Rakamin Academy

Restful API

Rest API & Middleware

Raih Mimpi #TanpaBatas

Outline Pembelajaran

Rakamin Academy

Restful API

- [ ] Metode HTTP Request Express JS
- [ ] Konsep dan Implementasi Restfull API Express JS
- [ ] Konsep Pagination
- [ ] Dokumentasi API
- [ ] Pengenalan Swagger
- [ ] Implementasi Swagger

Objektif sesi

- Siswa memahami tentang metode HTTP request pada Express JS
- Siswa memahami konsep dan implementasi restfull API Express JS
- Siswa memahami konsep pagination pada Express
- Siswa memahami cara implementasi pagination
- Siswa memahami dokumentasi API
- Siswa memahami penggunaan Swagger untuk dokumentasi API

Outline Pembelajaran

Rakamin Academy

Restful API

- [ ] Metode HTTP Request Express JS
- [ ] Konsep dan Implementasi Restfull API Express JS
- [ ] Konsep Pagination
- [ ] Dokumentasi API
- [ ] Pengenalan Swagger
- [ ] Implementasi Swagger

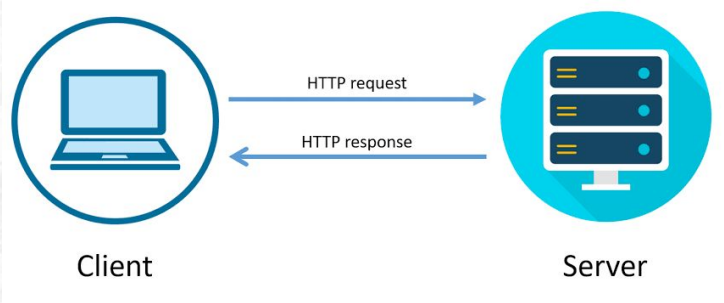
Metode HTTP Request Express JS

Hypertext Transfer Protocol (HTTP) dirancang untuk memungkinkan komunikasi antara klien dan server.

HTTP berfungsi sebagai protokol respons dari permintaan antara klien dan server.

Contoh: Klien (browser) mengirimkan permintaan HTTP ke server; kemudian server mengembalikan respons ke klien. Tanggapan berisi informasi status tentang permintaan dan mungkin juga berisi konten yang diminta.

HTTP method terdiri dari GET, PUT, POST, DELETE, PATCH, OPTION, HEAD.



Rakamin Academy

Contoh HTTP request pada express seperti berikut.

```
router.get('/', function(req, res){
   res.json(movies);
});
```

```
router.delete('/:id', function(req, res){
   var removeIndex = movies.map(function(movie){
      return movie.id;
   }).indexOf(req.params.id); //Gets us the index of movie with given id.

   if(removeIndex === -1){
      res.json({message: "Not found"});
   } else {
      movies.splice(removeIndex, 1);
      res.send({message: "Movie id " + req.params.id + " removed."});
   }
});
```

Outline Pembelajaran

Rakamin Academy

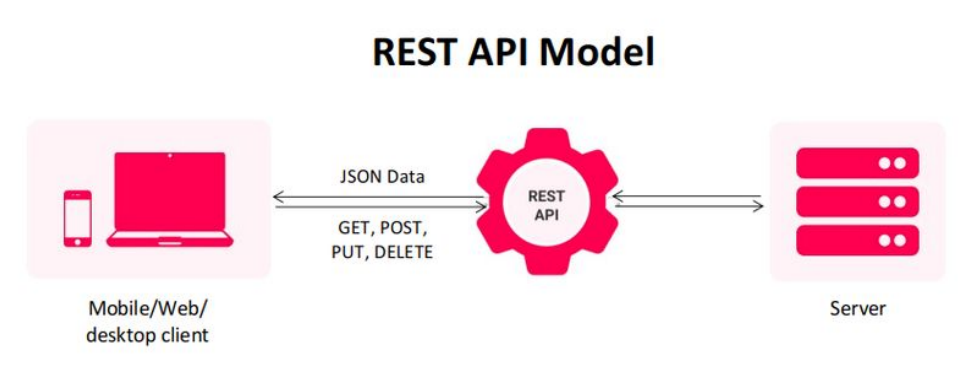
Restful API

- [x] Metode HTTP Request Express JS
- [ ] Konsep dan Implementasi Restfull API Express JS
- [ ] Konsep Pagination
- [ ] Dokumentasi API
- [ ] Pengenalan Swagger
- [ ] Implementasi Swagger

Konsep dan Implementasi Restfull API

API selalu diperlukan untuk membuat aplikasi mobile, website, menggunakan AJAX, dan menyediakan data ke klien. Gaya arsitektur populer tentang cara menyusun dan menamai API disebut REST (Representational Transfer State). HTTP 1.1 dirancang dengan mempertimbangkan prinsip REST. REST diperkenalkan oleh Roy Fielding pada tahun 2000 dalam Paper Fielding Disertations.

RESTful memberi kita hampir semua informasi yang kita perlukan untuk memproses request dari client.



Rakamin Academy

Pada kesempatan kali ini kita akan mencoba membuat aplikasi Express sederhana dengan mengimplementasikan restfull API. API yang akan kita buat berupa API untuk menampilkan dan mengolah list film. Berikut gambaran endpoint yang akan kita buat.

Method	URI	Details	Function
GET	/movies	Safe, cachable	Gets the list of all movies and their details
GET	/movies/1234	Safe, cachable	Gets the details of Movie id 1234
POST	/movies	N/A	Creates a new movie with the details provided. Response contains the URI for this newly created resource.
PUT	/movies/1234	Idempotent	Modifies movie id 1234(creates one if it doesn't already exist). Response contains the URI for this newly created resource.
DELETE	/movies/1234	Idempotent	Movie id 1234 should be deleted, if it exists. Response should contain the status of the request.
DELETE or PUT	/movies	Invalid	Should be invalid. **DELETE** and **PUT** should specify which resource they are working on.

Kita akan menggunakan JSON sebagai model transport data pada API ini karena cukup mudah diintegrasikan dengan javascript dan memiliki banyak kelebihan. Buatlah file **index.js** dan **movies.js** dan tuliskan kode berikut

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();

app.use(cookieParser());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

//Require the Router we defined in movies.js
var movies = require('./movies.js');

//Use the Router on the sub route /movies
app.use('/movies', movies);

app.listen(3000);
```

Selanjutnya kita bisa mulai membuat function router pada file **movies.js** seperti berikut.

```
var express = require('express');
var router = express.Router();
var movies = [
   {id: 101, name: "Fight Club", year: 1999, rating: 8.1},
   {id: 102, name: "Inception", year: 2010, rating: 8.7},
   {id: 103, name: "The Dark Knight", year: 2008, rating: 9},
   {id: 104, name: "12 Angry Men", year: 1957, rating: 8.9}
];

//Routes will go here
module.exports = router;
```

Untuk API kali ini kita akan menyimpan data di local server tanpa menggunakan database.

GET Routes

```
router.get('/', function(req, res){
   res.json(movies);
});
```

GET Routes (Spesifik ID)

```
router.get('/:id([0-9]{3,})', function(req, res){
   var currMovie = movies.filter(function(movie){
      if(movie.id == req.params.id){
         return true;
      }
   });
   if(currMovie.length == 1){
      res.json(currMovie[0])
   } else {
      res.status(404);//Set status to 404 as movie was not found
      res.json({message: "Not Found"});
   }
});
```

Rakamin Academy

POST Routes

```
router.post('/', function(req, res){
   //Check if all fields are provided and are valid:
   if(!req.body.name ||
      !req.body.year.toString().match(/^[0-9]{4}$/g) ||
      !req.body.rating.toString().match(/^[0-9]\.[0-9]$/g)){

      res.status(400);
      res.json({message: "Bad Request"});
   } else {
      var newId = movies[movies.length-1].id+1;
      movies.push({
         id: newId,
         name: req.body.name,
         year: req.body.year,
         rating: req.body.rating
      });
      res.json({message: "New movie created.", location: "/movies/" + newId});
   }
});
```

PUT Routes

```
router.put('/:id', function(req, res){
   //Check if all fields are provided and are valid:
   if(!req.body.name ||
      !req.body.year.toString().match(/^[0-9]{4}$/g) ||
      !req.body.rating.toString().match(/^[0-9]\.[0-9]$/g) ||
      !req.params.id.toString().match(/^[0-9]{3,}$/g)){

      res.status(400);
      res.json({message: "Bad Request"});
   } else {
      //Gets us the index of movie with given id.
      var updateIndex = movies.map(function(movie){
         return movie.id;
      }).indexOf(parseInt(req.params.id));

      if(updateIndex === -1){
         //Movie not found, create new
         movies.push({
            id: req.params.id,
            name: req.body.name,
            year: req.body.year,
            rating: req.body.rating
         });
         res.json({message: "New movie created.", location: "/movies/" + req.params.id});
      } else {
         //Update existing movie
         movies[updateIndex] = {
            id: req.params.id,
            name: req.body.name,
            year: req.body.year,
            rating: req.body.rating
         };
         res.json({message: "Movie id " + req.params.id + " updated.",
            location: "/movies/" + req.params.id});
      }
   }
});
```

Rakamin Academy

DELETE Routes

```
router.delete('/:id', function(req, res){
   var removeIndex = movies.map(function(movie){
      return movie.id;
   }).indexOf(req.params.id); //Gets us the index of movie with given id.

   if(removeIndex === -1){
      res.json({message: "Not found"});
   } else {
      movies.splice(removeIndex, 1);
      res.send({message: "Movie id " + req.params.id + " removed."});
   }
});
```

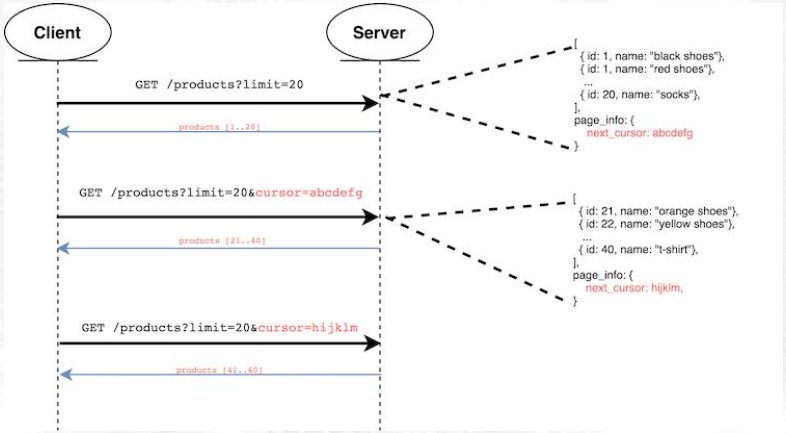
Outline Pembelajaran

Rakamin Academy

Restful API

- [x] Metode HTTP Request Express JS
- [x] Konsep dan Implementasi Restfull API Express JS
- [ ] Konsep Pagination
- [ ] Dokumentasi API
- [ ] Pengenalan Swagger
- [ ] Implementasi Swagger

Konsep dan Implementasi Pagination



Setelah membuat REStful API terkadang data yang kita peroleh dalam satu request sangatlah banyak. Konsep paginasi berperan penting dalam menyelesaikan masalah ini.

Paginasi adalah konsep menambahkan angka untuk mengidentifikasi urutan halaman. Di pagination, kita membatasi untuk mengurangi ukuran data yang diambil dari database. Dalam tutorial ini, kita akan mengimplementasikan pagination dalam istilah yang paling sederhana yaitu tanpa bantuan database.

Contohnya kita punya sebuah data yang cukup panjang seperti ini.

Kita bisa melakukan pagination pada Express dengan cara

```
get paginated results
app.get('/users/paginate', (req, res) => {      You, 1 second ago •
  const page = parseInt(req.query.page);
  const limit = parseInt(req.query.limit);

  // calculating the starting and ending index
  const startIndex = (page - 1) * limit;
  const endIndex = page * limit;

  const results = {};
  if (endIndex < model.length) {
    results.next = {
      page: page + 1,
      limit: limit,
    };
  }

  if (startIndex > 0) {
    results.previous = {
      page: page - 1,
      limit: limit,
    };
  }

  results.results = model.slice(startIndex, endIndex);

  res.json(results);
});
```

Untuk mengakses kita bisa melakukan hal berikut untuk request pagination

```
http://localhost:3000/users/paginate?page=1&limit=2
```

Jika kita lakukan test pada postman maka akan menghasilkan response seperti ini


```
{
    "next": {
        "page": 2,
        "limit": 2
    },
    "results": [
        {
            "id": 1,
            "full_name": "Kendre Abelevitz"
        },
        {
            "id": 2,
            "full_name": "Rona Walas"
        }
    ]
}
```

Outline Pembelajaran

Rakamin Academy

Restful API

- [x] Metode HTTP Request Express JS
- [x] Konsep dan Implementasi Restfull API Express JS
- [x] Konsep Pagination
- [ ] Dokumentasi API
- [ ] Pengenalan Swagger
- [ ] Implementasi Swagger

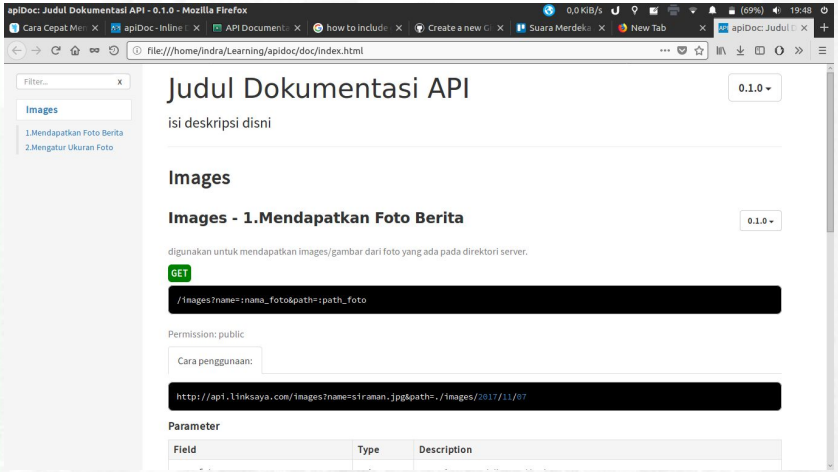
Dokumentasi API

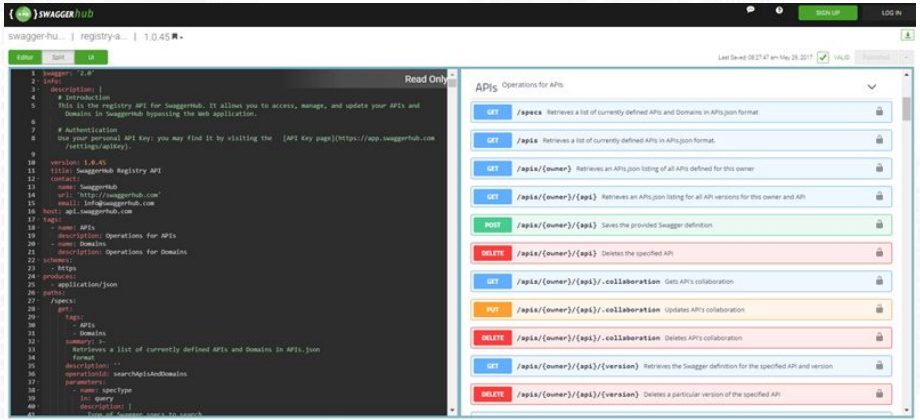
Restful API yang telah kita buat tentunya akan digunakan oleh developer lain (Frontend Developer). Oleh karena itu kita perlu memberitahu tentang cara penggunaan dan fitur-fitur apa saja yang tersedia pada API kita. Itulah mengapa pentingnya untuk membuat dokumentasi API agar developer lain mudah dalam menggunakan API yang telah kita buat.

Dokumentasi API, atau dokumen deskripsi API, adalah kumpulan referensi, tutorial, dan contoh yang membantu developer menggunakan API yang kita buat.

Dokumentasi API adalah sumber utama untuk menjelaskan apa saja yang bisa dilakukan dengan API dan cara memulainya. Selain itu juga berfungsi sebagai tempat bagi developer untuk melihat kembali tentang sintaks ataupun fungsionalitas API.

Berikut beberapa contoh dokumentasi API





Selain itu dokumentasi API juga bisa berupa file postman. Contohnya seperti yang ada pada link berikut. [Postman Example]

Outline Pembelajaran

Rakamin Academy

Restful API

- [x] Metode HTTP Request Express JS
- [x] Konsep dan Implementasi Restfull API Express JS
- [x] Konsep Pagination
- [x] Dokumentasi API
- [ ] Pengenalan Swagger
- [ ] Implementasi Swagger

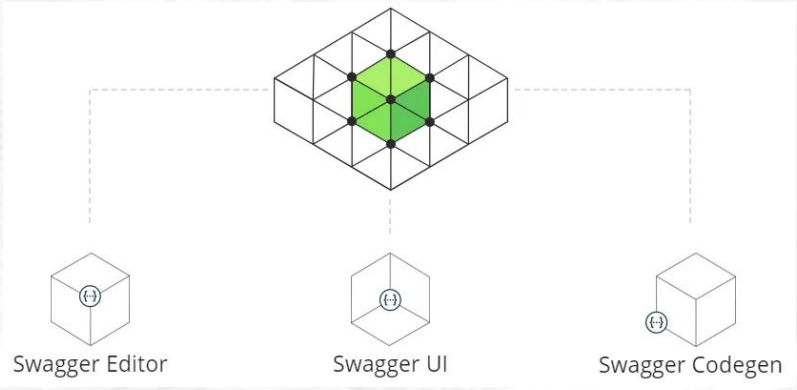
Pengenalan Swagger

Pada kesempatan kali ini kita akan membahas tentang Swagger. Setelah membangun API apakah kita pernah membuat dokumentasi API?

Bagaimana cara kita membuat dokumentasi tersebut agar mudah digunakan, diatur, atau bahkan dapat diotomatisasi?

Salah satunya kita dapat menggunakan Swagger. Swagger adalah tools Open API yang digunakan untuk mendokumentasikan API atau web service yang dibangun. Swagger dapat diakses melalui https://swagger.io/.

Swagger memiliki beberapa tool yang bisa kita manfaatkan. Adapun tool yang disediakan adalah sebagai berikut.



1. **Swagger Editor**
   Editor API untuk mendesain API dengan OpenAPI Specification dan dapat disimpan dalam format yaml atau json.
2. **Swagger UI**
   UI interaktif yang dapat digunakan di browser untuk memvisualisasikan definisi OpenAPI Specification.
3. **Swagger Codegen**
   Code generator yang dapat digunakan untuk membuat server stubs dan client SDKs secara langsung dari OpenAPI Specification yang telah kita buat.

Outline Pembelajaran

Rakamin Academy

Restful API

- [x] Metode HTTP Request Express JS
- [x] Konsep dan Implementasi Restfull API Express JS
- [x] Konsep Pagination
- [x] Dokumentasi API
- [x] Pengenalan Swagger
- [ ] Implementasi Swagger

Implementasi Swagger

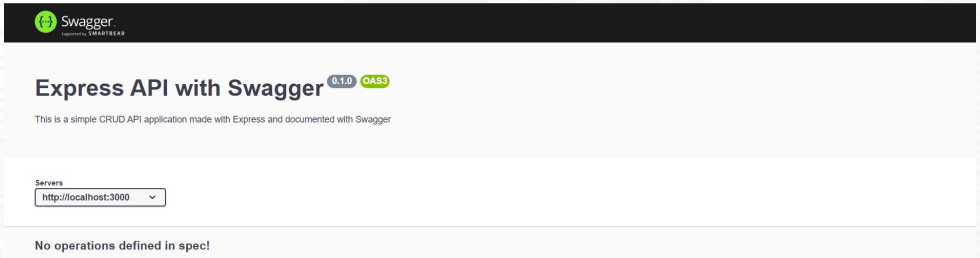
Untuk menggunakan swagger pada aplikasi express kita perlu menginstall 2 package yaitu **swagger-ui-express** dan **swagger-jsdoc** dengan cara seperti berikut.

```
npm i swagger-ui-express swagger-jsdoc
```

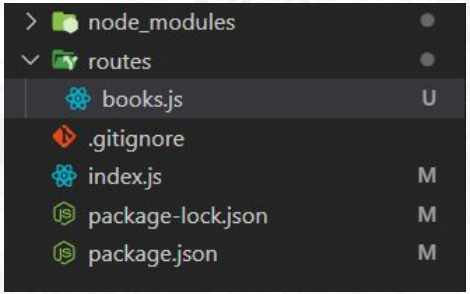
Selanjutnya kita bisa menggunakan swagger dengan cara menuliskan kode pada **index.js** seperti ini.

```
var express = require('express');      Hint: File is a CommonJS module; it may be converted to an ES
var swaggerJsdoc = require('swagger-jsdoc');
var swaggerUi = require('swagger-ui-express');

var app = express();

const options = {
  definition: {
    openapi: '3.0.0',
    info: {
      title: 'Express API with Swagger',
      version: '0.1.0',
      description:
        'This is a simple CRUD API application made with Express and documented with Swagger',
    },
    servers: [
      {
        url: 'http://localhost:3000',
      },
    ],
  },
  apis: ['./routes/*'],
};

const specs = swaggerJsdoc(options);
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(specs));


app.listen(3000);
```

Jika kita membuka http://localhost:3000/api-docs maka akan muncul tampilan seperti berikut.

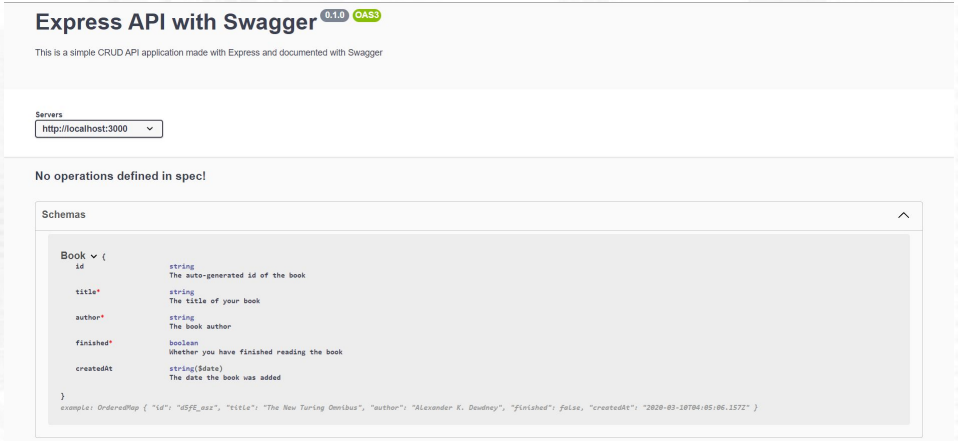


Pada contoh kali ini kita akan mengimplementasikan pada API sederhana yang menampilkan list dari beberapa buku. Dengan struktur folder seperti berikut.



```
/**
 * @swagger
 * components:
 *   schemas:
 *     Book:
 *       type: object
 *       required:
 *         - title
 *         - author
 *         - finished
 *       properties:
 *         id:
 *           type: string
 *           description: The auto-generated id of the book
 *         title:
 *           type: string
 *           description: The title of your book
 *         author:
 *           type: string
 *           description: The book author
 *         finished:
 *           type: boolean
 *           description: Whether you have finished reading the book
 *         createdAt:
 *           type: string
 *           format: date
 *           description: The date the book was added
 *       example:
 *         id: d5fE_asz
 *         title: The New Turing Omnibus
 *         author: Alexander K. Dewdney
 *         finished: false
 *         createdAt: 2020-03-10T04:05:06.157Z
 */
```

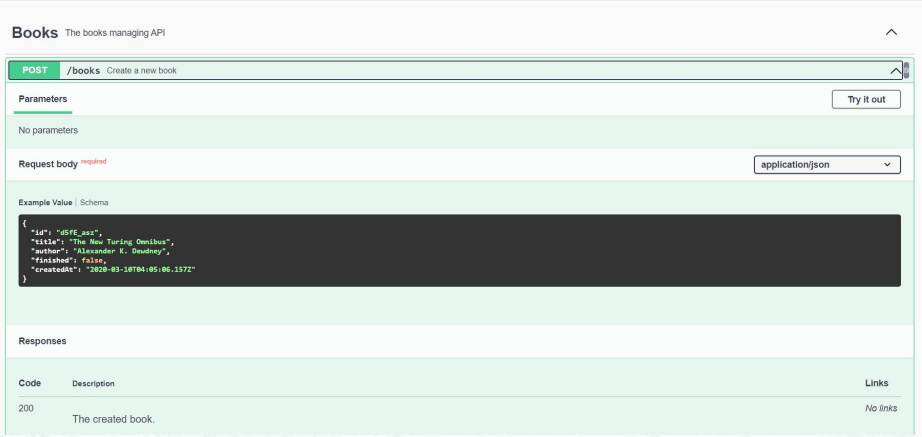
Langkah selanjutnya membuat API Model dengan menuliskan kode berikut tepat diatas kode router didalam file **books.js**



Maka tampilan dari swagger akan berubah seperti pada gambar disamping. Penggunaan JSDoc pada Express swagger memungkinkan kita untuk mengkonfigurasi dokumentasi dengan cara menuliskan comment pada source code secara langsung.

Setelah itu kita bisa menuliskan dokumentasi penggunaan APi yang kita buat dengan cara menuliskan comment tambahan tepat dibawah comment schema sebelumnya. Hal tersebut akan menambahkan dokumenatsi API yang sudah kita buat.

```
/**
 * @swagger
 * tags:
 *   name: Books
 *   description: The books managing API
 * /books:
 *   post:
 *     summary: Create a new book
 *     tags: [Books]
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             $ref: '#/components/schemas/Book'
 *     responses:
 *       200:
 *         description: The created book.
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/Book'
 *       500:
 *         description: Some server error
 *
 */
```



Study Case

Berikut tersedia data world (data). Data yang ada pada database tersebut merupakan list negara yang ada di seluruh dunia. Kita perlu untuk membuat RESTful API untuk mengolah data seperti menambahkan negara baru, mengedit, menghapus, ataupun menampilkan data. Namun untuk menampilkan data yang begitu banyak dapat mempengaruhi performa kecepatan pengambilan data, oleh karena itu kita perlu mengimplementasikan pagination pada GET request.

Setelah RESTful API selesai dokumentasi yang baik juga diperlukan agar orang lain dapat mengakses API dengan mudah..

Cara Penyelesaian

Langkah-langkah yang dapat dilakukan adalah sebagai berikut

1. Import data ke database dan koneksikan dengan Express.
2. Buat file router dengan function GET, POST, DELETE, dan PUT di dalamnya.
3. Modifikasi function GET dengan menambahkan query pada url untuk melimit data
4. Install Package dokumentasi API Swagger
5. Konfigurasi Swagger didalam router yang telah dibuat sebelumnya

Setelah itu kita bisa menggunakan Swagger secara langsung untuk melakukan pengetesan pada API kita.

Outline Pembelajaran

Rakamin Academy

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

Outline Pembelajaran

Rakamin Academy

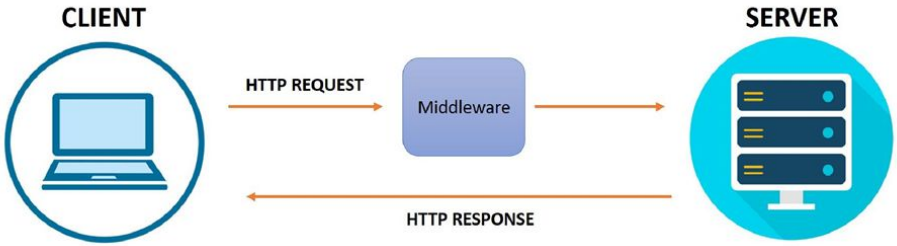
Middleware

- [ ] 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.



Rakamin Academy

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.

Rakamin Academy

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(...)
})
```

Outline Pembelajaran

Rakamin Academy

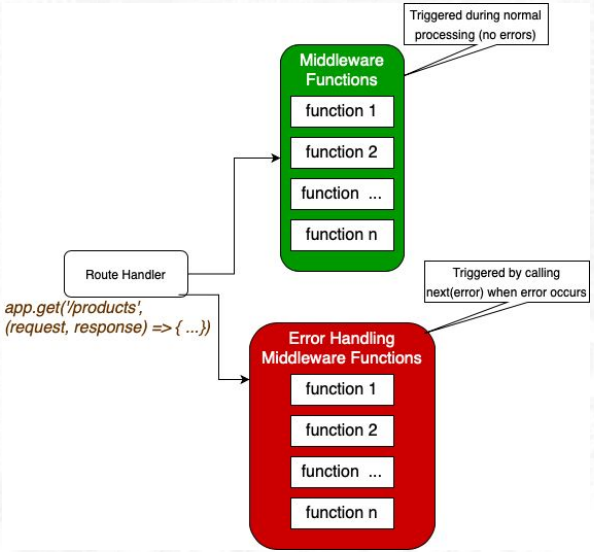
Middleware

- [x] 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
  }

})
```

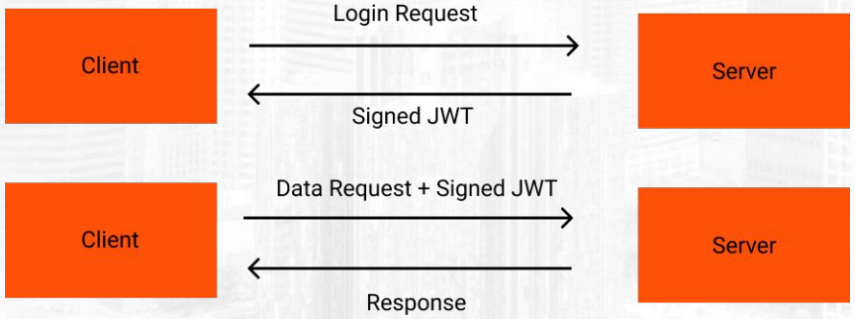
Outline Pembelajaran

Rakamin Academy

Middleware

- [x] Konsep dan implementasi Middleware
- [x] Handle Error Express JS
- [ ] Authentication dan Authorization API
- [ ] Logging pada Express JS

Authentication dan Authorization



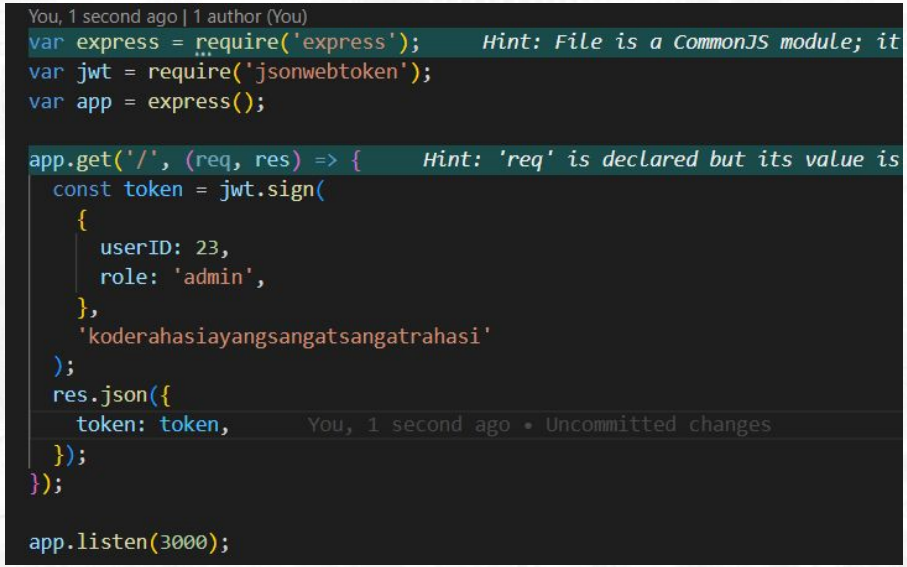
Secara sederhana, authentication adalah proses memverifikasi siapa pengguna (siapa user), dan authorization adalah proses memverifikasi apa yang dapat mereka akses (apa yang boleh user lakukan).

Pada kesempatan kali ini kita akan mengimplementasikan authentication dan authorization menggunakan JWT (JSON Web Token). JWT adalah cara yang baik untuk mentransmisikan informasi antar pihak dengan aman karena. Meskipun kita dapat menggunakan JWT dengan semua jenis metode komunikasi, saat ini JWT sangat populer untuk menangani authentication dan authorization melalui HTTP.


```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySUQiOjIzLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE2NzcxMjA3NTh9.
        A0J1tZbwW1gXr62L3StC3zwHjIVv0bfskfe8ubJtJ8M"
}
```

Rakamin Academy

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. Function ini akan men-encrypt data yang kita berikan menjadi token JWT.



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


```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySUQiOjIzLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE2NzcxMjA3NTh9.
        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'      You, now • Uncommitted changes
  );
  res.json({
    data: data,
  });
});
```

Ketika kita meminta request ke endpoint **/verify/eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySUQiOjIzLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE2NzcxMjA3NTh9.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 nev
  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.

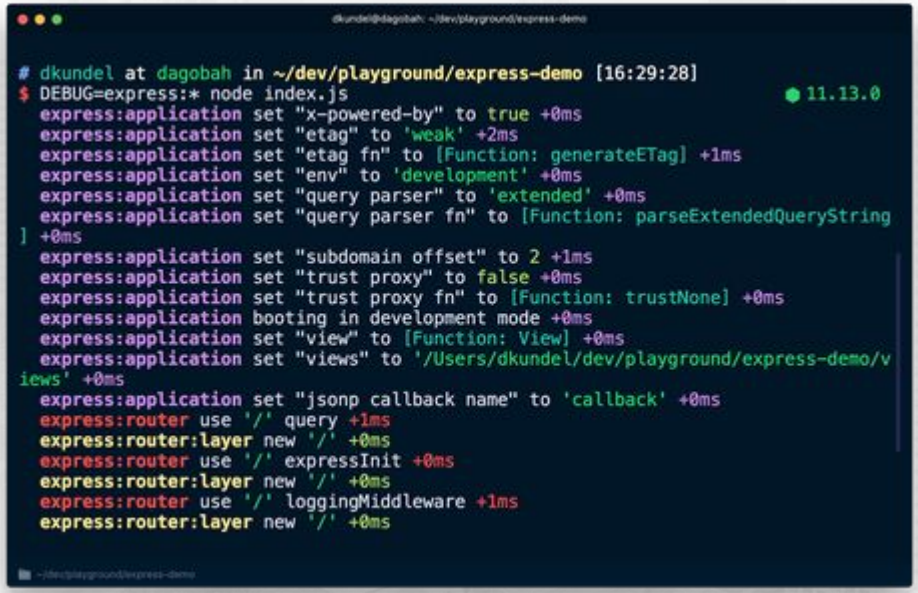
Outline Pembelajaran

Rakamin Academy

Middleware

- [x] Konsep dan implementasi Middleware
- [x] Handle Error Express JS
- [x] Authentication dan Authorization API
- [ ] Logging pada Express JS

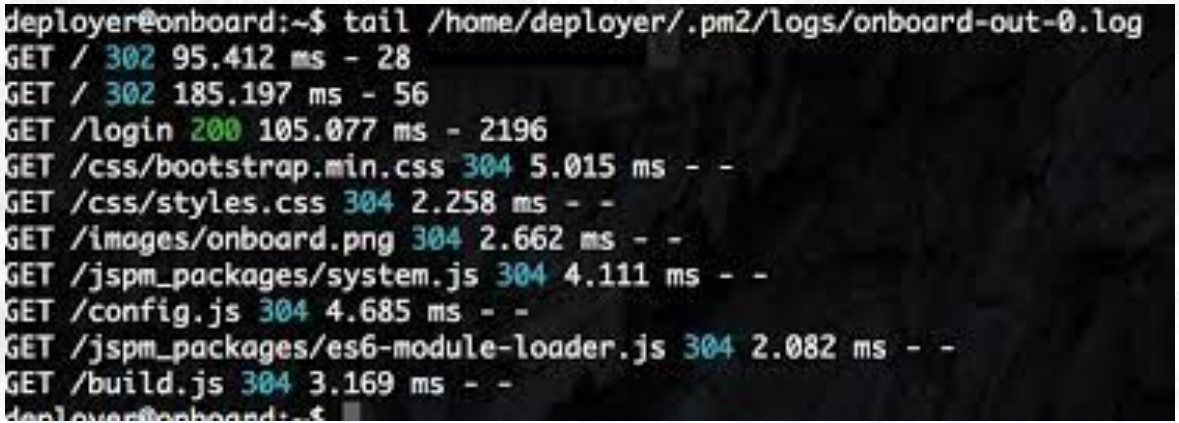
Logging pada Express



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

Rakamin Academy

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.



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
```

Outline Pembelajaran

Rakamin Academy

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-sematkan 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

Reference material

- Restful API Express
- Konsep Pagination dan Implementasinya
- Dokumentasi API
- Swagger


Rakamin
Academy
Thank you
Raih Mimpi #TanpaBatas