query* yang diatur untuk setiap *device* berdasarkan lebar maksimal dari layar yang digunakan.



Gambar 3.10. Halaman *projects*

Selanjutnya ada tampilan halaman *projects* yang mendeskripsikan tentang projek-projek yang sudah dikerjakan. Menggunakkan desain yang masih sama dengan challenge pemebelajaran *HTML*, *CSS* , dan *Framework CSS*.



Gambar 3.11. Halaman *experience*

Pada gambar 3.11, menampilkan halaman *experience* yang mendeskripsikan tentang pengalaman-pengalaman organisasi di almamater.*User* atau pengguna situs harus menekan tombol *accordion text*, jika *user* ingin membuka *accordion* yang ke-2.

Setelah membuat *responsive design* dari *website* portofolio, peserta Binar Academy wajib untuk mengumpul/*upload* keseluruhan *file* ke GitLab masing-masing menggunakkan terminal.

D.Pembelajaran OOP (*Object Oriented Programming*), DOM (*Document Object Model*), *Node.js*, dan *HTTP Server*

Pada bab ini, fasilitator membagi pembelajaran menjadi 4 materi, yaitu tentang OOP, DOM, *Node.js* dan *HTTP Server*. Materi yang diajarkan pertama mengenai OOP. Mempelajari tentang konsep OOP, memahami cara menggunakkan *function*, *class* dan *method* di JavaScript dan menerapkan 4 pilar OOP (*Inheritance*, *Encapsulation*, *Abstraction*, dan *Polymorphism*). Di materi selanjutnya fasilitator

mengajarkan tentang DOM atau bisa disebut *Document Object Model*. Peserta Binar i diajarkan tentang konsep tentang DOM, DOM di JavaScript, DOM *selector*, dan DOM *manipulation*. Selanjutnya, materi yang diajarkan tentang *node.js*, terdiri dari pemahaman *runtime environment*, *module* pada *node.js*, menginisiasi project menggunakan *Package Manager*, menggunakan *module* untuk *read & write file*, perbedaan menjalankan *code* di *node.js* dan *browser*. Terakhir, materi yang diajarkan fasilitator di bab ini tentang *HTTP Server*. Materi ini mengajarkan konsep dan fungsi *HTTP server*, melakukan *serving file HTML* dengan *HTTP Server*, membuat *endpoint* yang merespon dengan *file JSON*.

D.1Kriteria pembuatan *routing* dan fitur *filtering* pada *website* sewa mobil

Pada bab ini, fasilitator memberikan tugas untuk menambahkan *routing* dan fitur *filtering* pada *website* sewa mobil. Dengan kriteria yang diberikan, yaitu:

1. *HTTP Server* harus bisa menyajikan HTML yang dibutuhkan beserta JavaScript dan CSS.
2. Ketika ada *request* GET / maka *server* akan merespon dengan halaman home.
3. Ketika ada *request* GET /cars maka *server* akan merespon dengan halaman sewa mobil.
4. Terdapat fitur *filtering* di halaman sewa mobil. Fitur ini dapat menyaring berdasarkan tipe driver, waktu sewa dan jumlah penumpang.

D.2Implementasi

Pada proses pengerjaan *routing* dan fitur *filtering* ini dikerjakan secara individu. Langkah pertama pada pengerjaan tugas ini adalah membuat routing terlebih dahulu.

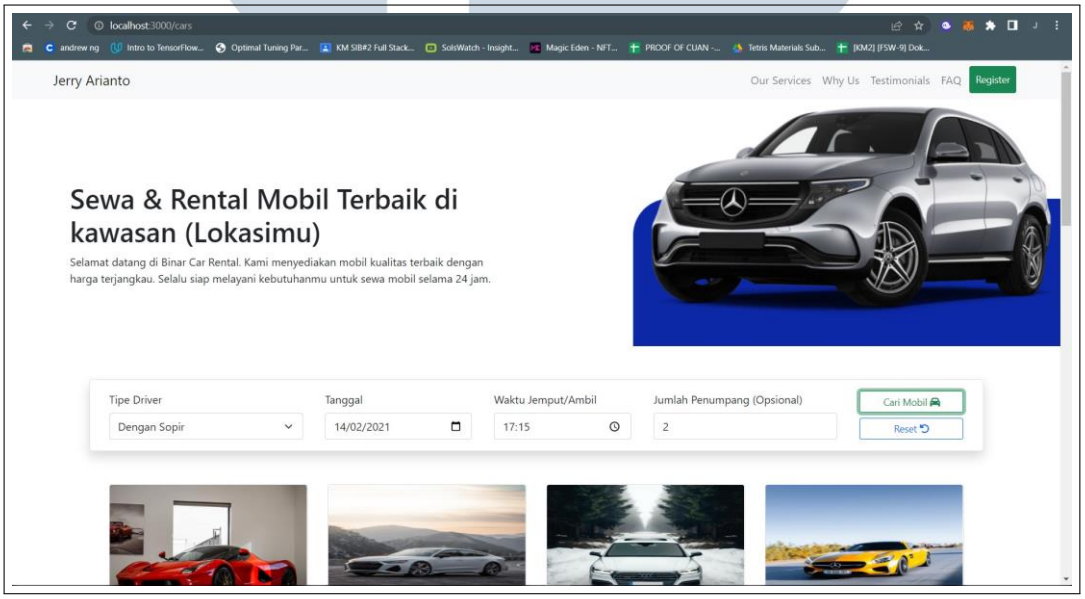
Gambar 3.12. *Routing*

Pada gambar 3.12, kode di baris ke-8 sampai ke-10 dan baris ke-11 sampai ke-13 akan merespon ke halaman *home* atau file index.html. Jadi, ada 2 jalur untuk menuju halaman *home*, pertama menggunakan ”/” dan yang kedua menggunakan ”/home”. Kedua-duanya menggunakan GET *method* karena hanya menampilkan data saja.

Selanjutnya, untuk baris ke-14 sampai ke-16 terdapat kode yang akan merespon ke halaman sewa mobil. Untuk masuk ke halaman ini, *user* harus menekan tombol ”sewa sekarang” yang ada di halaman *home*. Halaman sewa mobil memiliki *method* yang sama dengan halaman *home*, yaitu GET.

```
21  run = () => {
22    let d = this.tanggal.value + "T" + this.waktu.value;
23    let formdate = Date.parse(d);
24    console.log(formdate);
25    Car.list.forEach((car) => {
26      let penumpang = this.penumpang.value;
27      let driber = this.driver.value;
28      if (driber == "true") {
29        driber = true;
30      } else driber = false;
31
32      let asalehe = Date.parse(car.availableAt);
33
34      if (car.available == driber && asalehe >= formdate && car.capacity >= penumpang) {
35        const node = document.createElement("div");
36        node.className = "mb-5";
37        node.innerHTML = car.render();
38        this.carContainerElement.appendChild(node);
39      }
40    });
41  };
```

Gambar 3.13. *Filtering*



Gambar 3.14. Tampilan dari fitur *filtering*

Pada gambar 3.13, terdapat fitur *filtering* yang dapat dibandingkan berdasarkan 3 *variable*, yaitu tipe *driver*, waktu sewa dan jumlah penumpang. Di baris ke-25 terdapat perulangan *forEach* yang akan me*looping* data-data mobil yang akan disewa. Kemudian di baris ke-34 sampai ke-38, terdapat *if* sebagai pengambil keputusan dari fitur *filtering*. Ketika kondisi bernilai *true*, akan menampilkan data-data mobil yang tersedia di dalam sebuah *card*. Gambar 3.14 merupakan hasil dari fitur *filtering* yang didesain menggunakan *Bootstrap*

E.Pembelajaran *express.js, restful* API, *database, SQL, ORM* (*object relational mapping*

Pada pembelajaran ini, fasilitator membagi pembelajaran menjadi 5, yaitu tentang *express.js*, *restful* API, *database*, *SQL*, *ORM* (*object relational mapping*. Materi yang pertama membahas tentang konsep dari *express.js*, memahami *routing* pada *express*.js, *middleware* di *express.js*, dan penggunaan *view engine* untuk membuat HTML. Di materi kedua, terdapat *restful* API mempelajari tentang desain API, pengenalan konsep *rest* API dan *restful* API, *resource indentifiers, resource methods*, dan penerapan *rest API* pada *express*.js. Selanjutnya, materi ke-3 adalah *database*. Dalam materi ini, peserta Binar Academy diajarkan tentang memahami apa itu *database*, mengenal jenis-jenis DBMS, mengenal SQL dan NoSQL, cara melakukan *query* pada konsol DBMS, dan mengenal ERD. Lalu, di materi ke-4 mempelajari tentang memahami *query* dengan menggunakan SQL, memahami cara pengunaan DDL, dan memahami cara pengunaan DML. Di materi yang terakhir, fasilitator mengajarkan konsep ORM, cara melakukan instalasi ORM, definisi model, implementasi CRUD pada rest API dengan model.

E.1Kriteria pembuatan HTTP *server* yang dapat digunakan untuk melakukan manajemen data mobil

Fasilitator memberikan tugas untuk membuat HTTP server yang dapat digunakan untuk melakukan CRUD(*create, read, update, delete*). Dengan kriteria yang diberikan, yaitu:

1. Dapat membuat sebuah HTTP *server* dengan menggunakan *express*.js.
2. Dapat membuat sebuah HTTP *server* sesuai dengan kaidah *restful* API.
3. Dapat menggunakan *database management system* untuk membuat tabel, memodifikasi tabel, dan melakukan operasi CRUD.
4. Dapat membuat sebuah HTTP *server* yang dapat digunakan untuk melakukan operasi CRUD ke dalam *database* melalui HTTP *request*.

E.2Implementasi

Saat proses pembuatan HTTP *server* yang dapat digunakan untuk melakukan manajemen data mobil dilakukan secara berkelompok. Langkah per-

tama, kelompok membuat *database* menggunakkan ORM *Sequelize*.

```
{
  "development": {
    "username": "postgres",
    "password": "33745",
    "database": "databasee_development",
    "host": "127.0.0.1",
    "dialect": "postgres"
  },
  "test": {
    "username": "postgres",
    "password": "33745",
    "database": "databasee_test",
    "host": "127.0.0.1",
    "dialect": "postgres"
  },
  "production": {
    "username": "postgres",
    "password": "33745",
    "database": "databasee_production",
    "host": "127.0.0.1",
    "dialect": "postgres"
  }
}
```

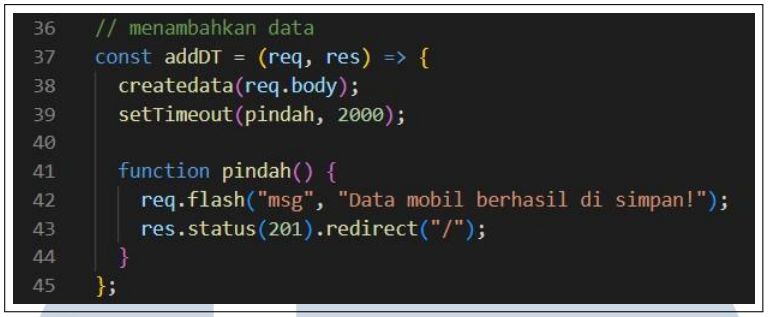
Gambar 3.15. Hasil implementasi *sequelize create database*

Pada gambar 3.15, menampilkan hasil konfigurasi *database* menggunakkan *sequelize* yang dipisah menjadi 3 jenis *environment*, yaitu *development*, *test*, *production*. Karena masih dalam tahap pengembangan *website*, maka *environment* yang digunakkan adalah *development*. Setelah selesai dikonfigurasi, *sequlize* memiliki fitur membuat *database* dengan perintah ”npx sequlize db:create”.

```
async up(queryInterface, Sequelize) {
  await queryInterface.createTable("cars", {
    id: {
      allowNull: false,
      autoIncrement: true,
      primaryKey: true,
      type: Sequelize.INTEGER,
    },
    name: {
      type: Sequelize.STRING,
    },
    size: {
      type: Sequelize.STRING,
    },
    image: {
      type: Sequelize.STRING,
    },
    price: {
      type: Sequelize.INTEGER,
    },
    createdAt: {
      allowNull: false,
      type: Sequelize.DATE,
    },
    updatedAt: {
      allowNull: false,
      type: Sequelize.DATE,
    },
  });
```

Gambar 3.16. Hasil konfigurasi fitur *migrate* dari *sequelize*

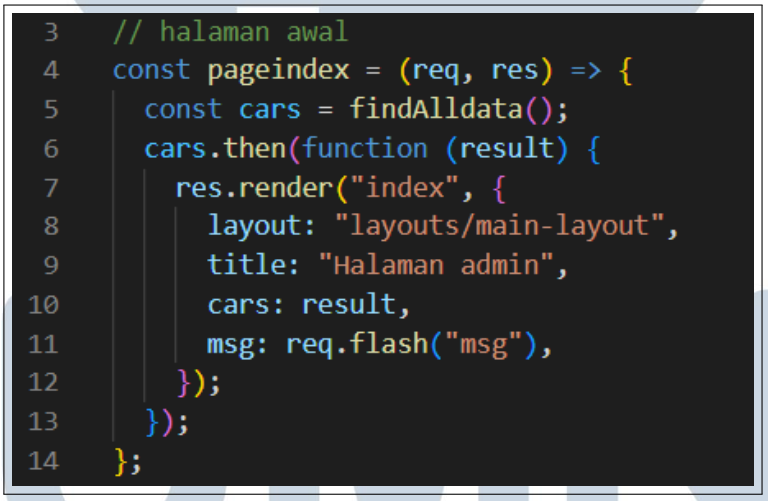
Ketika *database* sudah dibuat, maka langkah selanjutnya membuat tabel serta kolom tabel menggunakkan perintah ”npx sequelize model:generate –name cars –attributes name:string,size:string,image:string,price:integer”. Hasil implementasi dapat dilihat pada gambar 3.16. Ketika sudah dikonfigurasi, tambahkan perintah ”npx sequelize db:migrate” agar dapat memasukkan hasil konfigurasi ke dalam *database* dan tabel otomatis terbuat beserta kolom dan tipe datanya.



```
36  // menambahkan data
37  const addDT = (req, res) => {
38    createdata(req.body);
39    setTimeout(pindah, 2000);
40
41    function pindah() {
42      req.flash("msg", "Data mobil berhasil di simpan!");
43      res.status(201).redirect("/");
44    }
45  };
```

Gambar 3.17. Implementasi *create* data mobil

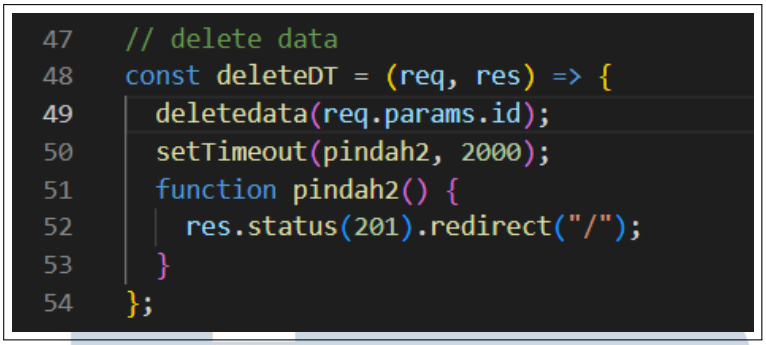
Pada gambar 3.17, menampilkan kode dari *create* data mobil. Pada baris ke-38 terdapat fungsi yang menambahkan data ke dalam *database* lewat *user input*. Setelah berhasil menambahkan data ke dalam *database*, *user* dilemparkan ke halaman *home* dengan jeda waktu 2 detik dan menampilkan pesan ”Data mobil berhasil di simpan!”.



```
3   // halaman awal
4   const pageindex = (req, res) => {
5     const cars = findAlldata();
6     cars.then(function (result) {
7       res.render("index", {
8         layout: "layouts/main-layout",
9         title: "Halaman admin",
10        cars: result,
11        msg: req.flash("msg"),
12      });
13    });
14  };
```

Gambar 3.18. Implementasi *read* data mobil

Pada gambar 3.18, merupakan kode dari fungsi *read* data mobil. Fungsi ini akan menampilkan semua data-data di *database* dan data tersebut akan diteruskan kepada file ejs. Di baris ke-5 terdapat fungsi *findAlldata* yang dapat mengakses *database* serta menampilkan semua data-data dalam *database*.



```
47  // delete data
48  const deleteDT = (req, res) => {
49    deletedata(req.params.id);
50    setTimeout(pindah2, 2000);
51    function pindah2() {
52      res.status(201).redirect("/");
53    }
54  };
```

Gambar 3.19. Implementasi *delete* data mobil

Pada gambar 3.19, mempertunjukkan fungsi yang berguna untuk *delete* data mobil. Fungsi ini akan mencari data mobil berdasarkan *id*nya. Setelah data mobil dihapus, *user* akan dilempar secara paksa ke halaman *home*.

```
56  // update data
57  const updateDT = (req, res) => {
58    updatedata(req.body);
59    setTimeout(pindah3, 2000);
60    function pindah3() {
61      req.flash("msg", "Data mobil berhasil di edit!");
62      res.status(201).redirect("/");
63    }
64  };
```

Gambar 3.20. Implementasi *update* data mobil

Untuk fungsi *update* mobil memiliki kode yang mirip dengan fungsi *create* mobil pada gambar 3.17. Sebelum dimasukkan ke *database*, fungsi ini akan mencari terlebih dahulu *id* dari data mobil yang ingin di*update*. Ketika *user* berhasil untuk meng*update* data, secara paksa user akan dipindahkan ke halaman *home* dan menampilkanTw sebuah pesan ”Data mobil berhasil di edit!”.

F.Pembelajaran *design pattern, asynchronous process, authentication & authorization,* dan *open API*

Pada bab ini, fasilitator membagi pembelajaran menjadi 4 materi, yaitu *design pattern, asynchronous process, authentication & authorization,* dan *open API*. Materi pertama, *design pattern* membahas tentang konsep dari *design pattern*, MVC (*model, view, controller*), *service repository pattern*, dan *microservice pattern*. Materi selanjutnya tentang asynchronous process, pada materi ini fasilitator membahas tentang memahami *asynchronous process*, mengenal cara penggunaan *callback*, dan

mengenal cara penggunaan *promise*. Pada materi ke-3 fasilitator membahas tentang mengenal konsep *authentication*, menerapkan *authentication* dengan menggunakan *encryption*, mengenal konsep *authorization*, menerapkan *authentication* dengan menggunakan *session based authentication*, dan *token based authentication* (JWT). Lalu, di materi terakhir ada open API yang membahas tentang konsep open API, mengenal struktur open API, memahami cara kerja swagger tool serta implementasinya, cara integrasi dokumentasi API, Mock API (*swagger codegen*).

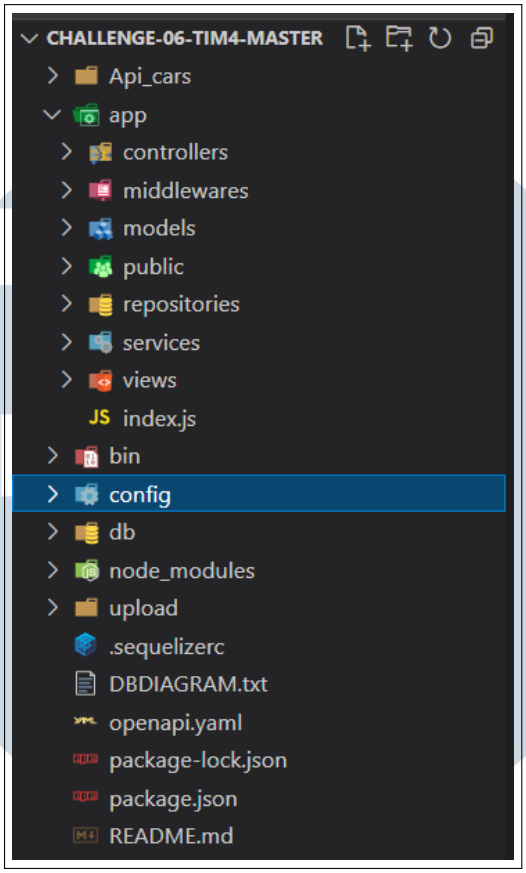
F.1Kriteria pembuatan REST API untuk melakukan manajemen data mobil dengan fitur *authentication*

Di bab ini, fasilitator memberikan tugas untuk membuat REST API yang terdapat *CRUD* (*create*, *read, update*, *delete*) dengan fitur *authentication*. Dengan kriteria yang diberikan, yaitu:

1. Mampu menerapkan *Service Repository Pattern* di dalam sebuah projek
2. Mampu membuat *asynchronous function* dan menjalankannya
3. Mampu menerapkan *Token Based Authentication* sebagai metode *authentication* di dalam REST API
4. Mampu membuat *Open* API *documentation* dari REST API yang akan dibuat

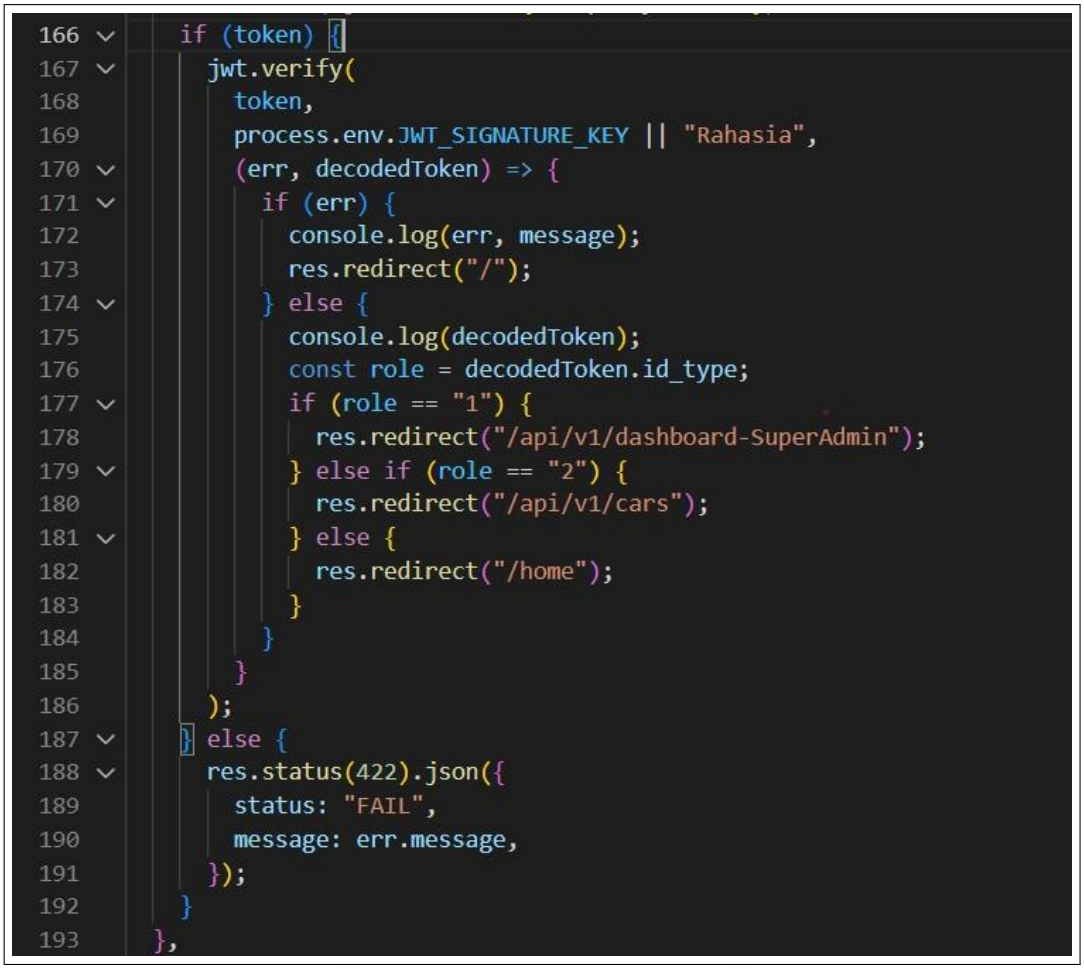
F.2Implementasi

Pada proses pembuatan REST API untuk melakukan manajemen data mobil dengan fitur *authentication* dilakukan secara berkelompok. Langkah pertama, kelompok membuat *service repository pattern* terlebih dahulu.



Gambar 3.21. Implementasi JWT (*JSON Web Token*) untuk *authentication*

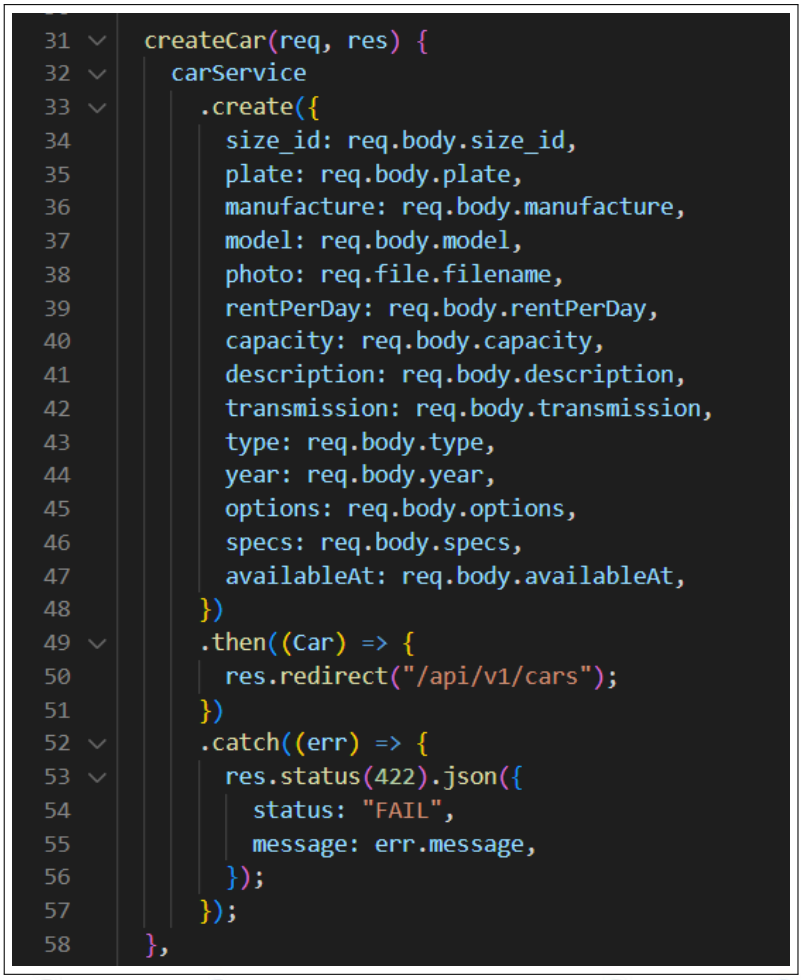
Pada gambar 3.21, terdapat sebuah folder *controllers*, *models*, *repositories*, *services*, dan *views* yang merupakan hasil implementasi dari *design pattern service repository*. Di bandingkan dengan *design pattern* MVC perbedaannya hanya di folder *services* dan *repositories*. Folder *services* berfungsi untuk menyimpan file yang berisi tentang logika bisnis dari sebuah aplikasi. Sedangkan, folder *repositories* dipakai untuk menyimpan file yang mengabstraksi *query* ke dalam *database*.



```
166   if (token) {
167     jwt.verify(
168       token,
169       process.env.JWT_SIGNATURE_KEY || "Rahasia",
170       (err, decodedToken) => {
171         if (err) {
172           console.log(err, message);
173           res.redirect("/");
174         } else {
175           console.log(decodedToken);
176           const role = decodedToken.id_type;
177           if (role == "1") {
178             res.redirect("/api/v1/dashboard-SuperAdmin");
179           } else if (role == "2") {
180             res.redirect("/api/v1/cars");
181           } else {
182             res.redirect("/home");
183           }
184         }
185       }
186     );
187   } else {
188     res.status(422).json({
189       status: "FAIL",
190       message: err.message,
191     });
192   }
193 },
```

Gambar 3.22. Implementasi JWT (*JSON Web Token*) untuk *authentication*

Pada gambar 3.22, menampilkan implementasi dari JWT untuk *authentication*. Di baris ke-167 terdapat *function* bawaan dari JWT yaitu *verify*. Fungsi *verify* berguna untuk mengecek apakah token yang dipakai adalah asli dan benar kebenarannya. Di dalam fungsi *verify* juga terdapat *callback function* yang berguna untuk pengecekan. Jika token tersebut tidak lolos terbukti kebenaran dan keasliannya, maka akan memunculkan *error message*. Lalu, di baris ke-177 jika kebenaran dan keasliannya terbukti, maka akan mengecek berdasarkan *role* yang diberikan. Ketika *role*nya bernilai 1 maka akan ditetapkan sebagai *superadmin*, jika bernilai 2 maka akan ditetapkan sebagai *admin*, selain dari angka 1 dan 2 akan ditetapkan sebagai user.



```
createCar(req, res) {
  carService
    .create({
      size_id: req.body.size_id,
      plate: req.body.plate,
      manufacture: req.body.manufacture,
      model: req.body.model,
      photo: req.file.filename,
      rentPerDay: req.body.rentPerDay,
      capacity: req.body.capacity,
      description: req.body.description,
      transmission: req.body.transmission,
      type: req.body.type,
      year: req.body.year,
      options: req.body.options,
      specs: req.body.specs,
      availableAt: req.body.availableAt,
    })
    .then((Car) => {
      res.redirect("/api/v1/cars");
    })
    .catch((err) => {
      res.status(422).json({
        status: "FAIL",
        message: err.message,
      });
    });
},
```

Gambar 3.23. Implementasi *create* mobil

Langkah selanjutnya, membuat CRUD untuk mengatur data mobil-mobil. Pada gambar 3.23 terdapat fungsi untuk menambahkan data mobil. Data mobil akan diambil menggunakan *request body* dari *input*an *user*. Ketika berhasil ditambahkan, akan dilempar ke halaman yang menampilkan data dari semua mobil. Saat terjadi *error*, akan ditangkap fungsi *catch* dan halaman tersebut akan merespon status 422 sekaligus menampilkan *error message*.

```
async updateCar(req, res) {
  carService
    .update(req.params.id, {
      size_id: req.body.size_id,
      plate: req.body.plate,
      manufacture: req.body.manufacture,
      model: req.body.model,
      photo: req.file.filename,
      rentPerDay: req.body.rentPerDay,
      capacity: req.body.capacity,
      description: req.body.description,
      transmission: req.body.transmission,
      type: req.body.type,
      year: req.body.year,
      options: req.body.options,
      specs: req.body.specs,
      availableAt: req.body.availableAt,
    })
    .then(() => {
      res.redirect("/api/v1/cars");
    })
    .catch((err) => {
      res.status(422).json({
        status: "FAIL",
        message: err.message,
      });
    });
},
```

Gambar 3.24. Implementasi *update* mobil

Setelah membuat fungsi menambahkan data mobil selesai, lalu kelompok membuat fungsi *update* mobil. Untuk implementasinya dapat dilihat pada gambar 3.24, secara keseluruhan kode yang ditulis sama dengan fungsi menambahkan data mobil.

```
89   async showCar(req, res) {
90     if (req.User.id_type === 1 || req.User.id_type === 2) {
91       carService
92         .get(req.params.id)
93         .then((Car) => {
94           res.status(200).json({
95             status: "OK",
96             data: Car,
97           });
98         })
99         .catch((err) => {
100          res.status(422).json({
101            status: "FAIL",
102            message: err.message,
103          });
104        });
105    } else {
106      res.status(403).json({
107        status: "FAIL",
108        message: "You are not authorized to perform this action",
109      });
110    }
111  },
```

Gambar 3.25. Implementasi *read* mobil

Fungsi selanjutnya akan menampilkan mobil berdasarkan id mobil tersebut. Sebelum menampilkan data mobil, terjadi pengecekkan *role* pada baris ke-90, yang dapat mengakses fungsi ini hanya *superadmin* dan *admin*. Jika tidak lolos saat pengecekka role, server akan merespon 403 dan jika lolos dari pengecekkan *role*, akan memanggil fungsi *get* untuk mendapatkan data dari id mobil. Jika berhasil halaman akan menampilkan data mobil dan merespon 200. Jika gagal halaman akan menangkap *error*, menampilkan *error message* dan server merespon 422.

```
113     async destroyCar(req, res) {
114       carService
115         .delete(req.params.id)
116         .then((Car) => {
117           res.redirect("/api/v1/cars");
118         })
119         .catch((err) => {
120           res.status(422).json({
121             status: "FAIL",
122             message: err.message,
123           });
124         });
125     },
126   };
```

Gambar 3.26. Implementasi *delete* mobil

Pada gambar 3.26 terdapat fungsi *delete* yang berfungsi untuk menghapus data mobil. Fungsi ini mirip seperti fungsi *read* mobil di gambar 3.25, karena keduanya sama-sama membutuhkan id dari mobil untuk melakukan langkah selanjutnya. Perbedaannya hanya tidak ada pengecekkan *role* di fungsi *delete* mobil dan tidak memanggil fungsi *read* melainkan *delete*.

G.Pembelajaran React.js, *React routing* dan *backend integration*, OAuth, redux

Pada pembelajaran di bab ini, terdapat 4 materi yang disampaikan oleh fasilitator. Pada materi pertama dibagi lagi menjadi 4 poin. Poin pertama mengajarkan tentang pemahaman dasar React.js. Poin kedua memahami konsep *component, state, serta property*. Poin ketiga memahami konsep dan cara melakukan *styling*. Poin keempat mengenal UI *framework* dan cara menggunakannya. Setelah itu, materi kedua membahas tentang React *routing* & *backend integration* materi ini dibagi menjadi 4 poin, yaitu memahami *routing* pada React, memahami konsep HTTP *request*, konsep & cara melakukan *file processing*, mengembangkan fitur *authentication* pada aplikasi React, dan konsep serta implementasi *private route*. Selanjutnya materi ketiga mengenai OAuth, materi ini mempelajari konsep OAuth, Google/-

Facebook OAuth. Terakhir, materi redux mempelajari tentang pemahaman dasar *state management*, memahami fungsi redux, cara menggunakkan redux *tool chain*, implementasi redux dalam React, mengenal redux thunk.

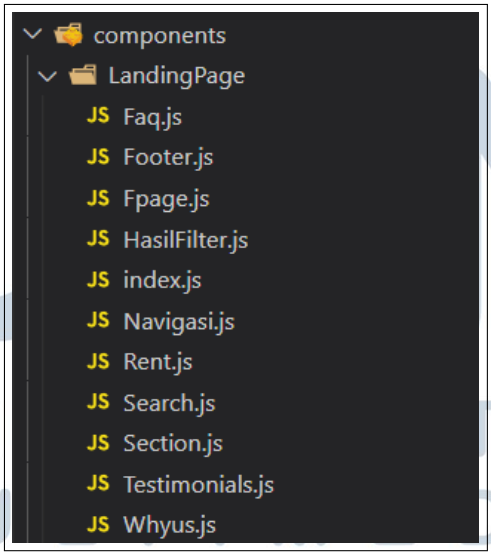
G.1Kriteria pembuatan *website* rental mobil menggunakkan React

Fasilitator meemberikan tugas membuat *website* rental mobil menggunakkan React yang *endpoint*nya sudah disediakan. Dengan kriteria yang diberikan, yaitu:

1. Mampu membuat aplikasi React menggunakkan OAuth
2. Mampu memyambungkan aplikasi React dengan sebuah *backend* menggunakan protokol HTTP

G.2Implementasi

Pada proses pembuatan *website* rental mobil dengan menggunakkan React dikerjakan secara individu. Langkah pertama adalah membuat aplikasi React dan membuat folder *components* supaya pengerjaan lebih efisien dan efektif.

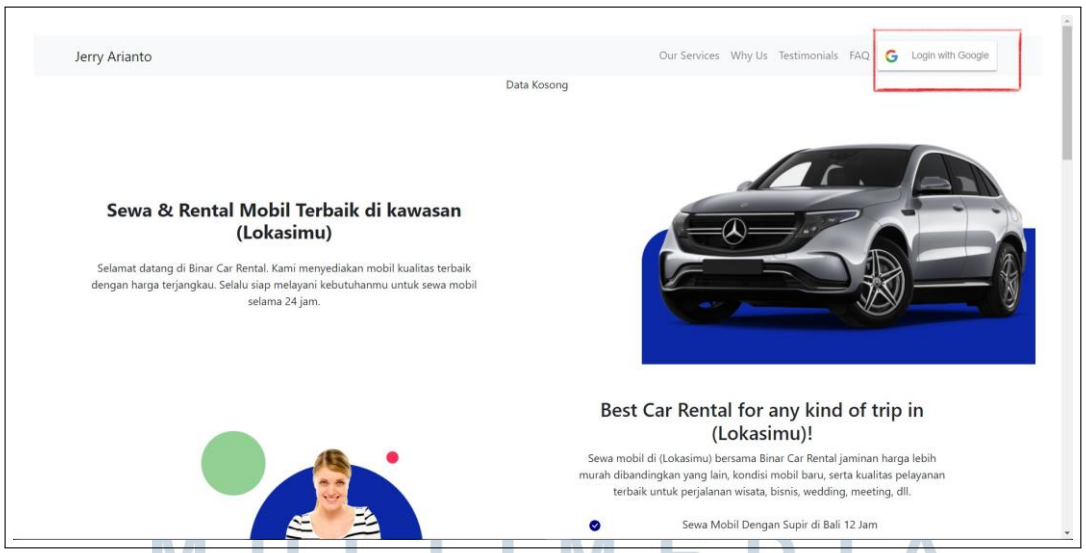


Gambar 3.27. Membuat folder *components*

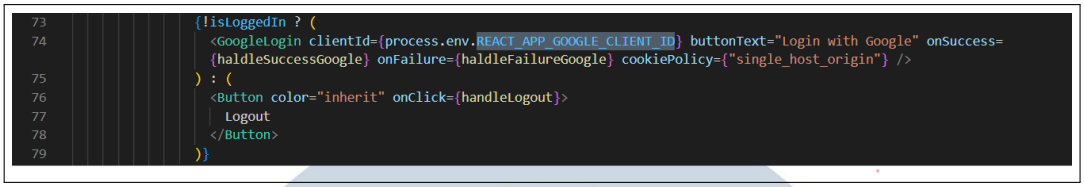
```jsx
function App() {
  return (
    <div className="App" style={{ padding: "30px" }}>
      {/* <LandingPage /> */}
      <Navigasi />
      <Fpage />
      <Section />
      <Whyus />
      <Testimonials />
      <Rent />
      <Faq />
      <Footer />
    </div>
  );
}

export default App;
```

Gambar 3.28. Memanggil *components*

Pada gambar 3.27, sudah ada folder *components* yang di dalamnya terdapat kode untuk menampilkan *section-section* atau fitur. Selanjutnya akan dipanggil dengan *file* app.js pada gambar 3.28 sehingga proses pengerjaan lebih rapih.



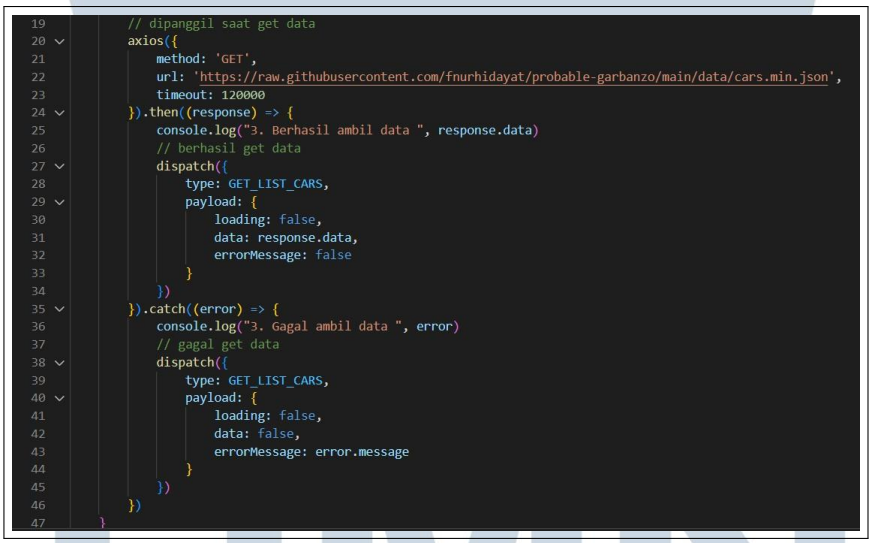
Gambar 3.29. Tampilan OAuth



```
{!isLoggedIn ? (
  <GoogleLogin clientId={process.env.REACT_APP_GOOGLE_CLIENT_ID} buttonText="Login with Google" onSuccess=
  {haldleSuccessGoogle} onFailure={haldleFailureGoogle} cookiePolicy={"single_host_origin"} />
) : (
  <Button color="inherit" onClick={handleLogout}>
    Logout
  </Button>
)}
```

Gambar 3.30. Implementasi OAuth

Kemudian, langkah selanjutnya mengimplementasi OAuth. OAuth yang digunakan adalah Google *Authenticator*. Fitur ini akan bekerja jika *user* mempunyai akun Google. Pada gambar 3.30, kode di baris ke-73 terdapat *decision making* untuk mengecek *user* sudah *login* atau belum. Ketika *user* belum *login*, *button* akan menjadi ”*login*”. Sebaliknya, ketika *user* sudah *login button* akan menjadi ”*logout*”



```
    // dipanggil saat get data
    axios({
        method: 'GET',
        url: 'https://raw.githubusercontent.com/fnurhidayat/probable-garbanzo/main/data/cars.min.json',
        timeout: 120000
    }).then((response) => {
        console.log("3. Berhasil ambil data ", response.data)
        // berhasil get data
        dispatch({
            type: GET_LIST_CARS,
            payload: {
                loading: false,
                data: response.data,
                errorMessage: false
            }
        })
    }).catch((error) => {
        console.log("3. Gagal ambil data ", error)
        // gagal get data
        dispatch({
            type: GET_LIST_CARS,
            payload: {
                loading: false,
                data: false,
                errorMessage: error.message
            }
        })
    })
}
```

Gambar 3.31. Implementasi *backend integration*

Terakhir, terdapat *backend integration* ini berfungsi untuk menarik API dari *endpoint* yang sudah disediakan. Data-data mobil akan muncul jika *user* sudah login. Pada gambar 3.31, kode di baris ke-20 menggunakkan *library* axios untuk membantu proses pengerjaan lebih mudah. Fungsi axios yang digunakkan mempunyai parameter *method* = GET, url yang sudah disediakan, serta *timeout* = 120000. Ketika berhasil, akan menampilkan pesan ”Berhasil ambil data ” di *console*. Lalu, ketika *error* akan ditangkap dan menampilkan pesan ”Gagal ambil data”.

H.Pembelajaran *websocket, next*.js, *media handling, eslint,* TDD (*testing, driven, developement*), *deployment* & CI/CD

Pada pembelajaran di bab ini, fasilitator membagi pembelajaran menjadi 6 materi, yaitu *websocket*, *next*.js, *media handling*, *eslint*, TDD (*testing*, *driven*, *developement*), *deployment* & CI/CD. Pada materi websocket atau materi pertama terdapat 2 poin. Poin pertama adalah pengenalan konsep *websocket* dan fungsinya, serta poin terakhir memahami contoh implementasi *Socket* IO. Materi tentang Next.js terdapat 4 poin. Poin pertama, pengenalan konsep dan fungsi SSR. Poin kedua, pengenalan Next.js. Poin selanjutnya, fitur-fitur pada Next.js dan poin terakhir penerapan Next.js. Pada materi ketiga membahas tentang *media handling*, fasilitator mengajarkan memahami cara melakukan *upload* dan penyimpanan *file* pada *website* dari *backend* serta *frontend*. Lalu, materi keempat membahas tentang konsep ESlint dan melakukan instalasi bersama konfigurasi ESlint di project Javascript. Materi kelima, pada materi ini membahas tentang konsep & jenis-jenis *testing*, pengenalan konsep TDD, melakukan testing di Javascript dengan Jest & Supertest, dan integrasi *testing* dengan Supertest. Pada materi terakhir, fasilitator mengajarkan konsep *deployment*, konsep CI/CD, implementasi *deployment* ke Heroku, dan implementasi CI/CD.

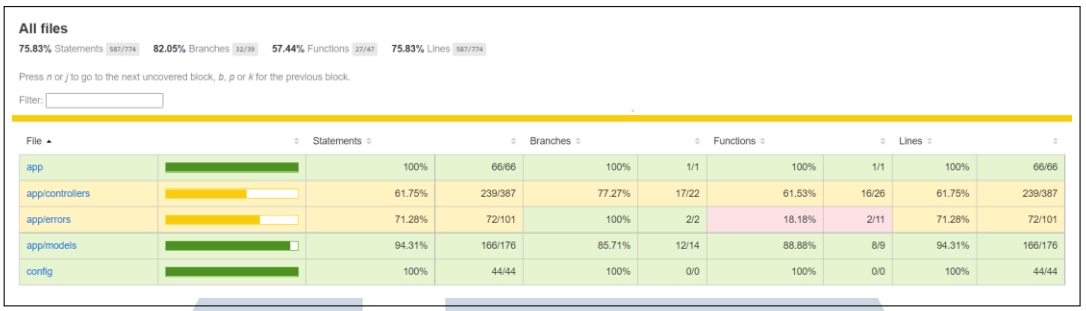
H.1Kriteria pengerjaan *hosting website*

Pada bab ini, fasilitator memberikan tugas untuk meng*hosting* sebuah *website*. Dengan kriteria yang diberikan, yaitu:

1. Melakukan unit testing pada kode yang ditulis
2. Melakukan *deployment* menggunakan Heroku

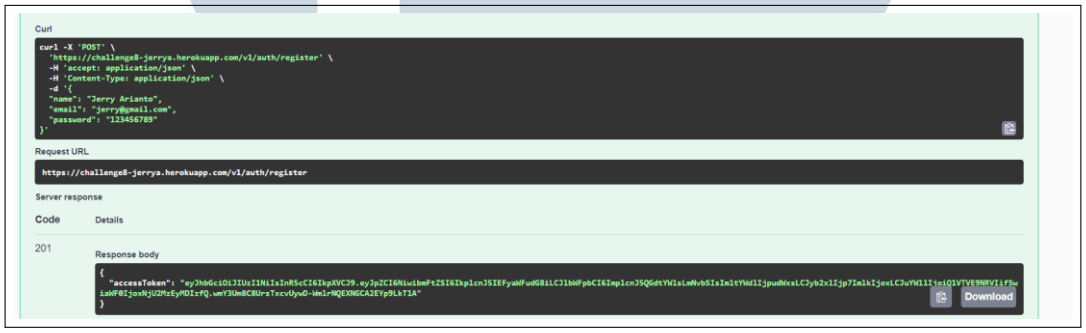
H.2Implementasi pengerjaan *hosting website*

Pada pengerjaan *hosting website* dikerjakan secara individu. Langkah awal untuk pembuatan *hosting website* adalah *unit testing* agar dapat mengecek kode-kode yang harus diperbaiki.



Gambar 3.32. Hasil *unit testing* dari *library supertest*

Pada gambar 3.32, menampilkan hasil *unit testing* dari supertest. Setelah dilihat dari tabel, hasil yang didapatkan untuk keseluruhan adalah 75.83%. Untuk folder *controllers* dan *errors* masih dapat dikembangkan dengan baik lagi dengan nilai 61.75% dan 71.28%.



Gambar 3.33. Hasil *deployment* menggunakan fitur *register*

Pada gambar 3.33, menampilkan hasil dari proses *register* dengan menggunakan *website* yang di*hosting*. Setelah mengirim *request* ke *server*, server menjawab dengan *response* 201 yang artinya *email* sudah terdaftar pada *database*.

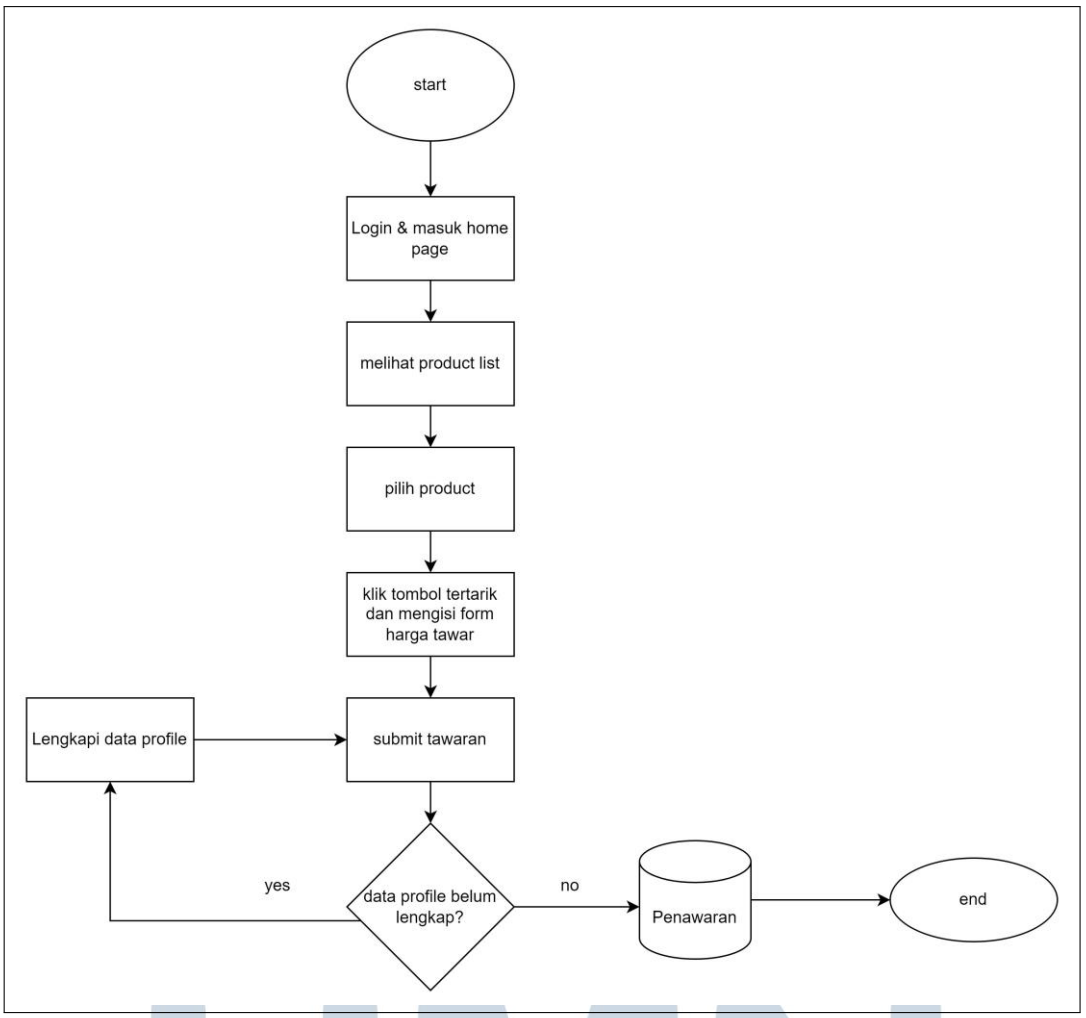
Gambar 3.34. Hasil *deployment* menggunakan fitur *login*

Setelah selesai *register*, proses selanjutnya adalah *login*. Saat *user* mencoba

login dengan menggunakan data yang telah terdaftar di *database*, maka *user* mendapat *response* 201 yang berarti berhasil *login* ke dalam *website*.

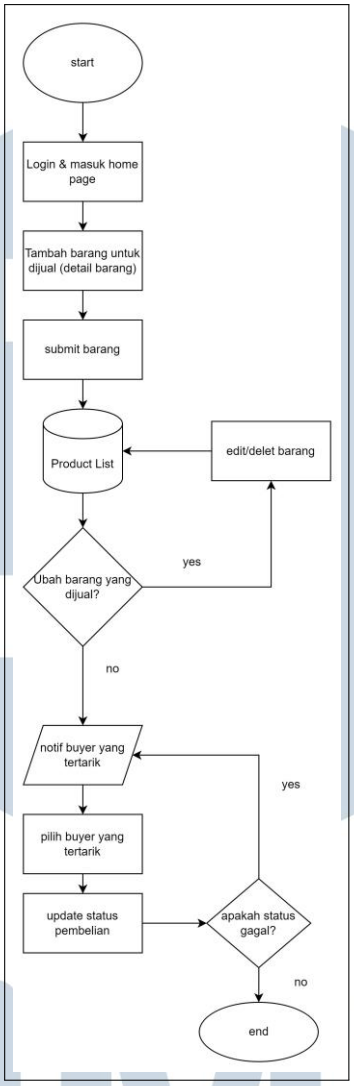
I.Proyek Akhir - Pembuatan *website e-commerce* sebagai *Backend Developer*

Salah satu syarat untuk lulus dalam kelas *fullstack website developer* di Binar Academy adalah mengerjakan proyek akhir. Tugas-tugas yang diberikan dari materi ke-1 sampai ke-8 merupakan bekal dalam pengerjaan proyek akhir. Proyek akhir dikerjakan secara berkelompok beranggotakan 6 orang yang dibagi 2 peran, yaitu frontend dan *backend*. Dalam pengerjaan proyek akhir diberi waktu selama 6 minggu (3 *sprint*) untuk membuat sebuah produk digital, berupa aplikasi *website* berdasarkan fitur (MVP) dan persyaratan teknis yang sudah ditentukan. Sebelum melakukan pengerjaan proyek akhir, kelompok menentukan *project owner* dan *scrum master* terlebih dahulu untuk dapat mengatur arah pengerjaan kelompok. *Website e-commerce* yang dikerjakan memiliki nama SecondHand. Sesuai dengan namanya *website* ini merupakan tempat jual-beli barang secara *online*, khususnya barang bekas. *Website* ini membuka dan menyediakan berbagai jenis kategori kebutuhan. *User* yang mendaftarkan diri pada aplikasi ini dapat berperan sebagai *seller* dan *buyer* dengan menggunakan 1 (satu) akun yang sama. *Website* ini akan mempertemukan *seller* dan *buyer* untuk dapat melakukan negosiasi barang dan melakukan transaksi langsung di luar platform.



Gambar 3.35. Flowchart sebagai *buyer*

Pada gambar 3.35, merupakan alur ketika *user* sukses *login* sebagai *buyer*. Ketika sudah login, *user* akan melihat daftar-daftar produk dari yang sudah didaftarkan *seller* ke *website e-commerce*. Jika *buyer* tertarik dengan produk yang dijual, buyer dapat menekan tombol ”menarik” dan akan menginput data ke dalam *database*, seperti harga penawaran. Ketika sudah selesai menginput, *buyer* dapat langsung *submit* tawaran. Setelah *submit* tawaran, terdapat *decision making*. Jika data profil belum lengkap, maka *buyer* secara paksa harus melengkapi data profilnya terlebih dahulu. Jika, *buyer* sudah melengkapi data profil, maka data yang diinput langsung disimpan ke dalam *database*.



Gambar 3.36. Flowchart sebagai *seller*

Pada gambar 3.36, merupakan alur ketika *user* sukses login sebagai *seller*. Saat *seller* berhasil login, maka tampilan pertama yang dilihat adalah *home page*. Seller dapat melakukan memanajemen produk (*create, read, update, dan delete*) di dalam *e-commerce* SecondHand. Ketika *seller* menambahkan data produk ke dalam *database* akan terjadi *decision making*. Jika *seller* ingin merubah barang yang dijual maka *seller* dapat *delete*/*edit* produk tersebut. Jika tidak, *seller* bisa menunggu sampai ada *buyer* yang minat dengan produk tersebut. Jika ada *buyer* yang minat dengan produk tersebut, secara sistem akan memunculkan notifikasi yang akan dikirim ke *email seller*. Setelah itu, *seller* berhak memilih/tidak memilih *buyer* serta terdapat *decision making*. Jika seller merasa tidak sesuai dengan harga yang ditawarkan, maka *seller* akan meng*update* status pembayaran menjadi ”gagal”

secara manual. Ketika *seller* puas dengan harga yang ditawarkan oleh *buyer*, maka *seller* secara manual akan meng*update* status pembayaran menjadi ”berhasil”.

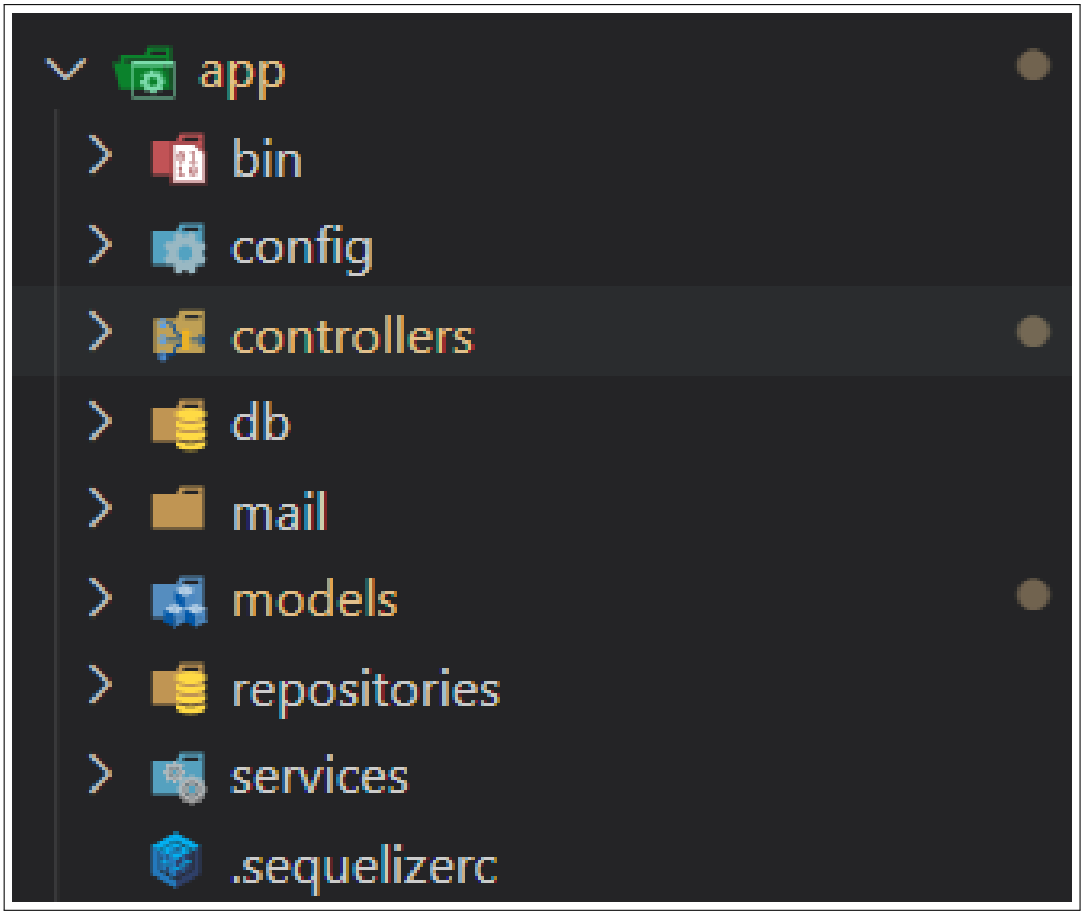
I.1Kriteria pembuatan *website* SecondHand sebagai *backend*

Berikut merupakan kriteria-kriteria yang diberikan untuk pembuatan *website* SecondHand sebagai backend, yaitu:

1. Menggunakan RESTful API
2. Menggunakan *design pattern* MVC(*model, service, controller* atau *Service Repository Pattern*
3. Menggunakkan *unit testing* Jest
4. Mengimplementasikan *deployment* dengan Heroku
5. Membuat API yang dapat diakses secara publik

I.2Implementasi pengerjaan *website* SecondHand

Proses pembuatan *website* SecondHand dilakukan secara berkelompok beranggotakan 6 orang yang dibagi menjadi tim *frontend* dan *backend*. Langkah awal, dalam pembuatan *website* SecondHand adalah menentukan *design pattern*.



Gambar 3.37. Implementasi Service Repository Pattern

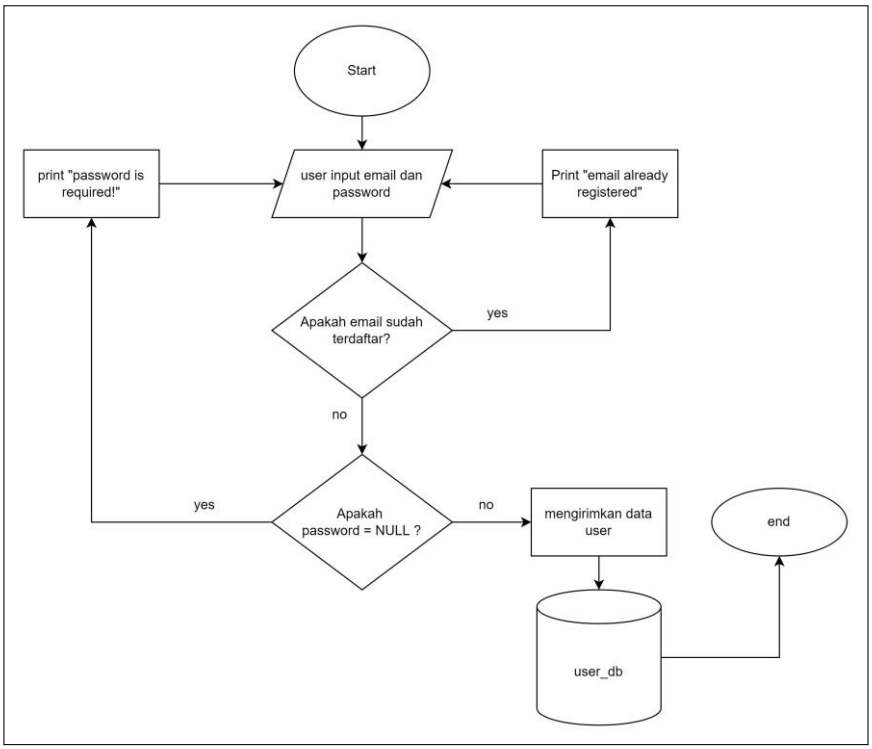
Pada gambar 3.37, menampilkan *Design pattern* yang digunakan, yaitu *Service Repository Pattern* karena menurut kelompok *Service Repository Pattern* dinilai lebih rapih daripada *design pattern* MVC. *Service Repository Pattern* tidak beda jauh dengan MVC (*model, view, controller*), hanya menambahkan folder *repository* dan *service*. Sehingga dalam pembuatan *code* di folder *controller* jauh lebih rapih dibanding menggunakkan *design pattern* MVC (*model, view, controller*).

Gambar 3.38. Implementasi *Config Database*

Pada gambar 3.38, menampilkan hasil konfigurasi *database* menggunakan *library* Sequelize. Untuk konfigurasinya, kelompok menggunakan *database* Postgre, password dan host yang disesuaikan oleh *databasenya*, dan menggunakan SecondHand_DB sebagai *database*. Sama seperti tugas pembelajaran ke-5 tentang (*database*), menggunakan perintah ”sequelize db:create” untuk membuat *database* secara langsung.

```
49    async register(req, res) {
50      const name = req.body.name;
51      const email = req.body.email;
52
53      // Check if user already exists
54      const existedUser = await userService.findId(email);
55      if (existedUser) {
56        return res.status(400).send({
57          status: "FAIL",
58          message: "Email telah digunakan",
59        });
60      }
61
62      const password = await encryptPassword(req.body.password);
63      if (req.body.password === "") {
64        res.status(422).json({
65          status: "FAIL",
66          message: "Password is required",
67        });
68        return;
69      }
70
71      userService
72        .create({ name, email, password })
73        .then((user) => {
74          res.status(201).json({
75            status: "REGISTER_SUCCESS",
76            data: user,
77          });
78        })
79        .catch((err) => {
80          res.status(422).json({
81            status: "FAIL",
82            message: err.message,
83          });
84        });
85    },
```

Gambar 3.39. Implementasi *register*

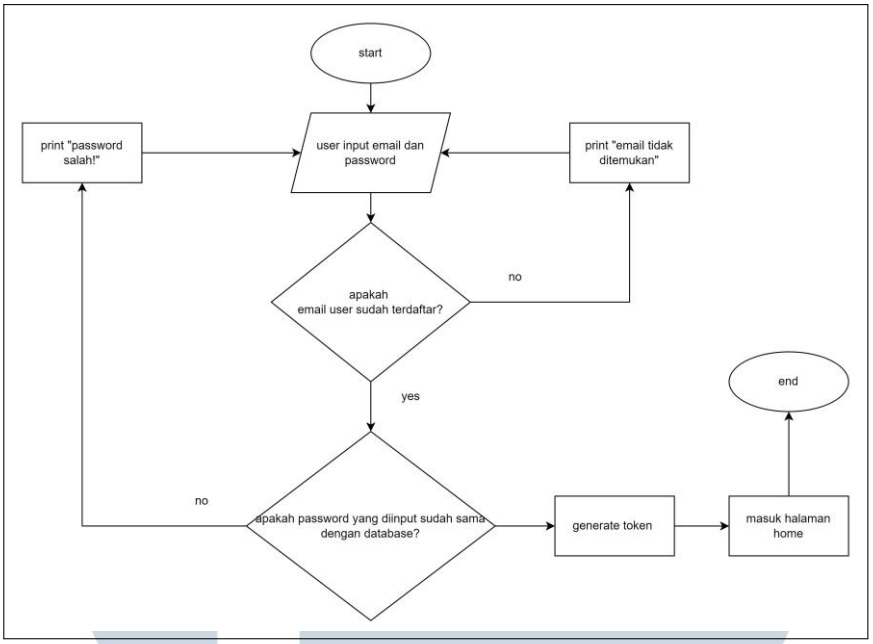


Gambar 3.40. *Flowchart register*

Pada gambar 3.39, menampilkan potongan kode untuk menggunakan fitur *register*. Di baris ke-54 sampai baris ke-60 merupakan kode untuk mengecek apakah *email* sudah terdaftar sebelumnya atau belum. Kemudian di baris ke-62 berfungsi untuk mengenkripsi *password* untuk meningkatkan keamanan sistem yang ada di *website* SecondHand. Untuk baris ke-63 sampai ke-69 merupakan kode untuk pengecekkan *password*. Jika *password* bernilai NULL, maka sistem akan me*return response* 422, dengan *status* bernilai *fail*, dan memberikan pesan ”*password is required*”. Di baris ke-71 sampai ke-84, menampilkan kode yang berfungsi untuk mendaftarkan *user* ke dalam *database*. Jika, data-data *user* berhasil ditambahkan ke dalam *database*, maka *server* akan mengirim *response* 201. Kemudian, jika *user* gagal dalam melakukan *register*, maka *server* akan me*response* 422.

```
87  async login(req, res) {
88    const email = req.body.email.toLowerCase(); // Biar case insensitive
89    const password = req.body.password;
90
91    let user = await User.findOne({
92      where: { email },
93    });
94    if (!user) {
95      res.status(404).json({ message: "Email tidak ditemukan" });
96      return;
97    }
98
99    const isPasswordCorrect = await checkPassword(user.password, password);
100   if (!isPasswordCorrect) {
101     res.status(401).json({ message: "Password salah!" });
102     return;
103   }
104
105   user = JSON.parse(JSON.stringify(user));
106   delete user.password;
107
108   const token = createToken(user);
109
110   res.status(200).json({
111     status: "LOGIN_SUCCESS",
112     token,
113     user,
114   });
115 },
```

Gambar 3.41. Implementasi *login*



Gambar 3.42. *Flowchart login*

Pada gambar 3.40, menampilkan potongan kode untuk menggunakan fitur *login*. Fitur *login* hanya bisa digunakan ketika *user* sudah mengirimkan data-datanya ke dalam *database*. Pertama, membuat *variable* bernama *email* dan *password* yang akan menangkap nilai yang diinput *user* yang ditampilkan di baris ke-88 dan baris ke-89. Kemudian, di baris ke-91 dan ke-97 terdapat kode untuk pengecekkan *email* yang diinput *user* sudah terdaftar atau belum. Jika, belum *server* akan mengirimkan *response* 404 dan pesan ”*email* tidak ditemukan”. Di baris ke-99 dan baris ke-103 merupakan kode untuk mengecek *password* yang diinput sudah sama dengan *database* atau belum. Ketika *password* yang diinput dengan di *database* tidak cocok, maka server akan mengirimkan *response* 401 dan pesan ”password salah!”. Pada baris ke-108, terdapat fitur yang dapat memberitahu apakah *user* sudah *login* atau belum. Fitur ini dapat men*generate token* setiap detiknya, yang artinya *token* ini bersifat unik. Ketika *user* sudah berhasil *login*, maka server akan menampilkan *response* 200 dan mengirimkan data *user* serta *token*.

3.3.2 Kendala yang Ditemukan

Terdapat kendala yang ditemukan saat melakukan studi independen, antara lain:

1. Kesulitan terhadap materi-materi yang rumit seperti ESlint, *Backend integra-*

tion dan OAuth.

2. Pada saat awal-awal. kelas *online* terasa canggung sehingga sulit untuk berkomunikasi dengan teman yang lain.

3.3.3 Solusi atas Kendala yang Ditemukan

1. Untuk mengatasi masalah kesulitan untuk memahami materi yang rumit, maka dilakukan pembelajaran menggunakan pendekatan mandiri, seperti membaca dokumentasi dan mencari informasi di Google serta aktif bertanya kepada fasilitator.
2. Memberikan saran dalam bentuk *form* sebagai *anonymous* kepada fasilitator untuk menyisipkan *ice breaking* pada pertemuan awal-awal supaya suasana menjadi cair dan mendapatkan kesempatan untuk berkomunikasi dengan teman kelas.