UNIVERSIDADE da MADEIRA

CTeSP CURSOS TÉCNICOS SUPERIORES PROFISSIONAIS

Tecnologias e Programação de Sistemas de Informação

Swagger
Sequelize

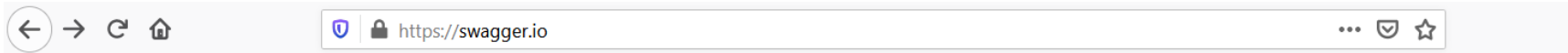Desenvolvimento Web - Back-End | David Jardim

Nodemon

- **Nodemon** is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected

Installation

```
npm install --save-dev nodemon
```

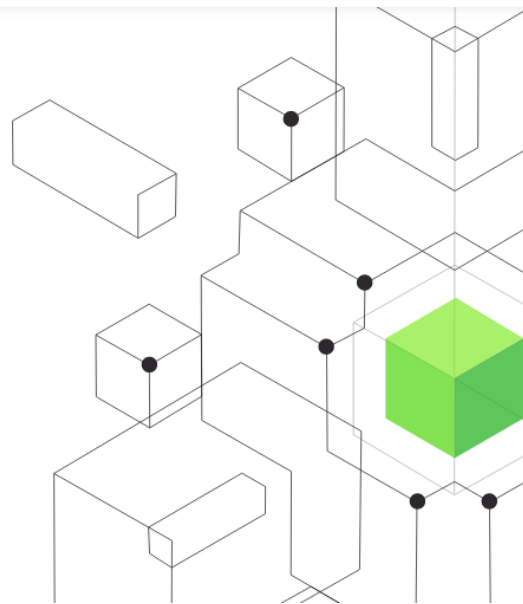## Usage

```
nodemon [your node app]
```

Swagger

- Swagger is an Interface Description Language for describing RESTful APIs expressed using JSON (**https://swagger.io/specification/**)

- Swagger is used together with a set of open-source software tools to design, build, document, and use RESTful web services

- Related NPM modules:
  - **swagger-jsdoc:** reads JSDoc-annotated code and generates an OpenAPI specification
  - **swagger-ui-express:** serve auto-generated swagger-ui generated API docs from express, based on a swagger.json file

Swagger from JSDoc - Configuration

```
1    const swaggerJsDoc = require('swagger-jsdoc');
2    const swaggerUi = require('swagger-ui-express');
3
4    const swaggerOptions = {
5        swaggerDefinition: {
6            info: {
15           definitions: {
43       },
44       apis: ["app.js"]
45   };
46
47   const swaggerDocs = swaggerJsDoc(swaggerOptions);
48   app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocs));
```

Swagger endpoint path

Swagger from JSDoc - Specification

```
4   const swaggerOptions = {
5       swaggerDefinition: {
6           info: {
7               version: "1.0.0",
8               title: "Ficha 7 API",
9               description: "Ficha 7 API Information",
10              contact: {
11                  name: "TPSI-DWB"
12              },
13              servers: ["http://localhost:3000"],
14          },
15          definitions: {
43      },
44      apis: ["app.js"]
45  };
```

Swagger from JSDoc - Model

```
15    definitions: {
16        "Person": {
17            "type": "object",
18            "properties": {
19                "id": {
20                    "type": "integer",
21                    "x-primary-key": true
22                },
23                "firstname": {
24                    "type": "string"
25                },
26                "lastname": {
27                    "type": "string"
28                },
29                "profession": {
30                    "type": "string"
31                },
32                "age": {
33                    "type": "integer",
34                    "format": "int64"
35                }
36            },
```

GET

Swagger from JSDoc - Usage

```
1   /**
2    * @swagger
3    * /person:
4    *     get:
5    *       tags:
6    *         - Person
7    *       summary: Gets a list of persons
8    *       description: Returns a list of persons
9    *       produces:
10   *           - application/json
11   *       responses:
12   *           200:
13   *                description: An array of persons
14   *                schema:
15   *                  $ref: '#/definitions/Person'
16   */
17  app.get('/person', (request, response) => {
```

POST

Swagger from JSDoc - Usage

```
/**
 * @swagger
 * /person:
 *    post:
 *      tags:
 *        - Person
 *      summary: Creates and stores a person
 *      description: Returns the id of the created person
 *      produces:
 *          - application/json
 *      parameters:
 *          - name: Model
 *            description: Sample person
 *            in: body
 *            required: true
 *            schema:
 *              $ref: '#/definitions/Person'
 *      responses:
 *          200:
 *              description: Successfully created
 */
app.post('/person', (request, response) => {
```

Swagger from JSON - Configuration

```
1  const swaggerUi = require('swagger-ui-express');
2  const swaggerDocument = require('./swagger.json');
3
4  // instanciar o express
5  const app = express();
6  app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
```

Swagger specification file

Swagger endpoint path

Swagger from JSON - Specification

```json
{
    "swagger": "2.0",
    "info": {
        "version": "1.0.0",
        "title": "Ficha 7 API",
        "description": "Ficha 7 API Information",
        "contact": {
            "name": "TPSI-DWB"
        },
        "servers": [
            "http://localhost:3000"
        ]
    },
    "paths": {
    "definitions": {
}
```

Swagger from JSON - Model

```
15  definitions: {
16      "Person": {
17          "type": "object",
18          "properties": {
19              "id": {
20                  "type": "integer",
21                  "x-primary-key": true
22              },
23              "firstname": {
24                  "type": "string"
25              },
26              "lastname": {
27                  "type": "string"
28              },
29              "profession": {
30                  "type": "string"
31              },
32              "age": {
33                  "type": "integer",
34                  "format": "int64"
35              }
36          },
```

POST

Swagger from JSON - Paths

```json
"paths": {
    "/person": {
        "post": {
            "tags": [
                "person"
            ],
            "summary": "Create person",
            "description": "This can only be done by the logged in user.",
            "operationId": "createPerson",
            "produces": [
                "application/xml",
                "application/json"
            ],
            "parameters": [
                {
                    "in": "body",
                    "name": "body",
                    "description": "Created person object",
                    "required": true,
                    "schema": {
                        "$ref": "#/definitions/Person"
                    }
                }
            ],
            "responses": {
                "default": {
                    "description": "successful operation"
                }
            }
        },
```

DELETE

Swagger from JSON - Paths

```
"/person/{id}": {
    "get": {
        "tags": [
            "person"
        ],
        "summary": "Get person by id",
        "description": "Get person by ID.",
        "operationId": "getPersonById",
        "produces": [
            "application/xml",
            "application/json"
        ],
        "parameters": [
            {
                "name": "id",
                "in": "path",
                "description": "The id that needs to be fetched. Use id 1 for testing. ",
                "required": true,
                "type": "string"
            }
        ],
        "responses": {
            "200": {
                "description": "successful operation",
                "schema": {
                    "$ref": "#/definitions/Person"
                }
            },
            "400": {
                "description": "Invalid id supplied"
            },
            "404": {
                "description": "Id not found"
            }
```

pet Everything about your Pets

Find out more: http://swagger.io ⌄

| POST | /pet Add a new pet to the store | 🔓 |

| PUT | /pet Update an existing pet | 🔓 |

| GET | /pet/findByStatus Finds Pets by status | 🔓 |

| GET | ~~/pet/findByTags~~ Finds Pets by tags | 🔓 |

| GET | /pet/{petId} Find pet by ID | 🔓 |

| POST | /pet/{petId} Updates a pet in the store with form data | 🔓 |

| DELETE | /pet/{petId} Deletes a pet | 🔓 |

| POST | /pet/{petId}/uploadImage uploads an image | 🔓 |

store Access to Petstore orders ⌄

| GET | /store/inventory Returns pet inventories by status | 🔓 |

| POST | /store/order Place an order for a pet |

POST `/store/order` Place an order for a pet

Parameters

<div style="text-align:right">Try it out</div>

Name	Description

body * required
object
(body)

order placed for purchasing the pet

Example Value   Model

```json
{
  "id": 0,
  "petId": 0,
  "quantity": 0,
  "shipDate": "2021-05-03T20:10:19.928Z",
  "status": "placed",
  "complete": false
}
```

Parameter content type

application/json ▾

Responses

Response content type application/xml ▾

Code	Description

200

successful operation

Example Value | Model

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Order>
        <id>0</id>
        <petId>0</petId>
        <quantity>0</quantity>
        <shipDate>2021-05-03T20:10:19.942Z</shipDate>
        <status>placed</status>
        <complete>false</complete>
</Order>
```

400

Invalid Order

17

npm   🔍 Search packages   **Search**   **Sign Up**   Sign In

Meet npm Pro: unlimited public & private packages + package-based permissions. **Learn more »**

sequelize

5.21.3 • `Public` • Published a month ago

📄 **Readme**	🔥 Explore `BETA`	📦 **15** Dependencies	🔗 **3300** Dependents	🏷 **493** Versions

Sequelize

`npm v5.21.3` `travis passing` `build passing` `downloads 1.74M/month` `coverage 96.3%`
`last commit about 15 hours ago` `merged PRs 2.46K` `stars 20.94K` `slack 6085` `node >=6.0.0` `license MIT`
`semantic-release`

Sequelize is a promise-based Node.js ORM for Postgres, MySQL, MariaDB, SQLite and Microsoft SQL Server. It features solid transaction support, relations, eager and lazy loading, read replication and more.

Sequelize follows SEMVER. Supports Node v6 and above to use ES6 features.

New to Sequelize? Take a look at the Tutorials and Guides. You might also be interested in the API Reference.

Table of Contents

- Installation
- Documentation
- Responsible disclosure
- Resources

Install

```
> npm i sequelize
```

⬇ Weekly Downloads

509 061

Version	License
5.21.3	MIT

Unpacked Size	Total Files
1.14 MB	124

Issues	Pull Requests
721	57

Homepage

🔗 sequelize.org/

Repository

🔗 github.com/sequelize/sequelize

Sequelize

- **Sequelize** is a promise-based Node.js **ORM** for Postgres, MySQL, MariaDB, SQLite and Microsoft SQL Server

- **Object-relational mapping** (ORM) in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages

- This creates, in effect, a "virtual object database" that can be used from within the programming language

MySQL Client for Node

```
connection.query(
  'SELECT * FROM `table` WHERE `name` = "Page" AND `age` > 45',
  function(err, results, fields) {
    console.log(results); // results contains rows returned by server
    console.log(fields); // fields contains extra meta data about results
  }
);
```

Sequelize

```
Post.findAll({
  where: {
    name: "Page",
    age: { [Op.lt]:45
    }
  }
});
```

Install Sequelize

```
# install sequelize
npm install --save sequelize
```

```
# install mysql2
npm install --save mysql2
```

Define a Model with JS code

A model is a class that extends Sequelize.Model.

```js
const User = sequelize.define('user', {
  // attributes
  firstName: {
    type: Sequelize.STRING,
    allowNull: false
  },
  lastName: {
    type: Sequelize.STRING
    // allowNull defaults to true
  }
}, {
  // options
});
```

- Internally, sequelize.define calls Sequelize.Model.init

- This code tells Sequelize to expect a table named **users** in the database with the fields **firstName** and **lastName**. The table name is automatically pluralized by default

- Sequelize also defines by default the fields **id** (primary key), **createdAt** and **updatedAt** to every model.

Synchronize the Models with the Database

```
sequelize.sync({ force: false })
    .then(() => {
        console.log('Database & tables created!');
    ]).then(function () {
        return Person.findAll();
    }).then(function (persons) {
        console.log(persons);
    });
});
```

- Synchronizes all the defined models to the DB

- If **force** is true, each model will DROP TABLE IF EXISTS

- https://sequelize.org/master/class/lib/seque lize.js~Sequelize.html#instance-method-sync

Insert multiple instances in bulk

```
Person.bulkCreate([
        { firstname: 'Pedro', lastname: 'Jardim', profession: 'IT', age:62 },
        { firstname: 'Manuel', lastname: 'Matos', profession: 'IT', age:12 },
        { firstname: 'Maria', lastname: 'Figueira', profession: 'IT', age:32 },
        { firstname: 'Ana', lastname: 'Duarte', profession: 'IT', age:82 },
        { firstname: 'Luís', lastname: 'Faria', profession: 'IT', age:42 }

]).then(function () {
    return Person.findAll();
}).then(function (persons) {
    console.log(persons);
});
```

- Create and insert multiple instances in bulk

- https://sequelize.org/master/class/lib/model.js~Model.html#static-method-bulkCreate

The Sequelize CLI

Installing CLI
$ npm install --save sequelize-cli

Bootstrapping (from project folder)
$ npx sequelize-cli init

This will create following folders
• config, contains config file, which tells CLI how to connect with database
• models, contains all models for your project
• migrations, contains all migration files
• seeders, contains all seed files

Sequelize Configurations

config/config.json

```json
{
  "development": {
    "username": "root",
    "password": "root",
    "database": "database"
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "test": {
    "username": "root",
    "password": null,
    "database": "database_test",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "production": {
    "username": "root",
    "password": null,
    "database": "database_production",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}
```

Create a model (and migration) with CLI

Execute from project directory:

$ npx sequelize-cli model:generate --name User --attributes
firstName:string,lastName:string,email:string

Attribute name

Attribute type

Model/table name

Create a model (and migration) with CLI

$ npx sequelize-cli model:generate --name User --attributes firstName:string,lastName:string,email:string

- This will create a file inside the *models* folder and the *migration* folder

- The file inside the *models* folder called *user.js* will be used to define the user table database in MySQL.

- The file inside the migration folder will be used to migrate the models to the MySQL database

Migrate models to the schema

to actually create that table in database you need to run **db:migrate** command

$ npx sequelize-cli db:migrate

- Will ensure a table called SequelizeMeta in database. This table records which migrations have run on the current database.

- Start looking for any migration files which haven't run yet. This is possible by checking SequelizeMeta table.

- Creates a table called Users with all columns as specified in its migration file.

Use models in app.js

- **Import models module:**

```
const models = require('./models');
const User = models.User;
```

- **Perform queries...**

CRUD Queries

```javascript
// Find all users
User.findAll().then(users => {
  console.log("All users:", JSON.stringify(users, null, 4));
});

// Create a new user
User.create({ firstName: "Jane", lastName: "Doe" }).then(jane => {
  console.log("Jane's auto-generated ID:", jane.id);
});

// Delete everyone named "Jane"
User.destroy({
  where: {
    firstName: "Jane"
  }
}).then(() => {
  console.log("Done");
});

// Change everyone without a last name to "Doe"
User.update({ lastName: "Doe" }, {
  where: {
    lastName: null
  }
}).then(() => {
  console.log("Done");
});
```

It is also possible to make raw SQL queries, if you really need them.

find - Search for one specific element in the database

```javascript
// search for known ids
Project.findByPk(123).then(project => {
  // project will be an instance of Project and stores the content of the table entry
  // with id 123. if such an entry is not defined you will get null
})

// search for attributes
Project.findOne({ where: {title: 'aProject'} }).then(project => {
  // project will be the first entry of the Projects table with the title 'aProject' || null
})


Project.findOne({
  where: {title: 'aProject'},
  attributes: ['id', ['name', 'title']]
}).then(project => {
  // project will be the first entry of the Projects table with the title 'aProject' || null
  // project.get('title') will contain the name of the project
})
```

They do *not* return plain objects but instead return model instances

Connection Pool (production)

If connecting to the database from a single process, you should create only one Sequelize instance. This connection pool can be configured through the constructor's options parameter (using options.pool):

```javascript
const sequelize = new Sequelize(/* ... */, {
  // ...
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
});
```

If connecting from multiple processes, you'll have to create one instance per process, but each instance should have a maximum connection pool size of such that the total maximum size is respected. For example, for a max connection pool size of 90 and three processes, the Sequelize instance of each process should have a max connection pool size of 30.

Documentation

- https://sequelize.org/v5/index.html

    - https://sequelize.org/v5/manual/getting-started.html#installing

    - https://sequelize.org/v5/manual/migrations.html

    - https://sequelize.org/v5/manual/querying.html

    - https://sequelize.org/v5/manual/models-usage.html

UNIVERSIDADE da MADEIRA

CTeSP CURSOS TÉCNICOS SUPERIORES PROFISSIONAIS