SofGuzman-Mx / **Non-Relational-Databases-for-Storing-JSON-Data**

Code Issues Pull requests Actions Projects Wiki Security Insights Set

This project aims to transform the existing platform into a dynamic and personalized experience for users. By leveraging a non-relational database (MongoDB) and a REST API (Express.js), Tattler will provide up-to-date information, personalized recommendations, and allow users to interact by adding comments, ratings, and new restaurants.

0 stars 0 forks 0 watching Branches Activity
Tags

Public repository

1 Branch 0 Tags Go to file t Go to file Add file Code

SofGuzman-Mx add image last push in GitBash d743da6 · 21 minutes ago

TattlerBackend	feat: Implement search and sorting...	11 hours ago
images	add image last push in GitBash	21 minutes ago
node_modules	feat: Implement search and sorting...	11 hours ago
README.md	feat: Add PUT method	10 hours ago
Sprint1_3350.zip	Add files Sprint 2	4 days ago
Sprint2_3350.zip	MVC structure	3 days ago
Sprint3_3350.zip	feature Sprint 3	21 minutes ago
package.json	feat: Implement search and sorting...	11 hours ago

README

Menu

- Sprint 1
- Sprint 2
- Sprint 3

Major Version: 1.1.0

1. Installation and Usage Prerequisites

- Node.js (v18.x or later
- npm (Node Package Manager)
- MongoDB (local instance or a cloud service like MongoDB Atlas)
- Git
- Postman or Insomnia for API testing

Sustainability (Long-Term Maintenance): Repository with version control practices.

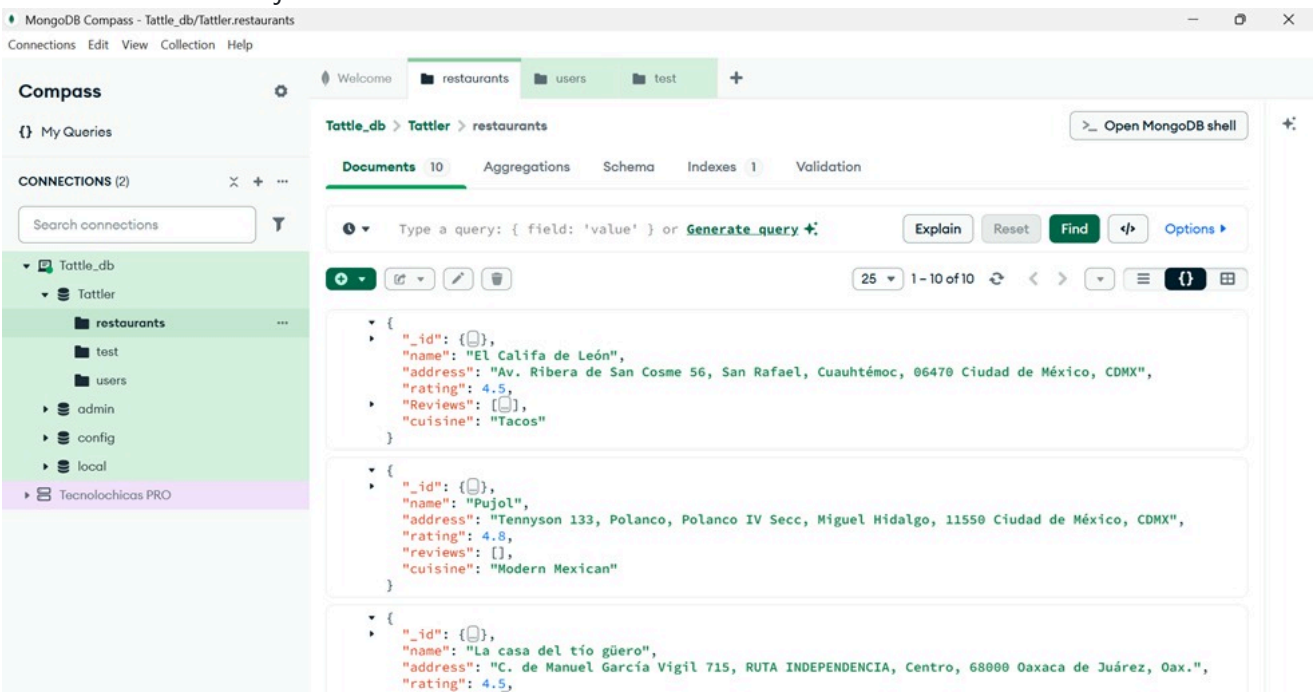
Scalability (Growthability): Solid foundation for scalability by using indexes in MongoDB, ensuring fast queries and Studio 3T optimizes a query, replacing an inefficient Collection Scan with a fast Index Scan.

Quality: ✅ Testing used with the GET, POST, PUT, and DELETE methods to the new search and sort features with screenshots in Postman as evidence of your quality assurance (QA) process.

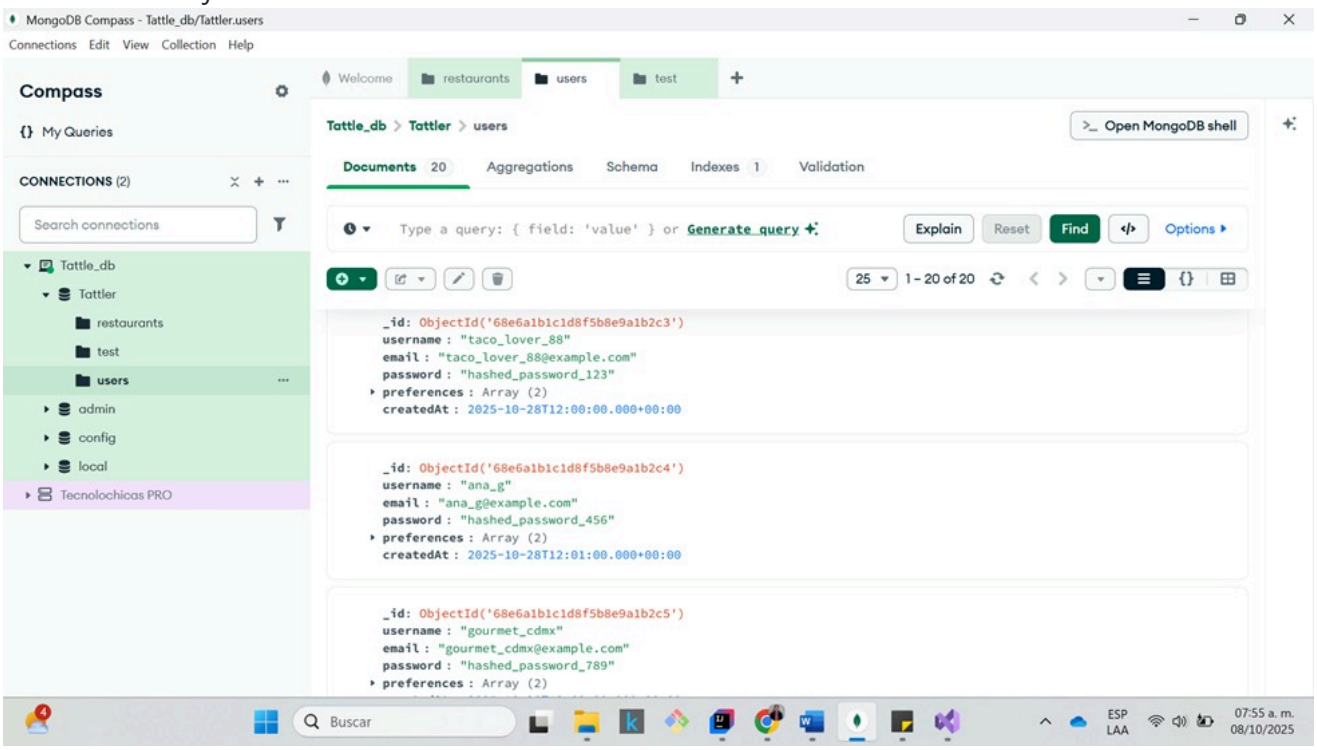
Sprint 1

Data base Setup in **Mongo DB Compass**

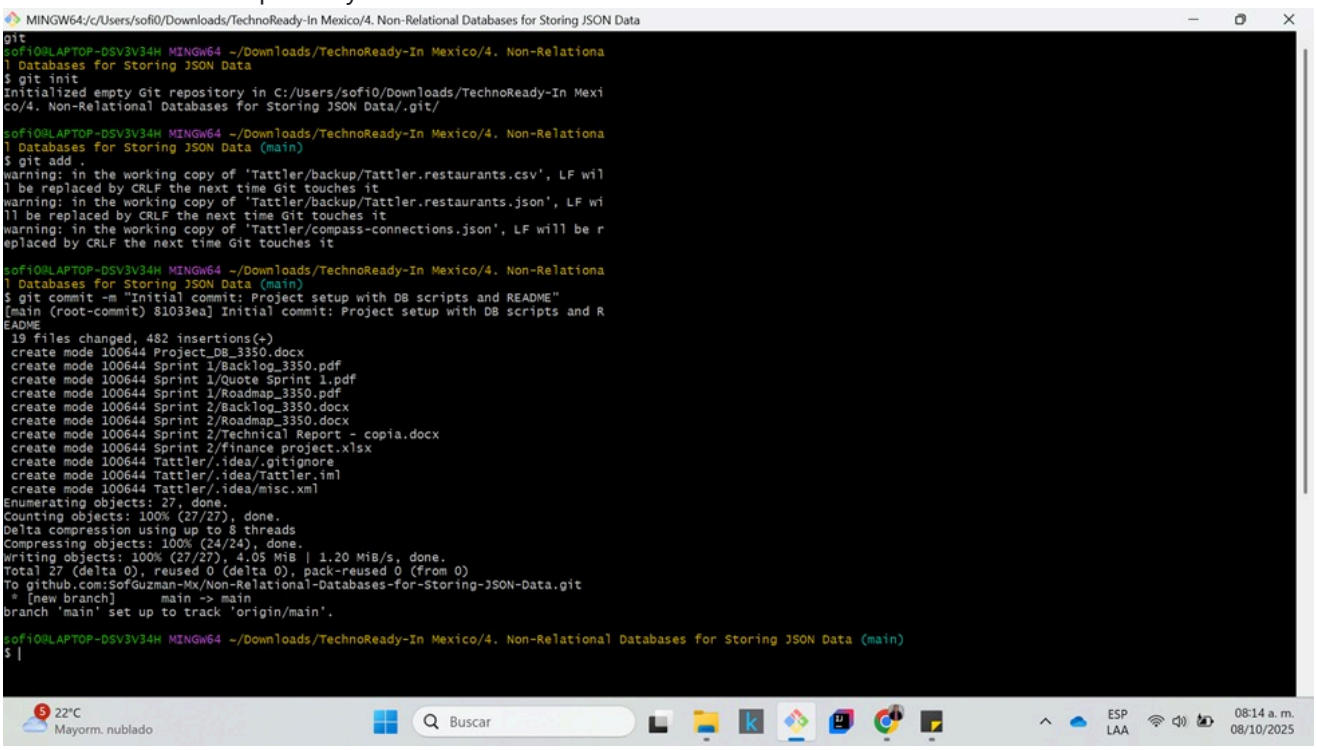
Collection restaurants.json



Collection users.json



first version of the repository



2. Clone the repository in Git:

```
git clone <git@github.com:SofGuzman-Mx/Non-Relational-Databases-for-Storing-JSON-Data.git>
```

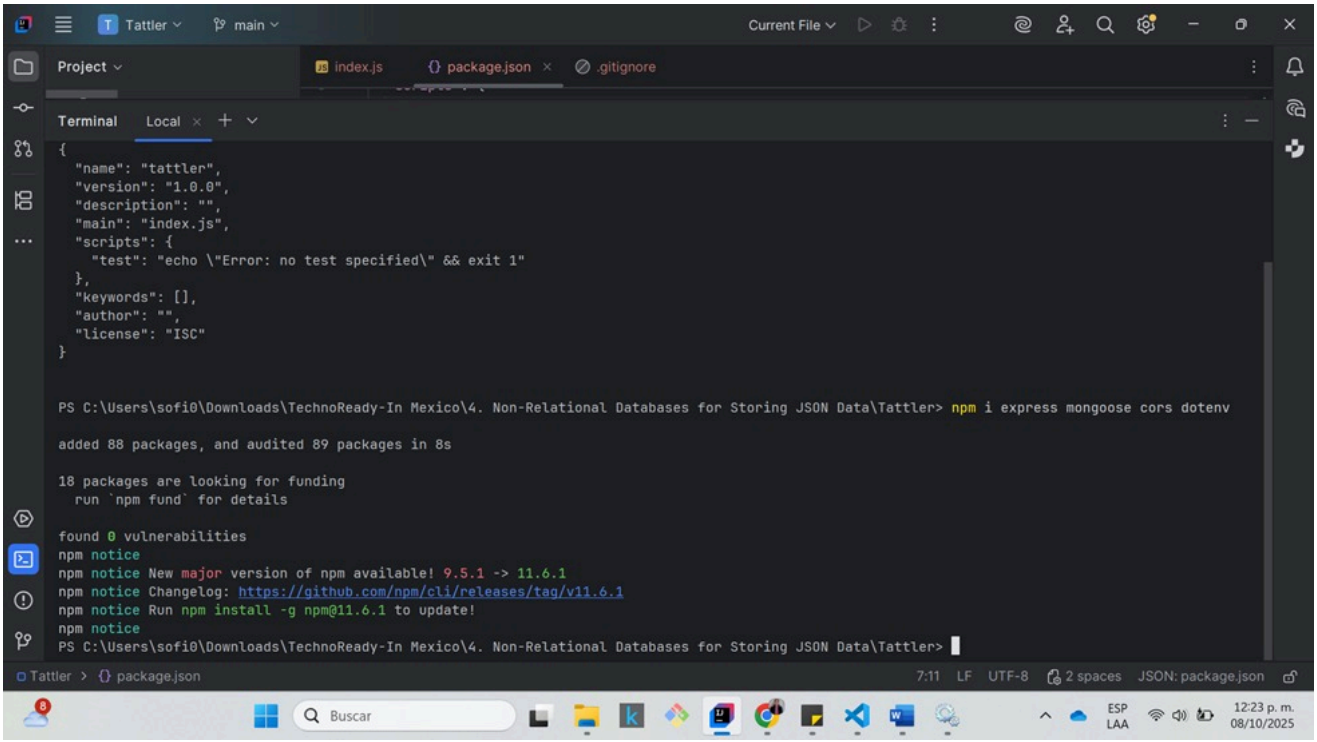
```
cd <Non-Relational-Databases-for-Storing-JSON-Data>
```

Sprint 2

Install dependencies

- moose
- express
- node.js

```
npm install
```



The import scripts require the mongodb and csv-parser packages

Set up environment variables Create a file named `.env` in the root of your project and add your MongoDB connection string.

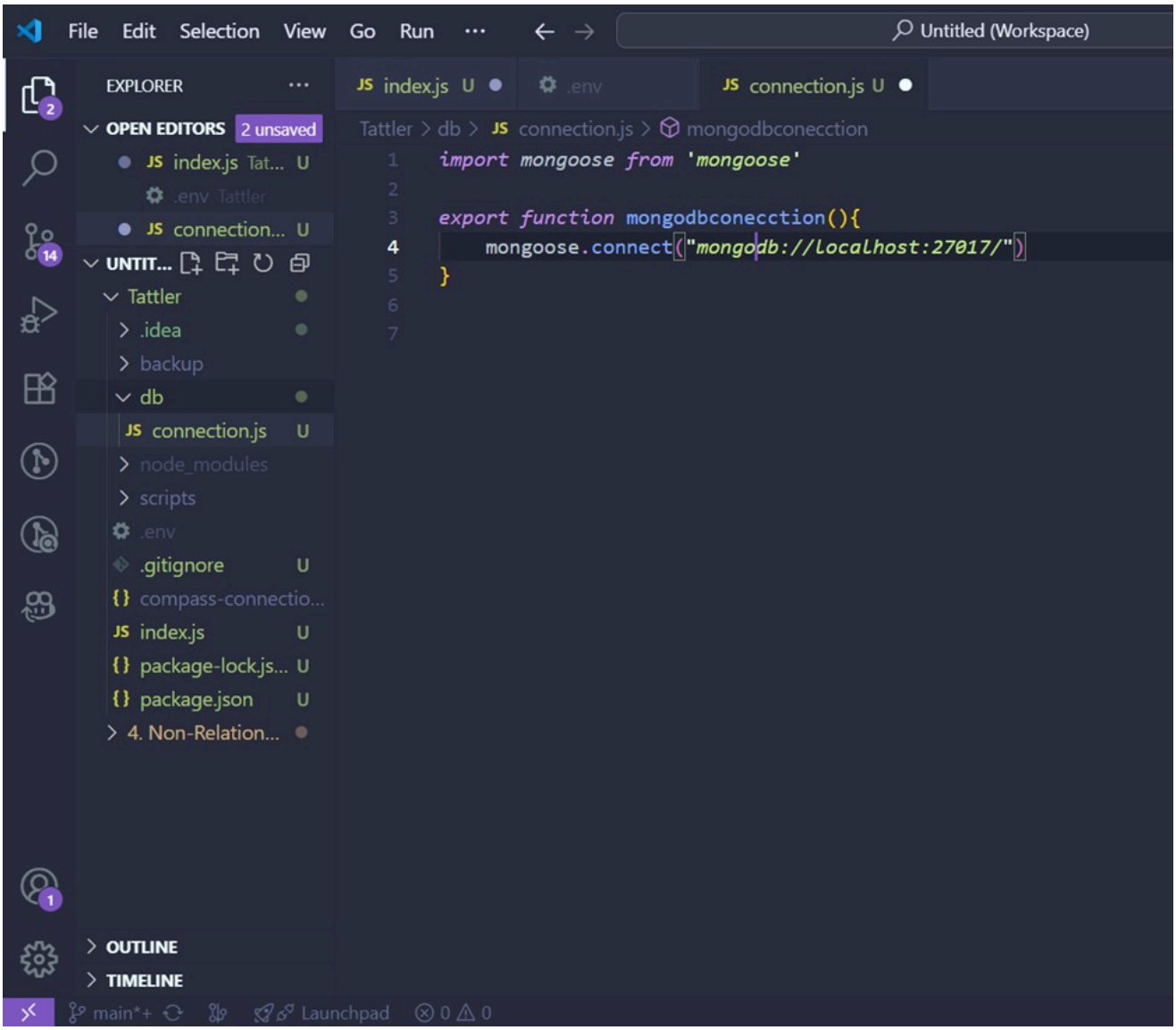
```
MONGO_URI="mongodb://localhost:27017/tattler_db"
```

Replace the MONGO_URI with your actual connection string from Studio 3T or MongoDB Atlas.

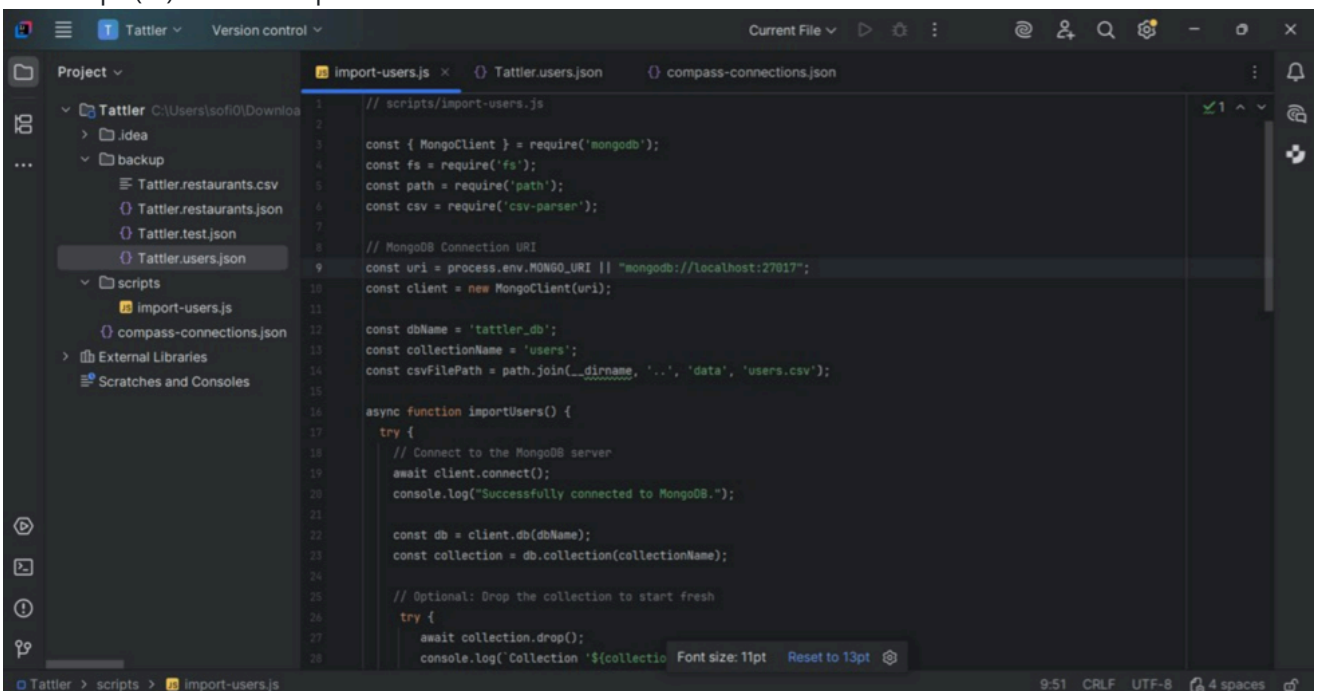
Running the Application To star the server, run the following command:

```
npm start
```

The API will be available at `http://localhost:3000`



JavaScript (JS) code to import users



Database Setup & Data Import You can populate the database in two ways:

A) Restoring from Backup:

Use the mongorestore command-line tool to restore the database from the provided backup files located in the db_backup/ directory.

```
mongorestore --uri="mongodb://localhost:27017" --db tattler_db db_backup/tattler_db
```

B) Using the Import Scripts (Recommended for CSV data):

Place your restaurants.csv and users.csv files in the data/ directory. Then, run the import scripts:

Import restaurants

```
node scripts/import-restaurants.js
```

Import users

```
node scripts/import-users.js
```

Import test

```
node scripts/import-test.js
```

These scripts will connect to your MongoDB instance, create the tattler_db database, and populate the restaurants and users collections.

Sprint 3

API Endpoints The primary endpoint with the new features is:

`GET /api/products`

Query Parameters

Guide: Testing the API with Postman / Insomnia

This guide will walk you through testing the new search, filter, and sort features of the `/api/products` endpoint.

Step 1: Add Sample Data Before you begin testing, make sure you have some diverse data in your MongoDB `products` collection. Using Studio 3T, add a few documents with different names, categories, and prices.

Example documents: [{

},

]

text search (find items with the word "mexican food" in the name or description): URL: Evidence:

Step 2: Set Up Your Request

1. Open Postman or Insomnia
2. Create a new GET request
3. Set the request URL to `http://localhost:3000/api/products`

Step 3: Test the features You will test the features by adding parameters to the "Params" or "Query" tab in your API client.

Test 1: Text Search Test 2: Filtering Test 3: Sorting Test 4: Combining All Features

With the new models defined, we can now properly rewrite your controller logic and set up your API routes.

Setting up user registration and login is a crucial step.

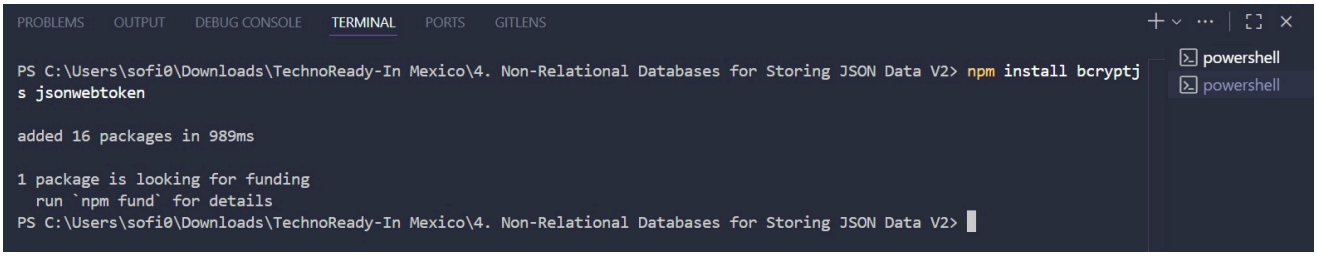
Step 1: Install Required Libraries

- bcryptjs
- jsonwebtoken

Open your terminal in the project folder and run:

```
npm install bcryptjs jsonwebtoken
```

result:



Step 2: Create the User Controller:This file will contain the logic for registering a new user and logging in an existing user.
controllers/userController.js

**Important Security Step: add a `JWT_SECRET` to your `.env` file. This is a secret key used to sign your tokens. Make it long and random.

```
JWT_SECRET="..."
```

```env
PORT=3000
MONGODB_URI="mongodb://localhost:27017/tattler_db"
JWT_SECRET="this-is-a-super-long-and-random-secret-key-for-my-tattler-app"
```

Step 3: Create the User Routes This file will define the API endpoints (`/register, /login`) and connect them to the controller functions you just created.

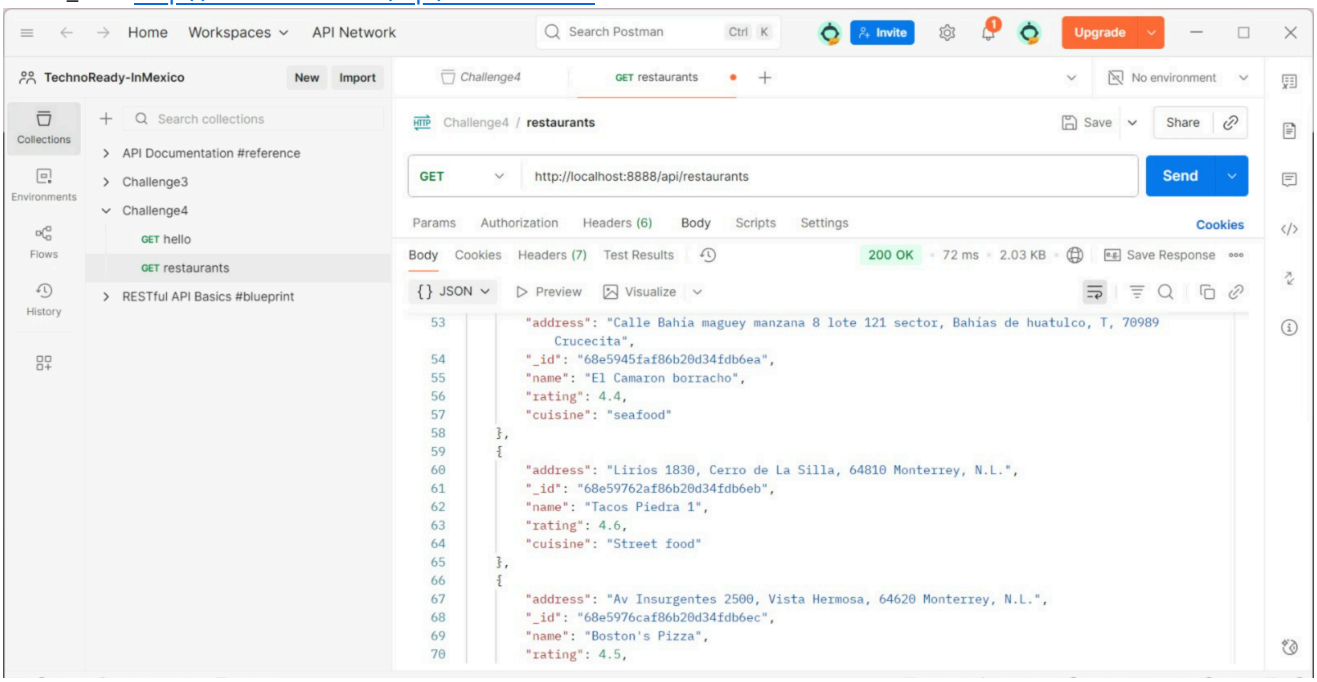
Create a new file: `routes/userRoutes.js`

Step 4:Update the Main Router Finally, let's tell your application to use these new user routes.

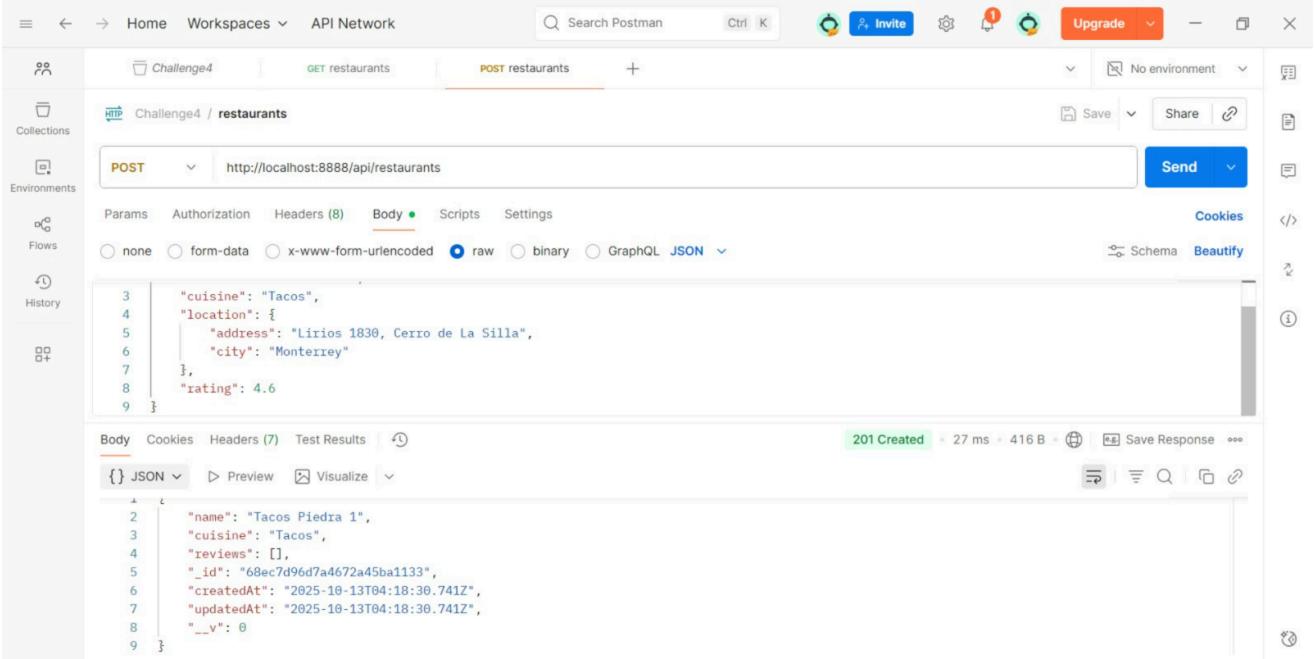
Update your Create a new file: `routes/index.js` file:

We completed our set of endpoints for user authentication! You can use a tool like Postman or Insomnia to test them:

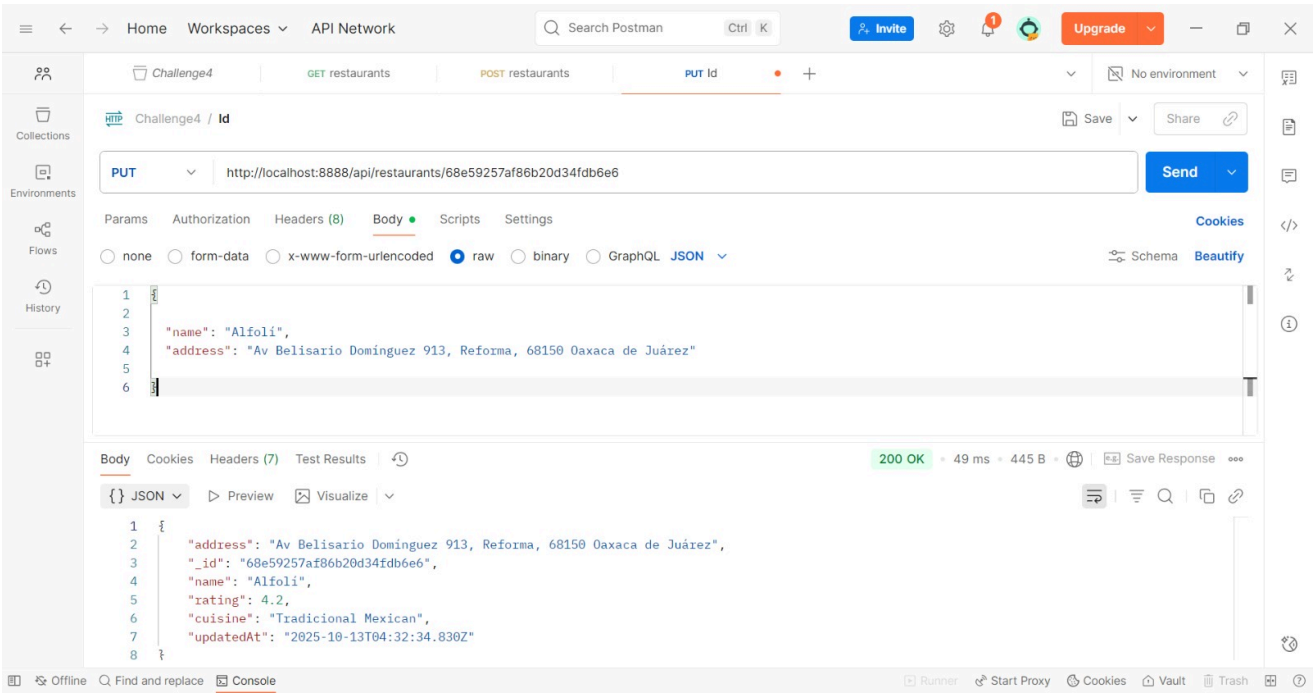
BASE_URL:[http://localhost:8888/api/restaurants`](http://localhost:8888/api/restaurants) **GET**



POST request to `BASE_URL` with a JSON body like:


Challenge4 / restaurants
POST
http://localhost:8888/api/restaurants
Send
Body
raw
JSON
"cuisine": "Tacos",
"location": {
"address": "Lirios 1830, Cerro de La Silla",
"city": "Monterrey"
},
"rating": 4.6
}
201 Created
27 ms
416 B
"name": "Tacos Piedra 1",
"cuisine": "Tacos",
"reviews": [],
"_id": "68ec7d96d7a4672a45ba1133",
"createdAt": "2025-10-13T04:18:30.741Z",
"updatedAt": "2025-10-13T04:18:30.741Z",
"__v": 0
}

PUT

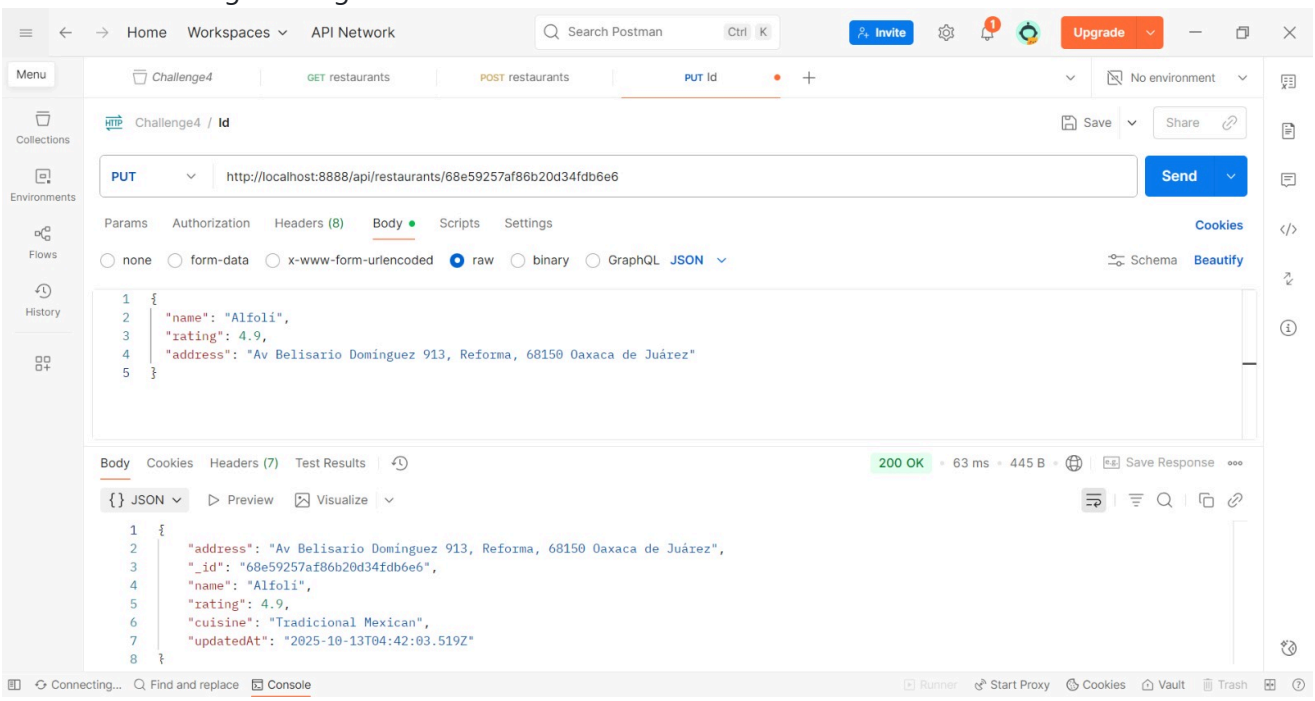

Challenge4 / Id
PUT
http://localhost:8888/api/restaurants/68e59257af86b20d34fdb6e6
Send
Body
raw
JSON
"name": "Alfoli",
"address": "Av Belisario Domínguez 913, Reforma, 68150 Oaxaca de Juárez"
200 OK
49 ms
445 B
"address": "Av Belisario Domínguez 913, Reforma, 68150 Oaxaca de Juárez",
"_id": "68e59257af86b20d34fdb6e6",
"name": "Alfoli",
"rating": 4.2,
"cuisine": "Tradicional Mexican",
"updatedAt": "2025-10-13T04:32:34.830Z"
}

JS Restaurant.js U ×

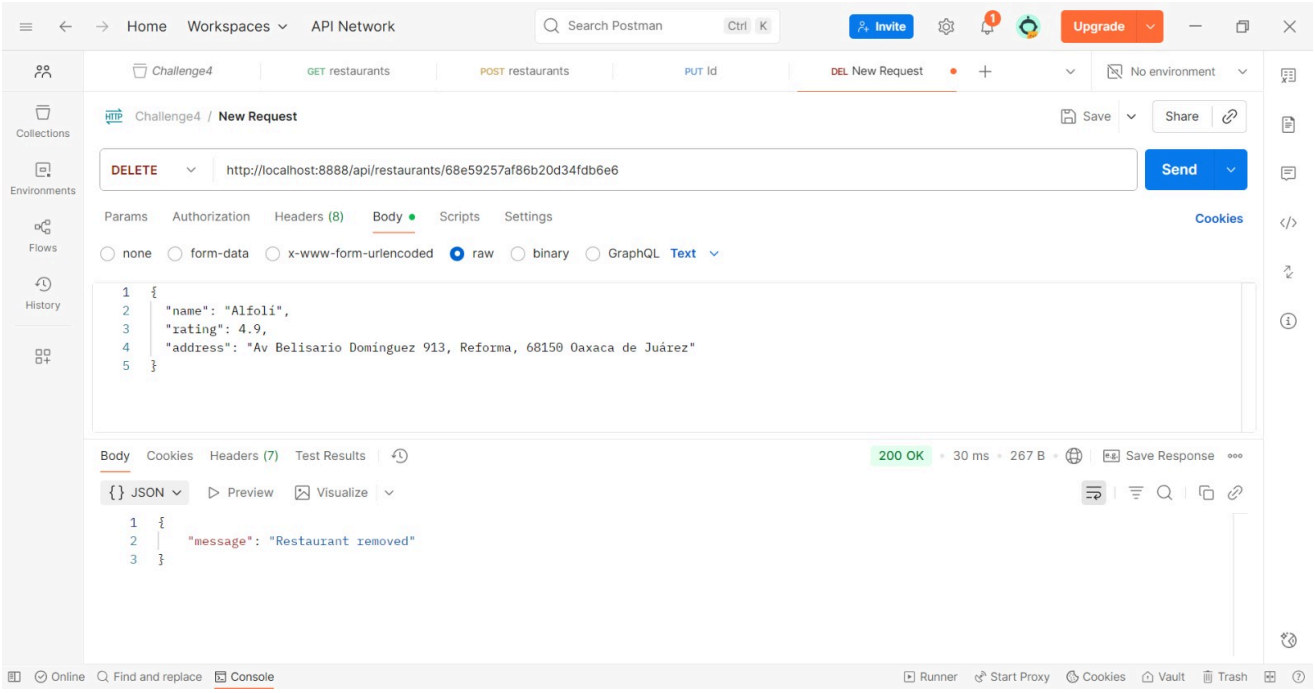
TattlerBackend > models > JS Restaurant.js > restaurantSchema

```
4   const restaurantSchema = new Schema({
10      cuisine: {
12          required: true,
13          trim: true
14      },
15      address: {
16          street: String,
17          city: String,
18          state: String,
19          zipcode: String
20      },
21      rating: {
22          type: Number,
23          default: 0
24      },
25      reviews: [{
26          type: Schema.Types.ObjectId,
27          ref: 'Review'
28      }]
29  }, {
30      timestamps: true
31  });
32
33  export default mongoose.model('Restaurant', restaurantSchema);
```

And now the rating is changed



DELET



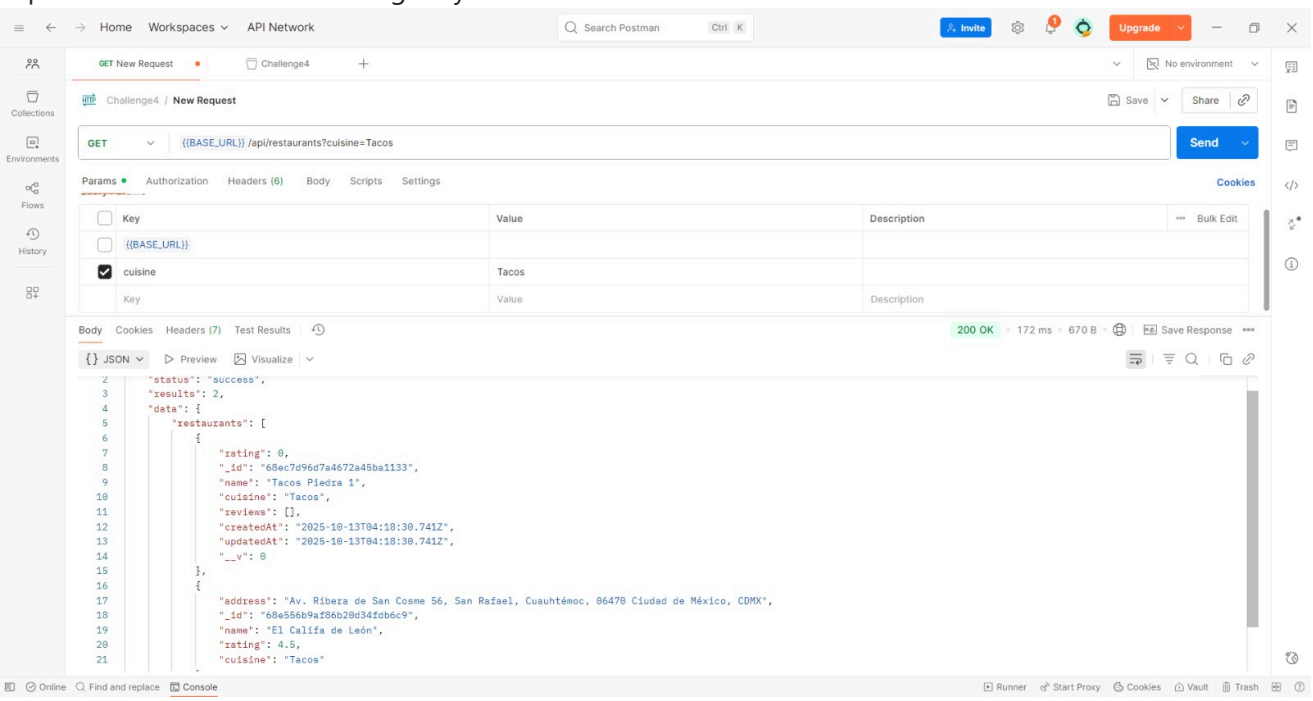
Tests in Postman

Search Test (Filtering)

- **Request 1: Search by cuisine type**

```
GET http://localhost:8888/api/restaurants?cuisine=Tacos
```

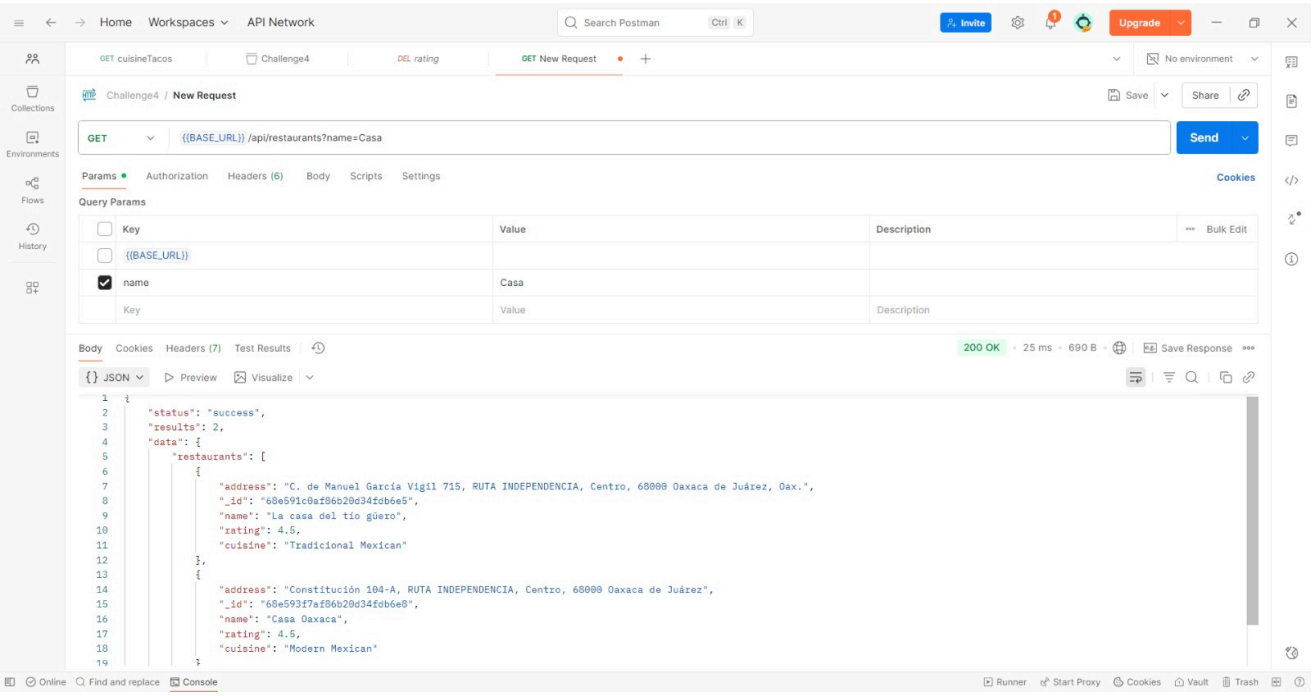
Expected Result: A list containing only restaurants whose field `cuisine` is "Tacos".



- **Request 2: Search by name**

```
GET http://localhost:8888/api/restaurants?name=casa
```

Expected Result: A list of restaurants that contain the word "house" in their name.



- **Request 3: Sort by rating (highest to lowest)**

```
GET http://localhost:8888/api/restaurants?sort=-rating
```

Expected Result: The full list of restaurants, sorted with highest ratings first.

Challenge4 / **nameRestaurant Copy**

GET {{BASE_URL}} /api/restaurants?sort=rating

Params • Authorization Headers (6) Body Scripts Settings

Query Params

Key	Value	Description
{{BASE_URL}}		
sort	rating	
Key	Value	Description

Body Cookies Headers (7) Test Results 200 OK

{} JSON Preview Visualize


```
3     "results": 10,
4     "data": {
5         "restaurants": [
6             {
7                 "rating": 0,
8                 "_id": "68ec7d96d7a4672a45ba1133",
9                 "name": "Tacos Piedra 1",
10                "cuisine": "Tacos",
11                "reviews": [],
12                "createdAt": "2025-10-13T04:18:30.741Z",
13                "updatedAt": "2025-10-13T04:18:30.741Z",
14                "__v": 0
15            },
16            {
17                "address": "Ignacio Allende 109, RUTA INDEPENDENCIA, Centro, 68000 Oaxaca de Juárez",
18                "_id": "68e59372af86b20d34fdb6e7",
19                "name": "Vaca Marina",
20                "rating": 4,
```


- **Request 4: Sort by name (alphabetically)**

```
GET http://localhost:8888/api/restaurants?sort=name
```

Expected Result: The complete list of restaurants, sorted alphabetically by name.

Challenge4 / **sortName**

GET {{BASE_URL}} /api/restaurants?sort=name

Params • Authorization Headers (6) Body Scripts Settings

Body Cookies Headers (7) Test Results 200 OK

{} JSON Preview Visualize


```
2   "status": "success",
3   "results": 10,
4   "data": {
5       "restaurants": [
6           {
7               "address": "C. Pensamientos 314, Reforma, 68050 Oaxaca de Juárez, Oax",
8               "_id": "68e5941daf86b20d34fdb6e9",
9               "name": "314 Punto Natural Reforma",
10              "rating": 4.6,
11              "cuisine": "Organic food"
12          },
13          {
14              "address": "Av Insurgentes 2500, Vista Hermosa, 64620 Monterrey, N.L.",
15              "_id": "68e5976caf86b20d34fdb6ec",
16              "name": "Boston's Pizza",
17              "rating": 4.5,
18              "cuisine": "American food"
19          },
20          {
21              "address": "Constitución 104-A, RUTA INDEPENDENCIA, Centro, 68000 Oaxaca de Juárez",
22              "_id": "68e593f7af86b20d34fdb6e8",
23              "name": "Casa Oaxaca",
24              "rating": 4.5,
25              "cuisine": "Modern Mexican"
26          },
27          {
28              "address": "Av. Ribera de San Cosme 56, San Rafael, Cuauhtémoc, 06470 Ciudad de México, CDMX",
29              "_id": "68e556b9af86b20d34fdb6c9",
```


Studio 3T isn't just for viewing data; it's for demonstrating that you can manage a database with professional tools, perform complex analyses, and optimize queries, which is directly linked to scalability and maintainability (sustainability).

Studio 3T was used to perform complex data analysis and verify the integrity of collections, as shown in the following image:

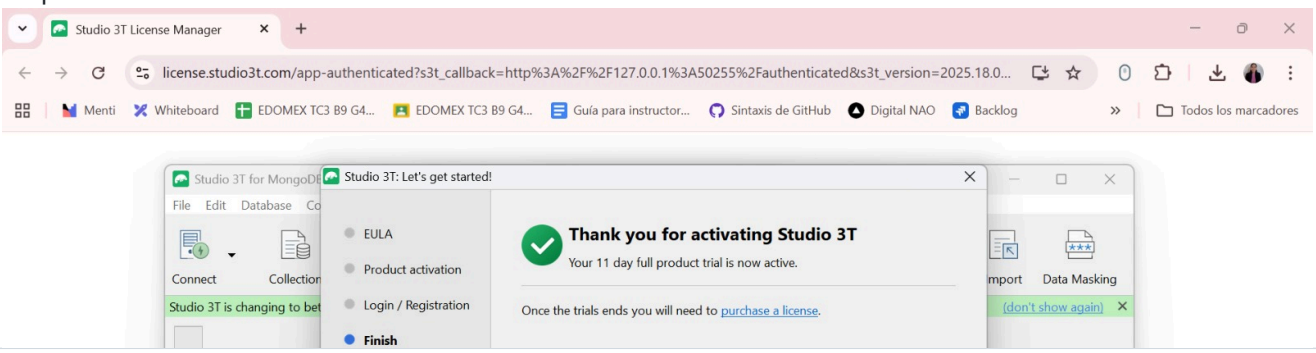
1. Connect to your Local Database

First, you need to tell Studio 3T where your MongoDB database is running.

```
cd TattlerBackend
```

```
node index.js
```

Step 1. Install Studio 3T



Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

- **JavaScript** 100.0%

Suggested workflows

Based on your tech stack

Node.js Configure

Build and test a Node.js project with npm.

Gulp Configure

Build a NodeJS project with npm and gulp.

Datadog Synthetics Configure

Run Datadog Synthetic tests within your GitHub Actions workflow

More workflows

Dismiss suggestions