

Middleware dan Request, Response, Next dan Error

Sio Jurnal Pipin, S.Kom.



Middleware

- Middleware merupakan fungsi yang mengaitkan ke dalam sebuah **routing process**, dan beberapa operasi di beberapa titik, tergantung pada apa yang ingin dilakukan.
- Biasanya digunakan untuk verifikasi permintaan atau respons, atau mengakhiri permintaan sebelum mencapai route tujuan / fungsi di **controller**.
- Verifikasi disini dapat berupa pengecekan cookies, token user, validasi url dst.
- Biasanya untuk kasus ini kita menggunakan
 - Morgan : <https://github.com/expressjs/morgan>
 - cookie-parser :
<http://expressjs.com/en/resources/middleware/cookie-parser.html>

Middleware | lanjutan

- Package Middleware tersebut Ini biasanya akan ditambahkan pada server yang digunakan dengan penulisan:

```
app.use((req, res, next) => { /* */ })
```

- Contoh:

```
M05-queryparams > . index.js > ...
1  var express = require("express");
2  var app = express();
3  var bodyParser = require("body-parser");
4
5  //https://github.com/expressjs/morgan
6  var logger = require("morgan");
7
```

```
SI0@DESKTOP-U2DMK91 MINGW64 ~/OneDrive - mikroskil.ac.id
$ node index.js
Server run
GET /api/budi/1222 200 4.761 ms - 28
GET /favicon.ico 404 1.269 ms - 150
```

- Saat ada user yang mengakses router maka middleware morgan yang berfungsi sebagai logger akan menampilkan informasi path url yang diakses oleh user beserta status code, dan informasi lainnya.

Middleware | lanjutan

- Selain itu, middleware dapat berupa fungsi yang sering digunakan untuk menjadi perantara, sekaligus verifikasi aktivitas sebelum menuju router.
- Middleware function dapat digunakan untuk router secara spesifik. Misal validasi role admin hanya pada router yang biasanya diakses oleh admin saja. Format:

```
const myMiddleware = (req, res, next) => {  
  /* ... */  
  next()  
}  
  
app.get('/', myMiddleware, (req, res) => res.send('Hello World!'))
```

Contoh: sebuah fungsi middleware untuk cek jwt token. Jika valid, maka fungsi **next()** akan mengizinkan lanjut pada router tujuan.

```
3 module.exports = (req, res, next) => {  
4   try {  
5     const token = req.headers.authorization.split(" ")[1];  
6     const decoded = jwt.verify(token, process.env.JWT_KEY);  
7     req.userData = decoded;  
8     next();  
9   } catch (error) {  
10    return res.status(401).json({ message: 'Auth failed' });  
11  }  
12 }
```

Req dan Res dalam Komunikasi Server

- ExpressJs memungkinkan untuk menangani Request dan Reponse secara manual.
- Proses Komunikasi server dengan ExpressJs yaitu:
 1. Memanggil *web server* yang telah disediakan NodeJS.
 2. Membaca URL yang ingin diakses pengguna, dan memanggil fungsi untuk memproses URL tersebut.
 3. Memproses HTTP Request dari *client*.
 4. Menghasilkan HTTP Response untuk *client*.
- **request**, yang merupakan instan dari ***http.IncomingMessage***, mewakili HTTP Request yang dikirimkan pengguna. Sebuah koneksi bisa saja menghasilkan beberapa HTTP Request. Kita akan membahas bagian ini lebih jauh nantinya.
- **response**, instan dari ***http.ServerResponse***, mewakili HTTP Response yang akan diberikan kepada pengguna.

Req, Res dan Next ()dalam Komunikasi Server | Lanjutan

```
app.use((req, res, next) => { /* */ })
```

- **Request** merupakan object HTTP request yang dikirim dari client ke server
- **Result** merupakan hasil result yang dikirim dari server ke client, ini adalah object yang kita gunakan untuk mengirim data ke client baik berupa data json, text biasa maupun html
- **Next** merupakan callback untuk melanjutkan proses life-cycle ke middleware berikutnya
- **Callback next()** merupakan callback untuk meneruskan proses atau data ke middleware berikutnya secara berurutan. Jika kita passing data / response ke error parameter maka proses tersebut akan terhenti di middleware tersebut dan mengirim data ke error handler yang dipasang di file utama, misal seperti app.js atau index.js.

Contoh Middleware

- Pada index.js buatlah sebuah middleware

```
12 //Middleware cek nim
13 const myMiddleware = (req, res, next) => {
14   if (req.params.nim === "123") {
15     console.log("Nim terverifikasi");
16     next();
17   } else {
18     const err = {
19       status: "error",
20       data: {
21         nim: req.params.nim,
22       },
23     };
24     next(err);
25   }
26 };
```

- Kemudian route dengan method get

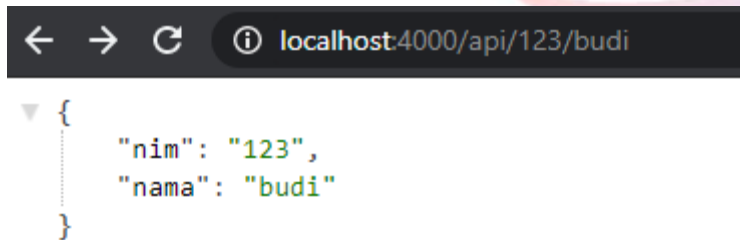
```
28 //route dengan method get
29 app.get("/api/:nim/:nama", myMiddleware, function (req, res) {
30   res.statusCode = 200;
31   //content-type pada expressjs
32   res.setHeader("Content-Type", "text/plain");
33   res.send(req.params);
34 });
```

- Dan fungsi untuk handel error sederhana

```
36 //ErrorHandling
37 app.use((error, req, res, next) => {
38   res.send(error);
39 });
```

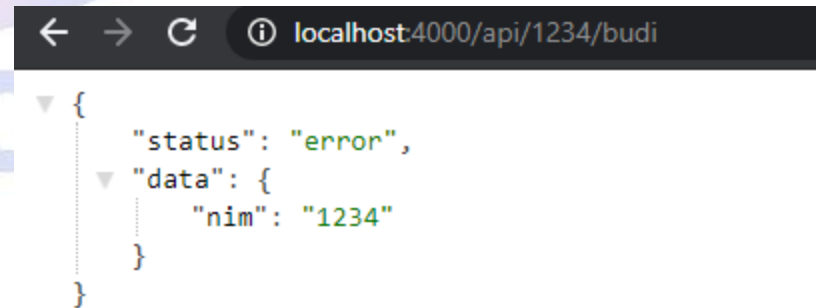
Contoh Middleware

- Fungsi middleware yang telah dibuat sebelumnya akan mengecek nim yang dikirim oleh user melalui url. Jika nim sama dengan verifikasi pada middleware maka akan lanjut pada fungsi tujuannya.
- Jika middleware mengecek itu tidak valid maka akan dilempar keluar (hentikan proses) ke fungsi error handling yang telah kita buat.



A screenshot of a web browser window with the address bar showing `localhost:4000/api/123/budi`. The response body is a JSON object: `{ "nim": "123", "nama": "budi" }`.

Request Valid



A screenshot of a web browser window with the address bar showing `localhost:4000/api/1234/budi`. The response body is a JSON object indicating an error: `{ "status": "error", "data": { "nim": "1234" } }`.

Tidak diproses ke fungsi tujuan