# Express JS

Back-end Web Development

# Mengenal Express JS

# Mengenal Express JS

- ″Fast, unopinionated, minimalist web framework for Node.js″ demikian tagline yang diusung oleh framework ini pada situs resminya https://expressjs.com/.
- ExpressJS merupakan framework web berbasis NodeJS yang memiliki berbagai fitur–fitur dasar untuk membangun aplikasi web, baik itu fullstack maupun web service. Fitur–fitur tersebut diantaranya: routing, middleware, dan template engine.
- Di samping popularitasnya yang tinggi sehingga banyak resource yang mendukungnya. Kelebihan lain dari framework ini adalah simple, minimalist dan to the point, artinya kita bisa membuat sesuatu dengan kode yang lebih singkat. Hal ini tentu akan memudahkan kita mempelajarinya. Framework ini cukup solid sehingga menjadi dasar dari banyak framework berbasis NodeJS lainnya.

# Instalasi & Konfigurasi

Kita bisa instalasi ExpressJS melalui perintah:

```
npm install express
```

Kemudian kita membuat file **index.js** yang berisi kode untuk menjalankan web server

JS index.js > ...

```
1  const express = require('express')
2  const app = express()
3  const port = 3000
4  app.listen(port, ()=>console.log(`Server running at http://localhost:${port}`))
```

# Routing Dasar

# Routing Dasar

Pada ExpressJS, penangan routing sangat sederhana, yaitu menggunakan format:

```
app.METHOD(ROUTE, CALLBACK)
```

Dimana :

- METHOD merupakan HTTP method yaitu get, post, put, dsb;
- ROUTE merupakan definisi routenya dalam bentuk string;
- CALLBACK merupakan fungsi yang dijalankan ketikan routing terpenuhi.

# Routing Dasar(Cont.)

index.js > ...

```
1   const express = require('express')
2   const app = express()
3   const port = 3000
4
5   app.get('/', (req, res) => res.send('Hello World!')) // <= tambahkan ini
6   app.listen(port, ()=>console.log(`Server running at http://localhost:${port}`))
```

# Routing Dasar (Cont.)

- app merupakan objek atau instance dari class Express`const express = require('express')`
- get() merupakan method HTTP **`GET`**, sehingga untuk method lainnya kita bisa sesuaikan saja **`post()`**, **`delete()`**, **`put()`**, dll.
- **`'/'`** merupakan definisi routenya atau dalam contoh ini merupakan route utama. Untuk membuat routing lain misalnya **`about`** maka cukup dengan menuliskannya dengan **`'/about'`**, demikian juga untuk route bertingkat kita juga bisa tuliskan sebagai berikut:**`'/page/about'`**
- **`(req, res) => res.send('Hello World!'))`** merupakan callback berbentuk fungsi closure/anonymous yang akan dieksekusi ketika routing ditemukan. Fungsi ini setidaknya memiliki dua argumen yaitu req (request) dan res (response), dimana fungsi ini menjalankan perintah **`res.send()`** yang berarti mengirimkan suatu response ke client. Responsenya pada contoh ini berupa teks **`Hello World!`**

# Method-method argurmen **`res`**

| Method | Keterangan |
| --- | --- |
| res.send() | Mengirim berbagai jenis response, di mana argumennya bisa berisi konten serta mengakhiri proses response |
| res.write() | Menulis konten ke response |
| res.end() | Mengakhiri proses response di mana argumennya bisa berisi konten |

```
app.get('/', (req, res) => {
  res.write('Hello ')
  res.write('World!')
  res.end()
})
```

# HTTP Method yg lain

```
app.post('/contoh', (req, res) => {
  res.send('request dengan method POST')
})

app.put('/contoh', (req, res) => {
  res.send('request dengan method PUT')
})

app.delete('/contoh', (req, res) => {
  res.send('request dengan method DELETE')
})
```

# HTTP Method **`all`**

Dengan cara ini maka route **`/universal`** bisa diakses dengan method apapun baik GET/POST/PUT/DELETE.

```
app.all("/universal", function (req, res) {
  res.send("request dengan method " + req.method);
});
```

# Routing Dinamis

Routing pada ExpressJS juga bisa menyertakan parameter, misalnya pada kasus aplikasi blog, dimana setiap artikel pada blog tersebut mempunyai routing tersendiri yang berisi parameter id artikel.

Perhatikan contoh routing berikut:

- route `/post/1` mengacu pada artikel blog dengan id satu,
- route `/post/2` mengacu pada artikel blog dengan id dua,
- route `/post/n` mengacu pada artikel blog dengan id yang ke sekian.

Berdasarkan penjelasan sebelumnya tentu kita bisa membuat deklarasi routingnya sebagai berikut:

```
app.get('/post/1', (req, res) => res.send('artikel-1'))

app.get('/post/2', (req, res) => res.send('artikel-2'))

app.get('/post/n', (req, res) => res.send('artikel-n'))
```

# Routing Dinamis (Params)

- Jika jumlah artikel kita hanya tiga buah maka tentu deklarasi seperti ini oke-oke saja, namun bayangkan jika kita punya ratusan atau ribuan artikel!.
- Selain tidak efisien karena deklarasi routingnya panjang, dengan cara ini maka setiap ada penambahan artikel berarti harus mengedit atau menambahkan kode routingnya.
- ExpressJS menyediakan mekanisme yang mudah untuk menangani kasus ini yaitu dengan routing dinamis. Sehingga deklarasi routingnya menjadi sebagai berikut:


- Definisi route untuk parameter dinamis adalah dengan menambahkan tanda **`:`** di depan nama parameternya **`:id`**. Untuk membaca data parameter tersebut, kita bisa gunakan argumen **`req`** yaitu **`req.params.id`**.

# Routing Dinamis (Query String)

- Cara lain untuk membuat dinamis routing adalah dengan menggunakan query string pada URL. Perhatikan contoh query string berikut:

```
/foods

/foods?page=2

/foods?page=2&sort=title
```

- Yaitu dengan menambahkan tanda tanya **?** di depan nama parameter, dan untuk mendefinisikan nilai dari parameter dengan menggunakan tanda sama dengan **=** diikuti nilai dari parameternya.
- Jika parameternya lebih dari satu maka dipisahkan dengan tanda **&**.

# Routing Dinamis (Query String)

```
app.get("/foods", (req, res) => {
  console.log(req.query);
  res.end();
});
```

Kode di atas hanya mencetak log **`req.query`** pada console tanpa menampilkan response apapun.

Apabila kita akses dengan URL **`http://localhost:3000/foods?page=2&sort=title`**, maka pada log akan muncul sebagai berikut:

```
{ page: '2', sort: 'title' }
```

# Routing Dinamis (Query String)

```
app.get('/foods', (req, res) => {
    const page = req.query.page ? req.query.page : 1
    res.write('Foods page: '+page+'\n')
    if (req.query.sort) res.write('Sort by: '+req.query.sort)
    res.end()
  })
```

# Hot Reload

Pada saat development, terutama ketika kita mengubah kode routing maka untuk mengimplementasikannya harus dengan merestart aplikasi kita, bahkan terkadang kita lupa belum merestartnya sehingga kodingan kita yang sebenarnya sudah benar, tidak berjalan seperti yang kita harapkan, yap karena belum di restart saja.

Akan cukup membantu jika ada tools yang otomatis merestart aplikasi ketika ada perubahan. Tools itu bernama **nodemon**.

```
npm install -g nodemon
```

Untuk menggunakan pustaka ini maka jalankan aplikasi melalui nodemon. Jika sebelumnya untuk menjalankan aplikasi menggunakan perintah **`node .`** maka sekarang gunakan **`nodemon .`**.

# Middleware

# Middleware

- Kata middleware dari sisi istilah berarti perangkat yang berada ditengah tengah. Pada ExpressJS, middleware bertindak sebagai penengah antaraclient request dengan pemrosesan request. Middleware bekerja melakukan **`intercept`** pada sistem routing di ExpressJS.
- Secara umum, middleware digunakan untuk menjalankan fungsi–fungsi tertentu ketika routing tertentu diakses.
- Sebagai contoh untuk menjalankan fungsi log access, dimana fungsi ini akan menyimpan atau menampilkan log dari setiap routing yang diakses. Contoh lain, untuk menjalankan fungsi authentication pada routing yang terproteksi atau perlu login.

# Middleware

- Middleware berupa sebuah fungsi, dimana fungsi ini memiliki tiga argumen yaitu req (objek request), res (objek response) dan next (fungsi).
- Fungsi middleware dapat menjalankan beberapa tugas seperti:
  - menjalankan kode program tertentu
  - membuat perubahan terhadap objek request dan response, termasuk menghentikan request.
  - menjalankan fungsi next() untuk melanjutkan request.

# Membuat Middleware

Sebagaimana yang dijelaskan sebelumnya, middleware berupa sebuah fungsi, dimana fungsi ini memiliki tiga argumen yaitu req (objek request), res (objek response) dan next (fungsi).

Middleware dapat diimplementasikan pada ExpressJs, dengan menggunakan method **use** pada objek express.

```
app.use(fungsi middleware)
```

# Menangani Log dengan Middleware

Misalnya kita akan membuat middleware yang berfungsi menampilkan log pada tiap request terhadap route apapun.

Adapun data log yang ingin ditampilkan adalah:

- data waktu saat request dilakukan **`Date.now()`** (timestamp),
- ip address client **`req.ip`** dan
- route yang diakses **`req.originalUrl`**.

```
const log = (req, res, next) => {
  console.log(Date.now() + " " + req.ip + " " + req.originalUrl);
  next();
};

app.use(log);
```

Jadi, setiap ada request maka middleware akan mencetak lognya pada terminal

ExpressJS juga menyediakan middleware official untuk menangani log yaitu morgan.

# Menangani Routing 404 dengan Middleware

- Contoh lain, kita bisa membuat middleware untuk menangani jika request dari client tidak menemukan routing yang cocok atau 404. Strategi ini cukup sederhana sebab sebenarnya tidak ada mekanisme yang dilakukan untuk mengecek ada atau tidaknya routing tersebut.
- Strateginya hanya mengikuti alur eksekusi kode routing yaitu dari atas ke bawah. Oleh karena itu supaya strategi ini berjalan dengan baik maka kode untuk middleware ini harus diletakkan setelah semua deklarasi routing.

```
// kode deklarasi routing

app.use((req, res, next) => {
  res.status(404).send("resource tidak ditemukan");
});
```

- Atau supaya lebih web service friendly maka kita bisa ubah menjadi format json.

```
app.use((req, res, next) => {
  res.json({
    status: "error",
    message: "resource tidak ditemukan",
  });
});
```

# Struktur file index.js dengan middleware

JS index.js > ...

```
1   /* file: index.js */
2
3   const express = require("express");
4   const app = express();
5   const port = 3000;
6
7   // middleware log
8   const log = (req, res, next) => {
9     console.log(Date.now() + " " + req.ip + " " + req.originalUrl);
10    next();
11  };
12  app.use(log);
13
14  // deklarasi routing
15
16  // middleware menangani 404
17  const notFound = (req, res, next) => {
18    res.json({
19      status: "error",
20      message: "resource tidak ditemukan",
21    });
22  };
23  app.use(notFound);
24
25  app.listen(port, () =>
26    console.log(`Server running at http://localhost:${port}`)
27  );
```

# Menangani Error dengan Middleware

Kita juga bisa menangani error dengan menggunakan middleware, namun argumennya sedikit berbeda, terdapat satu argumen tambahan yaitu **err** yang merepresentasikan objek error.

Implementasinya, middleware ini akan menangani semua jenis error yang mungkin timbul sehingga responnya tetap dapat kita kontrol.

ExpressJS juga menyediakan built-in middleware untuk menangani error: error handler

```
const errorHandling = (err, req, res, next) => {
  res.json({
    status: "error",
    message: "terjadi kesalahan pada server",
  });
};
app.use(errorHandling);
```

# Menangani Request Body dengan middleware

Request dari client terkadang juga menyertakan konten melalui body, terutama request dari form, atau terkadang request yang dikirimkan berformat JSON (application/json).

Untuk menangani request tersebut, kita bisa menggunakan middleware bawaan ExpressJS yaitu **`body-parser`**.

Middleware ini digunakan untuk memparsing object request tepatnya pada body baik yang diencode (application/x-www-form-url-encode) maupun yang berformat json dari request client sehingga menjadi mudah dibaca.

# Menangani Request Body dengan middleware

```
// kode lain

const bodyParser = require('body-parser') // <==

// kode lain

// penggunaan middleware body parser
// parse x-www-form-url-encode
app.use(bodyParser.urlencoded({ extended: true }))
// parse JSON
app.use(bodyParser.json())

// kode lain

app.post('/login', (req, res) => {
  const { username, password } = req.body
  res.send(`Anda login dengan username ${username} dan password ${password}`)
})

//kode lain
```

# Menangani Request Body dengan middleware

- Tanpa menggunakan middleware **`body-parser`** maka kita tidak bisa melakukannya dengan semudah ini **`const { username, password } = req.body`**.
- Pada contoh ini, kita menggunakan enctype: x-www-form-urlencoded untuk formnya. Enctype ini cocok untuk data sederhana atau JSON.
- Sejak ExpressJS versi 4.16+, fungsi middleware body-parser ini sudah built-in di dalam express, sehingga kita tidak perlu lagi menggunakan middleware tersebut.

```
// penggunaan middleware body parser
// parse x-www-form-url-encode
// app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.urlencoded({ extended: true }));
// parse JSON
// app.use(bodyParser.json());
app.use(express.json());
```

# Menangani Statis File dengan middleware

- Response dari ExpressJS tidak hanya berupa data berformat string, namun juga bisa berupa file statis yang ada di server. Untuk melayani file statis seperti gambar, file CSS, dan file JavaScript, kita dapat menggunakan built-in middleware express.static di Express
- Sebagai contoh kita punya sebuah direktori bernama **public** sebagai tempat dimana kita meletakkan file-file asset dari web kita. Maka yang perlu kita lakukan adalah mendefinisikan direktori tersebut sebagai direktori static.
- Dengan cara ini maka kita bisa mengakses file-file dalam folder public secara langsung. Misalnya jika di dalam folder tersebut terdapat file **contoh.pdf** maka kita bisa mengaksesnya melalui URL **http://localhost:3000/contoh.pdf**, maka file PDF akan terbuka atau terdownload jika tidak ada plugin PDF pada browser.

```
const express = require("express");
const app = express();
const port = 3000;
const path = require("path"); // <== tambahkan ini

// Kode Lain

app.use(express.static(path.join(__dirname, "public")));

// Kode Lain
```

# Menangani File Upload dengan middleware

- Sekarang kita akan belajar menangani file yang diupload oleh user. Bagaimana menangkap file tersebut dari form yang dikirimkan oleh user dan kemudian menyimpannya ke suatu direktori tertentu.
- Untuk memudahkan kita dalam menangani file upload maka kita bisa menggunakan middleware yang dibuat oleh ExpressJS yaitu **Multer**. **Multer** merupakan middleware untuk menangani file upload menggunakan multipart/form-data

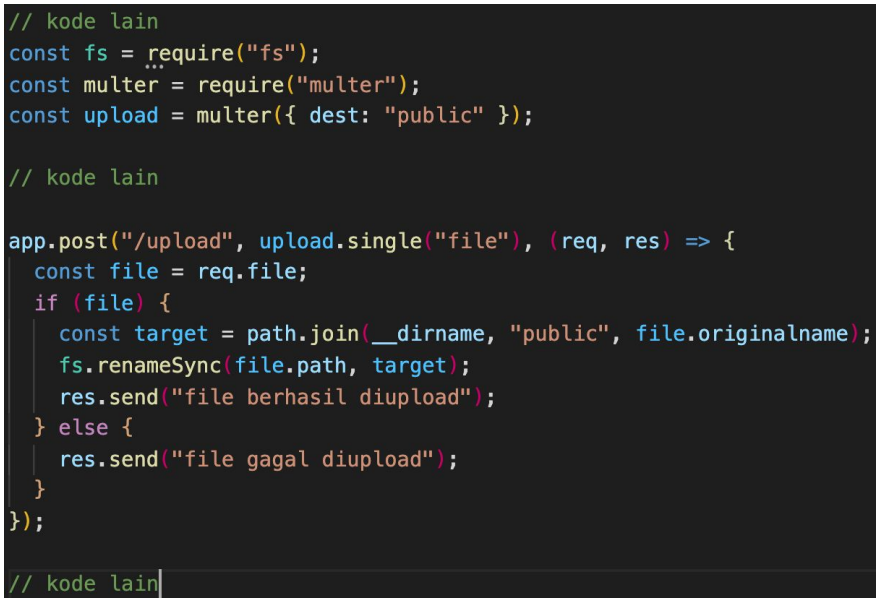
```
npm install multer
```

```
// kode lain
const multer = require("multer");
const upload = multer({ dest: "public" });

// kode lain
routers.post("/upload", upload.single("file"), (req, res) => {
  res.send(req.file);
});
// kode lain
```

# Menangani File Upload dengan middleware

- Maka jika kita akses menggunakan postman route POST upload, kemudian pada body form-data, kita tambahkan key **file** dan kita isi dengan file yang akan diupload
- Nama file yang diupload akan berbeda dengan nama aslinya dimana akan otomatis dihash
- Untuk mengubah nama filenya menjadi nama lain, maka kita bisa gunakan pustaka **`fs`** (bawaan NodeJS) melalui method **`renameSync`**. **`fs.renameSync(oldPath, newPath)`**



```
// kode lain
const fs = require("fs");
const multer = require("multer");
const upload = multer({ dest: "public" });

// kode lain

app.post("/upload", upload.single("file"), (req, res) => {
  const file = req.file;
  if (file) {
    const target = path.join(__dirname, "public", file.originalname);
    fs.renameSync(file.path, target);
    res.send("file berhasil diupload");
  } else {
    res.send("file gagal diupload");
  }
});

// kode lain
```

# Menangani CORS dengan middleware

- CORS singkatan dari cross origin resource sharing merupakan kondisi di mana suatu aplikasi web tertentu menggunakan atau mengakses resource (web service) dari lokasi lain yang berbeda origin (alamat domain, protokol (http or https) dan atau port-nya).
- Sebagai contoh CORS: aplikasi JS yang dihosting pada alamat domain https://abc.com dengan XMLHttpRequest mengakses web service dengan alamat https://xyz.com.
- Maka untuk alasan keamanan, browser secara default akan menolak akses tersebut. Itu artinya, akses resource secara default hanya diizinkan untuk mengakses ke origin yang sama.
- Ketika terjadi CORS, maka browser akan otomatis menambahkan HTTP header tertentu yang menginformasikan kepada server bahwa request tersebut masuk dalam kategori CORS.
- Untuk mengatasi CORS tersebut maka di sisi server perlu disetting agar request secara CORS diperbolehkan.
- ExpressJS mempunyai built-in middleware untuk menangani CORS yaitu bernama `cors`

# Menangani CORS dengan middleware

```
// kode lain

// require module middleware cors
const cors = require('cors')

// kode lain

// terapkan pada objek express
app.use(cors());

// kode lain
```