

Tecnologias e Programação de Sistemas de Informação

Swagger Sequelize

Desenvolvimento Web - Back-End | David Jardim

Cofinanciado por:



🔍 Search packages

Search

Sign Up

Sign In

Have ideas to improve npm? [Join in the discussion!](#) »

nodemon DT

2.0.7 • Public • Published 4 months ago

Readme

Explore BETA

10 Dependencies

2,985 Dependents

218 Versions



Install

```
> npm i nodemon
```

♥ Fund this package

📉 Weekly Downloads

3,410,920



Version

2.0.7

License

MIT

Nodemon

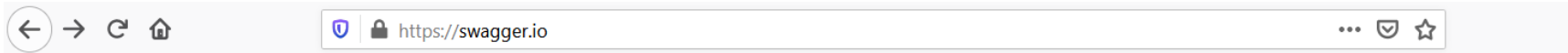
- **Nodemon** is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected

Installation

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

Usage

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



Swagger.
Supported by SMARTBEAR

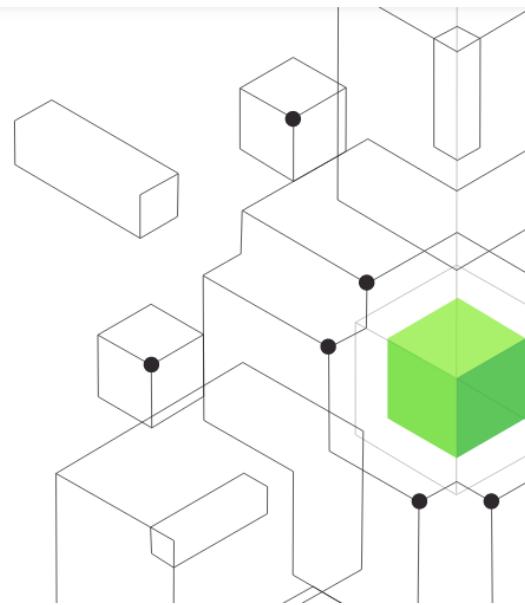
Why Swagger? ▾

Tools ▾

Resources ▾

API Development for Everyone

Simplify API development for users, teams, and enterprises with the Swagger open source and professional toolset. Find out how Swagger can help you design and document your APIs at scale.



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
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
```

```
definitions: {
  "Person": {
    "type": "object",
    "properties": {
      "id": {
        "type": "integer",
        "x-primary-key": true
      },
      "firstname": {
        "type": "string"
      },
      "lastname": {
        "type": "string"
      },
      "profession": {
        "type": "string"
      },
      "age": {
        "type": "integer",
        "format": "int64"
      }
    }
  },
}
```


Swagger from JSDoc - Usage

GET

```
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) => {
```

Swagger from JSDoc - Usage

POST

```
/**
 * @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

```
1  {
2      "swagger": "2.0",
3      "info": {
4          "version": "1.0.0",
5          "title": "Ficha 7 API",
6          "description": "Ficha 7 API Information",
7          "contact": {
8              "name": "TPSI-DWB"
9          },
10         "servers": [
11             "http://localhost:3000"
12         ],
13     },
14     "paths": {
138     "definitions": {
165     }
```

Swagger from JSON - Model

```
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
```

```
definitions: {
  "Person": {
    "type": "object",
    "properties": {
      "id": {
        "type": "integer",
        "x-primary-key": true
      },
      "firstname": {
        "type": "string"
      },
      "lastname": {
        "type": "string"
      },
      "profession": {
        "type": "string"
      },
      "age": {
        "type": "integer",
        "format": "int64"
      }
    }
  },

```

Swagger from JSON - Paths

POST

```
"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"
        }
      }
    }
  },
}
```

Swagger from JSON - Paths

DELETE

```
"/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

[Try it out](#)

Name

Description

body * required

object

(body)

order placed for purchasing the pet

Example Value | Model

```
{
  "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 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



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

Promise based?

- The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value
- A Promise is a proxy for a value not necessarily known when the promise is created
- A Promise is in one of these states:
 - pending: initial state, neither fulfilled nor rejected.
 - fulfilled: meaning that the operation was completed successfully.
 - rejected: meaning that the operation failed.

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
```

Connect and authenticate to DB using Sequelize in JS code

```
const sequelize = new Sequelize('ficha9', 'root', 'password', {  
  dialect: 'mysql'  
});
```

```
sequelize.authenticate()  
  .then(() => {  
    console.log("Connection has been established");  
  })  
  .catch(err => {  
    console.error("Unable to connect", err);  
  });
```

Define a Model with JS code

A model is a class that extends `Sequelize.Model`.

```
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/sequelize.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

```
{
  "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

```
// 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

```
// search for known ids
Project.findById(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):

```
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.

Associations/Relationships

Sequelize supports the standard associations: **One-To-One**, **One-To-Many**, **Many-To-Many**

This is achieved using the following associations:

- The **HasOne** association
- The **BelongsTo** association
- The **HasMany** association
- The **BelongsToMany** association

HasOne

One-To-One relationship exists between A and B

Foreign key is defined in model B

```
const A = sequelize.define('A', /* ... */);  
const B = sequelize.define('B', /* ... */);  
  
A.hasOne(B); // A HasOne B
```

BelongsTo

One-To-One relationship exists between A and B

Foreign key is defined in model A

```
const A = sequelize.define('A', /* ... */);  
const B = sequelize.define('B', /* ... */);  
  
A.belongsTo(B); // A BelongsTo B
```

HasMany

One-To-Many relationship exists between A and B

Foreign key is defined in model B

```
const A = sequelize.define('A', /* ... */);  
const B = sequelize.define('B', /* ... */);  
  
A.hasMany(B); // A HasMany B
```

BelongsToMany

Many-To-Many relationship exists between A and B using table C (mandatory) as junction table

Foreign keys are defined in table C as aID and bID

```
const A = sequelize.define('A', /* ... */);  
const B = sequelize.define('B', /* ... */);  
  
A.belongsToMany(B, { through: 'C', /* options */});
```


Options

onDelete & onUpdate

The possible choices are RESTRICT, CASCADE, NO ACTION, SET DEFAULT and SET NULL.

The defaults for the One-To-One associations is SET NULL for ON DELETE and CASCADE for ON UPDATE.

```
A.hasOne(B, {  
  onDelete: 'RESTRICT',  
  onUpdate: 'RESTRICT'  
});
```

JOIN queries in Sequelize

After defining the relationship/association

Use the **include** option in the query method

By default the **include** option acts as a LEFT OUTER JOIN

INNER JOIN need the **required** option defined as true

```
Const results = A.findAll({  
  include: {model: B, required: true}  
});
```

INNER JOIN	<div>1</div> <div>2</div> <div>3</div>	INNER JOIN	<div>A</div> <div>B</div> <div>C</div>	=	<div>1</div> <div>2</div>	<div>B</div> <div>A</div>	Only returns rows that meet the join condition
RIGHT OUTER JOIN	<div>1</div> <div>2</div> <div>3</div>	RIGHT OUTER JOIN	<div>A</div> <div>B</div> <div>C</div>	=	<div>1</div> <div>2</div>	<div>B</div> <div>A</div> <div>C</div>	Returns all rows from the table on the right side of JOIN and matched rows from the left side of the JOIN
LEFT OUTER JOIN	<div>1</div> <div>2</div> <div>3</div>	LEFT OUTER JOIN	<div>A</div> <div>B</div> <div>C</div>	=	<div>1</div> <div>2</div> <div>3</div>	<div>B</div> <div>A</div>	Returns all rows from the table on the left side of JOIN and matched rows from the right side of the JOIN
FULL OUTER JOIN	<div>1</div> <div>2</div> <div>3</div>	FULL OUTER JOIN	<div>A</div> <div>B</div> <div>C</div>	=	<div>1</div> <div>2</div> <div>3</div>	<div>B</div> <div>A</div> <div>C</div>	Returns all rows from both sides even if join condition is not met
CROSS JOIN	<div>1</div> <div>2</div> <div>3</div>	CROSS JOIN	<div>A</div> <div>B</div> <div>C</div>	=	<div>1</div> <div>1</div> <div>1</div> <div>2</div> <div>2</div> <div>2</div> <div>3</div> <div>3</div> <div>3</div>	<div>A</div> <div>B</div> <div>C</div> <div>A</div> <div>B</div> <div>C</div> <div>A</div> <div>B</div> <div>C</div>	Cartesian product between the two sides is a join but without a join condition. Returns all rows joined from both sides

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>



CTeSP

CURSOS TÉCNICOS
SUPERIORES PROFISSIONAIS