SMK Telkom Malang

JOB SHEET

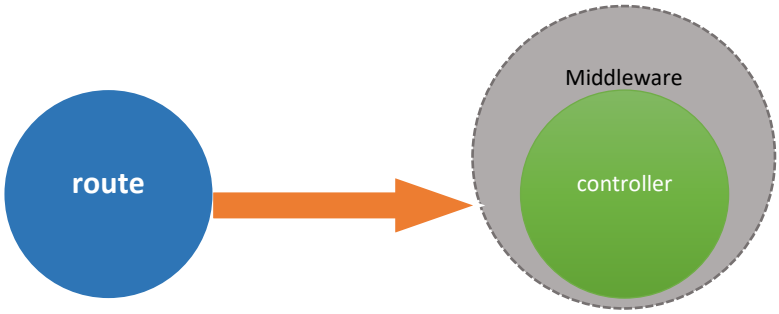
NODE JS

Middleware

Disusun Oleh:
TIM MGMP RPL
SMK TELKOM MALANG

KONSEP MIDDLEWARE

Setelah kita mempelajari CRUD menggunakan Library Sequelize pada modul sebelumnya, kita telah menerapkan sebuah code-based sederhana untuk membangun project "School Library". Pada code-based yang kita terapkan, terdapat sebuah proses yang disebut "middleware". Middleware merupakan sebuah proses yang dilalui dari proses routing ke controller. Hal tersebut diterapkan agar controller fokus pada proses sesuai dengan yang ditentukan tanpa harus melakukan proses lain seperti validasi inputan dan autentikasi user. Proses *middleware* ini bersifat opsional tergantung dari kebutuhan. Berikut ini diagram yang menggambarkan proses middleware.



Dalam modul ini kita akan membahas bagaimana cara untuk membuat proses middleware sederhana dan proses middleware untuk validasi request dan authorize user.

A. Pembuatan Middleware Sederhana

Dalam bagian ini, kita akan membuat middleware sederhana untuk lebih memahami konsep berjalannya middleware.

1. Buat folder baru dengan nama "**middlewares**" pada root folder project!
2. Pada folder **middlewares**, buat file baru dengan nama "simple-middleware.js"!
3. Buat fungsi middleware pertama seperti berikut ini!

```
/** create first simple middleware  */
const midOne = async (request, response, next) => {
    console.log(`Run Middleware One`)
    next()
    /** next() function used to continue to the controller
process */
}
```

4. Lakukan *export* pada fungsi yang telah dibuat sebelumnya!

```
/** export function to another file */
```

```
module.exports = {
    midOne
}
```

5. Pada file route untuk CRUD data buku (book.route.js), muat file "simple-middleware.js" yang telah dibuat menggunakan code berikut!

```
/** load function from simple-middleware */
const { midOne } = require(`../middlewares/simple-middleware`)
```

6. Setelah itu fungsi middleware "midOne" akan disisipkan pada route untuk mengambil data buku (sebagai contoh)!

```
/** create route to get data with method "GET" */
app.get("/",[midOne], bookController.getAllBooks)
```

7. Jalankan server backend dan akses pada Postman request untuk mengambil data buku! Setelah itu untuk membuktikan middlware "midOne" dijalankan, lihat pada terminal setelah request get data buku dijalankan!

```
Server of School's Library runs on port 8000
Run Middleware One
Executing (default): SELECT `id`, `isbn`, `title`, `author`,
datedAt` FROM `books` AS `book`;
```

B. Pembuatan Middleware untuk Validasi Input

Validasi input merupakan sebuah proses yang dijalankan saat ada data input/request dikirimkan. Proses ini digunakan untuk penyaringan data agar saat diterima oleh controller, data tersebut valid untuk diproses dibagian selanjutnya. Pada bagian ini kita akan menggunakan library tambahan untuk proses validasi data yaitu Library **Joi**. Untuk dokumentasi lengkap tentang penggunaan Joi dapat melihat pada laman berikut https://joi.dev/api/ .

Pada bagian ini kita akan membahas untuk proses validasi request pada CRUD data member.

1. Install library Joi melalui terminal dengan menjalankan perintah berikut!
   `npm i -s joi`
2. Buat file baru pada folder middleware dengan nama "member-validation.js" dan tuliskan code berikut!

```
/** load Joi Library */
const Joi = require(`joi`)

/** create function to validate request of member */
const validateMember = (request, response, next) => {
```

```
    /** define rules for request */
    const rules = Joi
        .object()
        .keys({
            /** name is required */
            name: Joi.string().required(),
            /** address is required */
            address: Joi.string().required(),
            /** contact is number only and required */
            contact: Joi.number().required(),
            /** gender is only "Male" and "Female" allowed */
            gender: Joi.string().valid(`Male`, `Female`)
        })
        .options({ abortEarly: false })

    /** get error of validation if it exists */
    let { error } = rules.validate(request.body)

    /** if error is exist */
    if (error != null) {
        /** get all error message */
        let errMessage = error.details.map(it =>
it.message).join(",")

        /** return error message with code 422 */
        return response.status(422).json({
            success: false,
            message: errMessage
        })
    }

    /** if error doesn't exist, continue to controller */
    next()
}

module.exports = { validateMember }
```

3. Buka file member.route.js dan muat middleware untuk validasi input pada member!

```
/** load middleware for validation request */
let { validateMember } = require(`../middlewares/member-
validation`)
```

4. Sisipkan fungsi middleware validasi pada route untuk menambah dan mengubah data member!

```
/** create route to add new member using method "POST" */
app.post("/", [validateMember],memberController.addMember)

/** create route to update member
 * using method "PUT" and define parameter for "id" */
app.put("/:id", [validateMember], memberController.updateMember)
```

5. Simpan semua perubahan file dan jalankan server backend!
6. Akses request tambah data member pada Postman dan coba berikan request data seperti berikut ini!

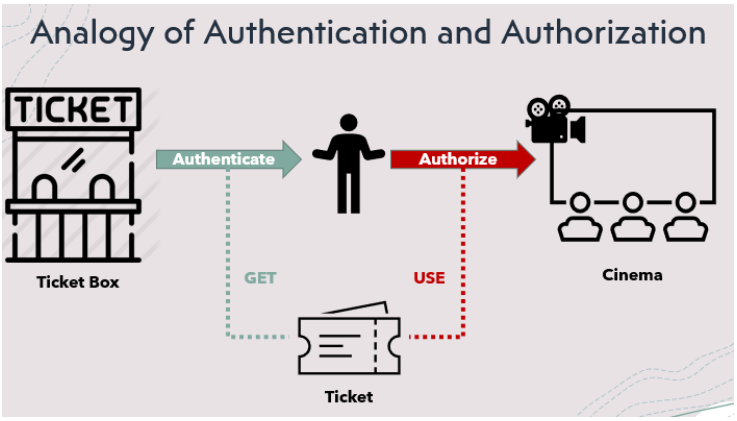
```
{
    "name":"Eddie",
    "address":"Kongo",
    "contact":"099182sss",
    "gender":"Males"
}
```

Setelah itu klik Send dan kita akan mendapatkan pesan error akibat request yang kita kirimkan tidak valid.

```
{
    "success": false,
    "message": "\"contact\" must be a number,\"gender\" must be one of [Male, Female]"
}
```

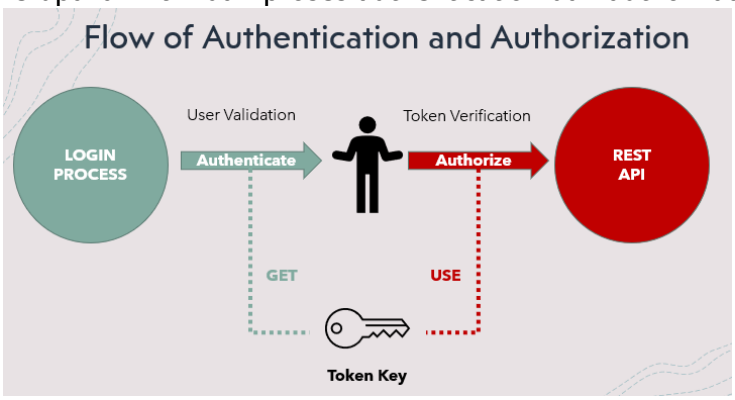
C. Pembuatan Authentication dan Middleware Authorization

Authentication adalah proses mengenal user yang akan masuk pada sebuah sistem. Proses pengenalan user menggunakan pencocokan data seperti *username* dan *password* yang tersimpan pada database. Ketika data tersebut cocok, maka akan dilakukan proses *generate* kode token. Kode token merupakan sebuah *credential* yang dibawa saat akan mengakses API agar mendapatkan izin untuk mengakses data atau melakukan proses yang diharapkan. Proses pemberian izin akses user terhadap API disebut dengan ***Authorization***. Pada proses tersebut dilakukan validasi token yang dibawa. Berikut ini merupakan analogi dari proses authentication dan authorization.



Analogi Proses *Authentication* dan *Authorization*

Berikut ini merupakan *flow* dari proses authentication dan authorization.



Flow Proses *Authentication* dan *Authorization*

Pada proses authentication terdapat proses generate token dan pada proses authorization terdapat proses validasi token. Untuk melakukan proses generate dan validasi token kita akan menggunakan Library JSONWEBTOKEN (jwt).

Untuk membuat token menggunakan JWT diperlukan sebuah *payload* dan *signature*. Payload merupakan data yang akan digenerate menjadi sebuah token. *Signature* merupakan data kunci untuk mengenerate ke bentuk token atau mengembalikan token ke bentuk payload semula. Berikut ini langkah-langkah pembuatan proses authentication dan authorization.

Langkah pembuatan proses Authentication

1. Install library jsonwebtoken dengan perintah **npm i -s jsonwebtoken**.
2. Pada folder controllers, buat file baru dengan nama auth.controller.js!
3. Buat code untuk authentication seperti berikut!

```
/** load express library */
const express = require(`express`)

/** load md5 library */
const md5 = require(`md5`)

/** load library jsonwebtoken */
```

```
const jwt = require(`jsonwebtoken`)

/** load model of admin */
const adminModel = require(`../models/index`).admin

/** create function to handle authenticating process */
const authenticate = async (request, response) => {
    let dataLogin = {
        username: request.body.username,
        password: md5(request.body.password)
    }

    /** check data username and password on admin's table */
    let dataAdmin = await adminModel.findOne({ where: dataLogin
})

    /** if data admin exists */
    if(dataAdmin){
        /** set payload for generate token.
         * payload is must be string.
         * dataAdmin is object, so we must convert to string.
         */
        let payload = JSON.stringify(dataAdmin)

        /** define secret key as signature */
        let secret = `mokleters`

       /** generate token */
        let token = jwt.sign(payload, secret)

        /** define response */
        return response.json({
            success: true,
            logged: true,
            message: `Authentication Successed`,
            token: token,
            data: dataAdmin
        })
    }

    /** if data admin is not exists */
    return response.json({
        success: false,
        logged: false,
        message: `Authentication Failed. Invalid username or
password`
    })
```

```
}

/** export function to another file */
module.exports = { authenticate }
```

4. Pada folder routes, buat file baru dengan nama "auth.route.js"!
5. Tuliskan code berikut untuk membuat routing pada proses authentication!

```
/** load library express */
const express = require(`express`)

/** initiate object that instance of express */
const app = express()

/** allow to read 'request' with json type */
app.use(express.json())

/** load function authentcation from auth's controller */
const {authenticate} = require(`../controllers/auth.controller`)

/** create route for authentication */
app.post(`/`, authenticate)

/** export app in order to load in another file */
module.exports = app
```

6. Pada file server.js, buat code untuk memuat routing proses authentication!

```
const auth = require(`./routes/auth.route`)
```

7. Kemudian buat code untuk menentukan prefix dari routing authentication!

```
app.use(`/auth`, auth)
```

8. Simpan dan jalankan server backend!
9. Buka aplikasi Postman, buat folder baru pada collection sebelumnya dengan nama authentication. Lalu buat request baru dengan memilih method POST dan masukkan Request URL dengan **http://localhost:8000/auth** dan masukkan request data pada body dengan tipe JSON seperti berikut ini!

```
{
    "username":"test",
    "password":"test"
}
```

Klik tombol Send dan simpan request yang diquat! Hasil output yang muncul seperti berikut!


```
{
    "success": false,
    "logged": false,
    "message": "Authentication Failed. Invalid username or password"
}
```


Untuk mendapatkan hasil authentication berhasil, maka isikan username dan password yang benar sesuai dengan yang tersimpan di table admin pada database. Berikut ini contoh output jika proses authentication berhasil.


```
{
    "success": true,
    "logged": true,
    "message": "Authentication Successed",
    "token": "eyJhbGciOiJIUzI1NiJ9.eyJpZCI6MSwibmFtZSI6IkFkbWluIDEiLCJjb250Y
WN0IjoiMDk4OTg4MjE4IiwiYWRkcmVzcyI6IkphcGFuIiwidXNlcm5hbWUiOiJhZG1pbiIsInBhc
3N3b3JkIjoiMjAyY2I5NjJhYzU5MDc1Yjk2NGIwNzE1MmQyMzRiNzAiLCJjcmVhdGVkQXQiOiIyM
DIyLTA1LTIwVDAwOjAwOjAwLjAwMFoiLCJ1cGRhdGVkQXQiOiIyMDIyLTA1LTIwVDAwOjAwOjAwL
jAwMFoifQ.m09S6Cp2Ibust3cWYXVTksLDogZrGg3Rd9Vb5bkKE_0",
    "data": {
        "id": 1,
        "name": "Admin 1",
        "contact": "098988218",
        "address": "Japan",
        "username": "admin",
        "password": "202cb962ac59075b964b07152d234b70",
        "createdAt": "2022-05-20T00:00:00.000Z",
        "updatedAt": "2022-05-20T00:00:00.000Z"
    }
}
```


Langkah-langkah pembuatan proses Authorization

1. Buka file auth.controller.js pada folder controller!
2. Buat fungsi baru dengan nama "authorize" berdampingan dengan function "authenticate" seperti berikut ini!

```
/** create function authroize */
const authorize = (request, response, next) => {
    /** get "Authorization" value from request's header */
    let headers = request.headers.authorization

    /** when using Bearer Token for authorization,
     * we have to split `headers` to get token key.
     * valus of headers = `Bearers tokenKey`
     */
```

```
    let tokenKey = headers && headers.split(" ")[1]

    /** check nullable token */
    if (tokenKey == null) {
        return response.json({
            success: false,
            message: `Unauthorized User`
        })
    }

    /** define secret Key (equals with secret key in
authentication function) */
    let secret = `mokleters`

    /** verify token using jwt */
    jwt.verify(tokenKey, secret, (error, user) => {
        /** check if there is error */
        if (error) {
            return response.json({
                success: false,
                message: `Invalid token`
            })
        }
    })

    /** if there is no problem, go on to controller */
    next()
}
```

3. Ekspor function authorization agar dapat di load oleh file lain!

```
/** export function to another file */
module.exports = { authenticate, authorize }
```

4. Setelah membuat function authorization, kemudian kita akan menerapkan function tersebut pada router yang memerlukan pengamanan, dalam bagian ini kita akan membuat pengamanan pada route data admin.
5. Buka file admin.route.js pada folder routes! Buat code untuk memanggil function authorize yang ada di file auth.controller.js seperti berikut!

```
/** load authorization function from controllers */
const { authorize } = require(`../controllers/auth.controller`)
```

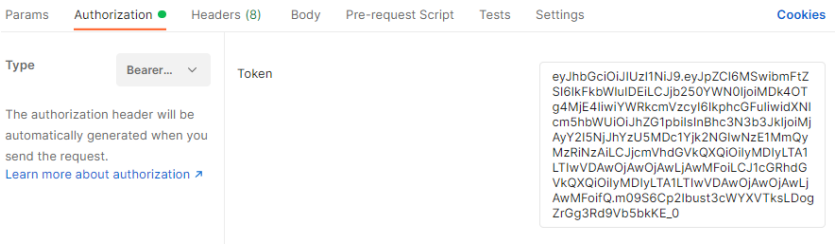
6. Sisipkan function authorize sebagai middleware pada setiap route yang telah dibuat!

```
/** create route to get data with method "GET" */
app.get("/", [authorize], adminController.getAdmins)

/** create route to add new admin using method "POST" */
app.post("/", [authorize], adminController.addAdmin)

/** create route to find admin
 * using method "POST" and path "find" */
app.post("/find", [authorize], adminController.findAdmin)

/** create route to update admin
 * using method "PUT" and define parameter for "id" */
app.put("/:id", [authorize], adminController.updateAdmin)

/** create route to delete admin
 * using method "DELETE" and define parameter for "id" */
app.delete("/:id", [authorize], adminController.deleteAdmin)
```

7. Simpan dan jalankan server backend!
8. Buka Aplikasi Postman dan akses request untuk menampilkan data admin! Jika dijalankan maka akan muncul response seperti berikut ini!

```
{
    "success": false,
    "message": "Unauthorized User"
}
```

   Hal tersebut terjadi karena kita tidak diijinkan untuk mengakses endpoint tersebut dan kita memerlukan token untuk bisa mengaksesnya.
9. Untuk mendapatkan token, silakan lakukan proses authentication dengan mengirimkan username dan password yang benar sesuai yang tersimpan pada table admin!
10. Copy kode token yang didapatkan pada proses authentication!
11. Masih pada Request untuk mendapatkan data admin, silakan tambahkan kode token pada tab **Authorization** dan pilih option **Bearer Token**. Paste kode token pada inputan yang disediakan!



12. Klik Send untuk mendapatkan response yang diinginkan yaitu mendapatkan data semua admin!

Tugas Praktikum!

1. Lengkapi proses validasi request pada setiap endpoint CRUD yang telah dibuat pada project "School Library"!
2. Lengkapi proses validasi token pada setiap endpoint yang telah dibuat pada project "School Library"! (kecuali endpoint untuk authentication).