REST APIs

Node.js Accelerator – Jan'23

Agenda

1. REST API introduction, anatomy of a REST API
2. Principles of REST
3. REST API using Express
4. Express Router
5. NestJS for building fast REST APIs
6. Swagger

Toptal®

Node.js Accelerator

Jan 2023

REST is an acronym for REpresentational State Transfer and an architectural style for distributed hypermedia systems.

- POST (create a resource or generally provide data)
- GET (retrieve an index of resources or an individual resource)
- PUT (create or replace a resource)
- PATCH (update/modify a resource)
- DELETE (remove a resource)

It all started in this dissertation

PUT vs PATCH

A common interview question you might get when talking about REST APIs is the difference between PUT and PATCH:

Patch is an operation that should update parts of a resource but not replace other parts that you didn't ask to, meanwhile PUT should update the entire resource if we are talking about REST apis.

More on patch in this link

You can start defining your API calls with resources and to each resource to use the HTTP methods available to create your own CRUD operation

```
configureRoutes(){
  this.app.route(`/users`)
    .get((req: express.Request, res: express.Response) => {
      res.status(200).send(`List of users`);
    })
    .post((req: express.Request, res: express.Response) => {
      res.status(200).send(`Post to users`);
    });

  this.app.route(`/users/:userId`)
    .all((req: express.Request, res: express.Response, next: express.NextFunction) => {
      // this middleware function runs before any request to /users/:userId
      // but it doesn't accomplish anything just yet---
      // it simply passes control to the next applicable function below using next()
      next();
    })
    .get((req: express.Request, res: express.Response) => {
      res.status(200).send(`GET requested for id ${req.params.userId}`);
    })
    .put((req: express.Request, res: express.Response) => {
      res.status(200).send(`PUT requested for id ${req.params.userId}`);
    })
    .patch((req: express.Request, res: express.Response) => {
      res.status(200).send(`PATCH requested for id ${req.params.userId}`);
    })
    .delete((req: express.Request, res: express.Response) => {
      res.status(200).send(`DELETE requested for id ${req.params.userId}`);
    });
}
```

In NestJS, the syntax changes due the Decorators usage but the REST pattern is still the same

```
@Controller( prefix: 'users')
export class UsersController {
  constructor(private readonly usersService: UsersService) {}

  @Post()
  create(@Body() createUserDto: CreateUserDto) {
    return this.usersService.create(createUserDto);
  }

  @Get()
  findAll() {
    return this.usersService.findAll();
  }

  @Get( path: ':id')
  findOne(@Param( property: 'id') id: string) {
    return this.usersService.findOne(+id);
  }

  @Patch( path: ':id')
  update(@Param( property: 'id') id: string, @Body() updateUserDto: UpdateUserDto) {
    return this.usersService.update(+id, updateUserDto);
  }

  @Delete( path: ':id')
  remove(@Param( property: 'id') id: string) {
    return this.usersService.remove(+id);
  }
}
```

Swagger with Express

Recipe:

- npm i swagger-ui-express swagger-jsdoc
- Configure your entry point and set the routes to be available
- Add the @openapi on the top of your routes and describe the usage of each route

```
/**
 * @openapi
 * /users:
 *   get:
 *     description: List users
 *     responses:
 *       200:
 *         description: Returns a list of users
 */
  .get(
      jwtMiddleware.validJWTNeeded,
      permissionMiddleware.onlyAdminCanDoThisAction,
      UsersController.listUsers
  )
```

```
const options = {
    definition: {
        openapi: "3.0.0",
        info: {title: "Toptal Rest Series"...},
        servers: [
            {
                url: "http://localhost:3000",
            },
        ],
    },
    apis: ["./**/**.routes.config.ts"],
};

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

More in this link

Quick recipe:

- npm install @nestjs/swagger swagger-ui-express
- Configure Swagger initial setup in the bootstrap function
- Enable CLI plugin: @nestjs/swagger/plugin
- Add @ApiProperty to the DTO
- Add @ApiResponse and/or variations in the controller
- Add @ApiTags to group requests based on the resource

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import {DocumentBuilder, SwaggerModule} from '@nestjs/swagger';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  const options = new DocumentBuilder()
    .setTitle('REST API')
    .setDescription('REST application')
    .setVersion('1.0')
    .build();

  const document = SwaggerModule.createDocument(app, options);

  SwaggerModule.setup( path: 'api', app, document);
  await app.listen( port: 3000);
}
bootstrap();
```

More in this link

Toptal®

ACTION ITEMS

Week 3

1. We strongly recommend you to explore below module through Udemy this week and learn about Node.js essentials

Module	Topic	Description	Udemy Link	
Self-paced learning on Udemy	ExpressJS	Learn to build API using ExpressJS, handling HTTP Requests, routing, Templating engines like Pug, handlebars	Click here	5 Hrs 6 min
Project	Incremental Project	Read the Project Requirement Part1 and Part2 Documents		


Toptal®