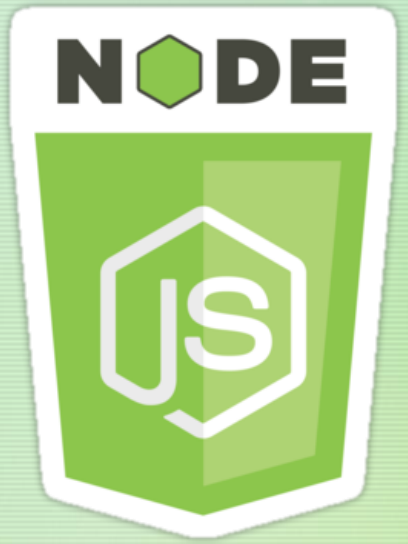
Documenting your Express API with Swagger



Swagger

Swagger is an open source set of tools that enable you to design, build, document, and use RESTful web services.

It was created to be mostly agnostic, which means that you can use it with pretty much any of your favorite languages and frameworks.

Application setup

For this tutorial, we won't cover anything related to Express API building.

I'm going to supply this ready-to-use example that you must clone to your local machine before proceeding to implementation.

Once you have this in your app, run the commands below in the terminal

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

Next, add the following imports to the beginning of the `server.js` file:

```
const swaggerJsdoc = require("swagger-jsdoc"),
swaggerUi = require("swagger-ui-express");
```

Those are the two respective objects representing the libraries we've imported. Add the following code before the app's `listen` function:

```
const options = {
  definition: {
    openapi: "3.0.0",
    info: {
      title: "Library API",
      version: "1.0.0",
      description:"A simple Express Library API",
    },
    servers: [
      {
        url: "http://localhost:3000/",
      },
    ],
  },
  apis: ["./routes/*.js"],
};

const specs = swaggerJsdoc(options);

app.use(
  "/api-docs",
  swaggerUi.serve,
  swaggerUi.setup(specs)
);
```

npm start

Creating the model

Go to `routes/books.js` and place the following code at the beginning of the file:

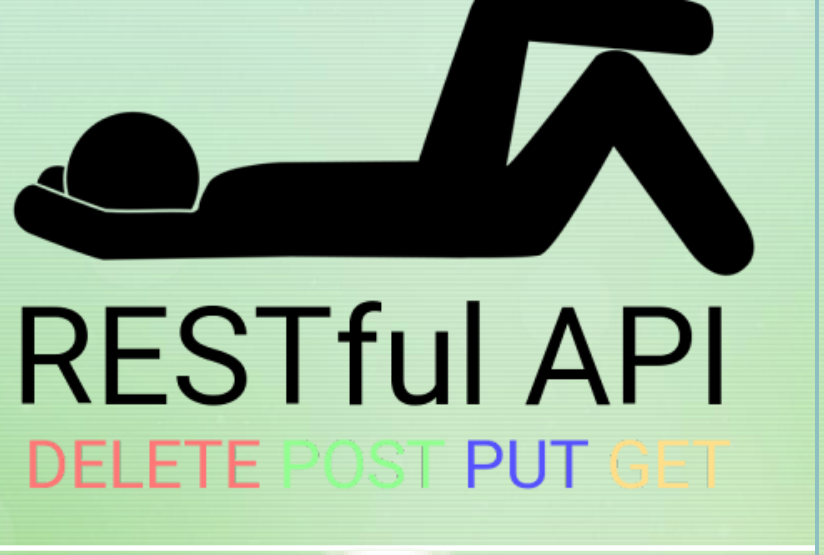
```
/**
*  @swagger
*  components:
*     schemas:
*       Book:
*         type: object
*         required:
*           - title
*           - author
*           - finished
*         properties:
*           id:
*             type: integer
*             description: The auto-generated id of the book.
*           title:
*             type: string
*             description: The title of your book.
*           author:
*             type: string
*             description: Who wrote the book?
*           finished:
*             type: boolean
*             description: Have you finished reading it?
*           createdAt:
*             type: string
*             format: date
*             description: The date of the record creation.
*         example:
*            title: The Pragmatic Programmer
*            author: Andy Hunt / Dave Thomas
*            finished: true
*/

/**
*  @swagger
*  tags:
*    name: Books
*    description: API to manage your books.
*/
```

Creating the model

The required property receives the list of attributes that are obligatory to be filled in the requests. This step is essential for letting people know what they must send when using your API.

The properties property describes the detailed information over your model attributes. Each attribute must have a name followed by its type, description (optional), and a format (you can validate values too). For a complete list of the available data types, please refer to Swagger Data Types.

Below, you can find the code for all operations CRUD:



get

```
/**
*  @swagger
*   /books/:
*     get:
*       summary: Lists all the books
*       tags: [Books]
*       responses:
*         "200":
*           description: The list of books.
*           content:
*             application/json:
*               schema:
*                 $ref: '#/components/schemas/Book'
*/
router.get("/", function (req, res) {
    res.status(200).json(books);
});
```

getById

```
/**
*  @swagger
*   /books/{id}:
*     get:
*       summary: Gets a book by id
*       tags: [Books]
*       parameters:
*         - in: path
*           name: id
*           schema:
*             type: integer
*           required: true
*           description: The book id
*       responses:
*         "200":
*           description: The list of books.
*           content:
*             application/json:
*               schema:
*                 $ref: '#/components/schemas/Book'
*         "404":
*           description: Book not found.
*/
router.get("/:id", function (req, res) {
    let book = books.find(function (item) {
        return item.id == req.params.id;
    });

    book ? res.status(200).json(book) : res.sendStatus(404);
});
```

post

```
/**
*
* @swagger
* /books/:
*     post:
*       summary: Creates 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'
*/

router.post("/", function (req, res) {
...
});
```

put

```
/**
*  @swagger
*   /books/{id}:
*     put:
*       summary: Updates a book
*       tags: [Books]
*       parameters:
*         - in: path
*           name: id
*           schema:
*             type: integer
*           required: true
*           description: The book id
*       requestBody:
*         required: true
*         content:
*           application/json:
*             schema:
*               $ref: '#/components/schemas/Book'
*       responses:
*         "204":
*           description: Update was successful.
*         "404":
*           description: Book not found.
*/

router.put("/:id", function (req, res) {
...
});
```

delete

```
/**
*  @swagger
*   /books/{id}:
*     delete:
*       summary: Deletes a book by id
*       tags: [Books]
*       parameters:
*         - in: path
*           name: id
*           schema:
*             type: integer
*           required: true
*           description: The book id
*       responses:
*         "204":
*           description: Delete was successful.
*         "404":
*           description: Book not found.
*/

router.delete("/:id", function (req, res) {
...
});
```

Conclusion

You may test each endpoint individually to make sure it’s working as precisely as your Postman requests.

Swagger is capable of way more than merely documenting your APIs. A quick read over the official docs will give you a better understanding of its power. Remember that documenting should be part of your team culture. Otherwise your docs won’t always be up to date.

Good luck!