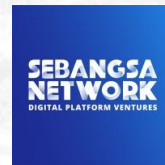


A Fathy Ahmad F

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

Restful API



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

Restful API



Metode HTTP Request Express JS



Konsep dan Implementasi Restfull API Express JS



Konsep Pagination



Dokumentasi API



Pengenalan Swagger



Implementasi Swagger

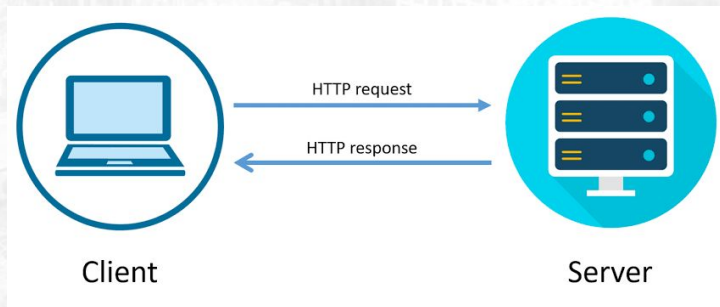
Metode HTTP Request Express JS

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.



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."});  
  }  
});
```

Restful API



Metode HTTP Request Express JS



Konsep dan Implementasi Restfull API Express JS



Konsep Pagination



Dokumentasi API



Pengenalan Swagger

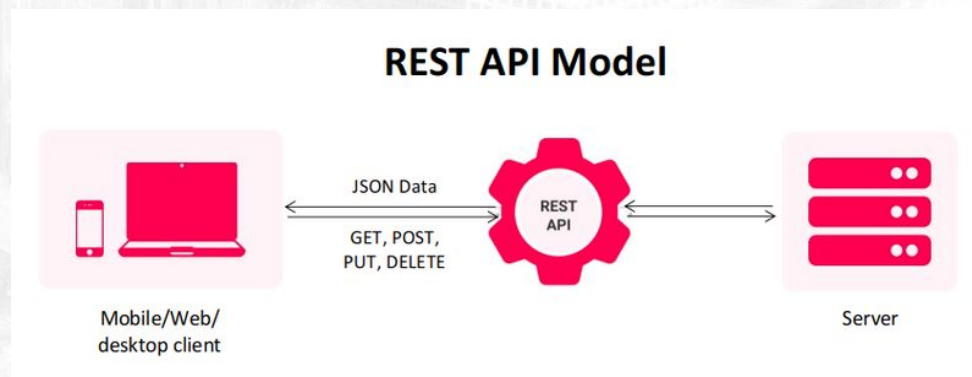


Implementasi Swagger

Konsep dan Implementasi Restful 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.



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"});  
  }  
});
```

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(/^[\d-]{4}$/g) ||
    !req.body.rating.toString().match(/^[\d-]\.[\d-]{1,2}$/g) ||
    !req.params.id.toString().match(/^[\d-]{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});
    }
  }
});
```

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."});
  }
});
```


Restful API

- ☒ Metode HTTP Request Express JS
- ☒ Konsep dan Implementasi Restfull API Express JS



Konsep Pagination



Dokumentasi API



Pengenalan Swagger

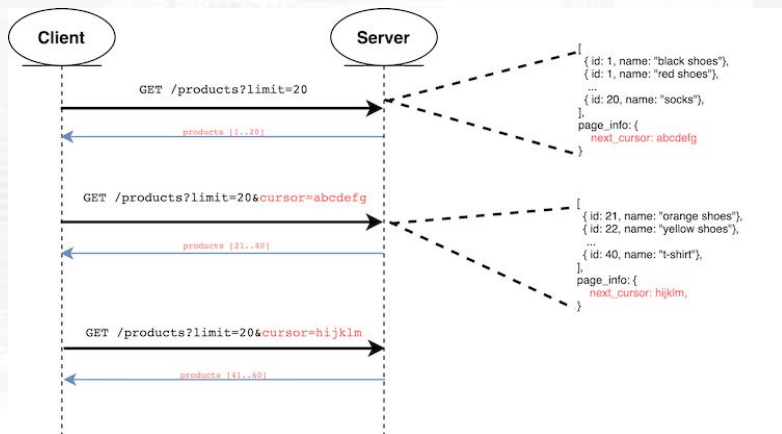


Implementasi Swagger

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"
    }
  ]
}
```


Restful API

- ☒ Metode HTTP Request Express JS
- ☒ Konsep dan Implementasi Restfull API Express JS
- ☒ 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.

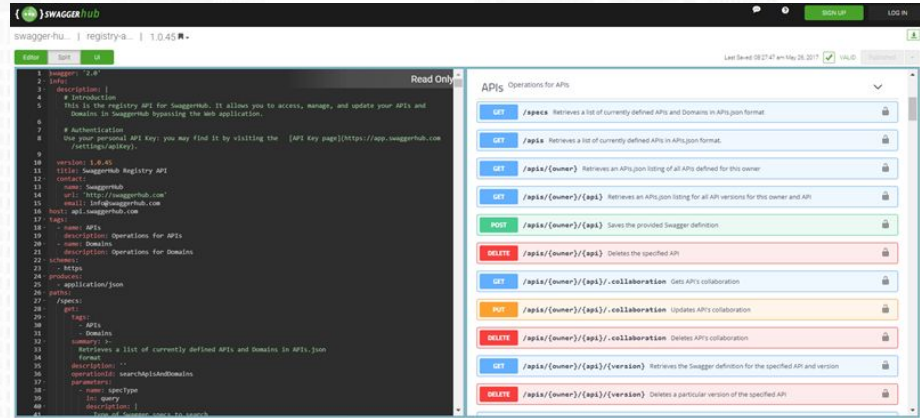
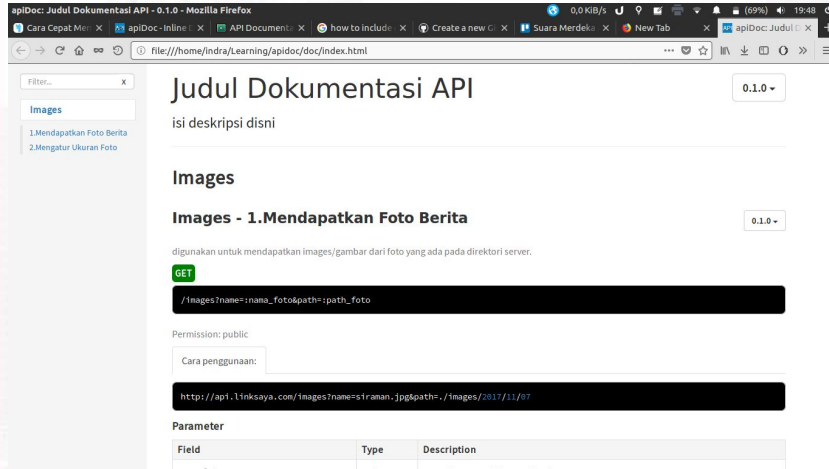


Dokumentasi API, atau dokumen deskripsi API, adalah kumpulan referensi, tutorial, dan contoh yang membantu developer menggunakan API yang kita buat.

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](#)

Restful API

- ☒ Metode HTTP Request Express JS
- ☒ Konsep dan Implementasi Restfull API Express JS
- ☒ Konsep Pagination
- ☒ 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**

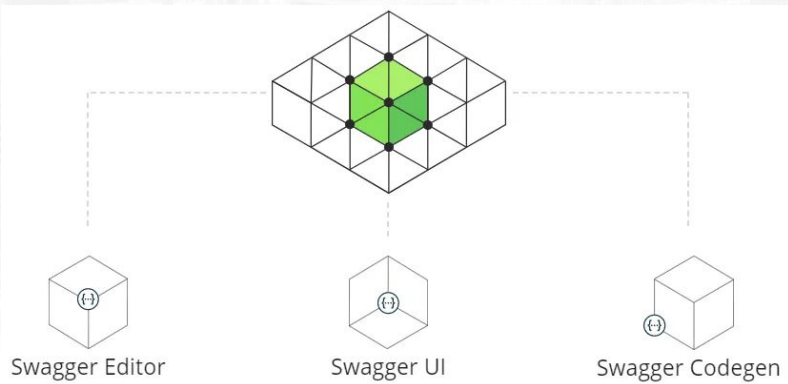
Editor API untuk mendesain API dengan OpenAPI Specification dan dapat disimpan dalam format yaml atau json.

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.



Restful API

- ☒ Metode HTTP Request Express JS
- ☒ Konsep dan Implementasi Restfull API Express JS
- ☒ Konsep Pagination
- ☒ Dokumentasi API
- ☒ 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');

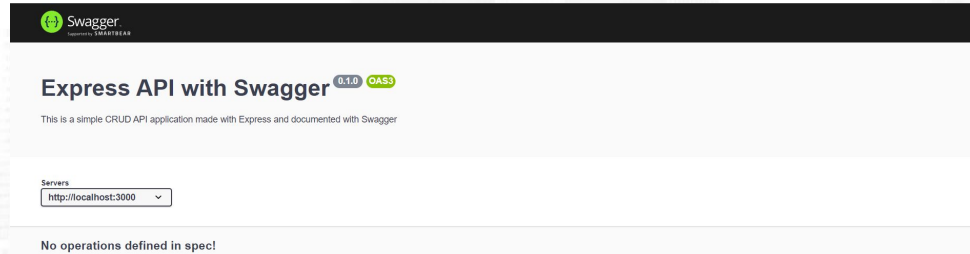
var app = express();

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/**'],
};

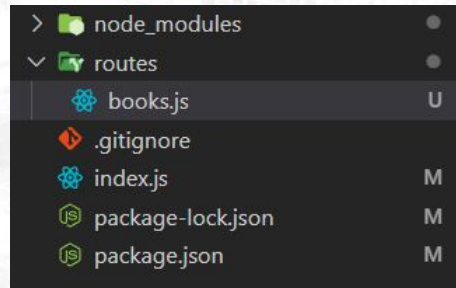
const specs = swaggerJsdoc(options);
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(specs));

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**

Express API with Swagger 0.1.0 OAS3

This is a simple CRUD API application made with Express and documented with Swagger

Servers

http://localhost:3000

No operations defined in spec!

Schemas

```
Book {
  id string
    The auto-generated id of the book
  title* string
    The title of your book
  author* string
    The book author
  finished* boolean
    Whether you have finished reading the book
  createdAt string(date)
    The date the book was added
}

example: OrderedMap { "id": "d5f5_oss", "title": "The Now Turing Omnibus", "author": "Alexander K. Dewdney", "finished": false, "createdAt": "2020-03-10T04:05:06.157Z" }
```

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
 */
```

Books The books managing API

POST /books Create a new book

Parameters

No parameters

Request body **required**

application/json

Example Value Schema

```
{
  "id": "d5ff-aaa",
  "title": "The New Turing Omnibus",
  "author": "Alexander K. Dewdney",
  "finished": false,
  "createdAt": "2020-03-18T04:05:06.157Z"
}
```

Responses

Code	Description	Links
200	The created book.	No links

Links

No links

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

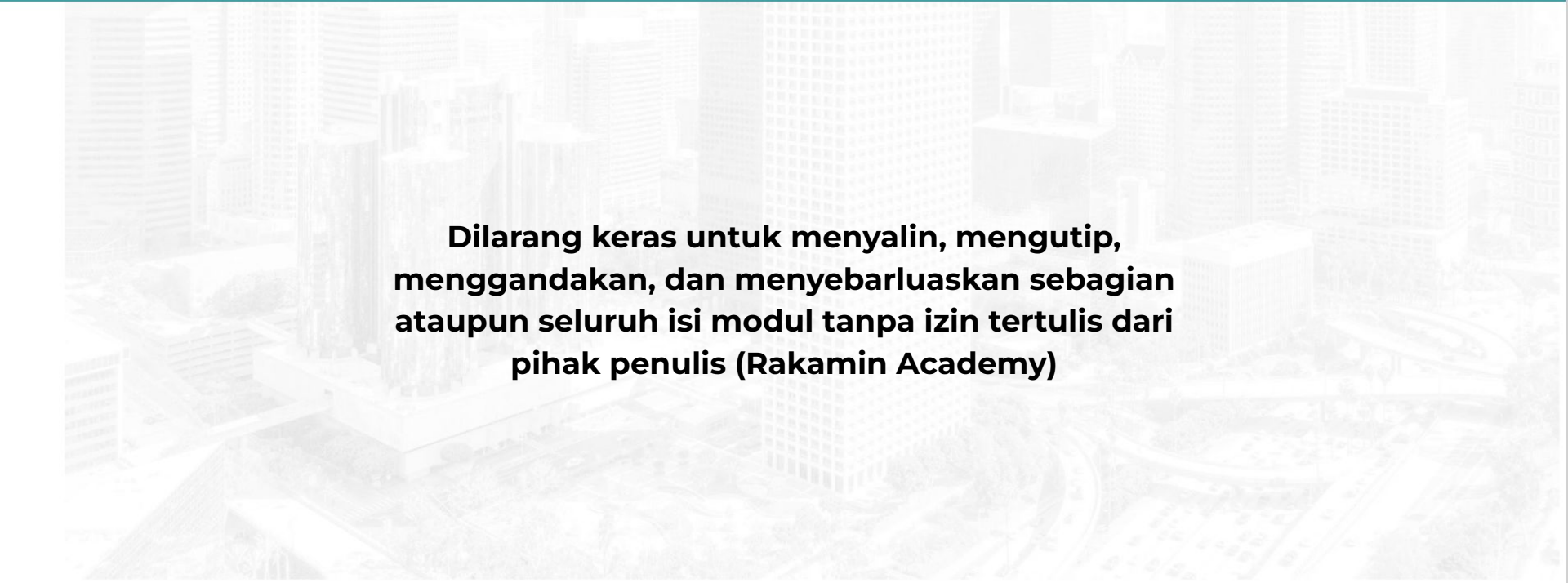
Setelah itu kita bisa menggunakan Swagger secara langsung untuk melakukan pengetesan pada API kita.

Reference material

- [Restful API Express](#)
- [Konsep Pagination dan Implementasinya](#)
- [Dokumentasi API](#)
- [Swagger](#)

Terima kasih!

Copyright Rakamin Academy

A faded, grayscale background image of a dense city skyline with numerous skyscrapers and buildings.

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