BAB 3
PELAKSANAAN KERJA MAGANG

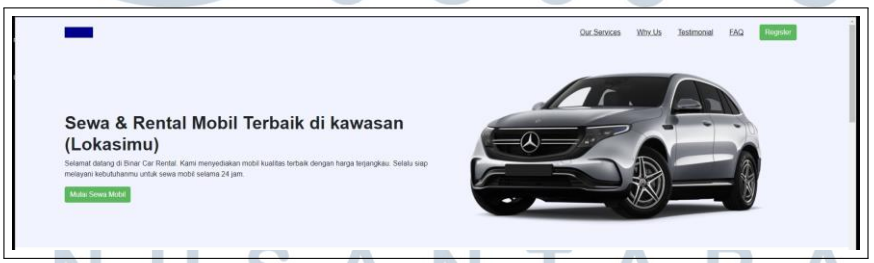
3.1 Kedudukan dan Koordinasi

Studi independen di Binar Academy dilaksanakan dengan mempelajari *Full Stack Website*. Terdapat kelas yang dibimbing oleh Kakak RaIhan Rafid, selaku fasilitator di *Full Stack Website developer* pada kelas kesatu. di Binar Academy memiliki tugas untuk mendengarkan materi dan mengerjakan tugas mingguan yang diberikan oleh fasilitator. Pelaksaaan kegiatan biasanya dilaksanakan melalui media *Zoom*, sedangkan *Discord*, dan *Telegram* biasanya digunakan untuk media komunikasi, dan juga menggunakan *github* untuk melakukan pengumpulan tugas, yang nantinya akan dicek oleh fasilitator.

3.2 Tugas yang Dilakukan

3.2.1 HTML, CSS, dan *Framework CSS*

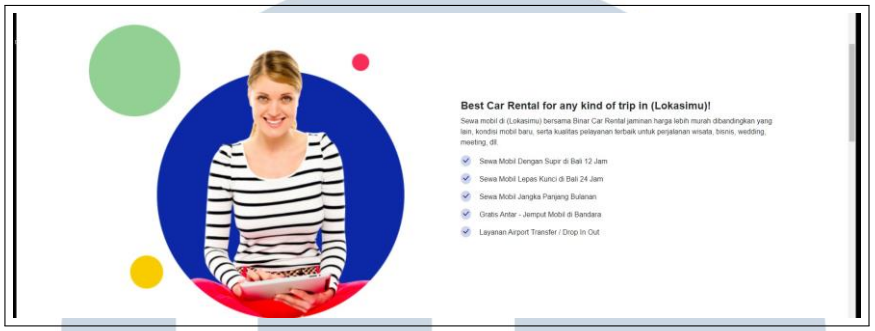
Pada *chapter* 1,mempelajari mengenai dasar-dasar *HTML* dan juga *CSS*, serta *Framework CSS*, pada *chapter* ini, fasilitator mengajarkan cara menggunakan *HTML* dan cara mengimplementasikan *CSS* baik secara *inline* maupun *external CSS*, dan dijelaskan cara untuk mengimplementasikan *Framework Boostrap* dan *Boostrap* yang digunakan adalah *Boostrap* 4, dan setelah semua materi selesai dijelaskan, fasilitator memberikan tugas untuk membuat tampilan peminjaman mobil dengan menggunakan *HTML,CSS,* dan juga *Framework Boostrap* 4, sedangkan *assetnya* disediakan dari *Figma*.



Gambar 3.1. *chapter* 1 - *Navbar*

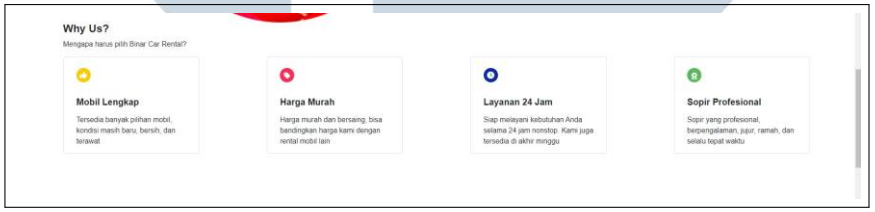
Pada *chapter* 1, membuat *landing page* yang terdiri dari beberapa komponen, yang pertama terdapat *navigation bar* yang dibuat dengan menggunakan

framework Boostrap 4, dan kita, sedangkan untuk *design* dan juga segala *asset* sudah tersedia pada Figma.



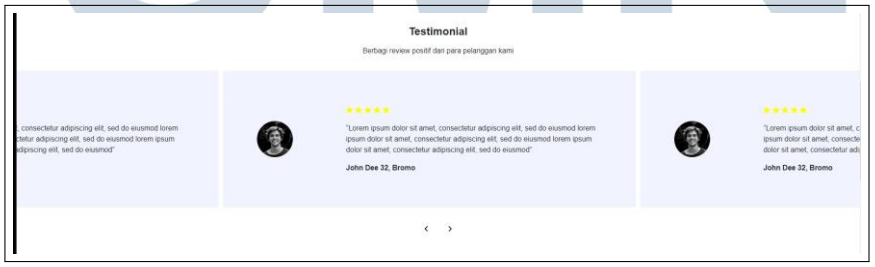
Gambar 3.2. chapter 1 - *Grid,Icon font-awesome*

Pada bagian ini, untuk memisahkan kedua belah *section* menggunakan *grid*, dan untuk *point-point* tersebut menggunakan *icon* yang di *import* dari *font-awesome*.



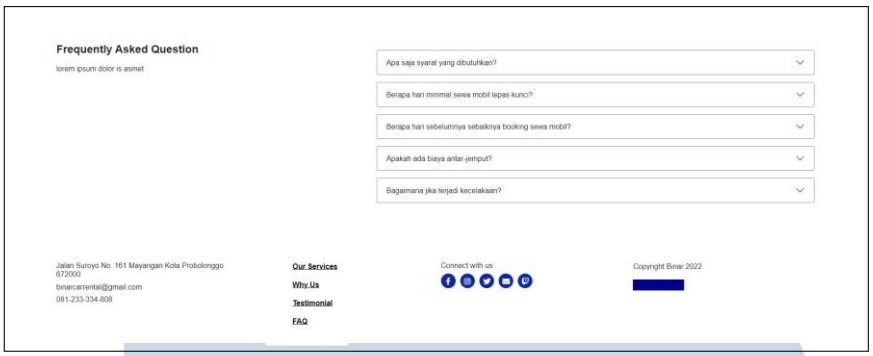
Gambar 3.3. *chapter* 1 - *Card, Row*, tampilan Why-Us?

Pada bagian *Why us*, untuk membuat kotak-kotak tersebut, menggunakan *card* lalu diberi *class row* agar tampilan menjadi lebih rapih, sedangkan untuk *icon* juga didapatkan memlalui *font-awesome*.



Gambar 3.4. Materi 1 - *Carousel*, tampilan Testimoni

Pada bagian Testimoni ini, dibuat dengan menggunakan *carousel* yang terdapat pada *Boostrap* 4, lalu di*custom* lagi menggunakan *CSS* agar, tampilan terlihat lebih rapih, sedangkan *icon* bintang, didapat melalui *font-awesome*.



Gambar 3.5. Chapter 1 - *Accordion dan Footer*

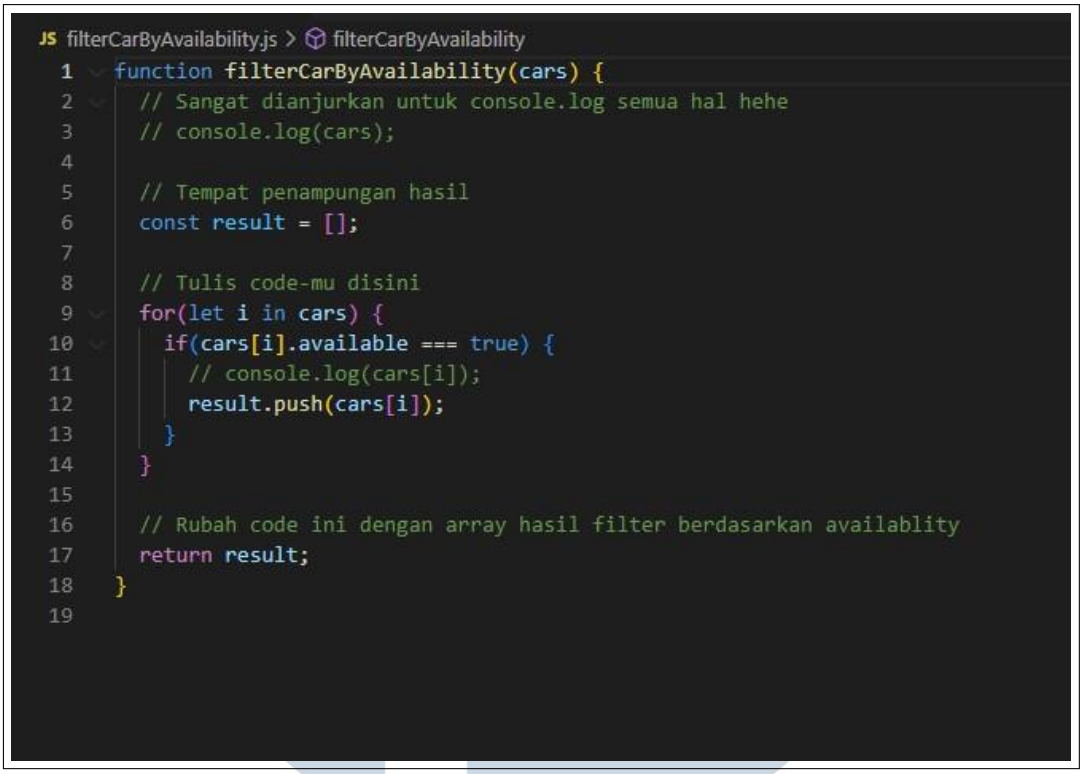
Pada bagian *FAQ*, dibuat menggunakan *accordion* yang diambil dari *Bootstrap* 4, pada bagian ini, setiap tulisan yang di klik, akan menampilkan *dropdown*., sedangkan untuk *footer* dibuat secara manual dengan ditambah *CSS* agar terlihat sesuai dengan *mock-up* yang tersedia pada Figma.

3.2.2 Javascript Introduction

Pada chapter 2, dijelaskan mengenai pemahaman dasar *javascript*, mempelejari algoritma dasar *javascript* dan fasilitator juga memberikan tugas untuk melakukan *filtering* terhadap data yang sudah disediakan, terdapat juga tugas untuk melakukan *filtering* dengan 3 kategori, yaitu tersedianya mobil atau tidak, *filtering* data berdasarkan tahun dari awal - akhir, dan *filtering* data berdasarkan tahun dari akhir - awal.

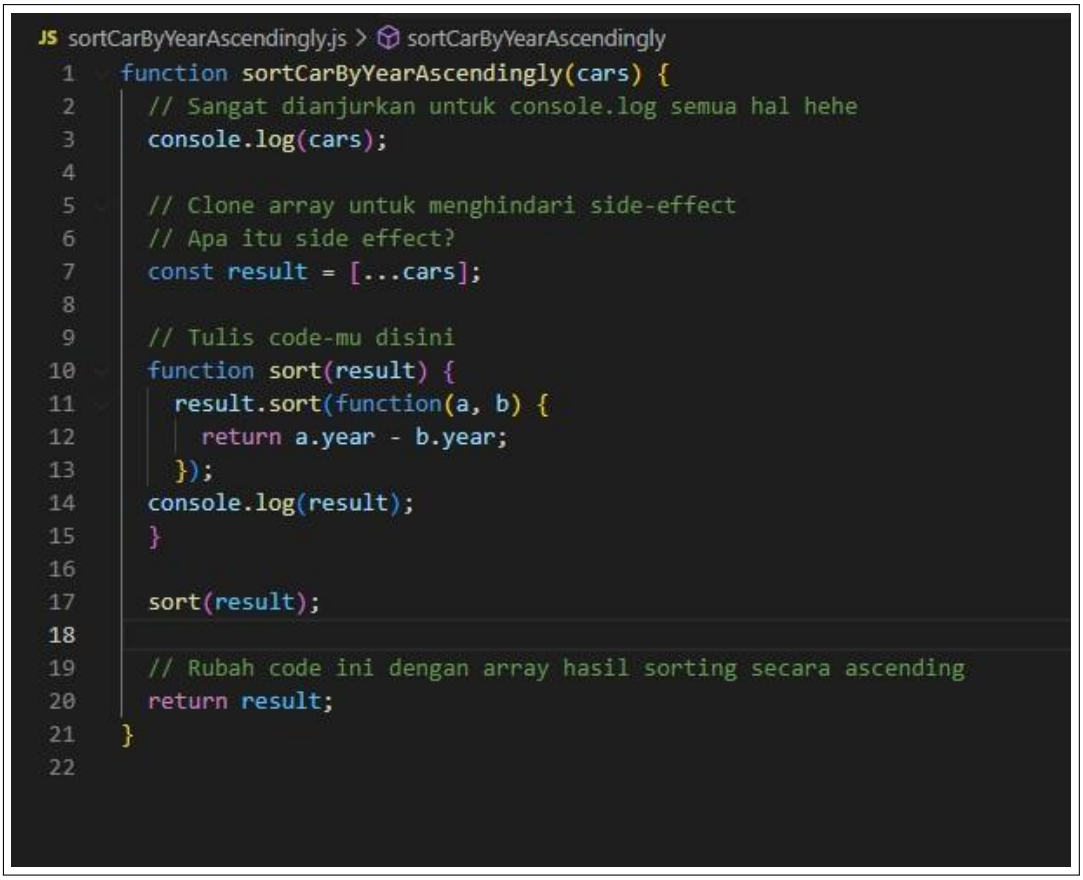
Gambar 3.6. *Chapter* 2 - *Filtering Javascript*

Pada *chapter* 2 tugas yang diberikan untuk perserta adalah melakukan 3 katergori *filter*, dan apabila hasil yang ditampilkan sudah seperti gambar diatas, berarti*filtering* data yang dilakukan berhasil.

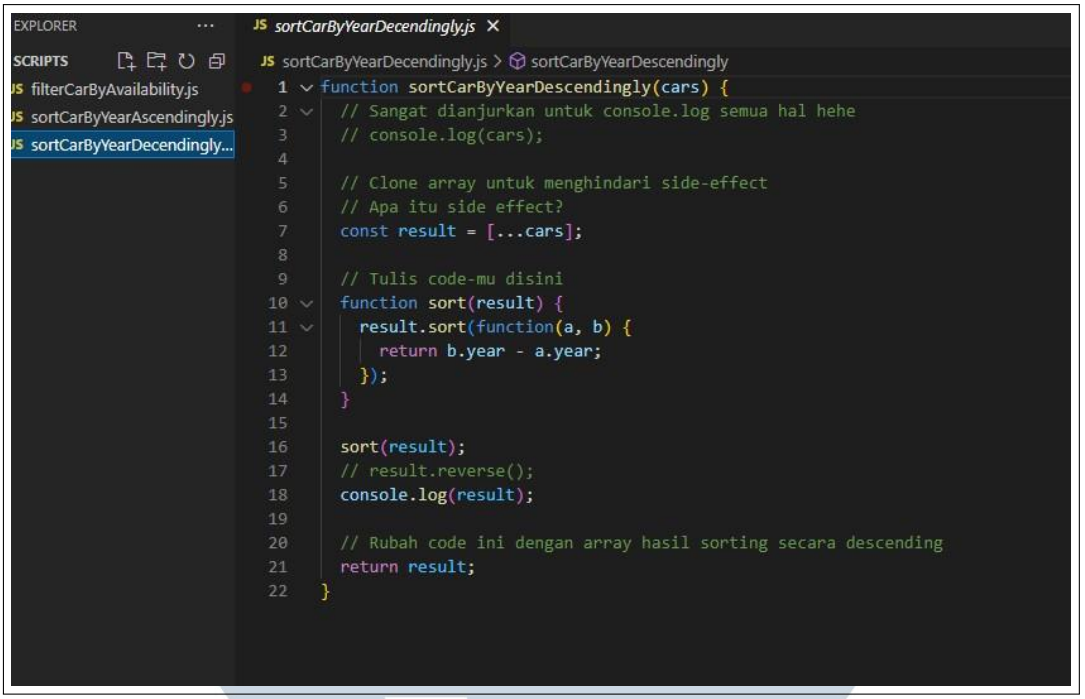


Gambar 3.7. *Chapter 2 - filterCarByAvailability*

Potongan *code* pada gamnbar 3.7 berfungsi untuk menunjukan ketersediaan mobil yang memiliki nilai *true* atau bisa dikatakan berasal dari yang memang *available*, dan hasil yang bernilai *true* akan dipush ke dalam *array result*



Gambar 3.8. *chapter* 2 - *sortCarByYearAscendingly*

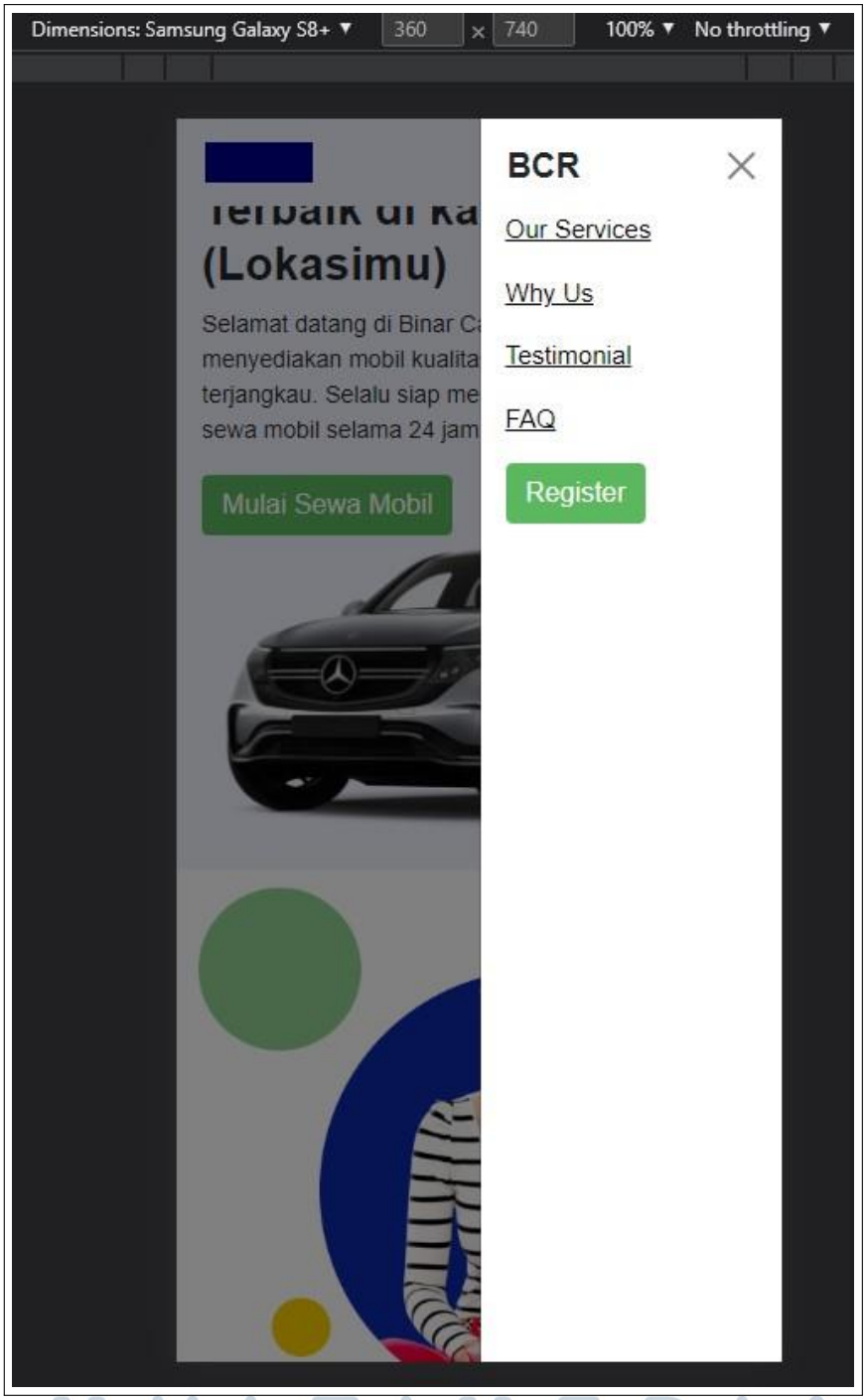


Gambar 3.9. *Chapter* 2 - *sortCarByYearDecendingly*

Potongan *code* [ada gambar 3.8 berfungsi untuk melakukan *sorting* atau menunjukan tahun mobil secara *ascending*, atau dari awal - akhir dan potongan *code* pada gambar 3.9 berfungsi untuk melakukan *sorting* atau menunjukan tahun mobil secara *descending*, atau dari akhir - awal.

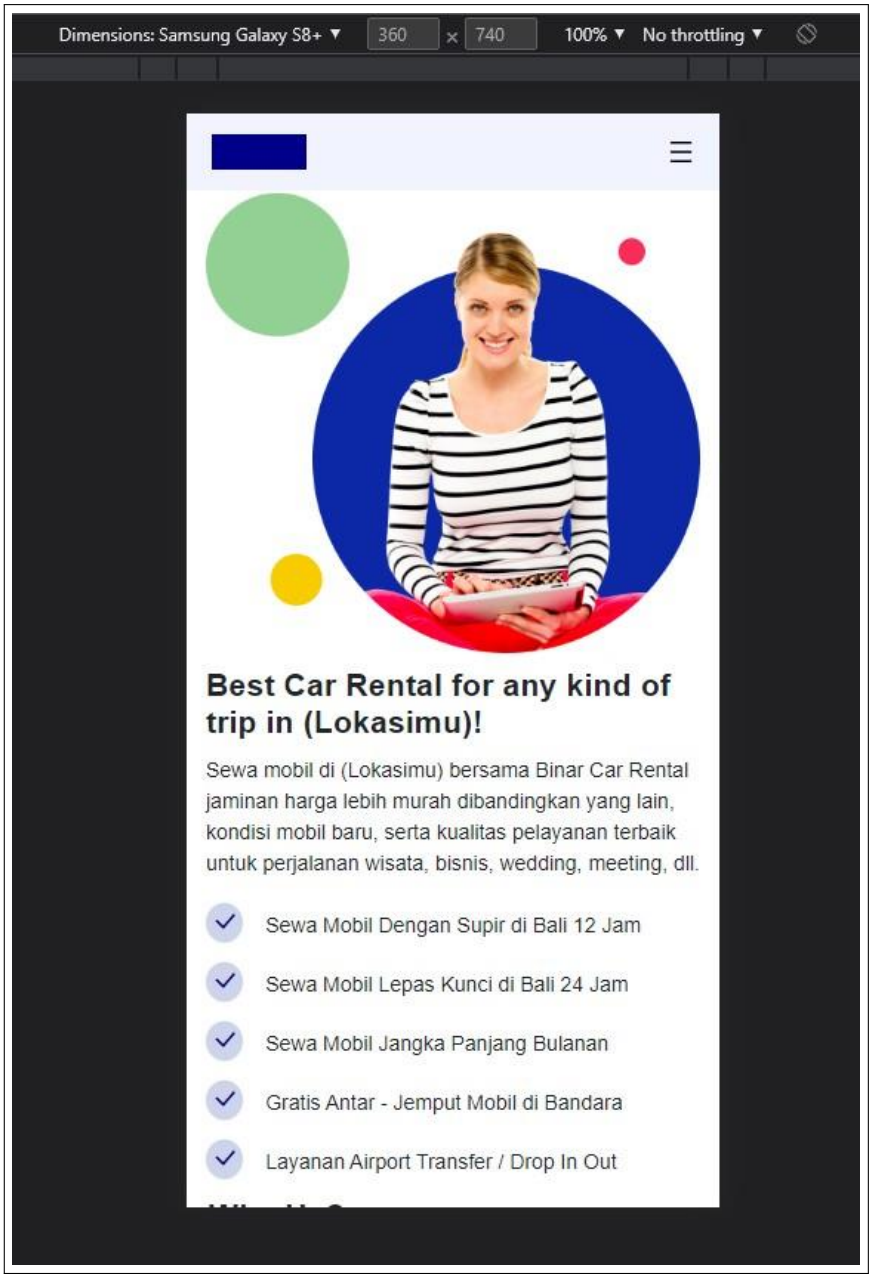
3.2.3 Responsive Website dan Gitlab

Pada *chapter* 3, dijelaskan cara untuk membuat *landing page* yang sudah dibuat pada *chapter* 1, menjadi *responsive*, yang berarti harus dapat membuat tampilan rapih juga bila tampilan diubah menjadi *handphone*, dan juga diajari untuk menggunakan *gitlab*,baik itu *push* maupun *pull* dokumen.



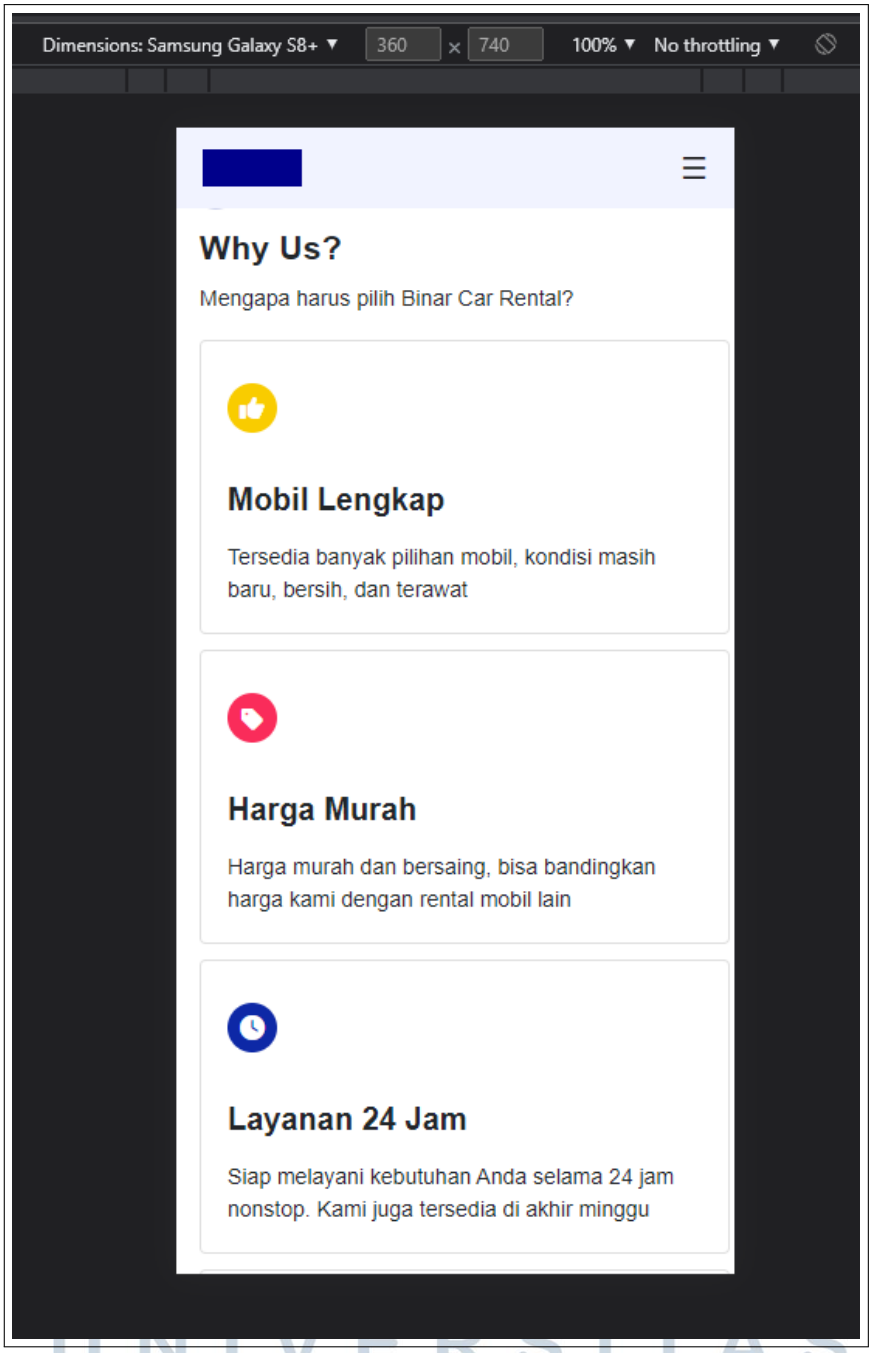
Gambar 3.10. *Chapter 3 - Navigation bar*

Gambar 3.10 menampilkan tampilan *navbar* yang berubah menjadi *side bar*, ketika diubah menjadi tampilan *mobile*. Untuk fungsi masi sama seperti sebelumnya, ketika diklik maka akan langsung melakukan *re-direct* ke bagian sesuai yang diklik



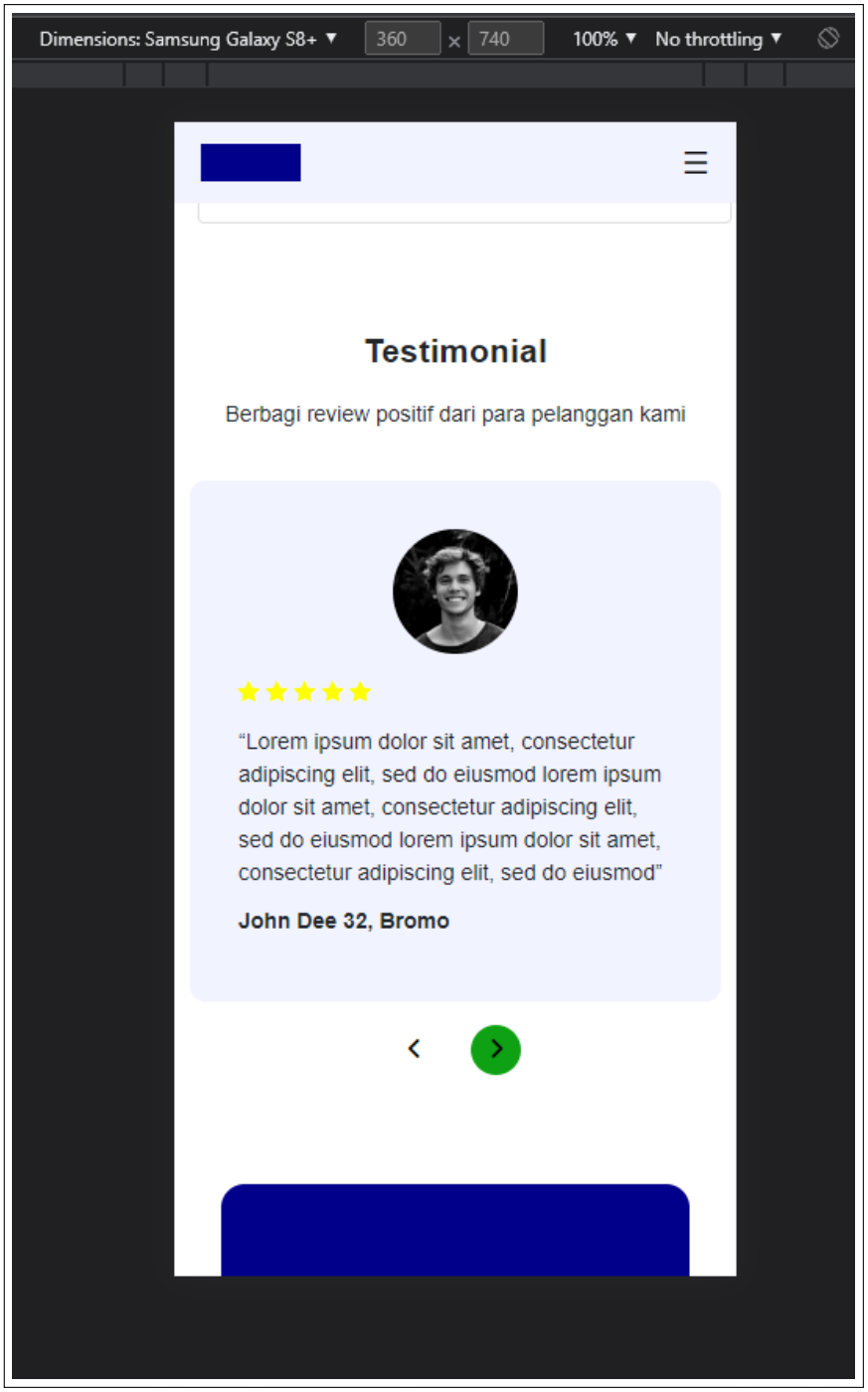
Gambar 3.11. *Chapter 3 - Landing Page*

Gambar 3.11 menampilkan *landing page* untuk *mobile* dan sudah tampil secara *responsive* dibuat dengan *css manual (sizing)* dengan menggunakan *media query*.



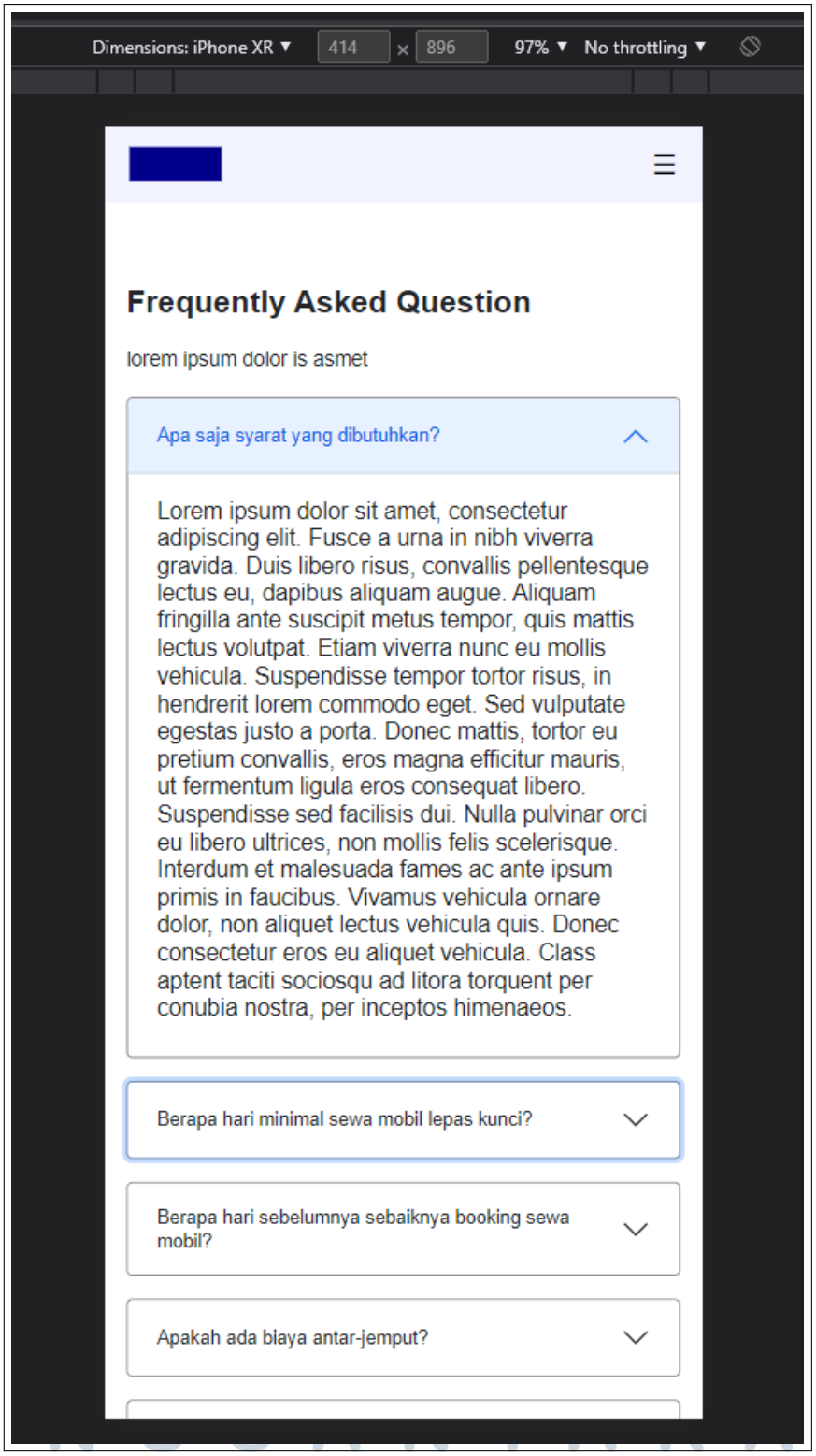
Gambar 3.12. *Chapter 3 - Why Us?*

Gambar 3.12 menampilkan *page* untuk *why us*, yang dibuat dengan *card*, dan ketika dalam bentuk *mobile* maka *card* akan turun kebawah .



Gambar 3.13. *Chapter 3 - Carousel*

Gambar 3.13 menampilkan halaman *carousel* yang sudah dibuat secara *responsive*, untuk tampilan sama seperti *carousel* pada umumnya, ketika diklik maka akan bergeser ke kiri atau ke kanan.

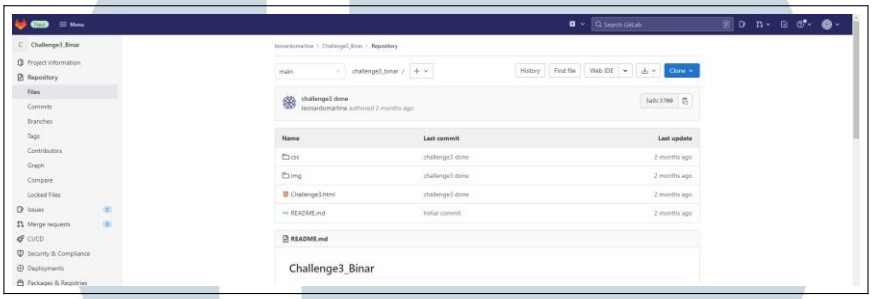


Gambar 3.14. *Chapter 3 - Accordion*

Gambar 3.14 menampilkan halaman *accordion* yang ketika diklik maka

akan menampilkan *text* dari *point* yang ditampilkan dan jika diklik kembali, maka *point* tersebut akan tertutup

Proses pembuatan *responsive page* dibuat secara manual dengan memodifikasi *CSS*, dan menggunakan *media query* untuk mengatur *size* yang sesuai untuk setiap *device* yang berbeda, berdasarkan lebar maksimal dari layar *device* yang ditentukan.

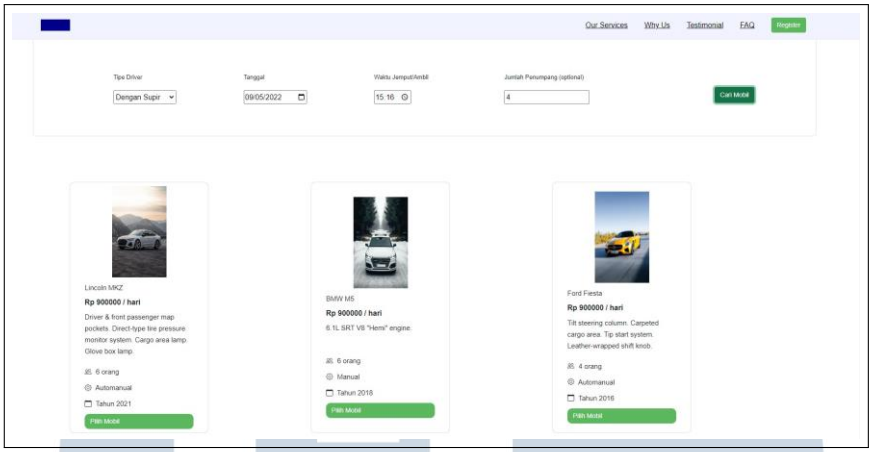


Gambar 3.15. *Chapter 3 - Gitlab*

Pada pengumpulan tugas kali ini, diwajibkan mengumpulkan melalui *Gitlab* dengan cara membuat *repository* baru yang nantinya melakukan*clone* menggunakan *git bash* lalu diwajibkan juga untuk melakukan *git add* serta git commit lalu tahap terakhir adalah melakukan *git push* kembali ke dalam *repository* tersebut.

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

Pada *Chapter* 4,mempelajari *OOP*, *DOM, HTT Server* dan juga *Node.js*. Pada pembelajaranm *OOP*,diajarkan konsep dasar mengenai cara penggunaan *OOP (Object Oriented Programming)* dan juga mempelajari 4 pilar *OOP* yaitu *Inheritance, Abstraction, Encapsulation, dan Polymorphism*. Untuk materi *DOM*, berikutnya mempelajari konsep dasar mengenai *DOM (Document Object Model)* , serta penggunaan *DOM* pada *Javascript*. Berikutnya adalah *HTTP Server*, perserta diajarkan untuk membuat data yang akan ditampilkan pada *server*, serta membuat *endpoint* yang merespon *file JSON*. Terakhir ada *node.js*, fasilitator mengajarkan untuk *menginstall package node.js* dan penggunaanya secara dasar.

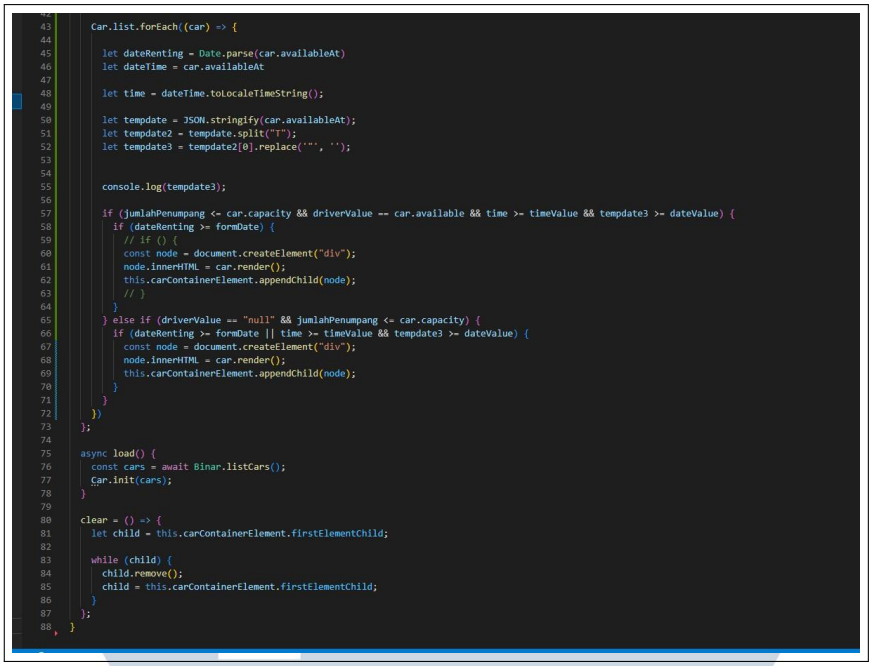


Gambar 3.16. *Chapter* 4 - Tampilan *Filter* mobil

Berikut adalah tampilan dari hasil pengerjaan *challenge* pada *chapter* 4, yaitu melakukan *filtering* sesuai dengan kondisi yang ditentukan, mulai dari kriteria tipe supir, tanggal, waktu dan juga jumlah penumpang. Selain itu juga diwajibkan untuk mengimplementasikan konsep DOM pada *code* yang dibuat.

```
class App {
  constructor() {
    this.clearButton = document.getElementById("clear-btn");
    this.loadButton = document.getElementById("load-btn");
    this.carContainerElement = document.getElementById("cars-container");
    this.submitButton = document.getElementById("submit-btn");
  }

  async init() {
    await this.load();


    this.clearButton.onclick = this.clear;
    this.loadButton.onclick = this.run;
    this.submitButton.onclick = this.filterCar;
  }

  run = () => {
    Car.list.forEach((car) => {

    });
  };

  filterCar = () => {
   let child = this.carContainerElement.firstElementChild;

    while (child) {
      child.remove();
      child = this.carContainerElement.firstElementChild;
    }

    let driverValue = document.getElementById("tipeDriverInput").value;
    let jumlahPenumpang = document.getElementById("jumlahPenumpangInput").value;
    let dateValue = document.getElementById("date").value;

    let timeValue = document.getElementById("time").value;

    let dateRent = (dateValue + "T" + timeValue);
    let formDate = Date.parse(dateRent);

    console.log(dateValue);

    Car.list.forEach((car) => {

      let dateRenting = Date.parse(car.availableAt)
      let dateTime = car.availableAt

      let time = dateTime.toLocaleTimeString();
```

Gambar 3.17. *Chapter* 4 - Tampilan *Filter code*



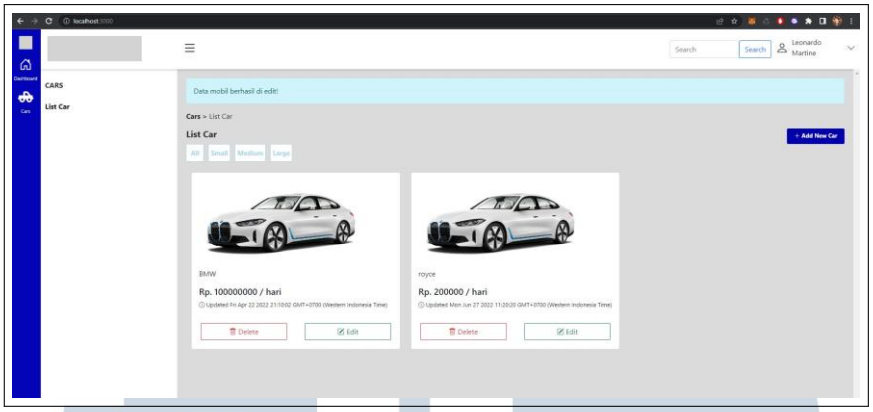
```
    Car.list.forEach((car) => {

      let dateRenting = Date.parse(car.availableAt)
      let dateTime = car.availableAt

      let time = dateTime.toLocaleTimeString();

      let tempdate = JSON.stringify(car.availableAt);
      let tempdate2 = tempdate.split("T");
      let tempdate3 = tempdate2[0].replace('"', '');


      console.log(tempdate3);

      if (jumlahPenumpang <= car.capacity && driverValue == car.available && time >= timeValue && tempdate3 >= dateValue) {
        if (dateRenting >= formDate) {
          // if () {
          const node = document.createElement("div");
          node.innerHTML = car.render();
          this.carContainerElement.appendChild(node);
          // }
        }
      } else if (driverValue == "null" && jumlahPenumpang <= car.capacity) {
        if (dateRenting >= formDate || time >= timeValue && tempdate3 >= dateValue) {
          const node = document.createElement("div");
          node.innerHTML = car.render();
          this.carContainerElement.appendChild(node);
        }
      }
    })
  };

  async load() {
    const cars = await Binar.listCars();
    Car.init(cars);
  }

  clear = () => {
    let child = this.carContainerElement.firstElementChild;

    while (child) {
      child.remove();
      child = this.carContainerElement.firstElementChild;
    }
  };
}
```

Gambar 3.18. *Chapter* 4 - Tampilan *Filter code*

Berikut adalah tampilan dari potongan *code* untuk melakukan *filter* pada mobil sesuai dengan kriteria yang sudah ditentukan pada *challenge* chapter 4, melakukan pengecekan dengan menggunakan *If*, pada *if* pertama jika jumlah penumpang lebih kecil dari kapasitas mobil , lalu berdsarkan tipe driver serta waktu dan tanggal, lalu di dalam kondisi tersebut, terdapat juga pengecekan waktu dan tipe driver sedangkan *if* kedua, bila *user* hanya memilih salah satu dari 4 kriteria yang tersedia.

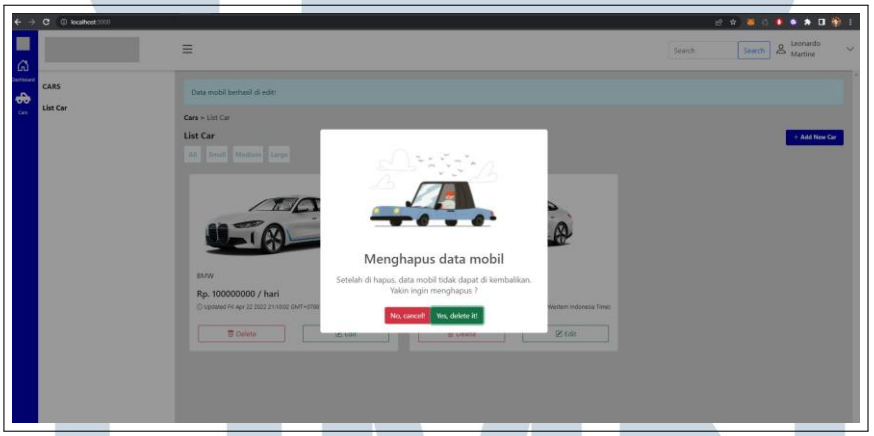
3.2.5 *Express.js, restful* API, *database,* dan ORM

Pada *chapter* 5,mempelejari konsep dasar *Express.js*, cara instalasi dan juga *request* serta respon *HTTP server*, yang dilanjut dengan cara menerapkan *rest API* pada *express.js*, dan juga belajar mengenai *database*, mempelajari apa perbedaan *SQL* dan *Non-SQL*, serta mempelajari cara implementasi pada data pada aplikasi *postgress* dan yang terkahir adalah belajar mengenai konsep ORM, proses instalasi dan cara menggunakannya.



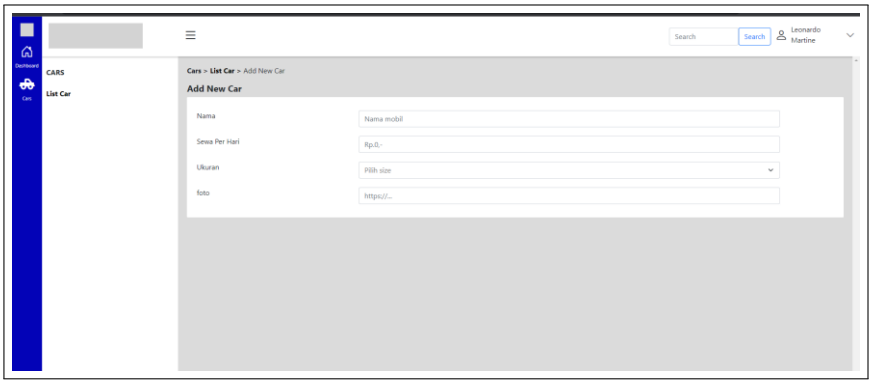
Gambar 3.19. *Chapter* 5 - Tampilan *edit* pada *chapter* 5

Pada gambar 3.19 terdapat *page* untuk melakukan *edit* atau melakukan *update* terhadap mobil yang sudah terdata sebelumnya.



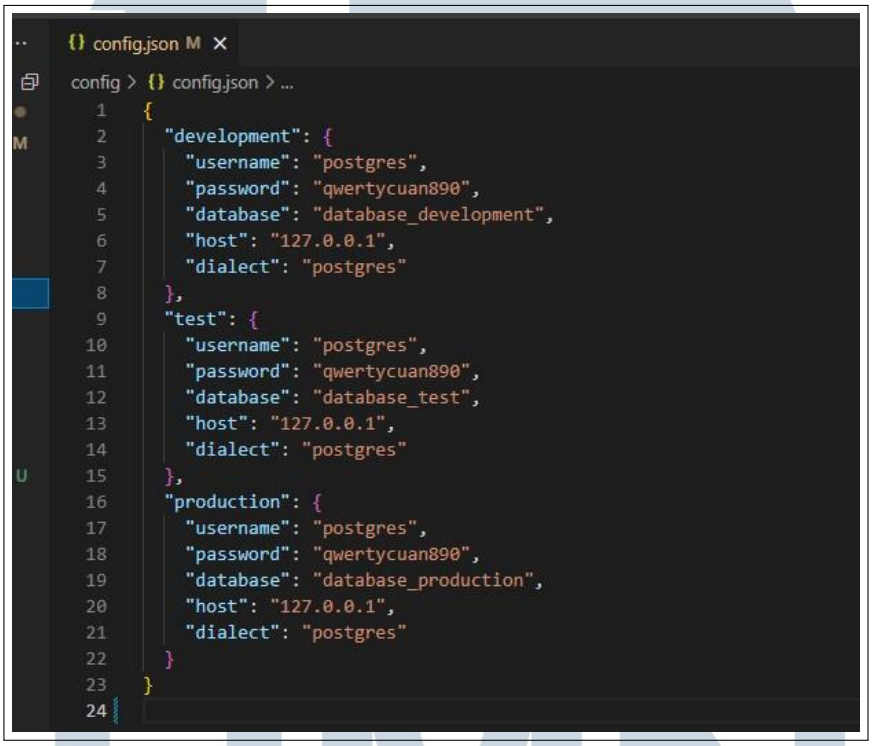
Gambar 3.20. *Chapter* 5 - Tampilan *delete* pada *chapter* 5

Pada gambar 3.20 terdapat tampilan untuk melakukan hapus data mobil yang sudah tersedia sebelumnya, ketika dihapus maka data mobil tersebut akan hilang



Gambar 3.21. *Chapter* 5 - Tampilan *add new car* pada *chapter* 5

Pada gambar 3.21 terdapat tampilan untuk menambahkan data mobil baru yang nantinya akan muncul pada halaman penjualan. Berikut adalah tugas yang diberikan pada *challenge chapter* 5, diwajibkan untuk membuat halaman *web* yang dibangun dengan menggunakan *express.js*, dan dapat melakukan *CRUD (create, read, update, dan delete)* serta untuk pengolahan datanya menggunakan *PostgreSQL*, dan wajib mengimplementasikan *rest-api*.



Gambar 3.22. *Chapter* 5 - Potongan code untuk konfigurasi *database*

Berikut adalah potongan code untuk menampilkan hasil konfigurasi *database* menggunakan *sequelize* yang dipisah menjadi 3 jenis *environment*, yaitu *development, test, production* dan karena masih dalam tahap pengembangan, maka *environment* yang dipakai adalah *development*.

```
41
42  const addDT = (req, res) => {
43    createdata(req.body);
44    setTimeout(pindah, 2000);
45
46    function pindah() {
47      req.flash("msg", "Data mobil berhasil di simpan!");
48      res.status(201).redirect("/");
49    }
50  };
51
52  const deleteDT = (req, res) => {
53    deletedata(req.params.id);
54    setTimeout(pindah2, 2000);
55    function pindah2() {
56      res.status(201).redirect("/");
57    }
58  };
59
60  const updateDT = (req, res) => {
61    updatedata(req.body);
62    setTimeout(pindah3, 2000);
63    function pindah3() {
64      req.flash("msg", "Data mobil berhasil di edit!");
65      res.status(201).redirect("/");
66    }
67  };
68
69  const filterDT = (req, res) => {
70    const cars = filterdata(req.params.size);
71    cars.then(function (result) {
72      res.render("index", {
73        layout: "layouts/main-layout",
74        title: "Halaman admin",
75        cars: result,
76        msg: req.flash("msg"),
77      });
78    });
79  };
```

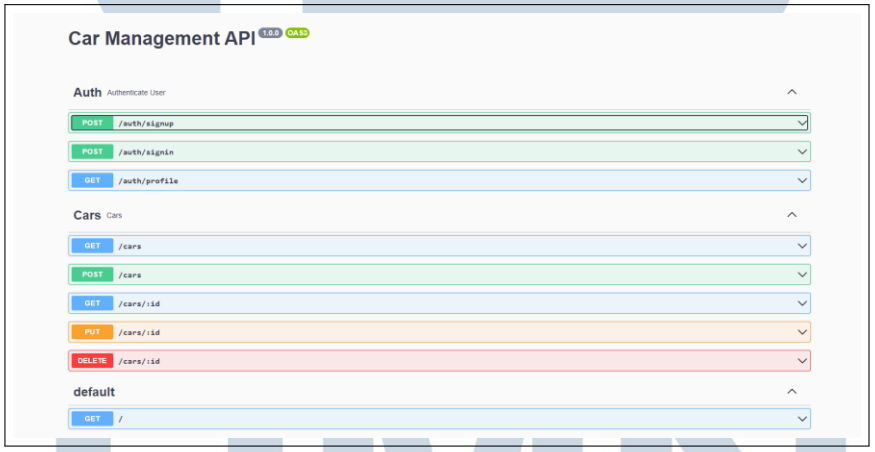
Gambar 3.23. *Chapter* 5 - Potongan *code Create,Read,Delete,Update*

Pada baris 42 - 50 merupakan potongan *code create car*, dimana pada baris 43, kita dapat melihat fungsi untuk menambahkan mobil secara manual, yang berikutnya akan masuk ke *database*, dan bila proses *create* sudah selesai maka tampilan akan kembali ke *home* dan memunculkan sebuah *alert* ”Data mobil berhasil disimpan!” Sedangkan pada baris 53 kita dapat melihat fungsi delete berdasarkan *id*, dan bila proses hapus sudah dilakukan maka tampilan akan kembali ke halaman *home* lagi. Berikutnya pada baris 60 - 66, merupakan potongan *code* untuk *update car*, *user* akan melakukan *update* secara manual, yang berikutnya fungsi akan mencari *id* yang sesuai, dan setelah proses *update* selesai, maka tampilan akan kembali ke *home* dan memunculkan *alert* ”Data mobil berhasil di edit!” dan pada baris 69- 79 merupakan potongan *code* untuk *read*, dimana pada bagian ini, *user*

dapat melihat semua data yang terdapat pada *database*.

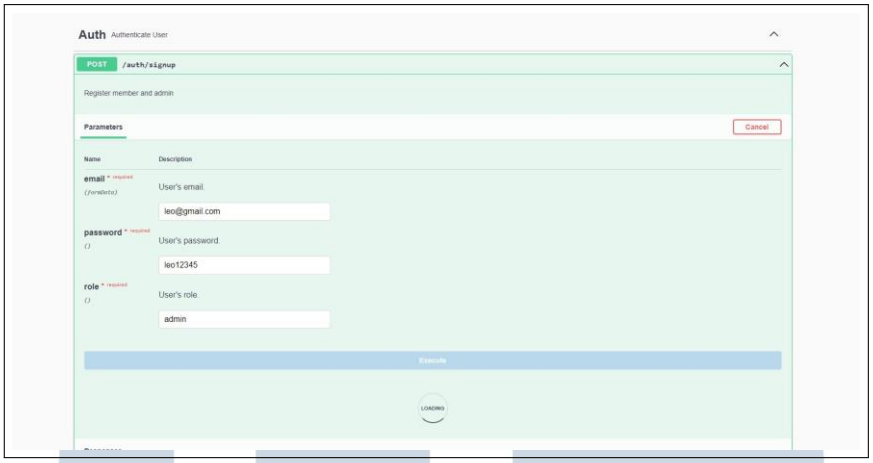
3.2.6 Design pattern, synchronous process, Authentication & Authorization, dan *open API*

Pada chapter 6, mempelajari *design pattern* membahas konsep dari *design pattern*, MVC (*model, view, controller*), *service repository pattern*, dan *microservice pattern*. Berikutnya adalah *asynchronous process*, mempelajari cara mengimplentasikan *callback*, dan *promise*, berikutnya mempelajari materi *authentication dan authorization* dan menerapkan *authentication* dengan menggunakan *session based authentication* dan *token based authentication (JWT)* dan yang terakhir mempelajari *open API* , membahas sturktur dan konsep*open API*, cara kerja *swagger tool* dan juga cara mengimplementasikannya.



Gambar 3.24. *Chapter* 6 - tampilan untuk *Swagger API*

Pada *chapter* 6, diwajibkan untuk dapat mengimplementasikan fungsi *GET, POST, UPDATE, DELETE* pada sebuah *open API* yang sudah diberikan yaitu *SwaggerUI*, seperti yang dapat kita lihat pada gambar, terdapat 2 *user*, yaitu *admin* dan *member* , dimana *admin* dapat melukan fungsi *delete* dan *post/create*, sedangkan *user* biasa hanya dapat melakukan fungsi *read/get*.



Gambar 3.25. *Chapter* 6 - Tampilan untuk *Form User*

Berikut adalah tampilan form user untuk melakukan registrasi, dimana terdapat pilihan untuk menjadi *admin* ataupun*member*, keduanya memiliki fungsinya masing-masing, baik *admin* maupun member keduanya harus membuat akun, yang disertai dengan *email* dan *password* tentunya

```
};

exports.createNewCarApi = async(request, response) => {
    const car = await carService.createNewCar(request);

    response.status(201).json({ data: car });
};

exports.updateCarApi = async(request, response) => {
    const car = await carService.updateCar(request, request.params.id);

    if (car == null) {
        response.status(404).json({ error: `Car not found with ids : ${request.params.id}` });
    } else {
        response.json({ message: "Updated successfully" });
    }
};

exports.deleteCar = async(request, response) => {
    const carById = await carService.findCarById(request.params.id);

    if (carById == null) {
        response.status(404).json({ error: `Car not found with ids : ${request.params.id}` });
    } else {
        carService.deleteCar(carById);
        response.json({ message: "Deleted successfully" });
    }
};
```

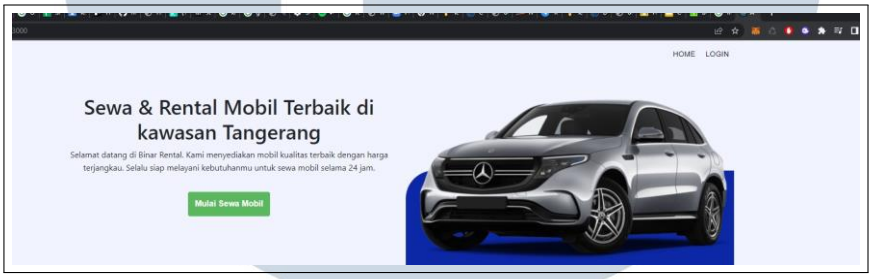
Gambar 3.26. *Chapter* 6 - tampilan untuk potongan *code CRUD*

Berikut adalah potongan *code* untuk *CRUD* yang mengatur data-data mobil , terdapat beberaa fungsi, yang pertama fungsi untuk menambah data mobil, yang apabila proses penambahan berhasil maka berikutnya akan dilempar ke halaman utama untuk menampilkan mobil,berikutnya ada fungsi *update* yang dicek berdasarkan *id* mobil, apabila proses *update* gagal maka akan menampilkan respon status 404 dan bila berhasil maka akan menampilkan *message "Updated successfully"*, berikutnya ada fungsi *delete*, sama seperti fungsi *delete* seperti biasanya,

apabila *error* menampilkan *response status* 404, dan apabila proses *delete* berhasil, maka akan menampilan *message " Deleted successfully".*

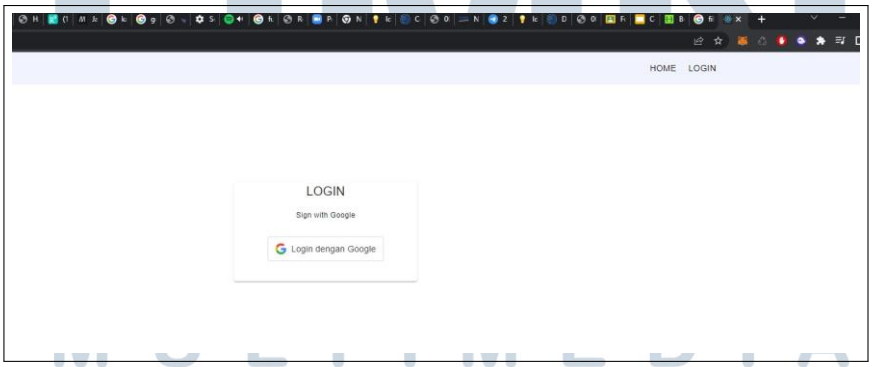
3.2.7 React.js, *React routing* dan *OAuth*

Pada *chapter* 7, mempelajari konsep dasar *react.js* yang meliputi *component dan styling* dengan menggunakan *UI framework*. Berikutnya mempelajari *react routing*, seperti cara meng*install* dan mengaplikasinnya kedalam tugas pengerjaan, dan yang terakhir ada *OAuth*, berikutnya mempeljari cara men*ingstall* dan bagaimana cara mengimplementasikanya ke dalam tugas mingguan, serta mempelajari cara menggunakan *react google login*.



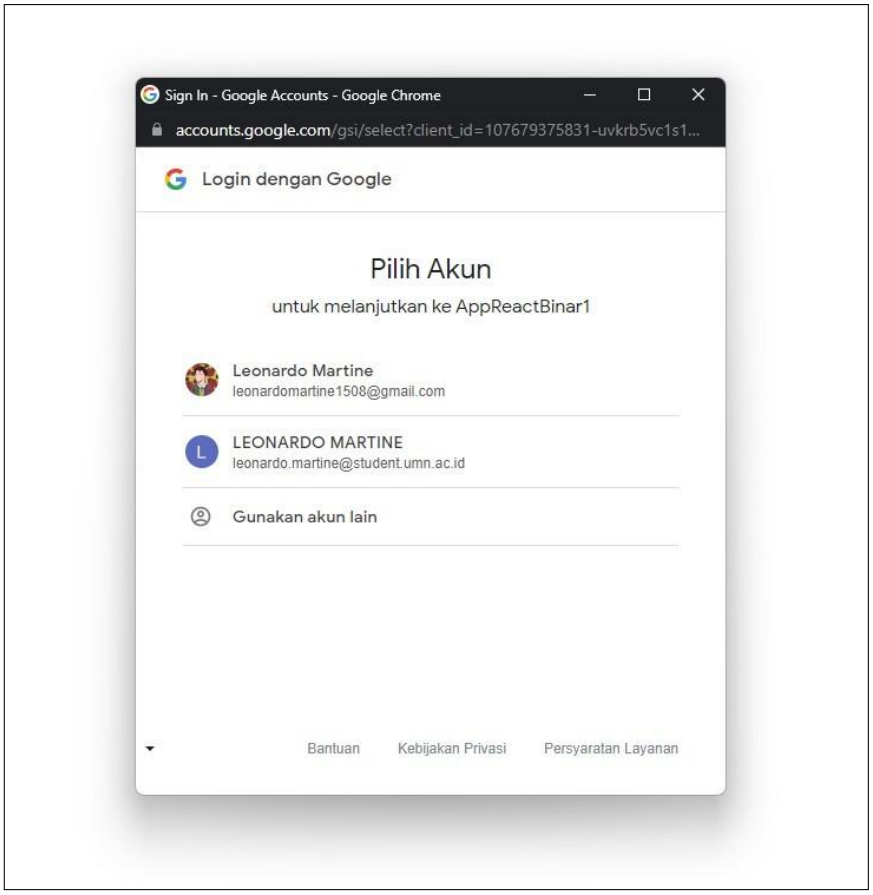
Gambar 3.27. *Chapter* 7 - Tampilan *home*

Pada gambar 3.27 terdapat tampilan *home*, sama seperti *landing page* yang sudah dibuat sebelumnya, yang nantinya untuk melakukan sesuatu akan *direct* ke *login page*.



Gambar 3.28. *Chapter* 7 - Tampilan *login* dengan *google*

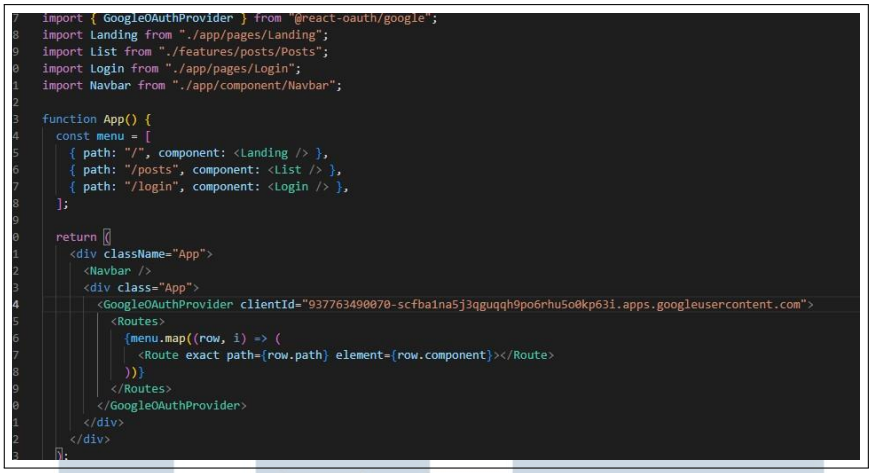
Pada gambar 3.28 tedapat tampilan untuk melakukan *login* dengan menggunakan *google*.



Gambar 3.29. *Chapter* 7 - Tampilan *login* dengan *google*

Pada gambar 3.29 tedapat tampilan untuk pemilihan akun *google* yang akan dipilih untuk melanjutkan proses *login*.

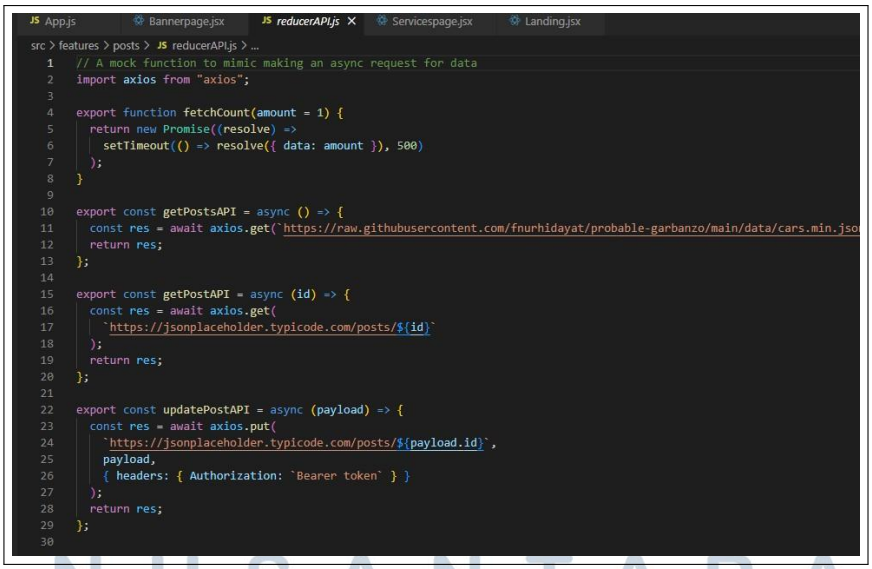
Pada *chapter* 7, membuat halaman *website* dengan *react.js* dan memiliki fitur *login* yang harus disertai dengan implementasi *Google Oauth*, dengan tujuan *user* dapat melakukan *login* dengan menggunakan *email google* yang sudah dimiliki *user* sebelelumnya, berikut adalah tampilan *home* yang disertai dengan fitur *login*, dimana untuk mengakses beberapa fitur, dan diwajibkan untuk *login* terlebih dahulu.



```
import { GoogleOAuthProvider } from "@react-oauth/google";
import Landing from "./app/pages/Landing";
import List from "./features/posts/Posts";
import Login from "./app/pages/Login";
import Navbar from "./app/component/Navbar";

function App() {
  const menu = [
    { path: "/", component: <Landing /> },
    { path: "/posts", component: <List /> },
    { path: "/login", component: <Login /> },
  ];

  return (
    <div className="App">
      <Navbar />
      <div class="App">
        <GoogleOAuthProvider clientId="937763490070-scfba1na5j3qguqqh9po6rhu5o0kp63i.apps.googleusercontent.com">
          <Routes>
            {menu.map((row, i) => (
              <Route exact path={row.path} element={row.component}></Route>
            ))}
          </Routes>
        </GoogleOAuthProvider>
      </div>
    </div>
  );
```

Gambar 3.30. *Chapter* 7 - Potongan *code* untuk Implementasi *Oauth*

Berikut merupakan tampilan *code* untuk mengimplementasikan *google OAuth*, proses implementasi *google OAuth* ini sendiri dibuat dengan membuka *console cloud google*, disini *user* akan membuat suatu *key access* yang nantinya akan dapat digunakan atau dapat memiliki koneksi dengan akun *google* yang *user* miliki, setelah proses pembuatan *OAuth client*nya sudah selesai, maka, *user* akan mendapatkan *client ID* yang dapat diimplementasikan kedalam projek, sehingga proses *login with google* telah berhasil.



App.js | Bannerpage.jsx | reducerAPI.js | Servicespage.jsx | Landing.jsx

src > features > posts > reducerAPI.js > ...

```
// A mock function to mimic making an async request for data
import axios from "axios";

export function fetchCount(amount = 1) {
  return new Promise((resolve) =>
    setTimeout(() => resolve({ data: amount }), 500)
  );
}

export const getPostsAPI = async () => {
  const res = await axios.get(`https://raw.githubusercontent.com/fnurhidayat/probable-garbanzo/main/data/cars.min.jso
  return res;
};

export const getPostAPI = async (id) => {
  const res = await axios.get(
    `https://jsonplaceholder.typicode.com/posts/${id}`
  );
  return res;
};

export const updatePostAPI = async (payload) => {
  const res = await axios.put(
    `https://jsonplaceholder.typicode.com/posts/${payload.id}`,
    payload,
    { headers: { Authorization: `Bearer token` } }
  );
  return res;
};
```

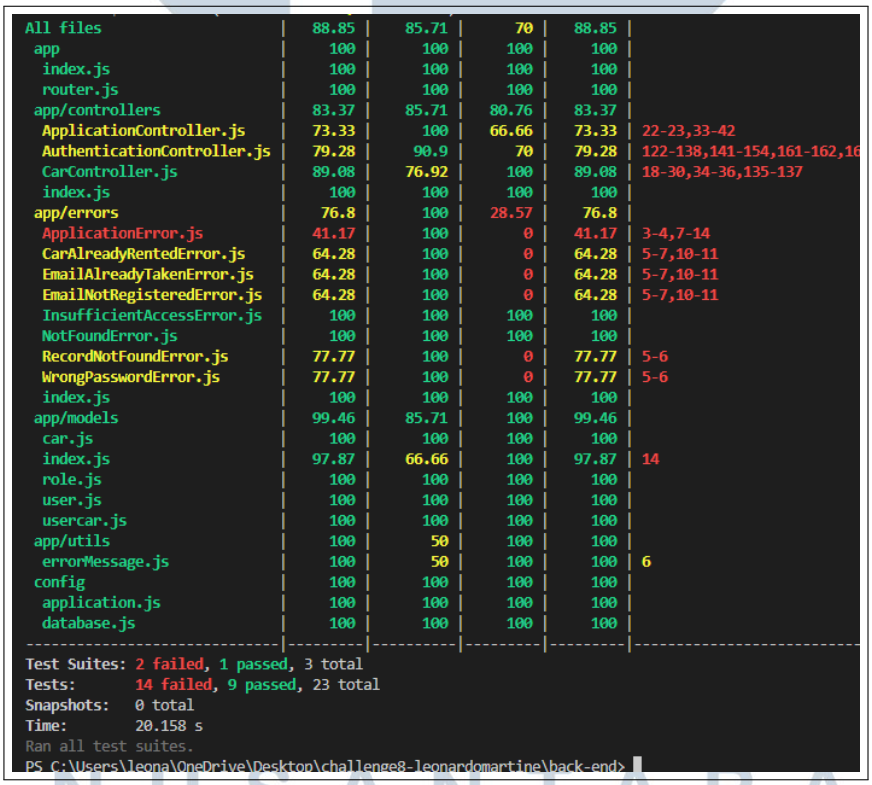
Gambar 3.31. *Chapter* 7 - Potongan *code* untuk *Backend Integration*

Implementasi *backend integration* sendiri memiliki fungsi untuk menarik *API* dari *endpoint* yang sudah tersedia, sehingga *user* harus *login* terlebih dahulu

agar dapat melihat fitur-fitur yang belum ditampilkan, *library axios* sendiri memiliki paramater *method= GET, url* yang sudah disediakan ,serta *time out* 500, tentu hal ini membuat proses pengerjaan menjadi lebih ringan dan menghemat waktu.

3.2.8 websocket, next.js, media handling, eslint, TDD (testing, driven, developement), deployment CI/CD

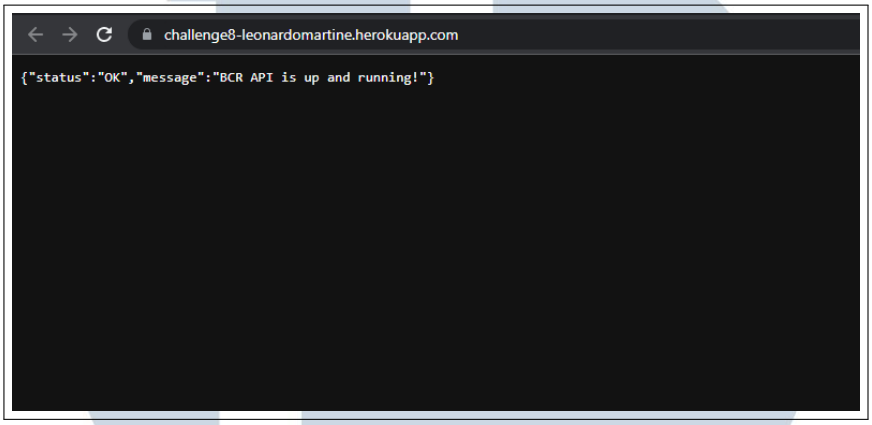
Pada *chapter* 8, mempelajari *websocket*, fungsi dan konsep dari *websocket* itu sendiri, serta bagaimana cara melakukan implementasi *socket IO*, berikutnya adalah *next.js*, berikutnya mempelajari fitur-fitur yang terdapat pada *next.js*, serta melakukan instalasi dan proses pengerjaan menggunakan *next.js*, berikutnya adalah *media handling*, berikutnya mempelajari bagaimana cara melakukan *save* dan *upload file* melalui *backend* dan juga *frontend* dan juga mempelajari konsep mengenai *TDD (testing, driven, development), desployment dan CI/CD*, dan tentunya juga bagaimana cara mengimplementasikannya secara langgsung terhadap suatu projek.



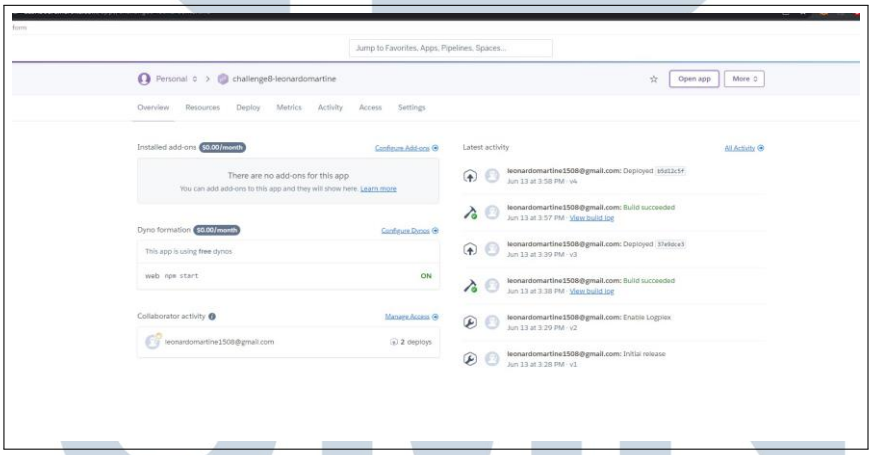
Gambar 3.32. *Chapter* 8 - Hasil Uji *Testing*

Berikut adalah hasil implementasi dari *testing* menggunakan *Jest*, dari total 23 tes yang diuji, terdapat 9 tes yang berhasil lolos dan 14 yang gagal, dan

hasil tes 88,85% *coverage* dan untuk angka dari hasil tes yang ditentukan sendiri adalah diharuskan untuk *minimal* 70 %, sehingga dengan hasil tes mencapai angka 88%, dapat dikatakan sudah sudah berhasil karena jauh di atas standar angka yang diberikan.



Gambar 3.33. *Chapter* 8 - Tampilan hasil *deploy* di *herokuapp*



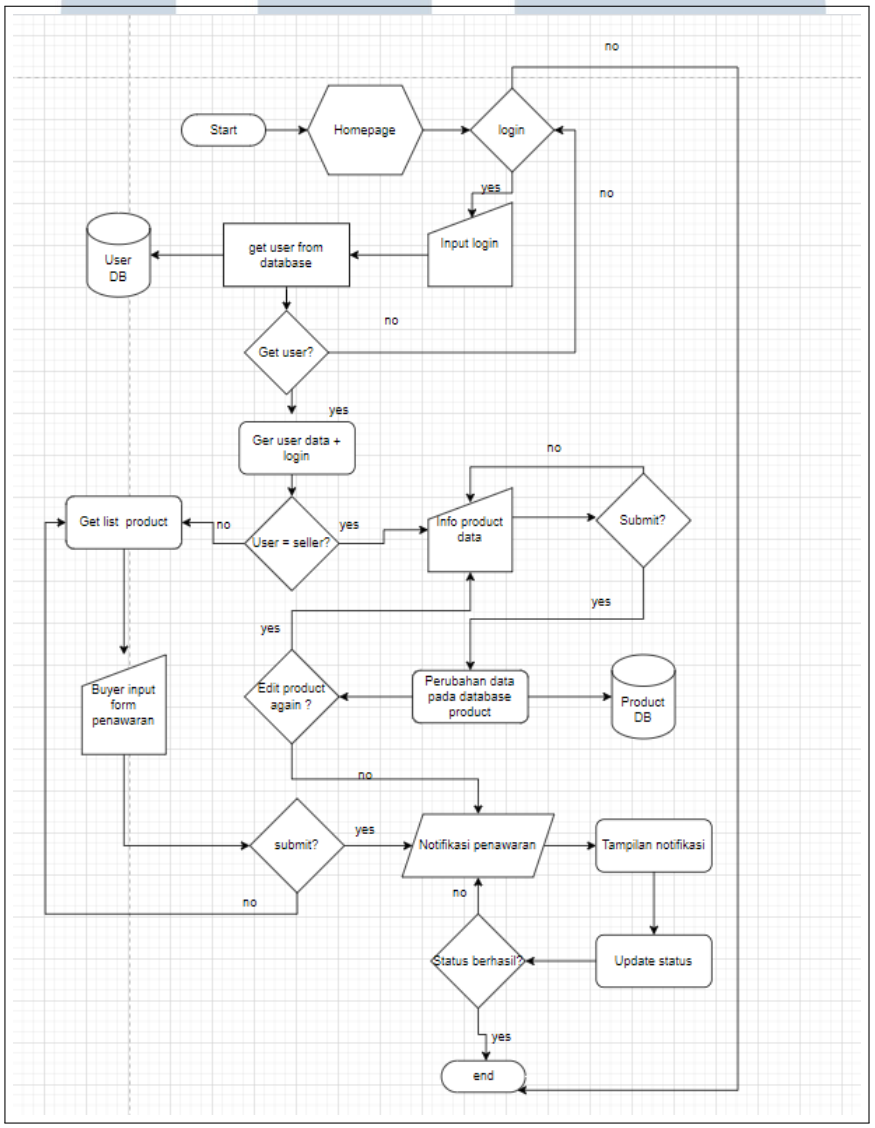
Gambar 3.34. *Chapter* 8 - Tampilan hasil *deploy* di *herokuapp*

Berikut adalah hasil dari *deployment* di *heroku*, hasil *deployment* dapat dinyatakan berhasil karena tampilan sudah memberikan *respone/status* ”OK”.

3.2.9 Progres Projek akhir

Setelah menyelesaikan *Chapter* 8, diarahkan untuk mengerjakan projek akhir yaitu membuat *website E-commerce* dengan pembagian kelompok, yang terdiri dari *back-end dan juga front-end*, sedangkan untuk bagian *front-end*, untuk pengerjaan projek akhir sendiri, setiap proses pengerjaan diatur di dalam *sprint*,

sprint sendiri adalah semacam agenda penjadwalan mengenai rangkaian atau tugas-tugas yang akan dilakukan dalam waktu tertentu, dan tugas sebagai *front-end* di *sprint* pertama adalah membuat *login page* secara *responsive* menggunakan *react-js* , lalu membuat *password show/unshow* serta melakukan integrasi *endpoint* pada *code back-end dan front-end*, agar terkoneksi. tetapi proses pengerjaan projek akhir ini juga baru dimulai ,sehingga proses pengerjaan masih akan berlanjut. Proses pengerjaan akan berjalan sesuai dengan *sprint* yang dibuat oleh *scrum master* dengan tujuan agar proses pengembangan *website* akan berjalan dengan baik dan teratur.



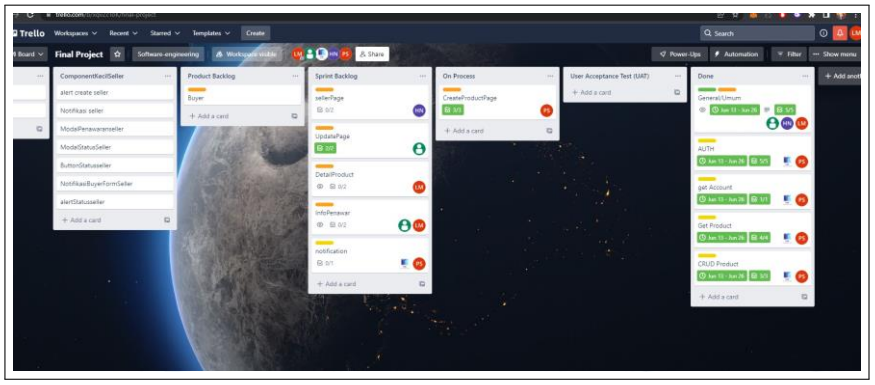
Gambar 3.35. Chapter 9 - Tampilan Flowchart Projek Akhir

Berikut adalah *flowchart* atau skema bagaimana aplikasi berjalan, pada

bagian pertama *user* akan masuk ke halaman *home*, tetapi untuk melakukan proses pembelian, *user* akan diarahkan ke halaman *login*, dan apabila *user* belum memiliki akun, maka *user* akan diarahkan ke halaman register, untuk melakukan registrasi akun, berikutnya *user* wajib melakukan *login* terlebih dahulu. pada aplikasi ini terdapat 2 *user*, yaitu *buyer* dan juga *seller*.

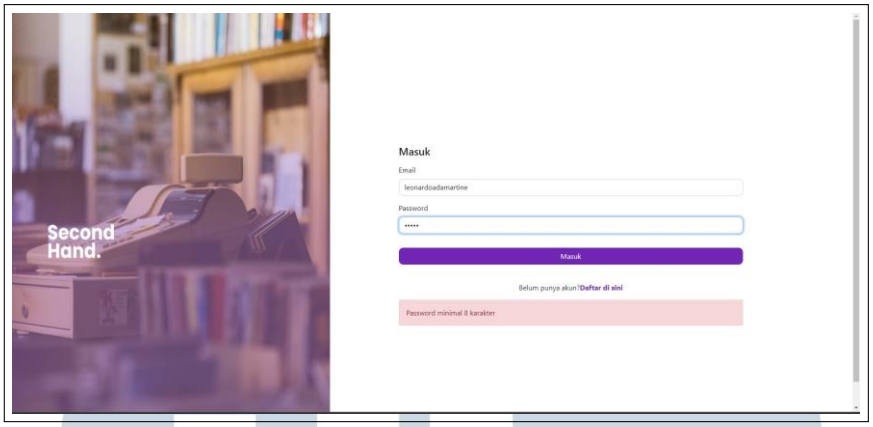
Buyer dapat melihat dan melakukan proses pembelian untuk segala jenis produk yang tersedia, untuk proses pembelian, *buyer* dapat melakukan sistem *bid*/penawaran terhadap barang yang ingin dibeli dengan menekan tombol ”menarik” , lalu diarahkan untuk mengisi *form* mengenai harga penawaran yang diajukan dan nantinya akan dikirimkan ke *seller*, dan menunggu untuk konfirmasi lebih lanjut dari *seller*.

Seller dapat menjual produk, atau menambah stok produk yang sudah habis, selain itu *seller* juga dapat melakukan *edit* dan *delete* terhadap suatu produk. *User* yang tertarik dengan produk yang dijual *seller*, dan mengirim harga penawaran akan masuk ke notifikasi *seller*, dan berikutnya adalah *seller* dapat memilih harga penawaran yang paling cocok, dan tentu *seller* dapat menolak penawaran *buyer* yang memang tidak sesuai dan jika proses penjual berhasil maka , *seller* dapat melakukan *update* terhadap produk yang sudah terjual tersebut.

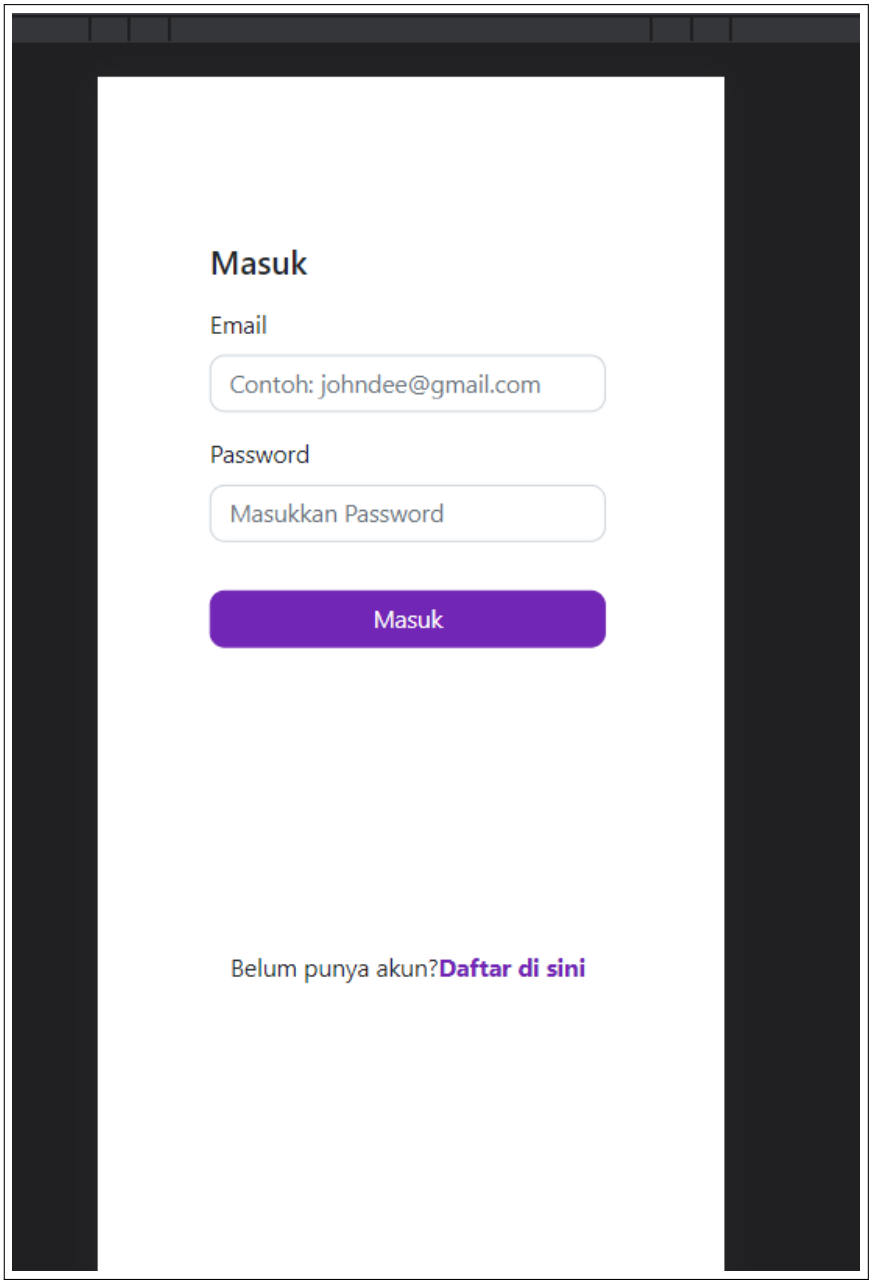


Gambar 3.36. Chapter 9 - Tampilan Sprint Final Project

Berikut adalah tampilan *sprint*, yang dibuat sebelum mengerjakan projek akhir. Setiap kelompok wajib membuat *sprint* dengan jadwal 2 minggu, pembagian tugas masing-masing sesuai dengan *job description* yang sudah ditentukan, dan pada *sprint* pertama untuk *front-end* mengerjakan bagian *login page, register page, dan aboutme page* secara responsive , dan tentunya melakukan *integrasi endpoint* juga sehingga dapat terhubung dengan*server* yang telah dibuat oleh *back-end*.

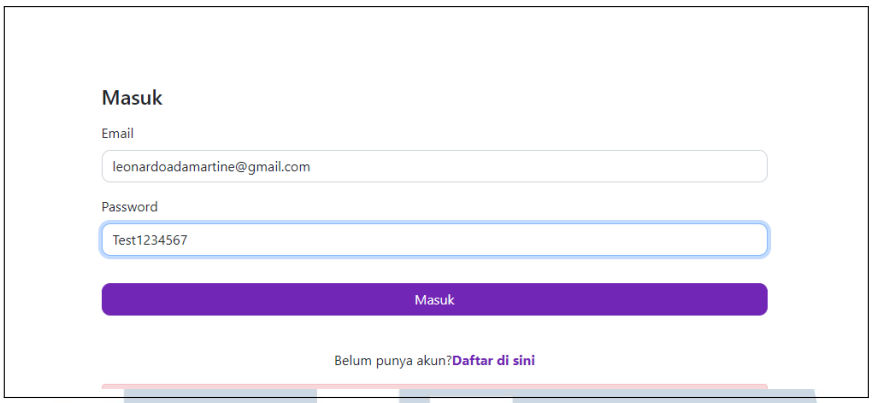


Gambar 3.37. *Chapter* 9 - Tampilan *login page*



Gambar 3.38. *Chapter* 9 - Tampilan *responsive login page*

Berikut adalah tampilan *login page* untuk tampilan *website* dan juga layar *handphone*, terdapat juga*button submit* yang diberi *hover* ketika ditekan, lalu *alert* yang muncul ketika *button submit* ditekan adalah karena *code* untuk tampilan *login page* sudah dintergrasikan dengan *API* yang dibuat oleh *back-end developer*, dimana pada *API back-end* terdapat beberapa kriteria, agar kita dapat *login*, seperti *password* yang dimasukkan harus benar, *email* dan *password* harus sudah tersedia atau dalam arti *user* sudah melakukan *registrasi* terlebih dahulu.



Gambar 3.39. *Chapter* 9 - Tampilan *login form* dengan *show/unshow password*

```
const [passwordShown, setPasswordShown] = useState(false);
const togglePassword = () => {
    // When the handler is invoked
    // inverse the boolean state of passwordShown
    setPasswordShown(!passwordShown);
  };
```

Gambar 3.40. *Chapter* 9 - Potongan *code* untuk *show/unshow password*

Tedapat juga fitur *show/unshow password*, ketika *user* menekan kolom *password*, atau ingin mengisi *password*, maka user dapat memperlihatkan *password* miliknya atau tidak dengan menekannya, dengan tujuan untuk memudahkan *user*, apabila *user* ingin memastikan *password* yang ditulisnya benar atau salah.

```
    <Row>
    <Col className="loginpict">
        <img src="/images/img.png" width="720px" height="960px" />
    </Col>
    <Col className="login">
<Container className="my-100">
    <h4 className="mb-3 text-left">Masuk</h4>

    <Form onSubmit={onLogin}>
        <Form.Group className="mb-3">
            <Form.Label>Email</Form.Label>
            <Form.Control
                type="text"
                ref={emailField}
                placeholder="Contoh: johndee@gmail.com"
                style={styleLabel}
            />
        </Form.Group>
        <Form.Group className="mb-3 ">
            <Form.Label>Password
            </Form.Label>

            {/* <div className="pass-login">
            <input className="border-0 outline-none" type={passwordShown ? "text" : "password"} placeholder="Masukkan Password"  ref={passwordField}  />
                <button className=" float-right border-0" onClick={togglePassword}><FontAwesomeIcon  icon={faEye}></FontAwesomeIcon></button>
                </div> */}
            <Form.Control

                type={passwordShown ? "text" : "password"}
                ref={passwordField}
                placeholder="Masukkan Password"
                style={styleLabel}
                onClick={togglePassword}
                FontAwesomeIcon={faEye}

            />
```

Gambar 3.41. *Chapter* 9 - Potongan *code* untuk menampilkan *login page*

Untuk *front-end* wajib menggunakan *react.js*, dan untuk *framework CSS*,

icon, dan sebagainya dibebaskan, untuk membuat *login page* seperti gambar 3.31, menggunakan *Booostrap* 5, dan sedikit *custom CSS* yang dibuat secara manual. terdapat juga *Form.Group* yang berfungsi untuk memisahkan *form* yang ingin digunakan, seperti pada *Form.Group* pertama untuk memberikan *input email* dan *Form.Group* yang kedua untuk *password*, sedangkan *Form.Label* adalah untuk memberikan *label* nama untuk suatu *input*, dan *Form.Control* berfungsi untuk memberikan hasil yang nantinya akan di*input*, seperti tipe data, lalu terdapat juga *placeholder*, yang memiliki fungsi untuk meletakan kata-kata dengan tujuan untuk membantu *user* dalam melakukan *login*.

3.3 Uraian Pelaksanaan Magang

Pelaksanaan studi indpenden dimulai pada tanggal 14 febuari hingga 22 juli, dan diadakan pada pukul 19.00 hingga 22.00 , diadakan melalui media *zoom* dan untuk melakukan komunikasi dengan sesama maupun fasilitator dilakukan melalui *telegram*. Proses pembelajaran di Binar sendiri dibagi menjadi 8 *chapter* yang dibagi menjadi 2 minggu pada setiap *chapternya*, pada minggu pertamna fasilitator akan menjelaskan materi yang sudah disediakan oleh Binar dan pada minggu kedua, selain itu diwajibkan juga untuk menjalankan tugas yang sudah diberikan berdasarkan materi yang sudah diajarkan oleh fasilitator. Setelah 8 *chapter* selesai, maka berikutnya dwajibkan untuk mengerjakan projek akhir, dengan mengimplementasikan semua pembelajaran yang sudah dilakukan pada *chapter* sebelumnya. Proses pengerjaan projek akhir dibagi menjadi 2 kategori yaitu *front-end* dan juga *back-end*, dan estimasi untuk proses pengerjaan projek akhir Binar sendiri adalah kurang lebih 1 bulan. dengan setiap minggunya terdapat *sprint* yang dibuat agar proses pengerjaan dapat berjalan dengan teratur dan terselesaikan dengan baik. Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1

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)* • Mengerjakan *challenge landing page*
3-4	• Mempelajari dan memahami data *structure, operator* & *expressions*, dan *basic javascript* • Mengerjakan *challenge filtering mobil dengan javascript*
5-6	• Mempelajari *responsive design* • Mempelajari dan memahami terminal & IDE, GIT, *web layouting* • Mengerjakan *challenge*, membuat *responsive web* • implementasi cara menggunakan Gitlab
7-8	• Menerapkan dan mengimplementasikan OOP, DOM, Node.js dan HTTP *Server* pada *website* • Mengerjakan challenge, menerapkan fitur filtering.

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

Minggu Ke -	Pekerjaan yang dilakukan
9-10	• Membuat database • Mempelejari dan memahami Express.js, *restful* API, *database*, dan ORM • Mengerjakan *challenge*, membuat HTTP *Server*
11-12	• 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, membuat tampilan *website* 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*

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

Minggu Ke -	Pekerjaan yang dilakukan
17-22	• Projek Akhir • Membuat sprint penjadwalan untuk pengerjaan projek • Mengerjakan bagian login form, intergrasi endpoint, responsive page

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala yang ditemukan pada saat melakukan studi independen :

1. Kurangnya pengetahuan pada materi *back-end* sehingga menhambat proses pengerjaan *challenge.*
2. Komunikasi cukup terkendala pada saat pertama kali proses pembelajaran dimulai.
3. Proses pengerjaan proyek akhir masih mengalami kendala, karena anggota tim memiliki kesibukan masing-masing.

3.4.2 Solusi atas Kendala yang Ditemukan pada saat melakukan studi independen :

1. Mengeksplor dan mempelejari lagi materi back-end sehingga proses pengerjaan challenge dapat berjalan dengan lebih baik lagi
2. Lebih membuka diri terhadap sesama, sehingga proses komunikasi dapat berjalan dengan lebih baik
3. Berusaha untuk membagi waktu dan juga memprioritaskan hal yang lebih penting, dan tetap mengerjakan kewajiban yang dimiliki.