POF - Planty of Food

REST API for manage purchasing group

Why

This project is the final practice for start2impact Node.js course.

How it works

This API has three data Schemas:

- Users: data of the subscripted users
- Products: data of the products
- Orders: data on which users and which products they have inserted in the specific order

Using the correct endpoints you can Create, Read, Update ore Delete (CRUD) what do you want. Finally, you can search through the intervals with some queries. For this project where used Node.js (with Express framework) and MongoDB database.

Libraries

- Express
- Mongoose
- dotenv (loads environment variables from a .env file into process.env)
- Morgan (Morgan is another HTTP request logger middleware for Node. js.)
- Express-mongo-sanitize (middleware which sanitizes user-supplied data to prevent MongoDB Operator Injection.)
- Express-rate-limit (Basic IP rate-limiting middleware for Express. Use to limit repeated requests to public APIs and/or endpoints such as password reset.)
- Helmet (Helmet helps secure Express apps by setting HTTP response headers.)
- Hpp (Express middleware to protect against HTTP Parameter Pollution attacks)
- Validator (A library of string validators and sanitizers. For strings only .)
- Xss-clean (Data sanitazation agains XSS Deprecate but working)

Installation

First of all, you need Node.js installed.
If you don't have it, you can download it here: Node.js
After the installation, you're ready to go.

Try the endpoint without clone

Try the endopoint here : `https://s2i-pof.onrender.com` !

1 - Clone the repository

`git clone https://github.com/boobaGreen/S2INodeJsPOF`

2 - Install the dependencies

`npm install`

3 - Start it

add this scripts at your "package.json" file :

```
"scripts": {
    "start": "SET NODE_ENV=development&&nodemon server.js",
    "start:prod": "SET NODE_ENV=production&&nodemon server.js",
    "debug": "ndb server.js"
  },
```

`npm start` - start in DEV mode default (error's message are set for developers) `npm start:prod` - start in PROD mode (error's messages are set for clients)

4 - Connect your MongoDB database

Create, if don't exist, a `config.env` file and the insert an enviroment variable named `DB` with your MongoDB connection string. Set a `PORT` , in the example is "3000" , if not set the default is 5000. There is a configFAKE.env file as an example and as a possible template.

EXAMPLE "config.env" :

```
PORT=3000
DB=mongodb://localhost:27017/NodePOFLocal
```

in the example above it is a local MongoDb database but you can also connect one in the cloud , the string will be approximately :

```
mongodb+srv://claudiodallara77:cA3K32LqwPsvsqWs@cluster0.3dfdb4w.mongodb.net/POF-S2I-NodeJs?retryWrites=true&w=majority
```

5 - Extra - Import already packaged data for Testing

The dev-data folder contains sample data to help test your project: `orders.json products.json` , and `orders.json` . There is also a file utility `import-dev-data.js` that is used to import sample files with one click or delete the entire database.

from the terminal in the main project folder launch the following command to import the files:

```
node dev-data\data\import-dev-data.js --import
```

to delete use instead:

```
node dev-data\data\import-dev-data.js --delete
```

6 - Test it with a client

Using something like Postman, you can start using this API on the the same port (in the example : 3000).

For convenience, with the button below you can access the entire collection relating to the project on postman with all the queries and documentation.

Run in Postman

Endpoints

I used the envoiroments variable {{URL}} which in DEV mode can be replaced directly by 127.0.0.1:3000 (or instead of 3000 the port you are using), and in the PROD phase it can be replaced by the domain real.

For the deploy site use the real name , in my case : `https://s2i-pof.onrender.com`

Users

You can get the entire users list with a GET request:

`/api/v1/users/`

or GET data for a specific user:

`/api/v1/users/:userID`

:userID must be a valid MongoDB id.
You can PATCH or DELETE user data with the same endpoint.

Finally, you can add a new user with a POST request:

`/api/v1/users/`

```
{
  "name": "insert an alphanumeric string (apostrophe exception), min 2 characters
, max 40",
  "surname": "insert an alphanumeric string (apostrophe exception), min 2
characters , max 40",
  "email": "insert a valid email"
}
```

Products

You can get the entire targets list with a GET request:

`/api/v1/products`

or GET data for a specific target:

`/api/v1/products/:productsID`

:productsID must be a valid MongoDB id.
You can PATCH or DELETE a target with the same endpoint.

Finally, you can add a new target with a POST request:

`/api/v1/products`

```
{
  "name": "an alphanumeric string (`-` exception) , min 3 characters, max 40"
}
```

Orders

You can get all the available orders with a GET request

`/api/v1/orders`

You can get all the order for one specific day (in this example 2023-10-27) with this GET request .

`/api/v1/orders/?createdAt[lte]=2023-10-27T23:59:59.999Z&createdAt[gte]=2023-10-27T00:00:00.000Z`

You can get all the order after a specific day&time (in this example 2023-10-27 , 20:30) with this GET request .

`/api/v1/orders/?createdAt[gte]=2023-10-27T11:58:52.255Z`

You can get all the order before a specific day&time (in this example 2023-10-27 , 11:58:52) with this GET request .

`/api/v1/orders/?createdAt[lte]=2023-10-27T11:58:52.000Z`

You can get all the orders that contain a specific product or a specific list of products by they "name" field , separate by a comma:

`/api/v1/orders/getOrdersByProductName?productNames=dattero,miele`

or GET data for a specific product by id:

`/api/v1/:ordersID`

You can PATCH or DELETE a target with the same endpoint.

For a new order, use a POST request:

`/api/v1/orders`

```
{
  "buyers": ["653afc5f684ac741822e6124", "653b7fd7d6b437985a02a501"],
  "products": ["653b7e3be0406350e6b284ce", "653b7e2fe0406350e6b284ca"]
}
```

the `"createdAt"` field will then be created which will be used for the filters and display order according to the main requirements

🔎 API feautures

In all the GET function you can set the fields you want to project in the response object or to not include("-", minus sign before the name without space)

EXAMPLE (exclude the "__v" field from this response): `/api/v1/products/?page=2&limit=3&fields=-v`

In all GetAllOrders,GetAllUsers and GetAllProducts you can set a `limit` and the page selected for every query . EXAMPLE (set : page 2 , limit 3): `/api/v1/products/?page=2&limit=3`.

You can `sort` the response of the GetAll functions : EXAMPLE (sort ascending by name): `/api/v1/products/?sort=name`.

For the Orders- "get all" the default orders is "-createdAt" (decrescent first, the newst before) , for switch use "sort=createdAt" (no minus sign)

📐 Database Architecture

The Product collection is very simple with only the "name" field. I only set the validation rules (see Endpoints section)

The User collection is very simple with the "name","surname" and the "email" fields. I only set the validation rules (see Endpoints section)

For "Order collection" however, different choices could already be made. Each order will have a list of related products and a list of buyers. Given the nature of small and widespread purchasing groups for example from small direct growers, I thought that the lists would never be too long (one - few). I therefore decided to register an array containing the IDs of the "buyers" and an array with the IDs of the products. I therefore decided that in the get phase the other details are also populated which are therefore virtual fields of both the product and the users

📄 License

MIT

📧 Contact Me

Any questions? Send me an e-mail here: claudiodallara77@gmail.com
You can find my Linkedin profile here: https://www.linkedin.com/in/claudio-dall-ara-244816175/