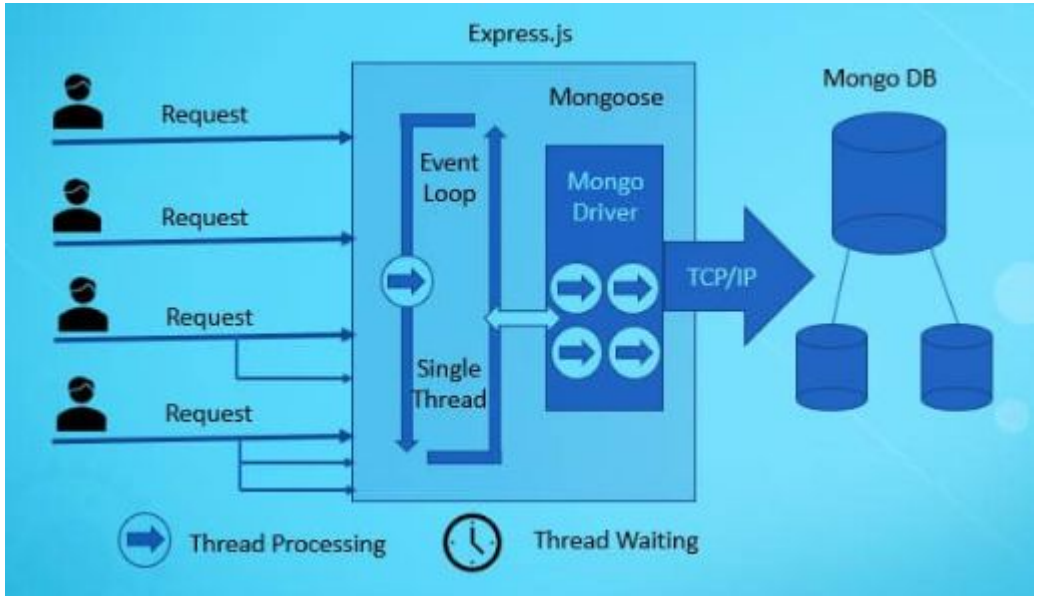
API Rest (NodeJS, Express)

SEMINARI 5

Óscar Boullosa
Itziar Mensa
Arnau Millán
Adrián Quirós
Maria Ubiergo

Stack MEAN

Stack MEAN



- ➔ **Request**
- ➔ **Response**

Stack MEAN

Stack MEAN

- **MongoDB** - NoSQL (non-relational) database that uses **JSON documents to store data**.
- **Express** - web application **framework for Node.js** that provides an abstraction over the HTTP protocol.
- **Angular** - frontend framework developed by Google that enables the **creation** of complex and scalable **web applications**.
- **Node.js** - JavaScript runtime **environment on the server**.

What is an API Rest?

What is an API Rest?

➔ API - **Application Programming Interface**

Is a way for software applications to communicate with each other.

➔ RESTful API - **Representational State Transfer**

Is designed to use the HTTP protocol and its methods (GET, POST, PUT, DELETE) to perform CRUD (Create, Read, Update, Delete) operations on resources.

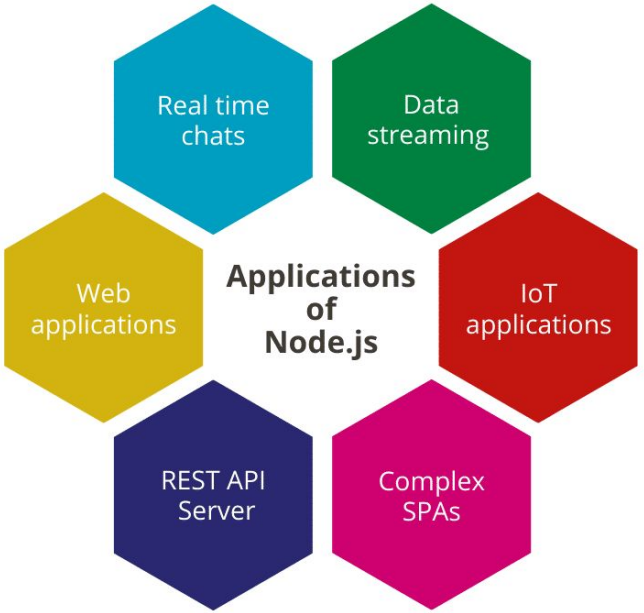
A REST API defines a set of resources, which represent information or actions available via the web, and a set of CRUD (Create, Read, Update, and Delete) operations that allow clients to interact with these resources.

These resources are identified through a unique URL and can be accessed and manipulated through HTTP methods (GET, POST, PUT, DELETE, etc.).

What is NodeJS?

- Is a JavaScript runtime built on the Chrome V8 engine. It allows developers to run JavaScript on the server-side.

- It is cross-platform, meaning it can run on various operating systems such as Windows, macOS, and Linux.



Installation

Global installation:

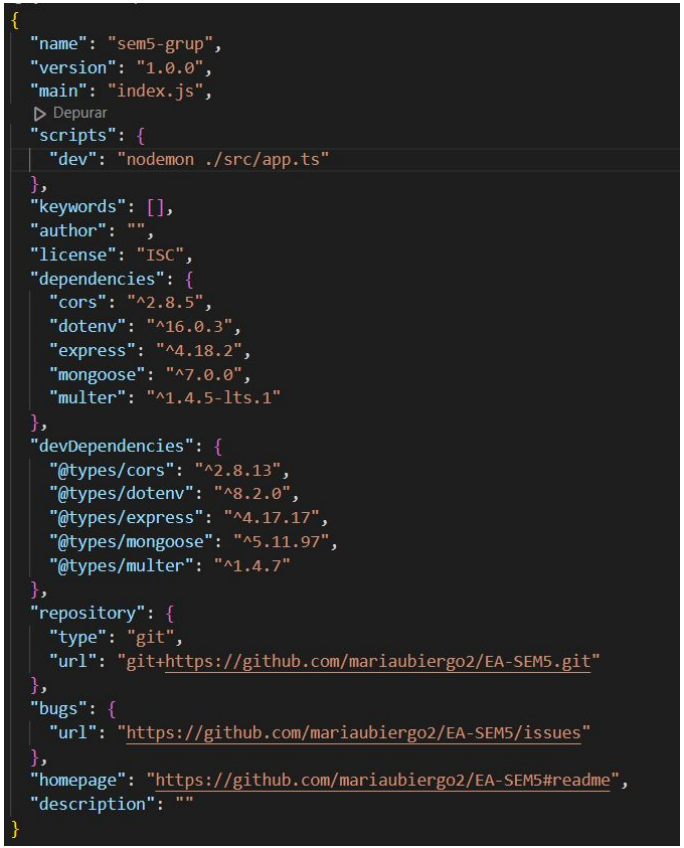
- npm typescript -g
- npm i ts-node -g
- npm i nodemon -g

Project installation:

- tsc --init
- npm init -y
- npm i express cors dotenv multer mongoose
- npm i @types/express @types/cors @types/dotenv @types/multer @types/mongoose -D

Now... Let's see the code!

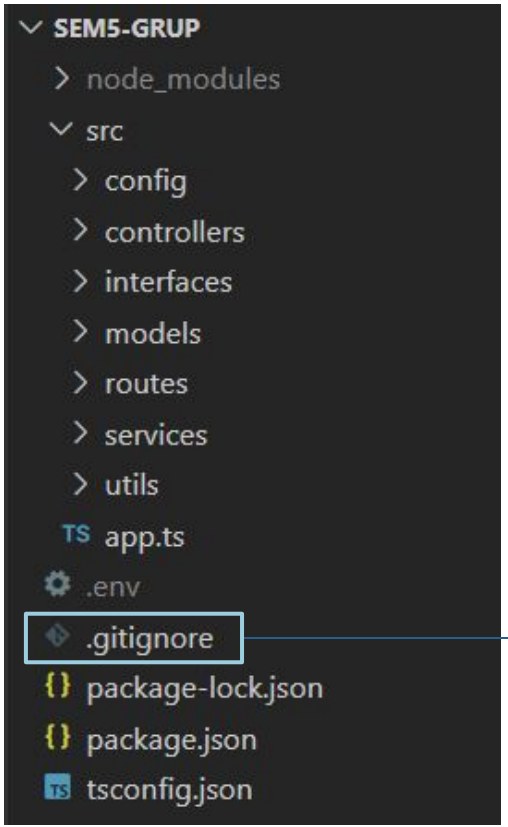
package.json



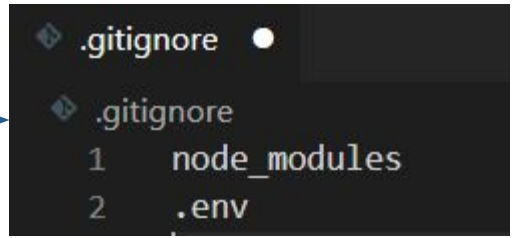
```
{
  "name": "sem5-grup",
  "version": "1.0.0",
  "main": "index.js",
  ▷ Depurar
  "scripts": {
    "dev": "nodemon ./src/app.ts"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "mongoose": "^7.0.0",
    "multer": "^1.4.5-lts.1"
  },
  "devDependencies": {
    "@types/cors": "^2.8.13",
    "@types/dotenv": "^8.2.0",
    "@types/express": "^4.17.17",
    "@types/mongoose": "^5.11.97",
    "@types/multer": "^1.4.7"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/mariaubiergo2/EA-SEM5.git"
  },
  "bugs": {
    "url": "https://github.com/mariaubiergo2/EA-SEM5/issues"
  },
  "homepage": "https://github.com/mariaubiergo2/EA-SEM5#readme",
  "description": ""
}
```

The caret (^) symbol is known as the caret operator and is used **to specify the version of a dependency** that will be installed. When the caret operator is used with a version, it indicates that a version compatible with the specified version is required, but it also allows for automatic minor updates.

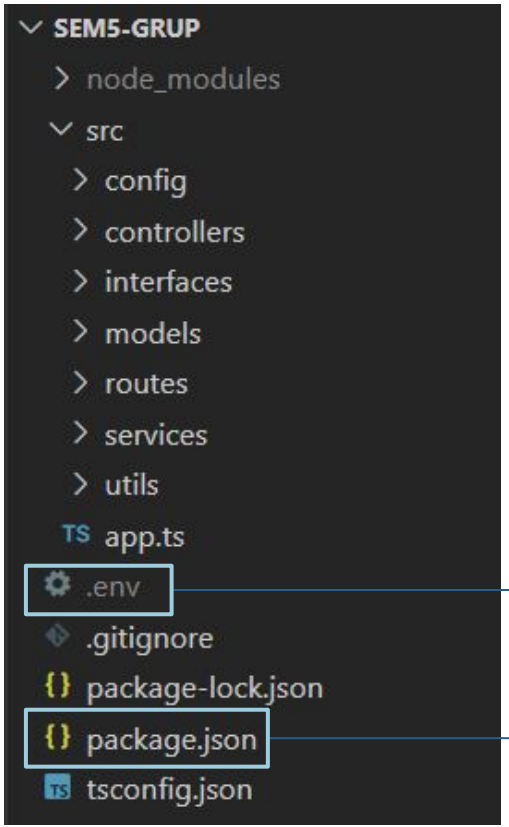
Structure of the API Rest



gitignore:



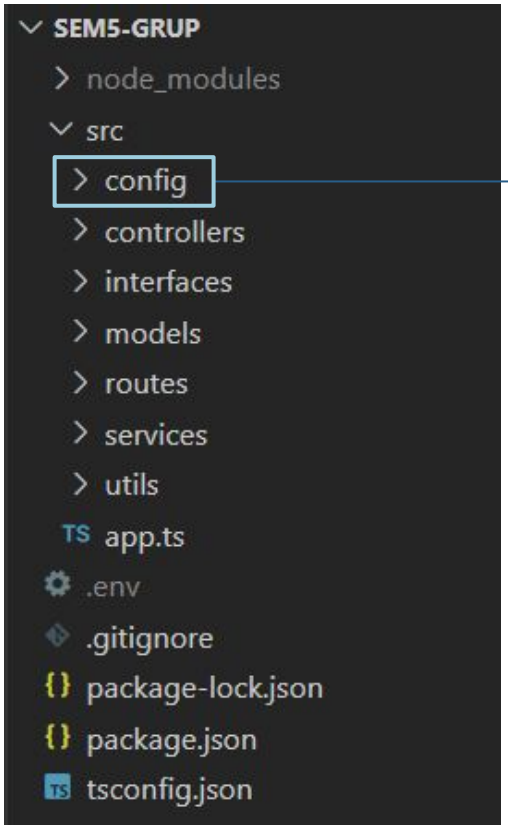
Structure of the API Rest



Variables: **PORT**
DB_URI (for MongoDB connection)

Dependencies

Structure of the API Rest



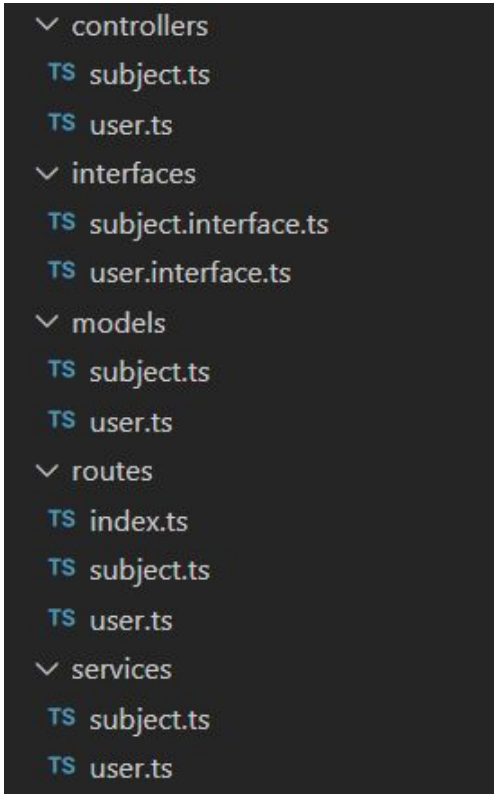
Configuration:

mongo.ts M

src > config > mongo.ts > ...

```ts
import "dotenv/config"; //Implements variables de entorno
import {connect} from "mongoose";

// To connect to MongoDB
async function dbConnect(): Promise<void>{
    const DB_URI=<string>process.env.DB_URI;
    await connect(DB_URI);
}

//Connection exported ready to be used
export default dbConnect;
```

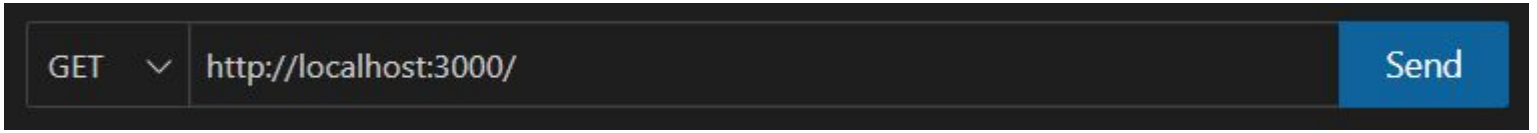
Structure of the API Rest



controllers	Uses the /services to do the requests. It contains the try-catch and throws errors.
interfaces	Definition of the interfaces, the structure of the models.
models	Using the interfaces, definition of the schemas. Defines the collections and the kind of schema it fed them.
routes	States the relation between each route and the action (/controllers) must be taken
services	In charge to connect with the DB

Structure of the API Rest

THUNDER CLIENT



/routes

```
13   router.get("/", getUsers);
14   router.get("/:id", getUser);
15   router.post("/", postUser);
16   router.put("/:id", updateUser);
17   router.delete("/:id", deleteUser);
```

/controllers

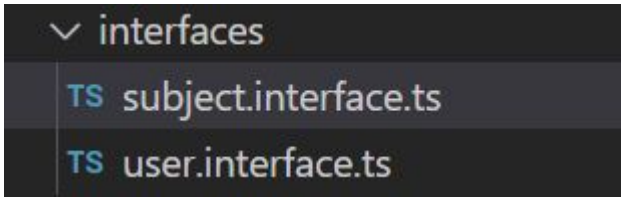
```
5   const getUser = async ({params}: Request, res: Response) => {
6       try{
7           const {id} = params;
8           const response = await get_User(id);
9           const data = response ? response : "NOT_FOUND";
10          res.send(data);
11      } catch (e) {
12          handleHttp(res, 'ERROR_GET_USER');
13      }
14  }
```

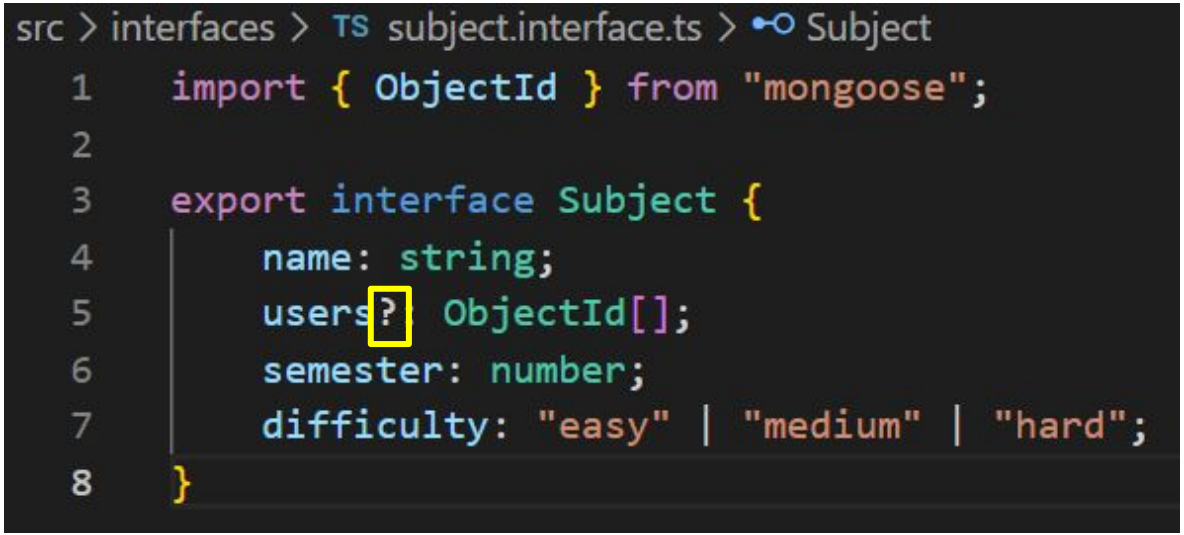
/services

```
16  const get_User = async (id: String) => {
17      const responseUser = await UserModel.findOne({_id:id})
18      return responseUser;
19  };
```

Interfaces

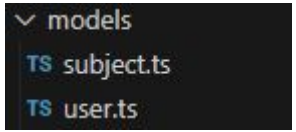
```
export interface User {
    name: string;
    surname: string;
    email: string;
    password: string;
}
```




src > interfaces > TS subject.interface.ts > Subject
1 import { ObjectId } from "mongoose";
2
3 export interface Subject {
4 name: string;
5 users?: ObjectId[];
6 semester: number;
7 difficulty: "easy" | "medium" | "hard";
8 }

Models

```ts
import {  Schema, Types, model, Model } from "mongoose";
import { User } from "../interfaces/user.interface";

const UserSchema = new Schema<User>(
    {
        name:{
            type: String,
            required:true,
        },
        surname:{
            type: String,
            required:true,
        },
        email:{
            type: String,
            required:true,
        },
        password:{
            type: String,
            required:true,
        },
    },
    {
        timestamps: true,
        versionKey: false,
    }
);

const UserModel = model('users', UserSchema);

export default UserModel;
```



```
import {  Schema, Types, model, Model } from "mongoose";
import { Subject } from "../interfaces/subject.interface";

const SubjectSchema = new Schema<Subject>(
    {
        name:{
            type: String,
            required:true,
        },
        users:{
            type: [Schema.Types.ObjectId],
            ref:'users',
        },
        semester:{
            type: Number,
            required:true,
        },
        difficulty:{
            type: String,
            enum: ["easy", "medium", "hard"],
            required:true,
        },
    },
    {
        timestamps: true,
        versionKey: false,
    }
);

const SubjectModel = model('subjects', SubjectSchema);

export default SubjectModel;
```

Services

```
import { Types } from "mongoose";
import { Subject } from "../interfaces/subject.interface";
import SubjectModel from "../models/subject";

const insertSubject=async(item:Subject)=>{
    const responseInsert=await SubjectModel.create(item);
    return responseInsert;
};

const getSubjects=async()=>{
    const responseItem=await SubjectModel.find({});
    return responseItem;
};

const getSubject=async(id:string)=>{
    const responseItem=await SubjectModel.findOne({_id:id});
    return responseItem;
};

.
.
.

export { insertSubject, getSubject, getSubjects, updateSubject, deleteSubject, matriculateSubject
```

Controller

src > controllers > TS subject.ts > ...

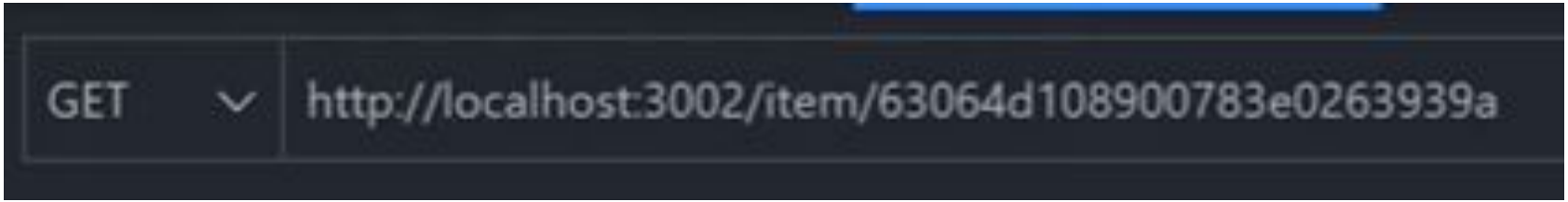
```ts
import { Request,Response } from "express";

const get_Subject = async({params}:Request,res:Response)=>{
    try{
        const {idSubject} = params;
        const response = await getSubject(idSubject);
        const data = response ? response:"NOT_FOUND";
        res.send(data);
    } catch(e){
        handleHttp(res,"ERROR_GET_SUBJECT");
    }
};

const get_Subjects = async (req:Request,res:Response) => {
};

const update_Subject = async ({params,body}:Request,res:Response)=>{
};

const post_Subject = async ({body}:Request,res:Response)=>{
};

const delete_Subject = async ({params}:Request,res:Response)=>{
};

const matriculate_Subject = async ({body}:Request,res:Response)=>{
};

export{get_Subject,get_Subjects,post_Subject,update_Subject,delete_Subject,matriculate_Subject};
```

Routes

src > routes > TS subject.ts > ...

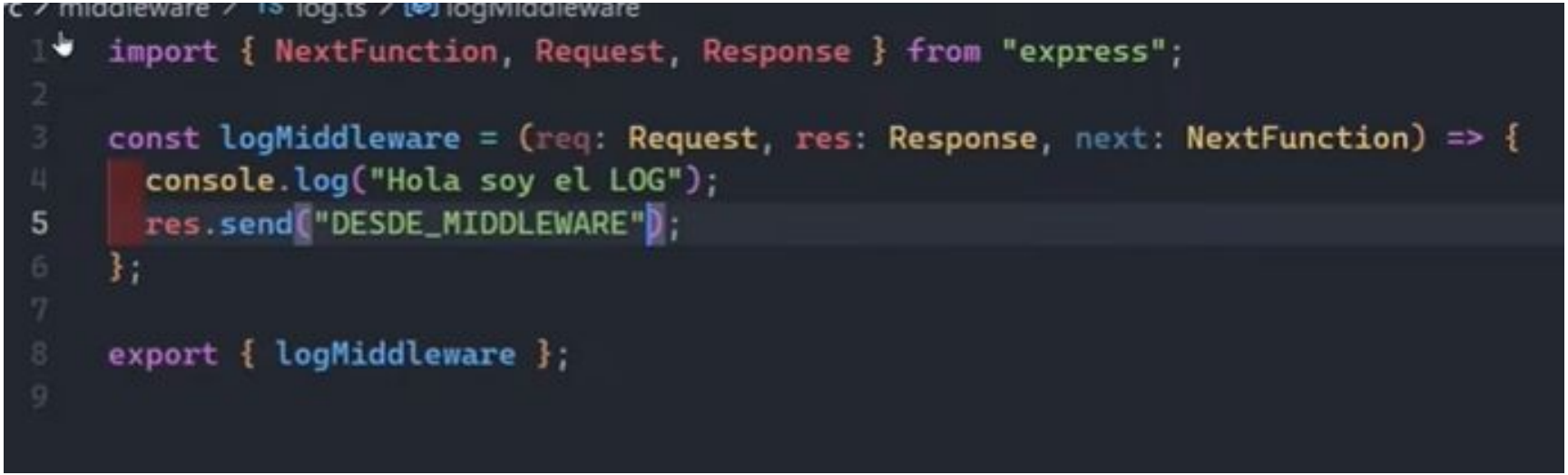
```ts
import { Request, Response, Router } from "express";

const router=Router();

router.get("/all",get_Subjects);
router.get("/:idSubject",get_Subject);
router.post("/",post_Subject);
router.put("/:idSubject",update_Subject);
router.delete("/:idSubject",delete_Subject);
router.post("/matriculate",matriculate_Subject);

export{router};
```

¿Middleware?

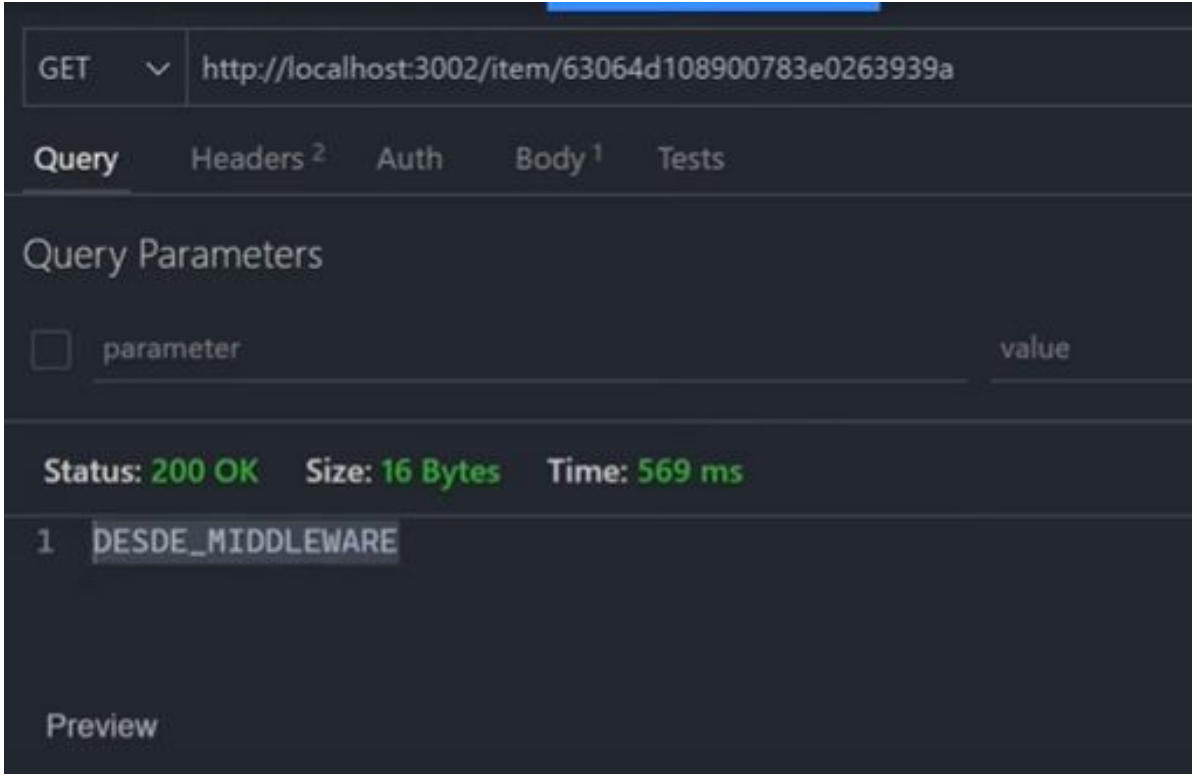



```
router.get("/:id", logMiddleware, getItem);
```

¿Middleware?


import { NextFunction, Request, Response } from "express";
const logMiddleware = (req: Request, res: Response, next: NextFunction) => {
console.log("Hola soy el LOG");
res.send("DESDE_MIDDLEWARE");
};
export { logMiddleware };

Middleware


GET
http://localhost:3002/item/63064d108900783e0263939a
Query
Headers 2
Auth
Body 1
Tests
Query Parameters
parameter
value
Status: 200 OK
Size: 16 Bytes
Time: 569 ms
1 DESDE_MIDDLEWARE
Preview

Middleware

ROUTE

MIDDLEWARE

CONTROLLER

Exercise

- Display a list of subject to which a user is matriculated.
- Display a list of users matriculated in a specific subject.
- [***OPTIONAL***] Search for users with the surname “Garcia” who are matriculated in the subjects of the second semester.