Swagger with NodeJS / Express

The objective of this document is to give some instructions about how to create/visualize swagger documents with applications written in NodeJS / Express. Any question/suggestion/fix can be sent to my email: rodrigo.giraldisilva@cgi.com.

What is Swagger?

Swagger is an open source software framework backed by a large ecosystem of tools that helps developers design, build, document, and consume RESTful Web services. While most users identify Swagger by the Swagger UI tool, the Swagger toolset includes support for automated documentation, code generation, and test case generation[1].

Our objective with Swagger

As we can read in definition, there are many possible uses for this framework. We will be using some node packages to generate swagger files and run a web server, which provides an intuitive UI where we can view the definition of the endpoints present in our NodeJS/Express application, like routes, parameters of request and possible responses. We will also be able to test these endpoints, doing real requests to our application.

Required packages (NPM)

There are many alternatives of node packages that can be used to achieve our objective but we will be using specifically two: swagger-jsdoc[2] and express-swaggerize-ui[3]. The first one is basically to generate a JSON file with all swagger definitions that we put in our code (comments/annotations), this JSON file can be used in different applications, like express-swaggerize-ui and Swagger Editor[4].

The second package (express-swaggerize-ui) creates a web application, which consumes the JSON file of swagger definitions. This package runs with express[5], this way we will also need to install it.

You can find sample applications of these two packages in their respective npmjs.com page.[2],[3].

Swagger/JSDoc comments

In order to generate the JSON file we need to write JSDoc[6] comments related to our endpoints, with routes, responses and models definitions. There is not a specific place to put these comments, as long as you specify in your express-swaggerize-ui in the “swaggerDefinition” object, “apis” array property. In my case, as we used express-generator to create the structure of our project, I decided to place the comments in the controllers and models.

In the models are the entities definitions, these definitions are used when we specify that a request/response requires/produces this type/definition. This is a good way to avoid repetition and also have a clear understanding of what that object is. What you need to do is specify all properties (name and type) of our model. In some cases, you do not have the same data type in swagger that you used in your real application. You can check which type are available in swagger at their official documentation [7].

The second part and most important are the endpoints definitions, in fact, you do not need to create a model in order to define an endpoint, sometimes you will not have an explicit type in your request nor response. As mentioned before, I put these comments in my controller methods because in our case is where we could have more information about parameters and responses, like possible response status and what which one returns.

Two good references that I followed to write the swagger comments were swagger petstore [8] and swagger-jsdoc documentation[9]. Swagger editor[4] is also a great tool to test and also see occurring errors. This editor can consume a JSON file or YAML. The displayed swagger definition in this tool is not in JSON format, it is in YAML, but at least in opinion it is helpful. To open your JSON file in this editor you can go to File/Import URL or File. If for any reason, you want to get the default swagger definition of the editor you can just delete all your current definition and refresh the page.

Swagger/JSDoc comments and NodeJS code examples

```
/**
 * @swagger
 * definitions:
 *   User:
 *     type: object
 *     required:
 *       - username
 *       - password
 *     properties:
 *       _id:
 *         type: string
 *       username:
 *         type: string
 *       password:
 *         type: string
 *       createdAt:
 *         type: string
 *       sharedSprints:
 *         type: array
 *         items:
 *           type: string
 *       __v:
 *         type: number
 */
var userSchema = new mongoose.Schema({
    username: String,
    password: String,
    createdAt: Date,
    sharedSprints: [String]
})
```

User swagger definition and NodeJS code

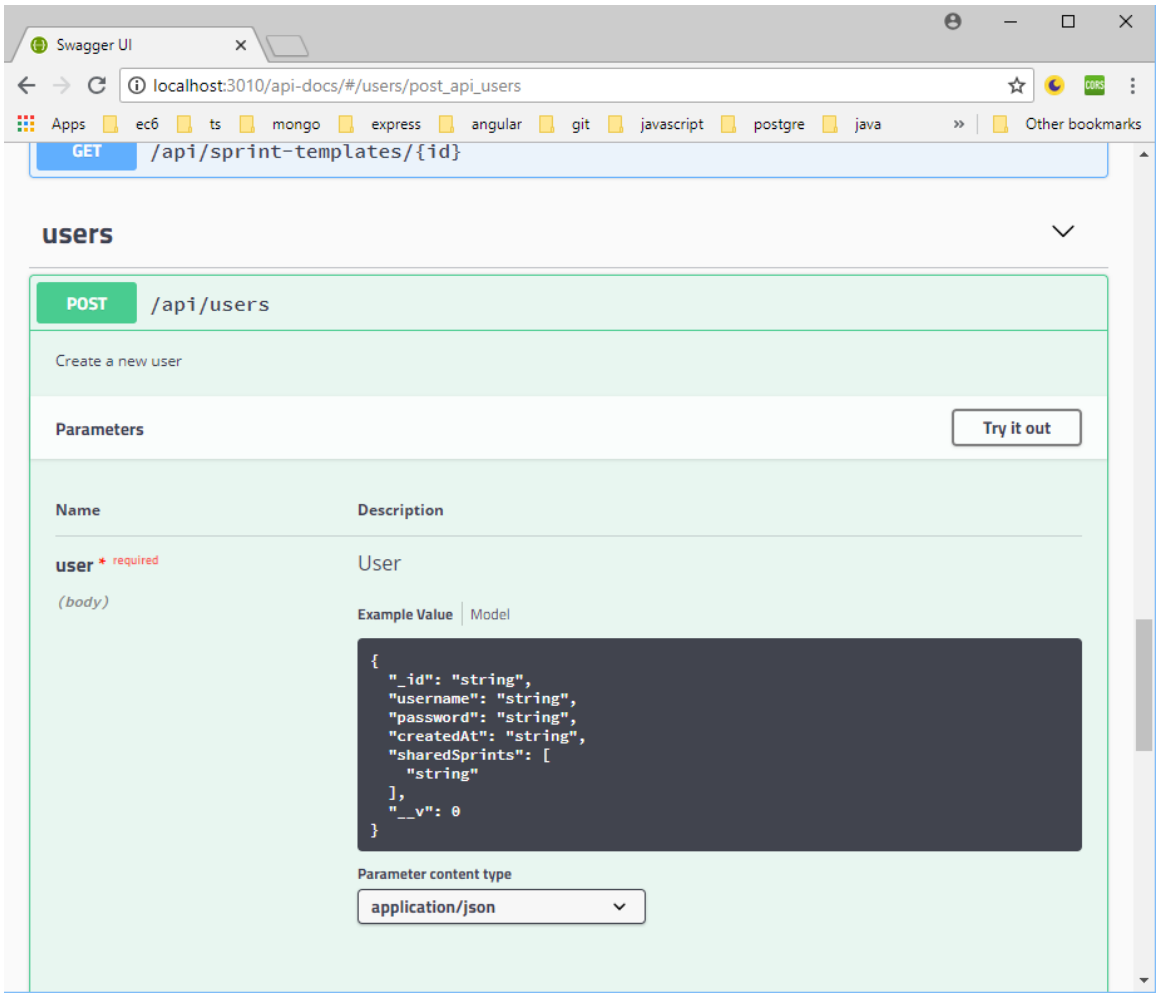
```
  }
}

/**
  * @swagger
  * /api/users/login:
  *   post:
  *     tags:
  *       - users
  *     parameters:
  *       - name: user
  *         description: User
  *         in: body
  *         required: true
  *         schema:
  *           $ref: '#/definitions/User'
  *     description: Log in a user
  *     produces:
  *       - application/json
  *     responses:
  *       200:
  *         description: ok
  *         schema:
  *           type: object
  *           properties:
  *             status:
  *               type: number
  *             data:
  *               $ref: '#/definitions/User'
  *             message:
  *               type: string
  *       400:
  *         description: bad request
  *       406:
  *         description: not acceptable
  */
exports.loginUser = async function (req, res, next) {
```

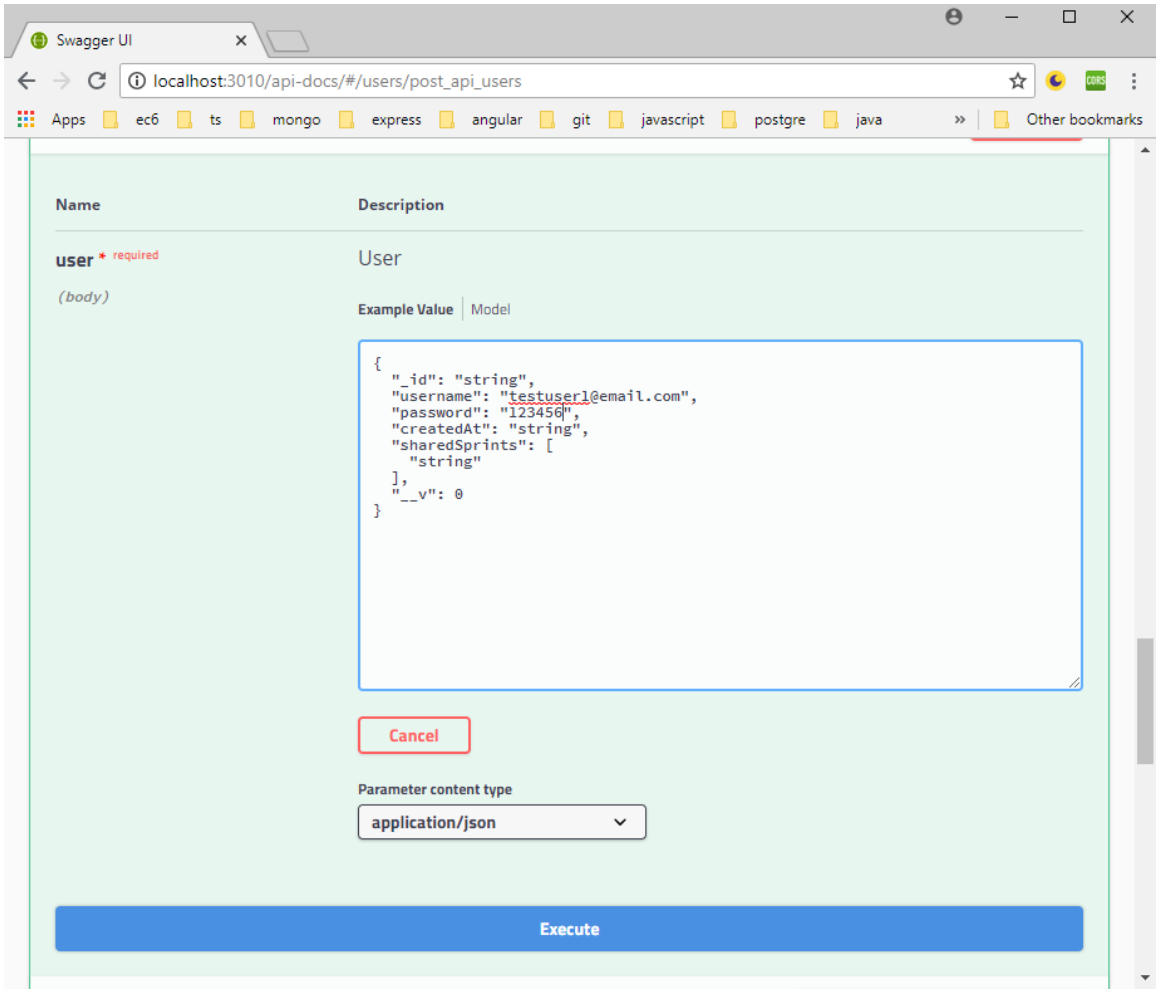
/api/users/login endpoint swagger definition

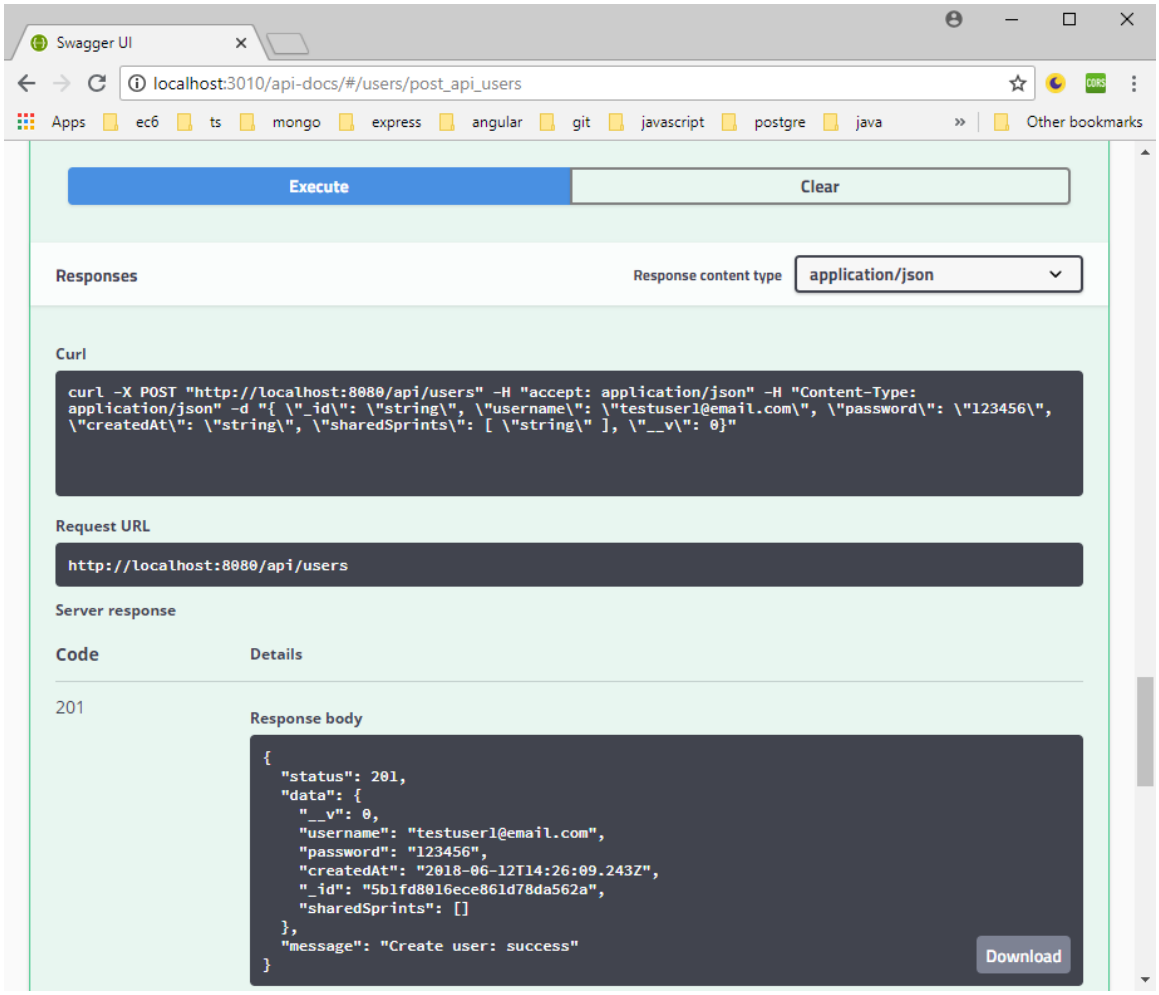
```
 *       400:
 *         description: bad request
 *       409:
 *         description: conflict
 */
exports.createUser = async function (req, res, next) {

  if (!req.body.username || !req.body.password) {
    return res.status(400).json({
      status: 400,
      message: "Create user: error"
    });
  }

  var newUser = {
    username: req.body.username,
    password: req.body.password
  }

  try {
    var usernameAvailable = await userService.usernameAvailable(newUser.username);
    if (!usernameAvailable) {
      return res.status(409).json({
        status: 409,
        message: "Username already exists"
      })
    }
    var createdUser = await userService.createUser(newUser);
    return res.status(201).json({
      status: 201,
      data: createdUser,
      message: "Create user: success"
    })
  } catch (e) {
    return res.status(400).json({
      status: 400,
      message: "Create user: error"
    })
  }
}
```

NodeJS method code

express-swaggerize-ui running application and request example




Swagger UI
localhost:3010/api-docs/#/users/post_api_users
Apps
ec6
ts
mongo
express
angular
git
javascript
postgre
java
Other bookmarks
Name
Description
user * required
(body)
User
Example Value
Model
{
"_id": "string",
"username": "testuser1@email.com",
"password": "123456",
"createdAt": "string",
"sharedSprints": [
"string"
],
"__v": 0
}
Cancel
Parameter content type
application/json
Execute


Swagger UI
localhost:3010/api-docs/#/users/post_api_users
Apps
ec6
ts
mongo
express
angular
git
javascript
postgre
java
Other bookmarks
Execute
Clear
Responses
Response content type
application/json
Curl
curl -X POST "http://localhost:8080/api/users" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"_id\": \"string\", \"username\": \"testuser1@email.com\", \"password\": \"123456\", \"createdAt\": \"string\", \"sharedSprints\": [\"string\"], \"__v\": 0}"
Request URL
http://localhost:8080/api/users
Server response
Code
Details
201
Response body
{
"status": 201,
"data": {
"__v": 0,
"username": "testuser1@email.com",
"password": "123456",
"createdAt": "2018-06-12T14:26:09.243Z",
"_id": "5b1fd8016ece861d78da562a",
"sharedSprints": []
},
"message": "Create user: success"
}
Download

Reference

[1] https://en.wikipedia.org/wiki/Swagger_(software)

[2] https://www.npmjs.com/package/swagger-jsdoc

[3] https://www.npmjs.com/package/express-swaggerize-ui

[4] https://editor.swagger.io/

[5] https://www.npmjs.com/package/express

[6] https://www.npmjs.com/package/express-generator

[7] https://swagger.io/docs/specification/data-models/data-types/

[8] http://petstore.swagger.io/

http://petstore.swagger.io/v2/swagger.json

[9] https://github.com/Surnet/swagger-jsdoc/blob/master/docs/GETTING-STARTED.md