PEMROGRAMAN WEB BERBASIS SERVICES
TUGAS AKHIR – AWONAPAKARYA

Wahyu Pambudi – 19312302 (Backend)
Muhammad Ali Nasir – 19312248 (FrontEnd)

PENJELASAN PROGRAM

Team AwonapaKarya membuat Website Berbasis Services dengan study Kasus Website **Website Inventaris Barang Sekolah** yang dikembangkan menggunakan Framework Javascript. Untuk backend menggunakan ExpressJs, sedangkan untuk Frontend menggunakan ReactJS**.**

Berikut ini tautan repository Github :

https://github.com/wahyupambudi/TA-GAB2-AwonapaKarya

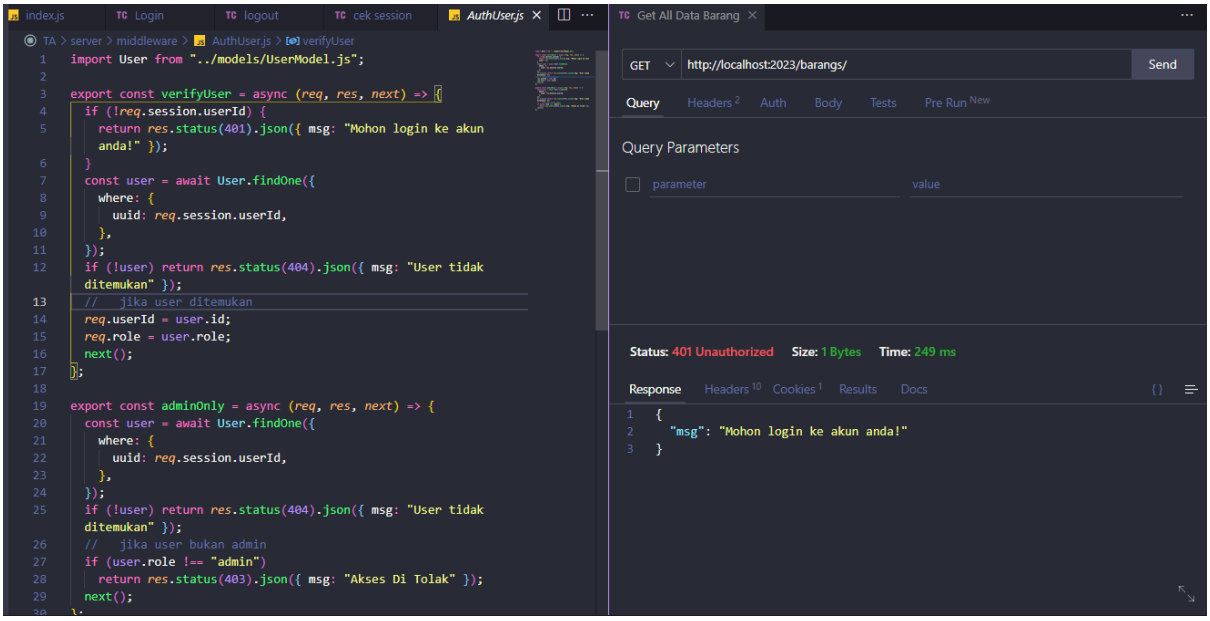
PENJELASAN PROGRAM BACKEND

Program Backend di kembangkan menggunakan NodeJs Framework ExpressJS. Menggunakan dependencies :

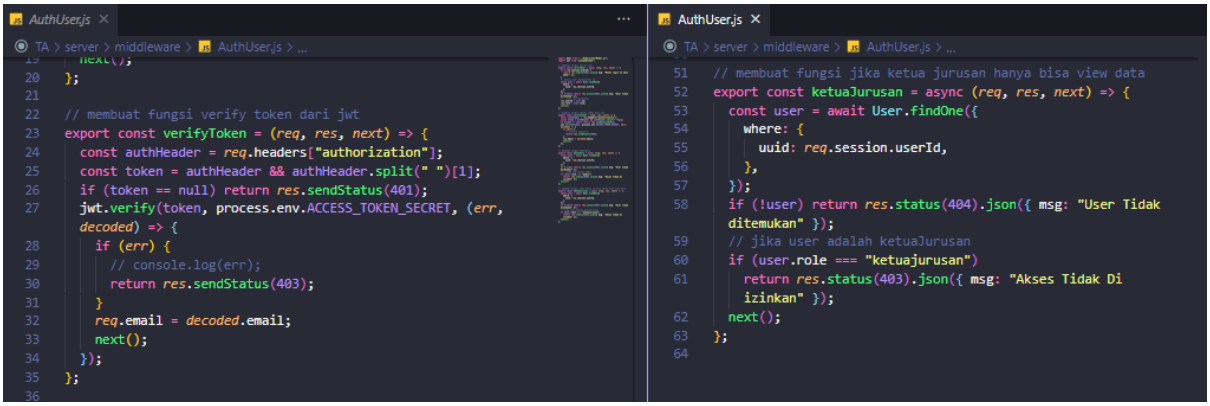
```
"dependencies": {
    "argon2": "^0.30.2", (hash password)
    "connect-session-sequelize": "^7.1.5", (session login user)
    "cookie-parser": "^1.4.6", (parser cookie jwt)
    "cors": "^2.8.5", (CORS)
    "dotenv": "^16.0.3", (untuk menyimpan variable .env)
    "express": "^4.18.2", (framework express)
    "express-fileupload": "^1.4.0", (express file upload)
    "express-session": "^1.17.3", (session express)
    "jsonwebtoken": "^9.0.0", (JWT Express)
    "mysql2": "^2.3.3",
    "sequelize": "^6.28.0"
  }
```

- **Middleware/AuthUser.js**
    - Untuk mengakses **REST API Server** (data barang/jasa dan data user). User diharuskan **login** terlebih dahulu, disini menggunakan middleware. Di dalam file A**uthUser.js** terdapat fungsi **verifyUser** jika session user tidak ditemukan maka akan menampilkan pesan “mohon login ke akun anda” hal ini berlaku disemua routers.

- Selain itu terdapat fungsi **adminOnly**, yang akan digunakan untuk routers / dashboard admin dalam melakukan manajemen user.



- Terdapat **verifyToken** yang digunakan untuk verifikasi token, jika terdapat token maka user bisa melakukan akses data ke REST API.
- Terdapat ketuaJurusan yang berguna untuk membatasi akses user dengan role ketua jurusan hanya bisa melihat (Read) data saja.
-



- **Login (Auth.js)**

```
TA > server > controllers > Auth.js > login > user > where
import User from "../models/UserModel.js";
import argon2 from "argon2";

// function login
export const login = async (req, res) => {
  const user = await User.findOne({
    where: {
      email: req.body.email,
    },
  });
  // jika user tidak ditemukan
  if (!user) return res.status(404).json({ msg: "User tidak ditemukan" });

  //   jika user ditemukan
  const match = await argon2.verify(user.password, req.body.password);

  //   jika password tidaak betul
  if (!match) return res.status(400).json({ msg: "Password Salah" });

  //   jika password cocok
  req.session.userId = user.uuid;
  //   get data user
  const uuid = user.uuid;
  const name = user.name;
  const email = user.email;
  const role = user.role;
  res.status(200).json({
    status: true,
    message: "Berhasil Login",
    dataUser: { uuid, name, email, role },
  });
};
```

- Proses login dengan mencari user berdasarkan email, jika tidak ditemukan akan menampilkan pesan “user tidak ditemukan”
- Melakukan pemeriksaan password dengan verify menggunakan argon2. Jika password tidak sesuai maka menampilkan pesan “password salah”.
- Jika berhasil maka akan melanjutkan proses berupa login dan akan membuat session. Dan akan menampilkan data berupa JSON, yaitu data user, id, name, email dan role user.

Update Proses Login

```
const accessToken = jwt.sign(
    { uuid, name, email, role },
    // proses env dari file .env
    process.env.ACCESS_TOKEN_SECRET,
    {
      expiresIn: "20m",
    }
  );
```

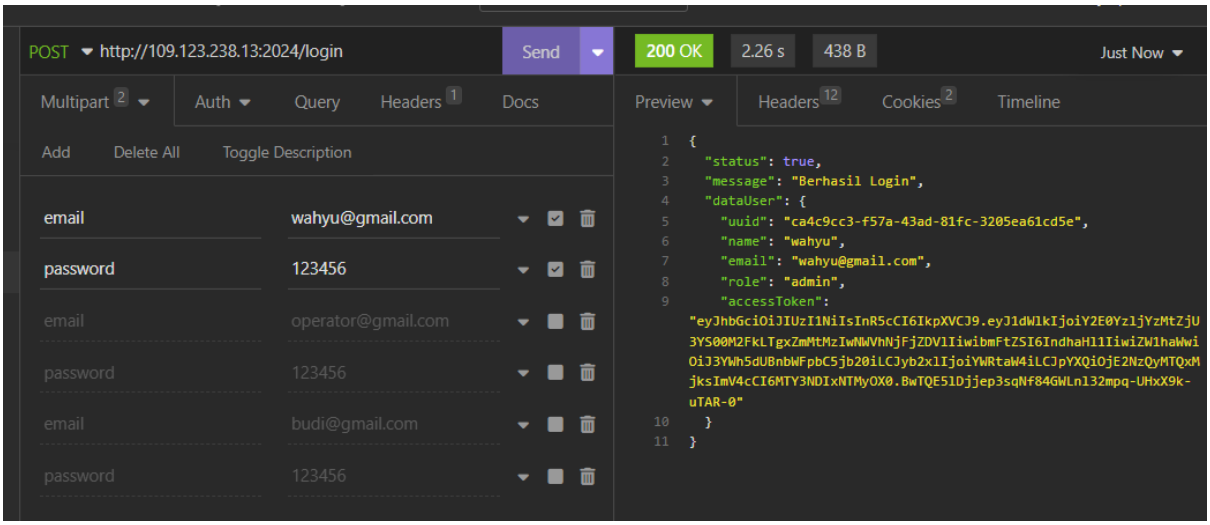
Digunakan untuk mendapatkan akses token jwt dari file .env, token tersebut bertahan selama 20 menit bersamaan dengan session login user.

```
// membuat variabel refreshToken dari data .env
const refreshToken = jwt.sign(
  { uuid, name, email, role },
  // proses env dari file .env
  process.env.REFRESH_TOKEN_SECRET,
  {
    expiresIn: "1d",
  }
);
// update refresh token ke database
await Users.update(
  { refresh_token: refreshToken },
  {
    where: {
      uuid: req.session.userId,
    },
  }
);

// membuat http only cookie
res.cookie("refreshToken", refreshToken, {
  httpOnly: true,
  maxAge: 24 * 60 * 60 * 1000,
});
res.status(200).json({
  status: true,
  message: "Berhasil Login",
  dataUser: { uuid, name, email, role, accessToken },
});
};
```

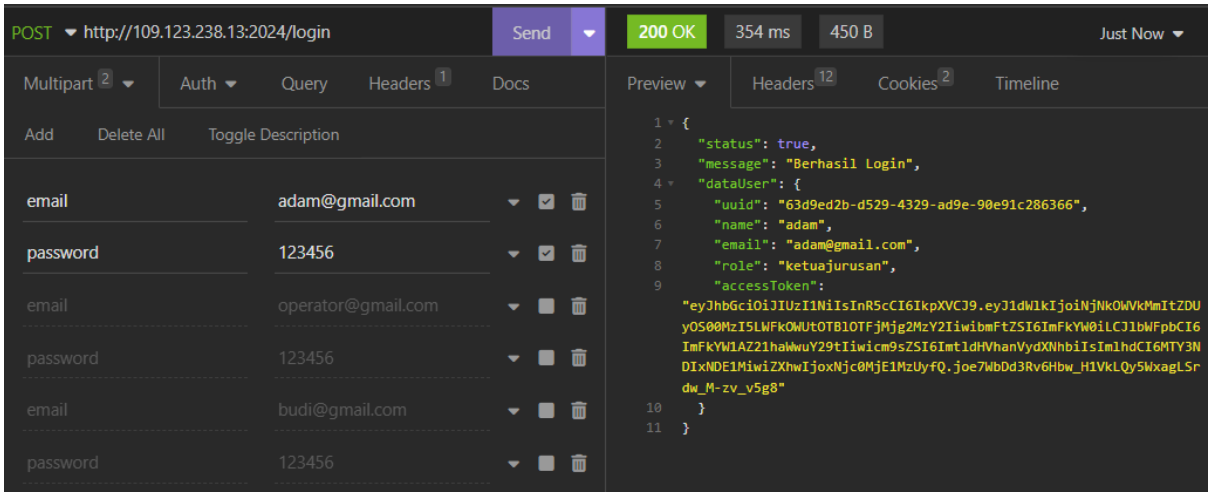
- Refresh Token berguna untuk memperbarui token setelah waktu dari access token habis, di dalam proses ini tidak perlu login ulang karena token sudah diberikan akses 1 hari.

- Namun jika pada session sudah habis waktunya maka tetap harus login ulang.

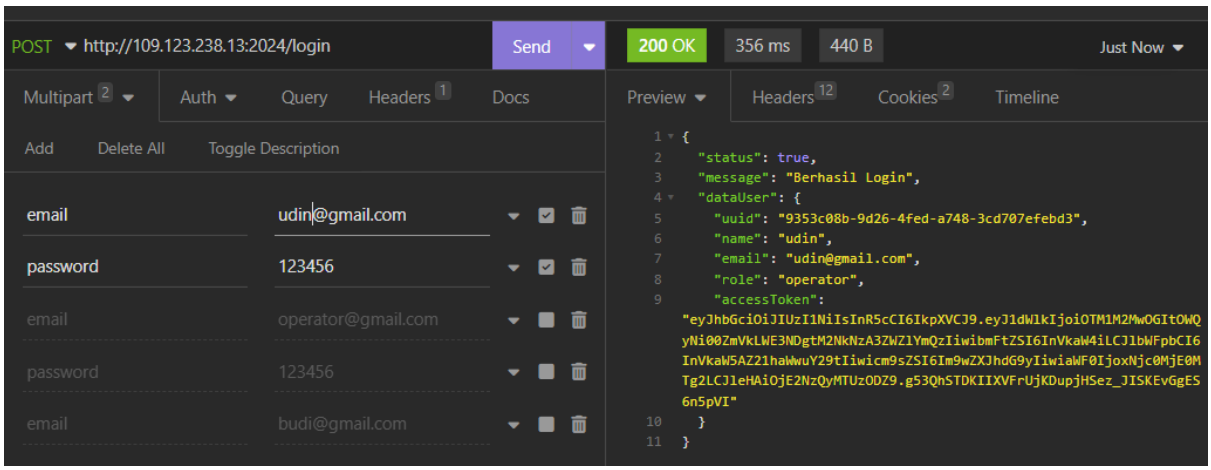
- Login Admin



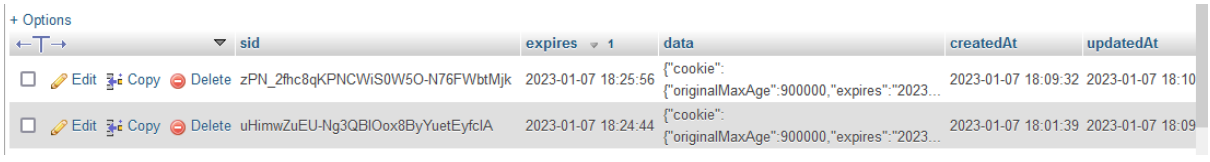
- Login Ketua Jurusan



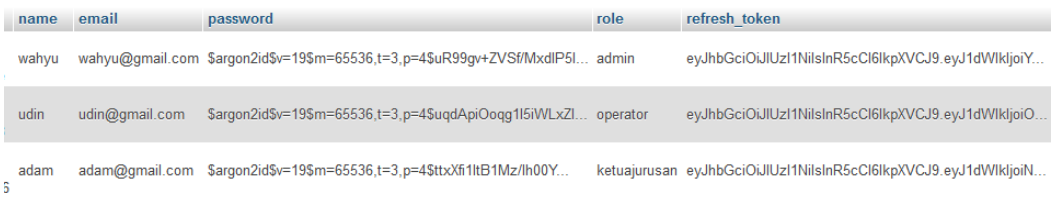
- Login Operator



- Check Session User
  Session user ketika login akan otomatis habis ketika sudah 15 Menit. Disini sampai jam 18.25 dan 18.24



- Check Refresh Token

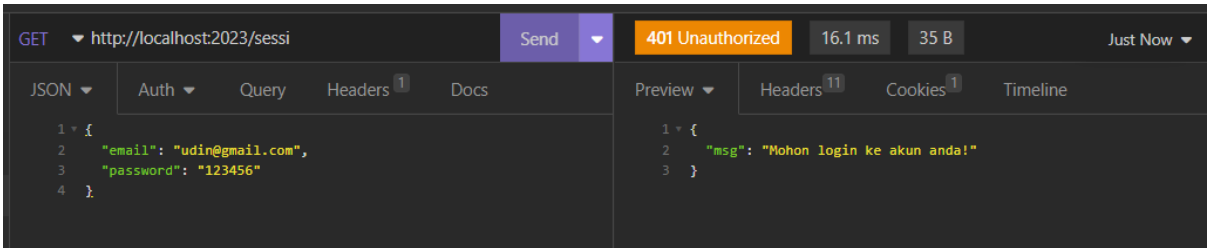


- Index.js yang digunakan untuk manajemen session menggunakan

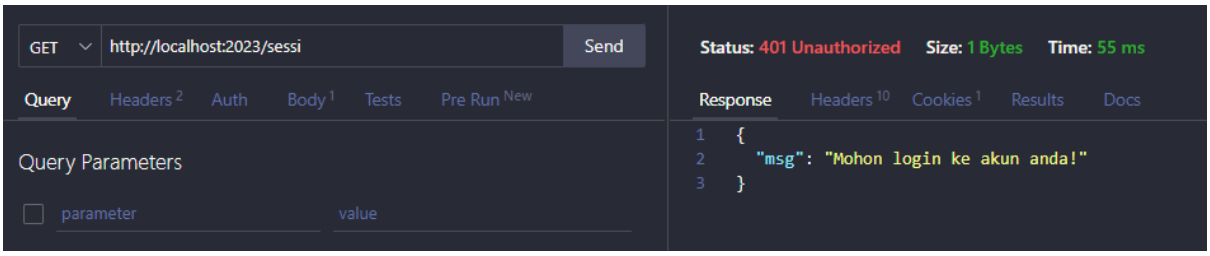
```
import session from "express-session";
import SequelizeStore from "connect-session-sequelize";
```

```
// digunakan untuk sessions
const sessionStore = SequelizeStore(session.Store);
const store = new sessionStore({
  db: db,
  checkExpirationInterval: 43200000, // interval hapus dari database setiap 12 jam
  expiration: 900000, // waktu session 15 menit
});

// membuat table session di database
// store.sync();

// definisikan session
app.use(
  session({
    secret: process.env.SESS_SECRET,
    resave: false,
    saveUninitialized: true,
    store: store,
    cookie: {
      // auto jika menggunakan http/s
      secure: "auto",
      maxAge: 900000,
    },
  })
);
```

- Ketika Session sudah habis
  - Login User



  - Login Admin



- Logout User (auth.js)

```
// function logout
export const logout = (req, res) => {
  req.session.destroy((err) => {
    // jika gagal logout
    if (err) return res.status(400).json({ msg: "Tidak dapat Logout" });
    // jika berhasil logout
    res.status(200).json({ msg: "Berhasil Logout" });
});
};
```

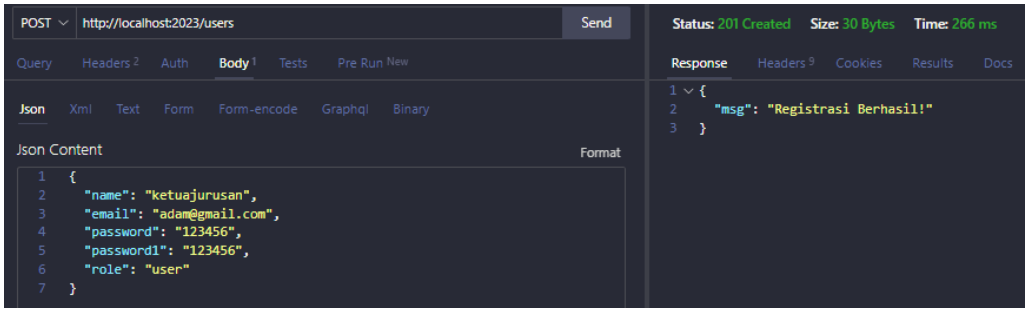
CRUD Pada Data User (Admin, User) pada controller Users.js

- Adduser

  Untuk menambah user, disini harus login sebagai admin, karena tujuan nya untuk inventaris data barang atau jasa/barang.

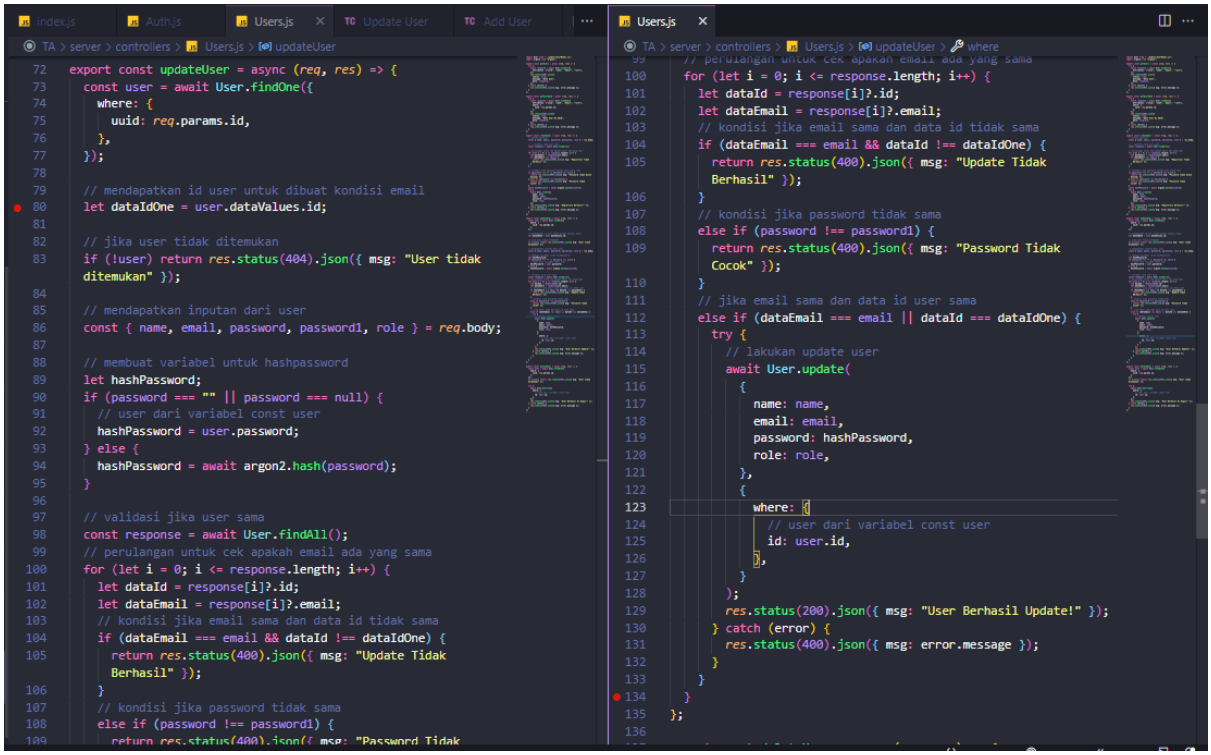
```
export const createUser = async (req, res) => {
  // proses destruct
  const { name, email, password, password1, role } = req.body;

  //   validasi jika user sama
  const response = await User.findAll();

  // perulangan untuk cek apakah email ada yang sama
  for (let i = 0; i <= response.length; i++) {
    let dataEmail = response[i]?.email;
    if (dataEmail === email) {
      return res.status(400).json({ msg: "Registrasi Tidak Berhasil" });
    }
  }

  // validasi jika password kosong atau tidak sama
  if (password === "" || password === null) {
    return res.status(400).json({ msg: "Password Tidak Boleh Kosong" });
  } else if (password !== password1) {
    return res.status(400).json({ msg: "Password Tidak Cocok" });
  }
  const hashPassword = await argon2.hash(password);
  try {
    await User.create({
      name: name,
      email: email,
      password: hashPassword,
      role: role,
    });
    res.status(201).json({ msg: "Registrasi Berhasil!" });
  } catch (error) {
    res.status(400).json({ msg: error.message });
  }
};
```

- membuat user dengan req body dari inputan
- Terdapat proses untuk pengecekan apakah ada email yang sama atau tidak.
- Jika password kosong atau tidak sama akan menampilkan pesan json.
- Jika berhasil maka password akan di hash menggunakan argon2.
- kemudian akan menjalankan proses create user.



- Update User
    - Untuk update user menggunakan method Patch. dengan mendapatkan id user menggunakan findOne berdasarkan id user.
    - Jika user tidak ditemukan tampilkan pesan json.
    - Kemudian menerima input dari body
    - Membuat kondisi untuk password jika password kosong maka akan menggunakan password yang lama, jika password di update maka akan melakukan update password baru dan akan melakukan hash password ulang.
    - Melakukan kondisi pengecekan email berdasarkan id
    - Jika email dan id sama maka akan melakukan update.

.



```
export const updateUser = async (req, res) => {
  const user = await User.findOne({
    where: {
      uuid: req.params.id,
    },
  });

  // mendapatkan id user untuk dibuat kondisi email
  let dataIdOne = user.dataValues.id;

  // jika user tidak ditemukan
  if (!user) return res.status(404).json({ msg: "User tidak
  ditemukan" });

  // mendapatkan inputan dari user
  const { name, email, password, password1, role } = req.body;

  // membuat variabel untuk hashpassword
  let hashPassword;
  if (password === "" || password === null) {
    // user dari variabel const user
    hashPassword = user.password;
  } else {
    hashPassword = await argon2.hash(password);
  }

  // validasi jika user sama
  const response = await User.findAll();
  // perulangan untuk cek apakah email ada yang sama
  for (let i = 0; i <= response.length; i++) {
    let dataId = response[i]?.id;
    let dataEmail = response[i]?.email;
    // kondisi jika email sama dan data id tidak sama
    if (dataEmail === email && dataId !== dataIdOne) {
      return res.status(400).json({ msg: "Update Tidak
      Berhasil" });
    }
    // kondisi jika password tidak sama
    else if (password !== password1) {
```

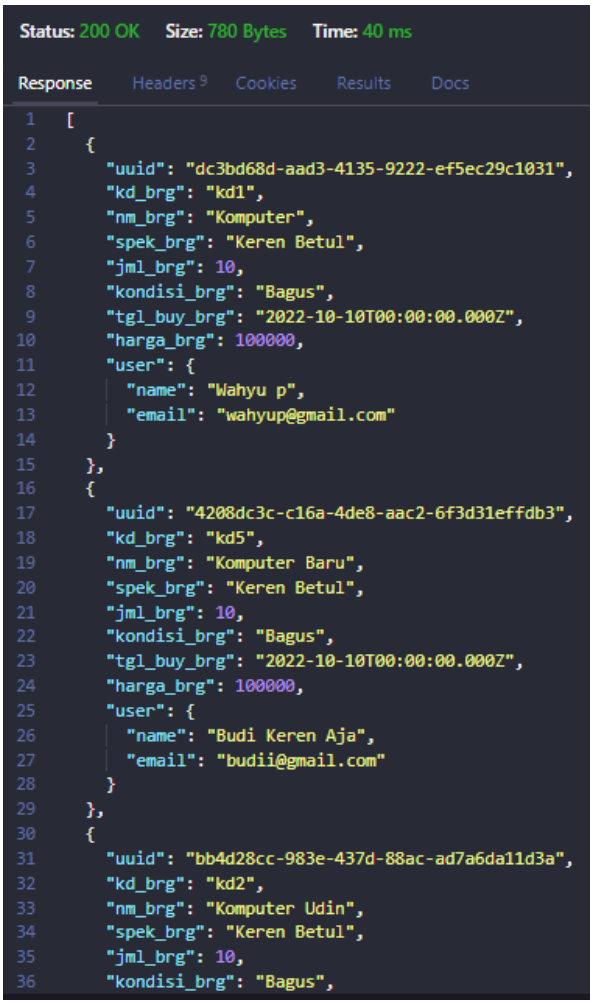
```
  for (let i = 0; i <= response.length; i++) {
    let dataId = response[i]?.id;
    let dataEmail = response[i]?.email;
    // kondisi jika email sama dan data id tidak sama
    if (dataEmail === email && dataId !== dataIdOne) {
      return res.status(400).json({ msg: "Update Tidak
      Berhasil" });
    }
    // kondisi jika password tidak sama
    else if (password !== password1) {
      return res.status(400).json({ msg: "Password Tidak
      Cocok" });
    }
    // jika email sama dan data id user sama
    else if (dataEmail === email || dataId === dataIdOne) {
      try {
        // lakukan update user
        await User.update(
          {
            name: name,
            email: email,
            password: hashPassword,
            role: role,
          },
          {
            where: {
              // user dari variabel const user
              id: user.id,
            },
          }
        );
        res.status(200).json({ msg: "User Berhasil Update!" });
      } catch (error) {
        res.status(400).json({ msg: error.message });
      }
    }
  }
};
```

- Delete User
    - Untuk delete user menggunakan method delete.
    - Jika user tidak ditemukan menampilkan json berupa User tidak ditemukan.
    - Jika user ditemukan berdasarkan uuid maka akan melakukan destroy atau menghapus user.

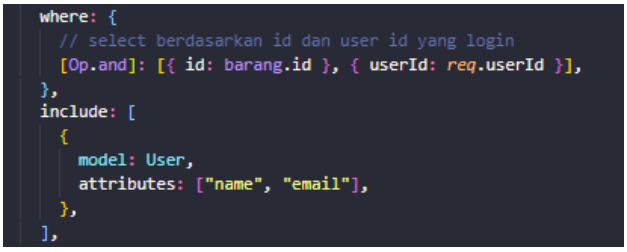
```
export const deleteUser = async (req, res) => {
  const user = await User.findOne({
    where: {
      uuid: req.params.id,
    },
  });
  if (!user) return res.status(404).json({ msg: "User tidak ditemukan" });

  try {
    await User.destroy({
      where: {
        // user dari variabel const user
        id: user.id,
      },
    });
    res.status(200).json({ msg: "User Berhasil Di Hapus!" });
  } catch (error) {
    res.status(400).json({ msg: error.message });
  }
};
```

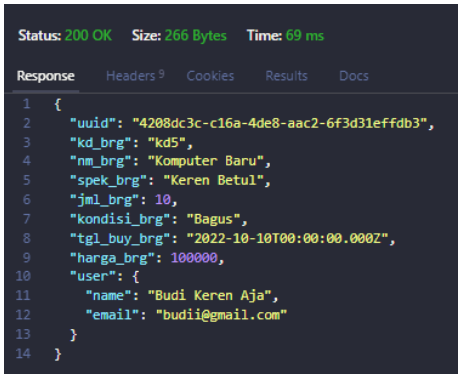
CRUD Pada Data Barang pada controller Barangs.js

- Get Semua Data Barang
    - Jika login sebagai admin, maka akan menampilkan seluruh data barang, yang di input oleh admin ataupun oleh user.
    - Jika login sebagai user data yang ditampilkan hanya berupa data yang di inputkan oleh user saja.



- Mendapatkan Barang berdasarkan id barang dan user yang login menggunakan op dari sequelize

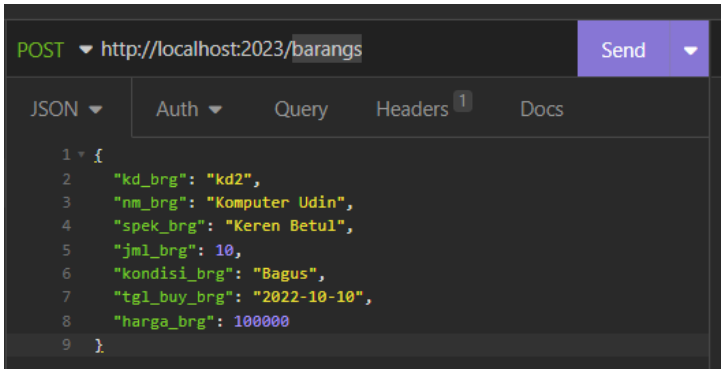




- Create Barang
  - Mendapatkan inputan dari json yang berisi data barang dari user (req body)
  - Kemudian melakukan pemeriksaaan untuk kode barang, bahwa kode barang tidak boleh sama.
  - Jika kondisi terpenuhi maka akan menjalankan proses membuat data barang.
  - Berdasarkan user yang menginput data maka dibutuhkan userId.

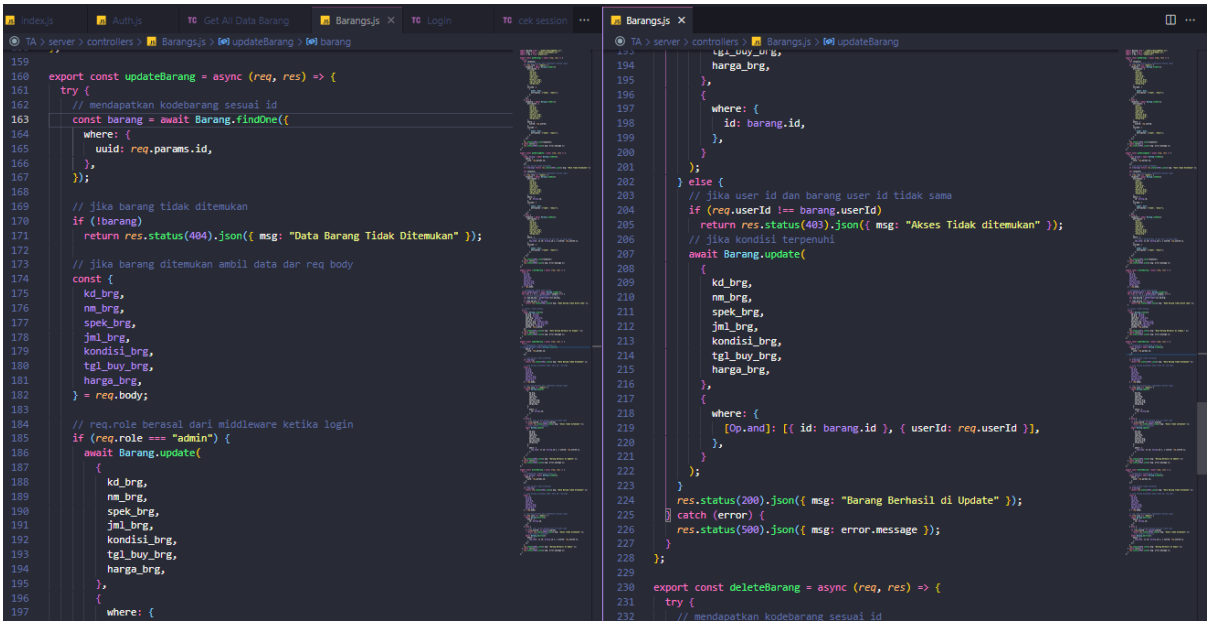
```
export const createBarang = async (req, res) => {
  const {
    kd_brg,
    nm_brg,
    spek_brg,
    jml_brg,
    kondisi_brg,
    tgl_buy_brg,
    harga_brg,
  } = req.body;

  // mendapatkan semua kode barang
  const getBarangAll = await Barang.findAll();
  for (let i = 0; i < getBarangAll.length; i++) {
    // console.log(getBarangAll[i].kd_brg);
    let new_kd_brg = getBarangAll[i].kd_brg;
    // jika kd_brg sama
    if (new_kd_brg === kd_brg)
      return res.status(500).json({ msg: "Kode Barang Tidak Boleh sama" });
  }

  // proses create barang
  try {
    await Barang.create({
      kd_brg: kd_brg,
      nm_brg: nm_brg,
      spek_brg: spek_brg,
      jml_brg: jml_brg,
      kondisi_brg: kondisi_brg,
      tgl_buy_brg: tgl_buy_brg,
      harga_brg: harga_brg,
      userId: req.userId,
    });
    res.status(201).json({ msg: "Data Barang Berhasil di Simpan." });
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
};
```

- Contoh method POST yang digunakan untuk input data barang menggunakan JSON.



- Update Barang
  - Untuk mendapatkan barang yang akan di update maka akan melakukan seleksi berdasarkan uuid data barang
  - Mendapatkan input dari user data barang
  - Kemudian jika user sebagai admin maka dapat melakukan update ke semua data barang dari user manapun.

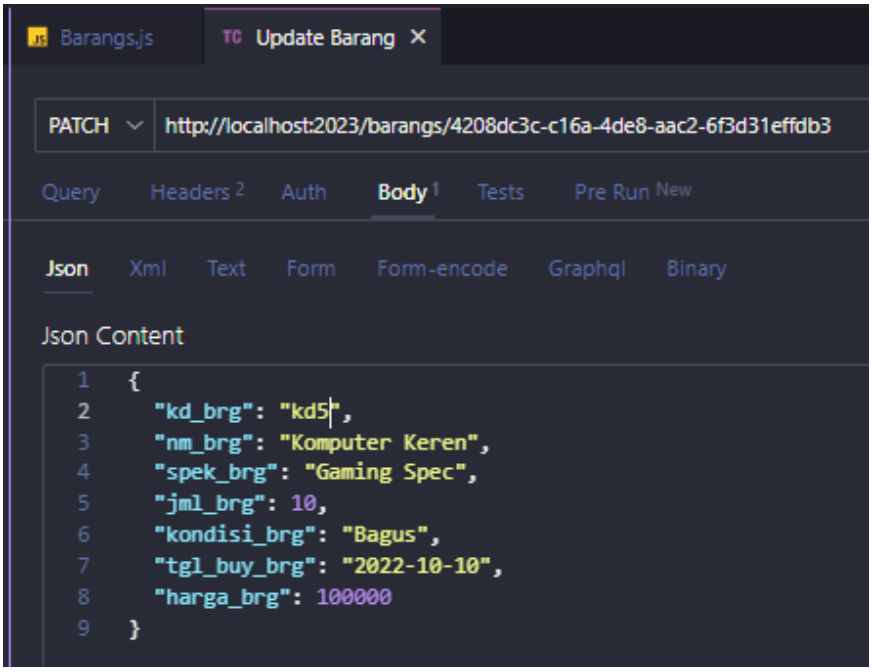
- Jika login sebagai user maka akan melakukan pemeriksaan uuid data user yang dikirim, dan jika kondisi terpenuhi maka akan melakukan Barang.Update.



```
export const updateBarang = async (req, res) => {
  try {
    // mendapatkan kodebarang sesuai id
    const barang = await Barang.findOne({
      where: {
        uuid: req.params.id,
      },
    });

    // jika barang tidak ditemukan
    if (!barang)
      return res.status(404).json({ msg: "Data Barang Tidak Ditemukan" });

    // jika barang ditemukan ambil data dar req body
    const {
      kd_brg,
      nm_brg,
      spek_brg,
      jml_brg,
      kondisi_brg,
      tgl_buy_brg,
      harga_brg,
    } = req.body;

    // req.role berasal dari middleware ketika login
    if (req.role === "admin") {
      await Barang.update(
        {
          kd_brg,
          nm_brg,
          spek_brg,
          jml_brg,
          kondisi_brg,
          tgl_buy_brg,
          harga_brg,
        },
        {
          where: {
```

```
          harga_brg,
        },
        {
          where: {
            id: barang.id,
          },
        }
      );
    } else {
      // jika user id dan barang user id tidak sama
      if (req.userId !== barang.userId)
        return res.status(403).json({ msg: "Akses Tidak ditemukan" });
      // jika kondisi terpenuhi
      await Barang.update(
        {
          kd_brg,
          nm_brg,
          spek_brg,
          jml_brg,
          kondisi_brg,
          tgl_buy_brg,
          harga_brg,
        },
        {
          where: {
            [Op.and]: [{ id: barang.id }, { userId: req.userId }],
          },
        }
      );
    }
    res.status(200).json({ msg: "Barang Berhasil di Update" });
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
};

export const deleteBarang = async (req, res) => {
  try {
    // mendapatkan kodebarang sesuai id
```

- Contoh proses update dari admin untuk data barang dari user budi



- Delete Barang
  - Proses hapus barang dengan cara mendapatkan data barang dengan findOne berdasarkan barang.id.
  - Jika user sebagai admin maka dapat menghapus seluruh data barang dari user lain.

- Jika login sebagai user maka hanya dapat menghapus untuk user yang login dan data id yang sama.

```
export const deleteBarang = async (req, res) => {
  try {
    // mendapatkan kodebarang sesuai id
    const barang = await Barang.findOne({
      where: {
        uuid: req.params.id,
      },
    });

    // jika barang tidak ditemukan
    if (!barang)
      return res.status(404).json({ msg: "Data Barang Tidak Ditemukan" });

    // req.role berasal dari middleware ketika login
    if (req.role === "admin") {
      await Barang.destroy({
        where: {
          id: barang.id,
        },
      });
    } else {
      // jika user id dan barang user id tidak sama
      if (req.userId !== barang.userId)
        return res.status(403).json({ msg: "Akses Tidak ditemukan" });
      // jika kondisi terpenuhi
      await Barang.destroy({
        where: {
          [Op.and]: [{ id: barang.id }, { userId: req.userId }],
        },
      });
    }
    res.status(200).json({ msg: "Barang Berhasil di Hapus" });
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
};
```

Penjelasan Program Front End

PENJELASAN PROGRAM FRONT END

Program FrontEnd di kembangkan menggunakan NodeJs Framework ReactJS. Menggunakan Dependencies :

```
"dependencies": {
  "@reduxjs/toolkit": "^1.9.1",
  "@testing-library/jest-dom": "^5.16.5",
  "@testing-library/react": "^13.4.0",
  "@testing-library/user-event": "^14.4.3",
  "axios": "^1.2.2",
  "bootstrap": "^5.1.0",
  "bulma": "^0.9.4",
  "react": "^18.2.0",
  "react-bootstrap": "^2.0.0-beta.5",
  "react-bulma-dropdown": "^0.1.10",
  "react-dom": "^18.2.0",
  "react-icons": "^4.7.1",
  "react-redux": "^8.0.5",
  "react-router-dom": "^6.6.1",
  "react-scripts": "5.0.1",
  "web-vitals": "^2.1.4"
},
```

Membuat Halaman Navbar dengan Framework Css Bulma :

Navbar.jsx M

client > src > component > Navbar.jsx > Navbar

```
1  import React from 'react'
2  import { NavLink, useNavigate } from "react-router-dom";
3
```

Import Navlink untuk mengarahkan setiap Menu dengan React-router-dom

```jsx
import React from 'react'
import { NavLink, useNavigate } from "react-router-dom";

const Navbar = () => {
  return (
    <div>
      <nav className="navbar is-info is-fixed-top has-shadow" role="navigation" aria-label="main navigation">
        <div className="navbar-brand">
          <NavLink to="/dashboard" className="navbar-item" href="https://bulma.io">
            <img src="https://bulma.io/images/bulma-logo.png" width="112" height="28"/>
          </NavLink>

          <a role="button" className="navbar-burger burger" aria-label="menu" aria-expanded="false" data-target="navbarBasicExample">
            <span aria-hidden="true"></span>
            <span aria-hidden="true"></span>
            <span aria-hidden="true"></span>
          </a>
        </div>

        <div id="navbarBasicExample" className="navbar-menu">
          <div className="navbar-start">

          </div>

          <div className="navbar-end">
            <div className="navbar-item">
              <div className="buttons">
                <a className="button is-primary">
                  <strong>Keluar</strong>
                </a>
              </div>
            </div>
          </div>
        </div>
      </nav>
    </div>
  )
}

export default Navbar
```

Membuat Halaman Sidebar dengan mengimport IoPerson, IoHome dari react-icons/io5 Agar bagian setiap menu memiliki Icon.

Navbar.jsx M Sidebar.jsx M

client > src > component > Sidebar.jsx > ...

```jsx
import React from "react";
import { NavLink } from "react-router-dom";
import { IoPerson, IoHome, } from "react-icons/io5";
```

```jsx
const Sidebar = () => {
  return (
    <div>
      <aside class="menu pl-2 has-shadow">
    <p class="menu-label">
      General
    </p>
    <ul class="menu-list">
      <li><NavLink to={"/dashboard"}><IoHome/> Dashboard</NavLink></li>
    </ul>
    <p class="menu-label">
      Administration
    </p>
    <ul class="menu-list">
      <li>
        <a class=""><IoPerson/> Manage Users</a>
        <ul>
          <li><NavLink to={"/admin"}>Admin</NavLink></li>
          <li><NavLink to={"/operator"}>Operator</NavLink></li>
          <li><NavLink to={"/kajur"}>Ketua Jurusan</NavLink></li>
        </ul>
      </li>
    </ul>
    <p class="menu-label">
      Belanja
    </p>
    <ul class="menu-list">
      <li><NavLink to={"/barangm"}>Barang Modal</NavLink></li>
      <li><a>Habis Pakai</a></li>
    </ul>
  </aside>
      </div>


  )
}
export default Sidebar
```

Membuat Halaman Login dengan :

Mengimport useDispatch, useSelector dari react-redux

Menimport LoginUser, reset

client > src > component > Login.jsx > ...

```jsx
import React, { useState, useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { useNavigate } from "react-router-dom";
import { LoginUser, reset } from "../features/authSlice";
```

```
const Login = () => {
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const dispatch = useDispatch();
    const navigate = useNavigate();
    const { user, isError, isSuccess, isLoading, message } = useSelector(
      (state) => state.auth
    );

    useEffect(() => {
      if (user || isSuccess) {
        navigate("/dashboard");
      }
      dispatch(reset());
    }, [user, isSuccess, dispatch, navigate]);

    const Auth = (e) => {
      e.preventDefault();
      dispatch(LoginUser({ email, password }));
    };
```

```
return (
  <div class="hero is-fullheight">
        <div class="hero-body is-justify-content-center is-align-items-center">
            <div class="columns is-flex is-flex-direction-column box">
                <div class="column">
                    <label for="email">Email</label>
                    <input class="input is-primary" type="text" placeholder="Email address"/>
                    <input
                  type="text"
                  className="input"
                  value={email}
                  onChange={(e) => setEmail(e.target.value)}
                  placeholder="Email"
                />
                </div>
                <div class="column">
                    <label for="Name">Password</label>
                    <input class="input is-primary" type="password" placeholder="Password"/>
                    <input
                  type="password"
                  className="input"
                  value={password}
                  onChange={(e) => setPassword(e.target.value)}
                  placeholder="******"
                />
                    <a href="forget.html" class="is-size-7 has-text-primary">forget password?</a>
                </div>
                <div class="column">
                    <button class="button is-info is-fullwidth" type="submit">Login</button>
                </div>
                <div class="has-text-centered">
                    <p class="is-size-7"> Don't have an account? <a href="#" class="has-text-primary">Sign up</a>
                    </p>
                </div>
            </div>
        </div>
    </div>
)
```

client > src > component > Barang.jsx > Barang

```
import React, { useState, useEffect } from "react";
import { Link } from "react-router-dom";
import axios from "axios";
```

```
const Barang = () => {
    const [barang, setBarangm] = useState([]);

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

    const getBarangm = async () => {
      const response = await axios.get("http://localhost:2023/barangs");
      setBarangm(response.data);
    };

    const deleteBarangm = async (barangId) => {
      await axios.delete(`http://localhost:2023/barangs/${barangId}`);
      getBarangm();
    };

  return (
    <div>
      <h1 className="title">Barang Modal</h1>
      <table className="table is-striped is-fullwidth">
        <thead>
            <tr>
                <th>Id</th>
                <th>Kode Barang</th>
                <th>Nama Barang</th>
                <th>Spesifikasi</th>
                <th>Jumlah Barang</th>
                <th>Tanggal Pembelian</th>
                <th>Harga Barang</th>
            </tr>
        </thead>
```

```
            <tbody>
            {barang.map((Barang, index) => (
                <tr key={Barang.uuid}>
                  <td>{index + 1}</td>
                  <td>{barang.name}</td>
                  <td>{barang.price}</td>
                  <td>{barang.name}</td>
                  <td>
                    <Link
                      to={`/products/edit/${barang.uuid}`}
                      className="button is-small is-info"
                    >
                      Edit
                    </Link>
                    <button
                      onClick={() => deleteBarangm(barang.uuid)}
                      className="button is-small is-danger"
                    >
                      Delete
                    </button>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      )
    }

    export default Barang
```