# Documentar y testear API

U-TAD

# Swagger (OpenAPI)

https://swagger.io/

https://editor.swagger.io/

Lo haremos con el API de Swagger para node.js:

npm i swagger-ui-express swagger-jsdoc

# Swagger (OpenAPI)

Crear la carpeta **docs/** y ahí el fichero **swagger.js** con las opciones (copiarlo de github)

Y en app.js

```
const swaggerUi = require("swagger-ui-express")
const swaggerSpecs = require("./docs/swagger")

...

app.use("/api-docs",
    swaggerUi.serve,
    swaggerUi.setup(swaggerSpecs)
)
app.use("/api", require("/routes"))
```

# Swagger (OpenAPI)

Y en **routes/auth.js**

```
/**
 * @openapi
 * /api/auth/register:
 *  post:
 *      tags:
 *      - User
 *      summary: "User registter"
 *      description: Register a new user
 *      requestBody:
 *          content:
 *              application/json:
 *                  schema:
 *                      $ref: "#/components/schemas/user"
 *      responses:
 *          '200':
 *              description: Returns the inserted object
 *          '401':
 *              description: Validation error
 *      security:
 *          - bearerAuth: []
 */
router.post("/register", validatorRegister, registerCtrl)
```

# Swagger (OpenAPI)

```
/**
 * @openapi
 * /api/auth/login:
 *  post:
 *      tags:
 *      - User
 *      summary: Login user
 *      description: ''
 *      requestBody:
 *          content:
 *              application/json:
 *                  schema:
 *                      $ref: "#/components/schemas/login"
 *      responses:
 *          '200':
 *              description: Returns the inserted object
 *          '401':
 *              description: Validation error
 */
router.post("/login", validatorLogin, loginCtrl)
```

# Swagger (OpenAPI)

```
/**
 * @openapi
 * /api/auth/update/{id}:
 *  put:
 *      tags:
 *      - User
 *      summary: Update user
 *      description: Update a user by an admin
 *      parameters:
 *          -   name: id
 *              in: path
 *              description: id that need to be updated
 *              required: true
 *              schema:
 *                  type: string
 *      requestBody:
 *          content:
 *              application/json:
 *                  schema:
 *                      $ref: "#/components/schemas/user"
 *      responses:
 *          '200':
 *              description: Returns the inserted object
 *          '401':
 *              description: Validation error
 *      security:
 *          - bearerAuth: []
 */
router.put("/update/:id", authMiddleware, checkRol(["admin"]), validatorGetUser, validatorUpdate, updateUser)
```

# Swagger (OpenAPI)

```
/**
 * @openapi
 * /api/auth/users:
 *  get:
 *      tags:
 *      - User
 *      summary: Get users in the System
 *      description: ''
 *      responses:
 *          '200':
 *              description: Returns the users
 *          '500':
 *              description: Server error
 *      security:
 *          - bearerAuth: []
 */
router.get("/users", authMiddleware, getUsers)
```

# Ejercicio

Documentar con swagger la rutas de “storage” y “tracks”

# Jest (unit testing)

- Automatización de pruebas unitarias con Jest

npm i jest supertest --save-dev

Pruebas con **nosql** con *ENGINE_DB* = *nosql* de .**env**

En **package.json**:

```
"scripts": {
    "test": "jest --forceExit",
    "start": "nodemon start"
  },
```

Y en **app.js** (al final)

```
module.exports = app
```

# Jest (unit testing)

En test/**app.test.js**

```
const request = require('supertest');
const app = require('../app')
describe('users', () => {
    var token = ""
    var id = ""
    it('should register a user', async () => {
        const response = await request(app)
            .post('/api/auth/register')
            .send({"name": "Menganito","age": 20,"email": "user25@test.com","password": "HolaMundo.01"})
            .set('Accept', 'application/json')
            .expect(200)
        expect(response.body.user.email).toEqual('user25@test.com')
        expect(response.body.user.role).toEqual('user')
        token = response.body.token
        id = response.body.user._id
    })
    it('should get the users', async () => {
        const response = await request(app)
            .get('/api/auth/users')
            .auth(token, { type: 'bearer' })
            .set('Accept', 'application/json')
            .expect(200)
        expect(response.body.pop().name).toEqual('Menganito')
    });
    it('should delete a user', async () => {
        const response = await request(app)
            .delete('/api/auth/users/'+id)
            .auth(token, { type: 'bearer' })
            .set('Accept', 'application/json')
            .expect(200)
        expect(response.body.acknowledged).toEqual(true)
    })
})
```

# Jest (unit testing)

Ejercicio:

Completar pruebas unitarias automáticas para el resto de peticiones.