Menggunakan library

**frontend**

npx create-react-app qte-coffee (create react app)

npm i tailwindcss (install tailwind css)

init tailwindcss (initializing tailwindcss)

npm axios (install axios)

npm I install moment-timezone

npm install react-router-dom

**backend**

npm i express

npm i bycrypt

npm i cors

npm i jsonwebtoken

npm i mongoose

npm i multer

npm i nodemon

npm dotenv

npm I express-validator

npm I body-parser

versi

"dependencies": {

"bcrypt": "^5.1.1",

"bcryptjs": "^2.4.3",

"body-parser": "^1.20.2",

"cors": "^2.8.5",

"dotenv": "^16.4.5",

"ejs": "^3.1.10",

"express": "^4.19.2",

"express-validator": "^7.0.1",

"jsonwebtoken": "^9.0.2",

"mongoose": "^8.3.4",

"multer": "^1.4.5-lts.1"

//

Backend kasih atas nama **(DONE)**

Ganti date option menjadi gmt+7 **(DONE)**

Pendapatan perharian get all, calculate price **(DONE)**

Done order masuk database **(DONE)**

Todo:

Total harga di database (edit backend) **(DONE)**

Card visual (frontend) **(DONE)**

**Referensi**

**Backend**

* Membuat backend expressjs:

<https://expressjs.com/en/starter/installing.html>

<https://medium.com/@naveednadaf/quick-node-express-js-project-setup-guide-88cd4d9a7af3>

* Menyambungkan backend dengan database mongodb:

<https://mongoosejs.com/docs/>

* bikin login:   
  <https://www.topcoder.com/thrive/articles/authentication-and-authorization-in-express-js-api-using-jwt>
* Routes, Controller:  
  <https://dev.to/ericchapman/nodejs-express-part-5-routes-and-controllers-55d3>

**Frontend:**

* Membuat project react:

<https://react.dev/learn/installation>

<https://reactjs.id/>

* Dokumentasi ReactJS:

<https://devdocs.io/react/>

* Styling TailwindCSS:

<https://tailwindcss.com/docs/installation>

**Dokumentasi Backend dengan Express.js dan MongoDB**

Pendahuluan

Dokumentasi ini menjelaskan cara membangun backend sederhana menggunakan Express.js dan MongoDB. Proyek ini mencakup fitur-fitur seperti autentikasi pengguna, pembuatan pesanan, dan middleware untuk memeriksa peran pengguna. Struktur direktori proyek adalah sebagai berikut:

**- controllers/**

- orderController.js

- userController.js

**- middleware/**

- middleware.js

- roleMiddleware.js

**- models/**

- Order.js

- User.js

**- routes/**

- orderRoutes.js

- user.routes.js

**- validation/**

- validation.js

- .env

- index.js

**Konfigurasi Awal**

Buat file `.env` untuk menyimpan konfigurasi lingkungan:

PORT=5000

MONGODB\_URI=mongodb://localhost:27017/qtecoffee

JWT\_SECRET=ealahkoasiudhaisuh

**Model**

Model digunakan untuk mendefinisikan skema database.

**models/Order.js**

const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({

alias: {

type: String,

required: true

},

items: [

{

category: {

type: String,

required: true

},

name: {

type: String,

required: true

},

price: {

type: Number,

required: true

},

quantity: {

type: Number,

required: true

}

}

],

createdAt: {

type: Date,

default: Date.now

}

});

const Order = mongoose.model('Order', orderSchema);

module.exports = Order;

Kode di atas mendefinisikan skema pesanan (orderSchema). Setiap pesanan memiliki alias, items (array dari objek yang mencakup kategori, nama, harga, dan kuantitas), dan createdAt yang otomatis diisi dengan waktu saat pesanan dibuat.

**models/User.js**

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

username: { type: String, required: true, unique: true },

password: { type: String, required: true },

role: { type: String, enum: ['admin', 'staff'], required: true }

});

const User = mongoose.model('User', userSchema);

module.exports = User;

Kode di atas mendefinisikan skema pengguna (userSchema). Setiap pengguna memiliki username, password yang dienkripsi, dan role (admin atau staff).

**Controller**

Controller digunakan untuk menangani logika bisnis dan interaksi dengan database.

**controllers/orderController.js**

const Order = require('../models/Order');

// Membuat order

exports.createOrder = async (req, res) => {

try {

const { alias, items } = req.body;

if (!alias || !items || !Array.isArray(items) || items.length === 0) {

return res.status(400).json({ message: 'Alias and items are required and items must be a non-empty array' });

}

for (let item of items) {

if (!item.category || !item.name || !item.price || !item.quantity) {

return res.status(400).json({ message: 'Each item must include category, name, price, and quantity' });

}

}

const newOrder = new Order({ alias, items });

await newOrder.save();

res.status(201).json({ message: 'Order placed successfully', order: newOrder });

} catch (error) {

console.error('Error placing order:', error);

res.status(500).json({ message: 'Error placing order' });

}

};

// Mengambil order dari database

exports.getOrders = async (req, res) => {

try {

const orders = await Order.find();

res.status(200).json(orders);

} catch (error) {

console.error('Error fetching orders:', error);

res.status(500).json({ message: 'Error fetching orders' });

}

};

**createOrder**: Fungsi ini menerima data pesanan dari request body dan menyimpan pesanan baru ke database. Jika ada kesalahan dalam input data, fungsi ini akan mengembalikan pesan kesalahan yang sesuai.

**getOrders**: Fungsi ini mengambil semua pesanan dari database dan mengembalikannya dalam response.

**controllers/userController.js**

require("dotenv").config();

const User = require("../models/User");

const bcryptjs = require("bcryptjs");

const jsonwebtoken = require("jsonwebtoken");

exports.LoginUser = async (req, res) => {

const { username, password } = req.body;

const dataUser = await User.findOne({ $or: [{ username: username }] });

if (dataUser) {

const passwordUser = await bcryptjs.compare(password, dataUser.password);

if (passwordUser) {

const data = {

id: dataUser.\_id,

};

const token = jsonwebtoken.sign(data, process.env.JWT\_SECRET);

return res.status(201).json({

message: "berhasil",

token: token,

role: dataUser.role,

});

} else {

return res.status(404).json({

status: false,

message: "Password yang dimasukkan salah",

});

}

} else {

return res.status(404).json({

status: false,

message: "Username atau Email yang dimasukkan salah",

});

}

};

LoginUser: Fungsi ini menerima username dan password dari request body. Jika username ditemukan di database dan password yang diberikan cocok, fungsi ini akan menghasilkan token JWT yang mengandung ID pengguna. Token ini digunakan untuk autentikasi di endpoint lainnya.

**Middleware**

Middleware digunakan untuk menangani autentikasi dan otorisasi.

**middleware/middleware.js**

require("dotenv").config();

const jsonwebtoken = require("jsonwebtoken");

const User = require('../models/User');

module.exports = async (req, res, next) => {

const token = req.header("Authorization");

if (!token) {

return res.status(401).json({

message: "Anda perlu login",

});

}

try {

const decode = jsonwebtoken.verify(token, process.env.JWT\_SECRET);

req.id = decode.id;

const user = await User.findById(decode.id);

if (!user) {

return res.status(401).json({

message: "User not found",

});

}

req.user = user;

next();

} catch (e) {

res.status(401).json({

message: "Illegal login",

});

}

};

**Penjelasan**

Middleware ini memeriksa apakah token yang diberikan valid. Jika valid, middleware menambahkan ID pengguna dan data pengguna ke objek request dan melanjutkan ke middleware berikutnya.

**middleware/roleMiddleware.js**

const jwt = require('jsonwebtoken');

const roleMiddleware = (roles) => {

return (req, res, next) => {

const token = req.header('Authorization');

if (!token) {

return res.status(401).json({ message: 'No token, authorization denied' });

}

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

if (!roles.includes(decoded.role)) {

return res.status(403).json({ message: 'Access denied' });

}

req.user = decoded;

next();

} catch (err) {

res.status(401).json({ message: 'Token is not valid' });

}

};

};

module.exports = roleMiddleware;

**Penjelasan**

Middleware ini memeriksa apakah peran pengguna termasuk dalam daftar peran yang diizinkan untuk mengakses rute tersebut. Jika tidak, middleware akan mengembalikan pesan Access denied.

**Validasi**

Validasi input menggunakan express-validator.

**validation/validation.js**

const { check, validationResult } = require("express-validator");

exports.runValidation = (req, res, next) => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

return res.status(404).json({

status: false,

message: errors.array()[0].msg,

});

}

next();

};

exports.validationLogin = [

check("username", "username tidak boleh kosong").notEmpty(),

check("password", "password tidak boleh kosong").notEmpty()

];

**Penjelasan**

**runValidation:** Middleware ini memeriksa hasil validasi dan mengembalikan pesan kesalahan jika ada input yang tidak valid.

**validationLogin:** Middleware ini memastikan bahwa username dan password tidak kosong.

**Routes**

Routes digunakan untuk mendefinisikan endpoint.

**routes/orderRoutes.js**

const express = require('express');

const router = express.Router();

const orderController = require('../controllers/orderController');

router.post('/', orderController.createOrder);

router.get('/', orderController.getOrders);

module.exports = router;

**Penjelasan**

**POST /:** Rute ini memanggil fungsi createOrder dari orderController untuk membuat pesanan baru.

**GET /:** Rute ini memanggil fungsi getOrders dari orderController untuk mengambil semua pesanan dari database.

**routes/user.routes.js**

const express = require("express");

const router = express.Router();

const middleware = require("../middleware/middleware.js")

const { LoginUser } = require("../controllers/userController.js");

const { runValidation, validationLogin } = require("../validation/validation");

const roleMiddleware = require('../middleware/roleMiddleware');

// USER AUTH

router.post("/login", validationLogin, runValidation, LoginUser);

router.get('/history', roleMiddleware(['admin']), (req, res) => {

res.send('Dashboard');

});

module.exports = router;

**Penjelasan**

**POST /login:** Rute ini memanggil fungsi LoginUser dari userController untuk melakukan autentikasi pengguna.

**GET /history:** Rute ini hanya dapat diakses oleh pengguna dengan peran admin, menggunakan roleMiddleware untuk memeriksa peran pengguna.

**Entry Point**

File utama untuk menginisialisasi server.

**index.js**

require('dotenv').config();

const express = require('express');

const bodyParser = require('body-parser');

const mongoose = require('mongoose');

const cors = require('cors');

const RouteUser = require('./routes/user.routes');

const coffeeMenuRoutes = require('./routes/coffeeMenuRoutes');

const orderRoutes = require('./routes/orderRoutes');

const app = express();

console.log('MONGODB\_URI:', process.env.MONGODB\_URI);

console.log('PORT:', process.env.PORT);

mongoose

.connect(process.env.MONGODB\_URI, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('Database connected'))

.catch((e) => console.error('Database connection error:', e));

app.use(cors());

app.use(bodyParser.json());

app.use('/user', RouteUser);

app.use('/menu', coffeeMenuRoutes);

app.use('/orders', orderRoutes);

app.listen(process.env.PORT, () => {

console.log('Server running on port', process.env.PORT);

});

**Penjelasan**

Mongoose: Menghubungkan aplikasi ke database MongoDB menggunakan URI yang disimpan dalam file .env.

Express: Membuat server Express dan mengatur middleware seperti cors dan bodyParser.

Routes: Menyusun rute untuk pengguna, menu, dan pesanan.

Server: Menjalankan server pada port yang ditentukan dalam file .env.

**Kesimpulan**

Penelitian ini berhasil membangun backend aplikasi menggunakan Express.js dan MongoDB yang efektif dan efisien. Beberapa poin utama yang dapat disimpulkan adalah:

1. Efisiensi Pengembangan: Express.js memfasilitasi pengembangan server yang modular dan mudah dikelola.

2. Keamanan Data: Penggunaan JWT untuk autentikasi dan bcrypt.js untuk hashing password memastikan keamanan data pengguna.

3. Manajemen Pesanan: Sistem ini memungkinkan penyimpanan dan pengambilan data pesanan dengan validasi input yang ketat, mengurangi kesalahan data.

4. Fleksibilitas dan Skalabilitas: MongoDB menawarkan fleksibilitas dalam penyimpanan data dan kemampuan untuk skala sesuai pertumbuhan aplikasi.

5. Middleware untuk Pengamanan Akses: Middleware memastikan hanya pengguna terotorisasi yang dapat mengakses fitur tertentu, menjaga integritas dan keamanan aplikasi.

**Dokumentasi Frontend dengan Reactjs**

1. App.js

Tujuan:

- `App.js` adalah file utama dalam aplikasi React. Ini bertanggung jawab untuk menentukan rute aplikasi dan halaman yang akan ditampilkan.

Alur Kerja:

- Menggunakan `react-router-dom` untuk menentukan rute.

- Setiap rute menunjuk ke komponen halaman yang sesuai.

2. pages/Dashboard.js

Tujuan:

- `Dashboard.js` adalah halaman utama dashboard untuk aplikasi Coffee Shop.

- Ini menampilkan informasi tentang pemesanan, riwayat penjualan, dan pendapatan harian.

- Menyediakan fungsionalitas untuk menambahkan pesanan dan logout.

Alur Kerja:

- Menggunakan `useEffect` untuk memeriksa token pengguna setelah komponen dipasang.

- Menyediakan tombol untuk navigasi antar halaman.

- Menggunakan `Routes` dari `react-router-dom` untuk menangani navigasi internal.

3. pages/Landing.js

Tujuan:

- `Landing.js` adalah halaman login untuk aplikasi Coffee Shop.

- Ini memungkinkan pengguna untuk masuk ke aplikasi dengan mengirimkan username dan password.

Alur Kerja:

- Menggunakan `axios` untuk mengirim permintaan HTTP ke backend.

- Menangani penanganan kesalahan dan pesan kesalahan.

4. pages/Menu.js

Tujuan:

- `Menu.js` adalah halaman untuk menampilkan menu kopi dan melakukan pemesanan.

Alur Kerja:

- Menampilkan kategori menu dan item-menu dari objek `menuOptions`.

- Menggunakan state untuk menyimpan pesanan dan jumlah yang dibayar.

- Menghitung total dan kembalian dari pesanan.

- Menggunakan `axios` untuk mengirim pesanan ke backend.

5. components/HistoryPenjualan.js

Tujuan:

- `HistoryPenjualan.js` adalah komponen untuk menampilkan riwayat penjualan.

Alur Kerja:

- Menggunakan `useEffect` untuk mengambil data pesanan dari backend saat komponen dipasang.

- Menampilkan pesanan dalam bentuk tabel dengan detail alias, item, total harga, dan tanggal.

6. components/menuOption.js

Tujuan:

- `menuOption.js` berisi daftar menu kopi yang tersedia.

Alur Kerja:

- Menyediakan objek yang berisi kategori dan item-menu kopi.

7. components/Pemesanan.js

Tujuan:

- `Pemesanan.js` adalah komponen untuk menampilkan formulir pemesanan.

Alur Kerja:

- Menyediakan formulir untuk memilih kategori dan item-menu.

- Menggunakan `axios` untuk mengirim pesanan ke backend.

8. components/PendapatanHarian.js

Tujuan:

- `PendapatanHarian.js` adalah komponen untuk menampilkan pendapatan harian.

Alur Kerja:

- Menggunakan `useEffect` untuk mengambil data pesanan dari backend saat komponen dipasang.

- Menghitung total pendapatan harian dan menampilkannya dalam bentuk tabel.