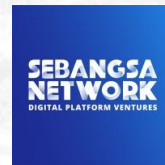


# A Fathy Ahmad F

Seorang *self-taught* Software Engineer  
berhabitat di Makassar, Sulawesi Selatan yang  
sekarang bekerja remote di salah satu  
perusahaan Singapore.

## Experience Background



**A. Fathy Ahmad F**

3 years experience as Software  
Engineer

# Express JS

## Middleware



# Middleware

- ☐ Konsep dan implementasi Middleware
- ☐ Handle Error Express JS
- ☐ Authentication dan Authorization API
- ☐ Logging pada Express JS

# Objektif sesi

- Siswa memahami konsep dan cara pengimplementasian middleware pada express
- Siswa mengetahui cara handle error pada Express
- Siswa mengetahui implementasi otentikasi dan otorisasi pada Express
- Siswa mengetahui cara Logging pada Express



# Middleware



Konsep dan implementasi Middleware



Handle Error Express JS



Authentication dan Authorization API

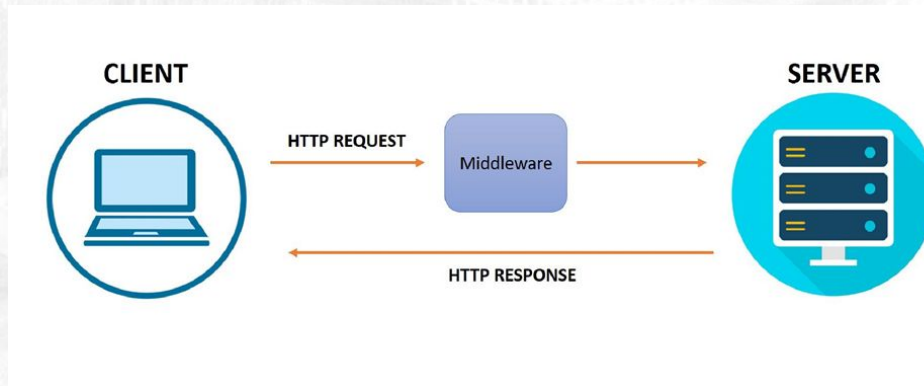


Logging pada Express JS

# Konsep dan Implementasi Middleware

Middleware pada dasarnya adalah ‘penengah’. Kalau dalam aplikasi middleware adalah sebuah aturan yang harus dilewati terlebih dahulu sebelum masuk ke dalam sebuah sistem atau keluar dari sebuah sistem.

Analoginya seperti ini, misalnya kita ingin mendaftar di sebuah pekerjaan yaitu menjadi seorang programmer, maka sebelum kita masuk menjadi karyawan, harus melewati prosedur terlebih dahulu. Misalnya, kita akan dicek apakah usianya sudah layak untuk bekerja. Semua tahapan-tahapan tersebut disebut dengan middleware.



Berikut contoh sederhana middleware pada Express js

```
var express = require('express');
var app = express();

//Middleware function to log request protocol
app.use('/things', function(req, res, next){
  console.log("A request for things received at " + Date.now());
  next();
});

// Route handler that sends the response
app.get('/things', function(req, res){
  res.send('Things');
});

app.listen(3000);
```

Fungsi diatas akan menjalankan console.log setiap kita mencoba request ke endpoint /things. Middleware menggunakan function **next()** untuk memberikan perintah melanjutkan ke proses berikutnya.

Contoh lainnya penggunaan middleware pada level router adalah sebagai berikut

```
const requireJsonContent = (request, response, next) => {
  if (request.headers['content-type'] !== 'application/json') {
    response.status(400).send('Server requires application/json')
  } else {
    next()
  }
}
```

Kode diatas merupakan function middleware yang akan memberikan response error 400 ketika request body type yang dikirimkan bukan application/json

Selanjutnya kita bisa menggunakan kode diatas pada function router yang kita miliki

```
app.post('/products', requireJsonContent, (request, response) => {
  ...
  ...
  response.json(...)
})
```



# Middleware



Konsep dan implementasi Middleware



Handle Error Express JS



Authentication dan Authorization API

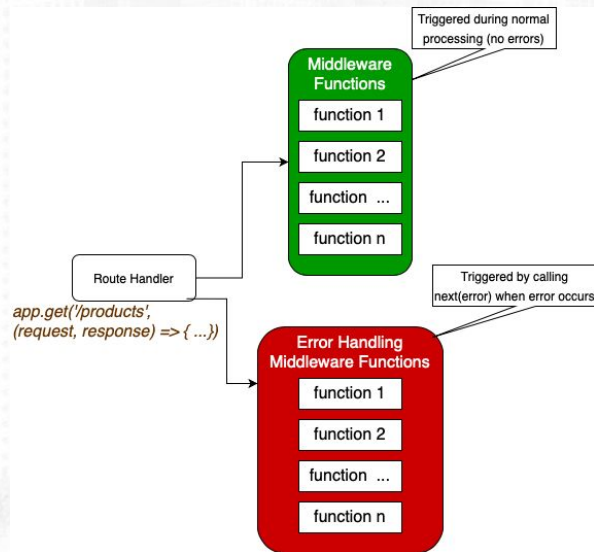


Logging pada Express JS

# Error Handling

Error handling pada express dilakukan dengan menggunakan middleware. Pada function middleware untuk error handling dibutuhkan 4 arguments yaitu **err, req, res, next**. Berikut contoh sederhana error handling untuk menampilkan response error setiap request.

```
app.use(function(err, req, res, next) {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
```



Kita juga bisa menggunakan Error handling pada router. Contohnya seperti berikut

```
const errorHandler = (error, request, response, next) {
  // Error handling middleware functionality
  console.log( `error ${error.message}` ) // log the error
  const status = error.status || 400
  // send back an easily understandable error message to the caller
  response.status(status).send(error.message)
}

app.get('/products', async (request, response) => {
  try {
    const apiResponse = await axios.get("http://localhost:3001/products")

    const jsonResponse = apiResponse.data

    response.send(jsonResponse)
  } catch(error) {
    next(error) // calling next error handling middleware
  }
})
```

# Middleware

☒ Konsep dan implementasi Middleware

☒ Handle Error Express JS



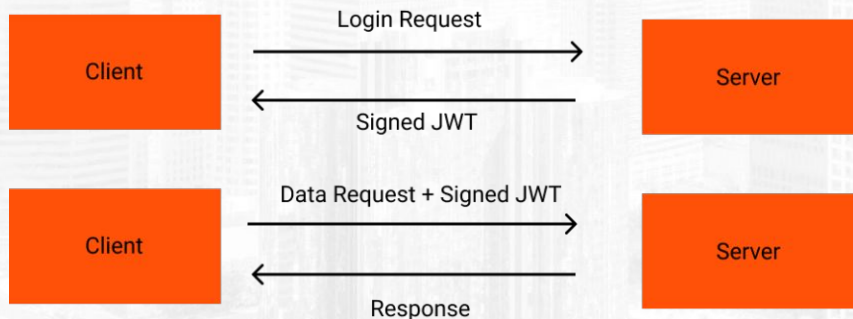
Authentication dan Authorization API



Logging pada Express JS



# Authentication dan Authorization



Secara sederhana, authentication adalah proses memverifikasi siapa pengguna (siapa user), dan authorization adalah proses memverifikasi apa yang dapat mereka akses (apa yang boleh user lakukan).



Untuk pengimplementasiannya pertama-tama kita perlu menginstal beberapa package dengan cara menjalankan perintah “npm install **jsonwebtoken**”. Selanjutnya kita bisa menggunakan JWT pada Express dengan cara seperti dibawah. Funtion ini akan men-encrypt data yang kita berikan menjadi token JWT.

```
You, 1 second ago | 1 author (You)
var express = require('express');    Hint: File is a CommonJS module; it
var jwt = require('jsonwebtoken');
var app = express();

app.get('/', (req, res) => {    Hint: 'req' is declared but its value is
  const token = jwt.sign(
    {
      userID: 23,
      role: 'admin',
    },
    'koderahasiayangsangatsangatrahasi'
  );
  res.json({
    token: token,
  });
});

app.listen(3000);
```

You, 1 second ago • Uncommitted changes

Ketika kita menjalankan aplikasi express dengan kode seperti sebelumnya maka API akan memberikan response seperti ini.

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySUQiOiJzLCJyb2x1IjoieYWRtaW4iLCJpYXQiOiJlE2NzcxMjA3NTh9.A0J1tZbwW1gXr62L3StC3zwHjIVv0bfskfe8ubJtJ8M"
}
```

Untuk mendapatkan informasi yang sudah kita encrypt menjadi token JWT (decrypt) kita bisa menggunakan function seperti ini

```
app.get('/verify/:token', (req, res) => {
  const data = jwt.verify(
    req.params.token,
    'koderahasiayangsangatsangatrahasi'
  );
  res.json({
    data: data,
  });
});
```



Ketika kita meminta request ke endpoint

**/verify/eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySUQiOiJzLCJyb2x1IjoieYWRtaW4iLCJpYXQiOiJlE2NzcxMjA3NTth9.A0J1tZbwW1gXr62L3StC3zwHjIVv0bfskfe8ubJtJ8M** pada postman maka API akan memberikan response data sebelumnya yang sudah kita ubah menjadi token seperti ini

```
{
  "data": {
    "userID": 23,
    "role": "admin",
    "iat": 1677120758
  }
}
```

Dengan konsep ini kita bisa mengimplementasikannya untuk memverifikasi user pada saat meminta request kepada server dengan membaca token yang mereka kirimkan.

Untuk membuat JWT token lebih aman kita bisa menentukan kapan token tersebut expired dengan cara menambahkan parameter tambahan pada function **jwt.sign** seperti berikut.

```
app.get('/', (req, res) => {    Hint: 'req' is declared but its value is new
  const token = jwt.sign(
    {
      userID: 23,
      role: 'admin',
    },
    'koderahasiayangsangatsangatrahasi',
    { expiresIn: '1h' }
  );
  res.json({
    token: token,
  });
});
```

You, 3 hours ago • modul 2 ...

Function diatas berarti token JWT hanya berlaku selama 1 jam saja.

# Middleware

☒ Konsep dan implementasi Middleware

☒ Handle Error Express JS

☒ Authentication dan Authorization API



Logging pada Express JS

# Logging pada Express

```

dkundel@dagobah: ~/dev/playground/express-demo
# dkundel at dagobah in ~/dev/playground/express-demo [16:29:28]
$ DEBUG=express:* node index.js 11.13.0
express:application set "x-powered-by" to true +0ms
express:application set "etag" to 'weak' +2ms
express:application set "etag fn" to [Function: generateETag] +1ms
express:application set "env" to 'development' +0ms
express:application set "query parser" to 'extended' +0ms
express:application set "query parser fn" to [Function: parseExtendedQueryString] +0ms
express:application set "subdomain offset" to 2 +1ms
express:application set "trust proxy" to false +0ms
express:application set "trust proxy fn" to [Function: trustNone] +0ms
express:application booting in development mode +0ms
express:application set "view" to [Function: View] +0ms
express:application set "views" to '/Users/dkundel/dev/playground/express-demo/views' +0ms
express:application set "jsonp callback name" to 'callback' +0ms
express:router use '/' query +1ms
express:router:layer new '/' +0ms
express:router use '/' expressInit +0ms
express:router:layer new '/' +0ms
express:router use '/' loggingMiddleware +1ms
express:router:layer new '/' +0ms
  
```

Log server adalah seluruh informasi dari order atau permintaan yang dikirimkan oleh klien kepada server dengan waktu yang tepat beserta hasil pemrosesannya.



Untuk kesempatan kali ini kita akan menggunakan package **morgan** untuk melakukan logging pada Express. Kenapa morgan? Morgan menyederhanakan tugas mencatat permintaan HTTP pada Express. Biasanya (tanpa morgan), developer perlu membuat semua logika logging secara manual. Kita perlu menginstruksikan Node.js/Express.js apa, bagaimana, dan di mana harus menyimpan logging. Morgan melakukan itu semua untuk developer. Morgan juga dilengkapi dengan beberapa preset bawaan yang telah ditentukan sebelumnya, menghemat waktu dan tenaga untuk mengatur sendiri semua pencatatan.

```
deployer@onboard:~$ tail /home/deployer/.pm2/logs/onboard-out-0.log
GET / 302 95.412 ms - 28
GET / 302 185.197 ms - 56
GET /login 200 105.077 ms - 2196
GET /css/bootstrap.min.css 304 5.015 ms - -
GET /css/styles.css 304 2.258 ms - -
GET /images/onboard.png 304 2.662 ms - -
GET /jspm_packages/system.js 304 4.111 ms - -
GET /config.js 304 4.685 ms - -
GET /jspm_packages/es6-module-loader.js 304 2.082 ms - -
GET /build.js 304 3.169 ms - -
deployer@onboard:~$
```

Pertama kita perlu menginstall morgan dengan cara

```
npm install morgan
```

Lalu kita bisa menuliskan kode berikut untuk melakukan logging sederhana pada express

```
const express = require('express');  
  
const morgan = require('morgan');  
  
const app = express();  
app.use(morgan('tiny'));  
  
app.listen(8080, () => {  
  console.log('Server listening on port :8080');  
});
```

Option 'tiny' pada morgan memberikan kita data logging yang simple dan minimalist

```
Server listening on port :8080
```

```
GET / 404 1.616 ms - 139
```

Ketika kita menggunakan opsi 'common' maka logging informasi akan memberikan data yang lebih rinci.

```
Server listening on port :8080
```

```
::ffff:127.0.0.1 - - [22/Aug/2020:17:07:31 +0000] "GET / HTTP/1.1" 404 139
```

# Middleware

- ✓ Konsep dan implementasi Middleware
- ✓ Handle Error Express JS
- ✓ Authentication dan Authorization API
- ✓ Logging pada Express JS



# Study Case

Berikut tersedia data customer rental dvd dengan status keaktifan yang berbeda. [\(data\)](#)

Kita ingin membuat API untuk menampilkan list film yang hanya bisa diakses oleh customer aktif saja. Kita juga perlu membuat API login untuk mensimulasikan proses login ( menggunakan email saja sebagai identifier ). Agar lebih mudah melakukan debugging dan pengontrolan request kita juga perlu mengimplementasikan logger.

# Cara penyelesaian

Langkah-langkah yang bisa kita lakukan adalah:

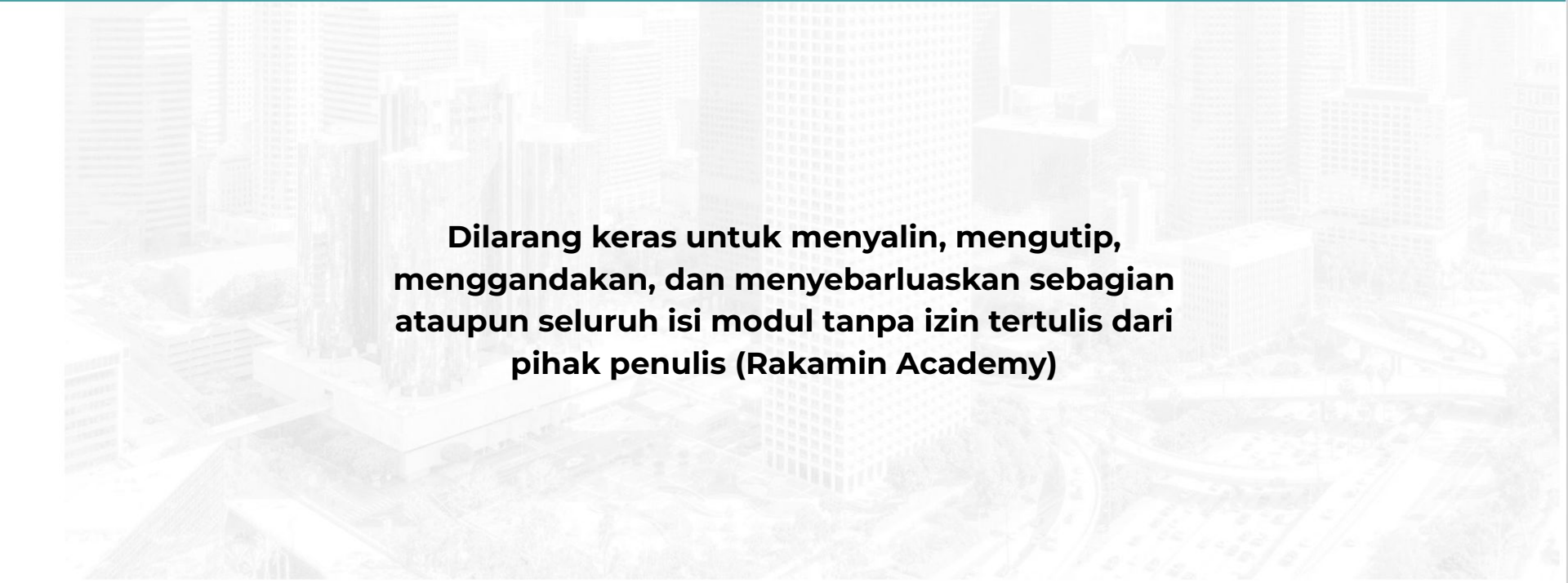
1. Import SQL data ke Postgress
2. Koneksikan Express dengan database
3. Membuat middleware authentication dan authorization untuk customer
4. Membuat API
  - A. Login API berupa POST request dengan email sebagai body. Api ini akan memberikan response berupa token JWT
  - B. Customer API berupa GET request dengan men-semicolon JWT token pada url sebagai identifier
5. Menginstall package **morgan** untuk aplikasi logger
6. Setting morgan pada **index.js**

# Reference material

- [Middleware Express](#)
- [Authentication dan Authorization Express](#)
- [Logging Express](#)

**Terima kasih!**

# Copyright Rakamin Academy

A faded, light-colored background image of a city skyline with various skyscrapers and buildings.

**Dilarang keras untuk menyalin, mengutip,  
menggandakan, dan menyebarkan sebagian  
ataupun seluruh isi modul tanpa izin tertulis dari  
pihak penulis (Rakamin Academy)**