CS628 Full-Stack Development – Web App

**HOS08A: Node, Express, and Express Router**

Created by Veerendra Jagatha on Aug 17, 2023

Reviewed by Sam Chung on Aug 19, 2023

School of Technology & Computing (STC)

City University of Seattle (CityU)

**Before You Start**

* **Screenshots may be different from your environment.**
* The directory path shown in screenshots may be different from yours.
* Version numbers may not match the most current version at the time of writing. If given the option to choose between the stable release (long-term support) or the most recent, please select the **stable release** rather than the beta-testing version.
* There might be subtle discrepancies along with the steps. Please **use your best judgment** while going through this cookbook-style tutorial to complete each step.
* If you are not familiar with a terminal, command line, and bash scripts, check out this video: <https://youtu.be/Dp7uw9c6QH8>
* All the steps and concepts in this tutorial are from references, so if you encounter problems, please **try to read and compare the references to solve the problem**. If you still can't solve the problem, please contact your course TA.
* **Avoid copy-pasting code from the book or the GitHub repository**. Instead, type out the code yourself. Resort to copy-pasting only when you are stuck and find things not working as expected.
* Some steps may not be explained in detail. If you are not sure what to do:

1. Consult the resources from the course.
2. If you cannot solve the problem after a few tries (usually 15 -30 minutes), ask a TA for help.

#### **Readings and Examples:**

* Visit [CS 628 Repository for Examples](https://github.com/samchung0117/cs628-examples).
  + Select the related module.
  + Visit the README.md file.
  + Find examples for your practices.

**Learning Outcomes**

* Section 1: Accessing GitHub Codespaces.
* Section 2: Setting up the project.
* Section 3: Setting up connection to MongoDB Atlas
* Section 4: Setting up Server API Endpoints
* Section 5: Testing the API Endpoints with Postman
* Section 6: Pushing your work to GitHub.

**Section 1: Accessing GitHub Codespaces**

Refer the steps from [TA Center](https://cityuseattle.github.io/docs/git/github_codepsace/) to get started with this week’s module GitHub Codespace.

**Section 2: Setting up the project**

Node.js is an open-source, server-side runtime environment that allows you to execute JavaScript code on the server. It's designed to enable developers to build scalable and efficient network applications.

Let us now check the node version in the current codespace.

1. Once your Codespace is open, you will have access to an integrated terminal.
2. In the terminal, you can use the following command to check the installed Node.js version:

**>>node -v**

****

In this HOS, we will be setting up Backend for our application. Create a folder named “backend” under the root directory in codespace and navigate to that folder using the following commands.

**>> mkdir backend**

**>> cd backend**

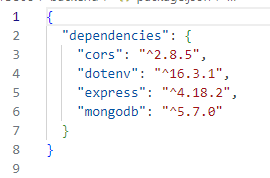
We will also install the required dependencies. Run the following command in the terminal.

**>> npm install mongodb express cors dotenv**

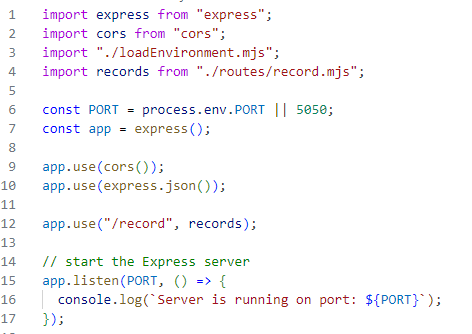
The command above installs the following dependencies:

1. The installation of MongoDB includes the MongoDB database driver, enabling your Node.js applications to establish connections with the database and manage data effectively.
2. When you install 'express', you're adding the Node.js web framework to your environment. Express is a rapid, versatile, and streamlined web application framework tailored for Node.js. It equips developers with a toolkit and capabilities to craft web and API applications.
3. The installation of 'cors' introduces a Node.js package that facilitates cross-origin resource sharing, allowing resources from different origins to be shared securely.
4. By installing 'dotenv', you bring in a module designed to load environment variables from a '.env' file into the 'process.env' file. This separation of configuration details from the code promotes cleaner and more organized development practices.

We can check our installed dependencies using the package.json file.



Now let’s create a file called s”erver.mjs” with the following code,

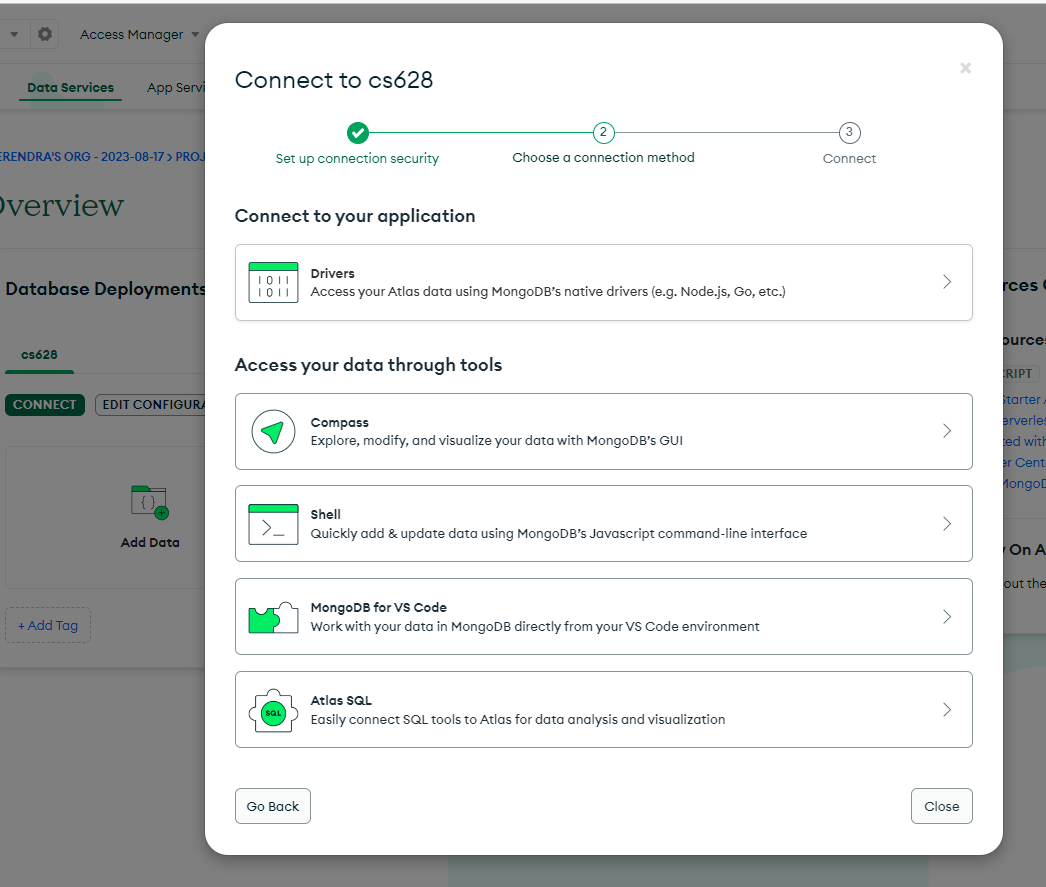


If you're wondering about the meaning of .mjs, it's important to note that both .js and .mjs files are utilized in Node.js to run JavaScript code. Nevertheless, they diverge significantly in how they manage modules. In .js files, Node.js employs the CommonJS module system, requiring the use of require() to import modules and module.exports to export them. Conversely, .mjs files utilize the ES Modules (ESM) system, employing import and export statements for handling modules.

In this code we are importing express and cors. const port process.env.port will access the port variable from the config.env.

**Section 3: Setting up connection to MongoDB Atlas**

In order to set up connection to MongoDB Atlas, we first need the connection string. Log in to your MongoDB Atlas account and then under the overview section, click on Connect and then Drivers.

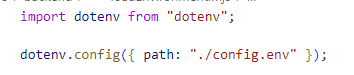


Copy the connection string and update your password in it like we did in our previous module and save it. We will use it in next steps.

Having obtained the connection string, return to the 'backend' directory and create a 'config.env' file. Within this file, assign the connection string to a fresh variable named ATLAS\_URI as shown below.



You will additionally require a module for loading this environment variable. In the backend directory, create a `loadEnvironment.mjs` file and input the provided code.



Create a new directory named 'db' within the backend directory. Inside this directory, create a file named 'conn.mjs'. Within 'conn.mjs', insert the following code to establish a connection with the database.



**Section 4: Setting up Server API Endpoints**

Return to the "backend" directory and generate the required directory and files using the following commands in the terminal for setting up API Endpoints:

**>> cd ../backend**

**>> mkdir routes**

**>> touch routes/record.mjs**

Now insert the code in your records.mjs from the hos-examples file “records.mjs”.

The code defines a router with various routes for performing CRUD (Create, Read, Update, Delete) operations on a collection named "records."

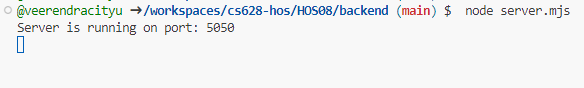
Express Router is a feature provided by the Express web application framework in Node.js that allows you to modularize and organize your application's routes and middleware in a structured manner.

* The first route responds to a GET request at the root path ("/") and retrieves all records from the "records" collection.
* The second route handles a GET request with a parameterized route ("/:id"), fetching a single record based on the provided ID.
* The third route deals with a POST request, allowing the creation of a new record in the "records" collection.
* The fourth route responds to a PATCH request ("/:id") and updates a record based on the provided ID with the data in the request body.
* The fifth route handles a DELETE request ("/:id"), deleting a record from the "records" collection based on the given ID.

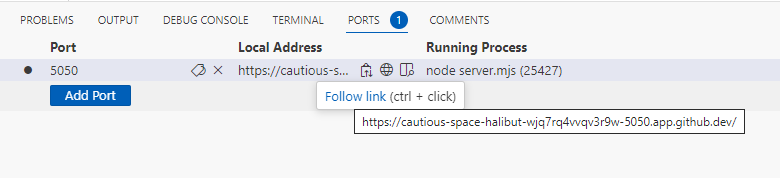
Let us now run the server using the following command in the terminal.

**>> node server.mjs**

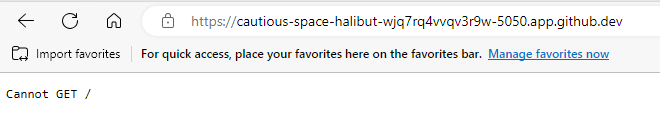
If everything works as expected, you will see this message in the terminal,

****

Now click on Ports tab beside the Terminal and open the local address in the new tab.



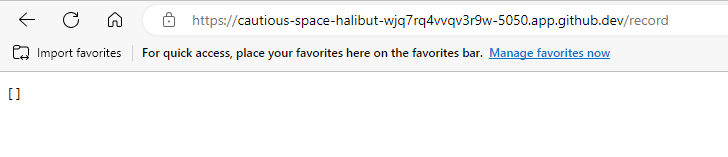
You will see the following message,



In our server.mjs file we are using **“/record**” to mount the records router to a specific route path within the Express application. This means that whenever a request is made to the "/record" route, the functionality defined in the records router will be executed.



Append **“/record”** to the above URL and refresh the browser. The error will go away, and an empty array will be returned. This is expected as we have not inserted any records into our database.

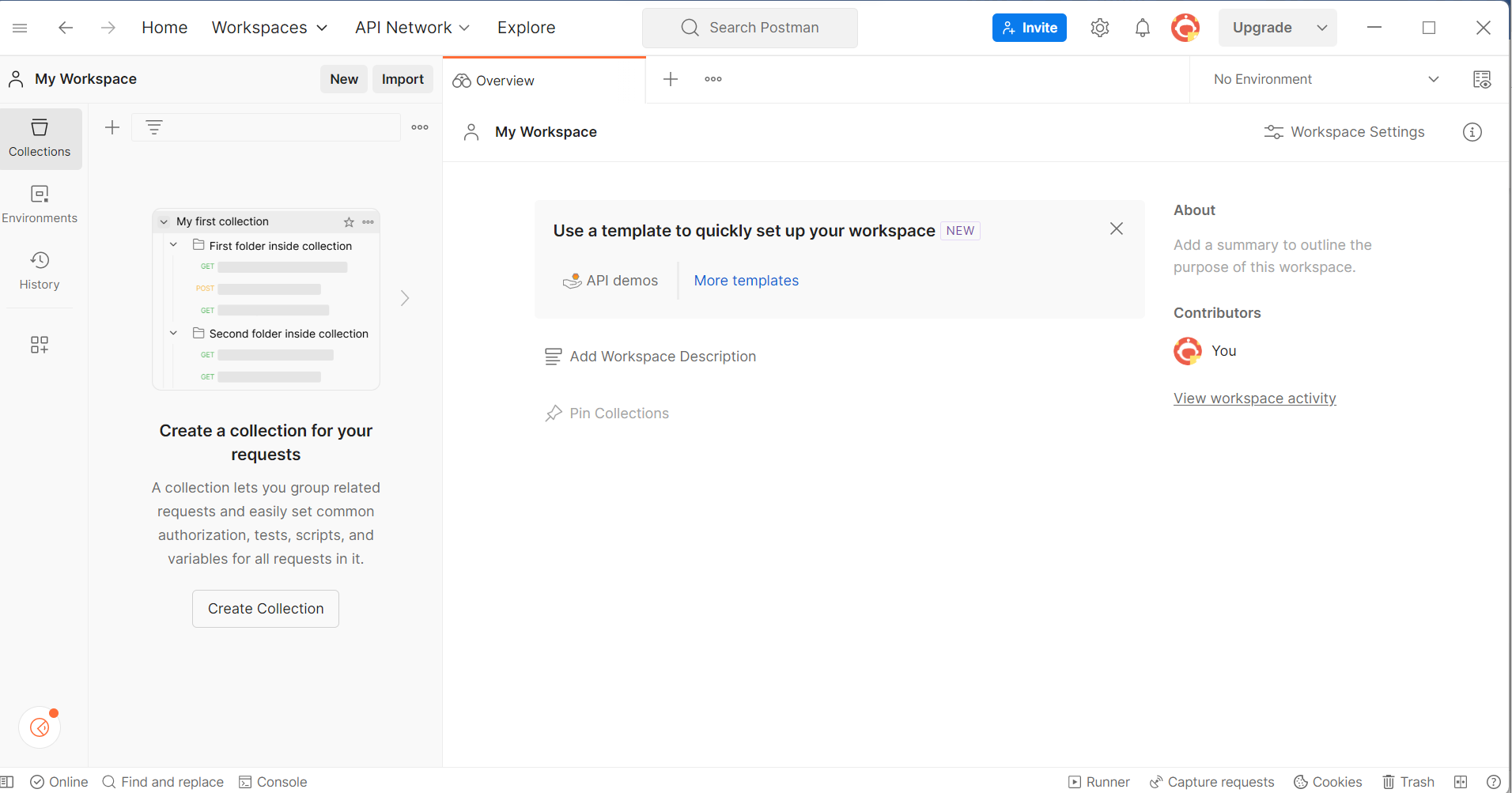


**Section 5: Testing the API Endpoints with Postman**

Let us now test our API using the postman tool. Download and install the postman app from this [link](https://www.postman.com/).

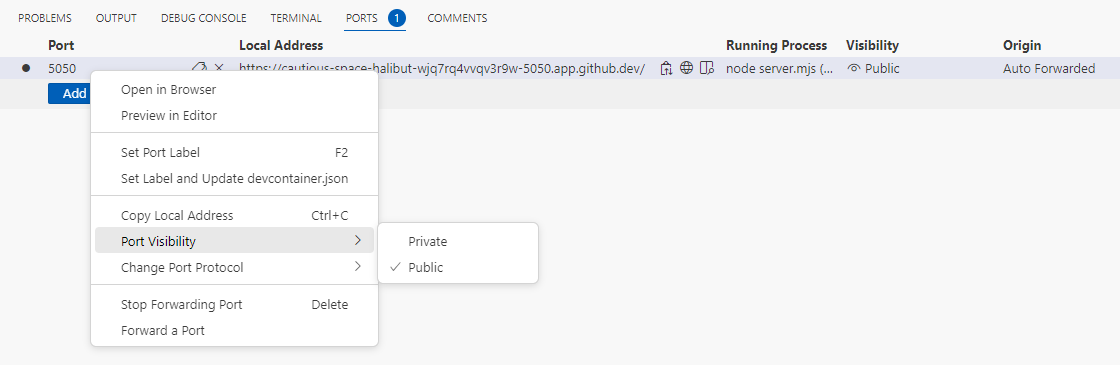
Postman is a popular tool for testing, developing, and documenting APIs. It simplifies the process of working with APIs by providing a user-friendly interface that allows developers to make HTTP requests, view responses, and test various API functionalities.

You can create an account or use google for signing in the postman application. After you sign in, you will see the following screen.

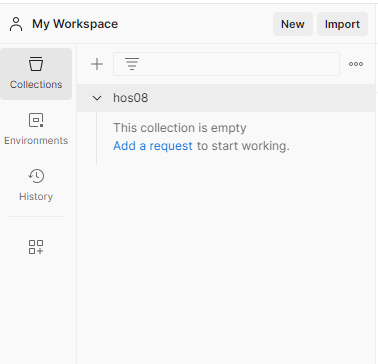


Before we start making the request, we need to make the port public for postman to access it.

In the ports tab of GitHub codespace, make it public as shown below by right clicking on the port number 5050.



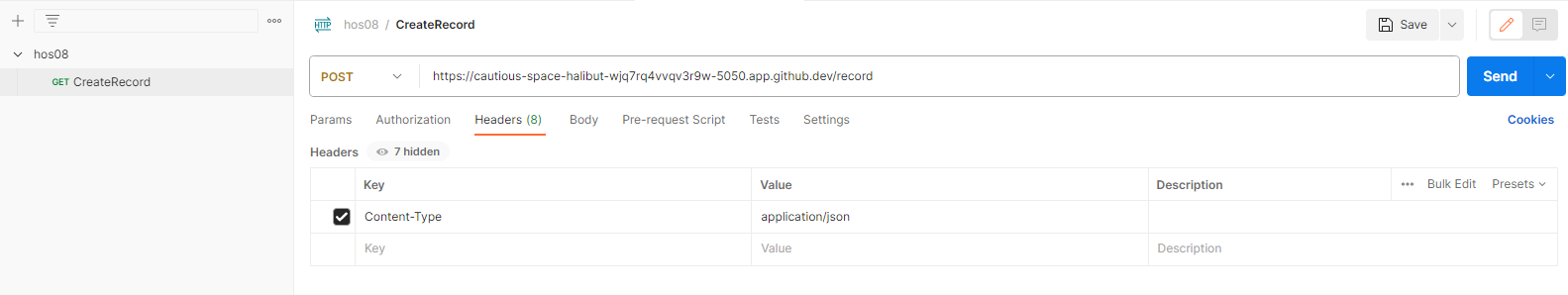
Start by creating a collection and give it a name hos08. After this click on Add a request to create your first request.



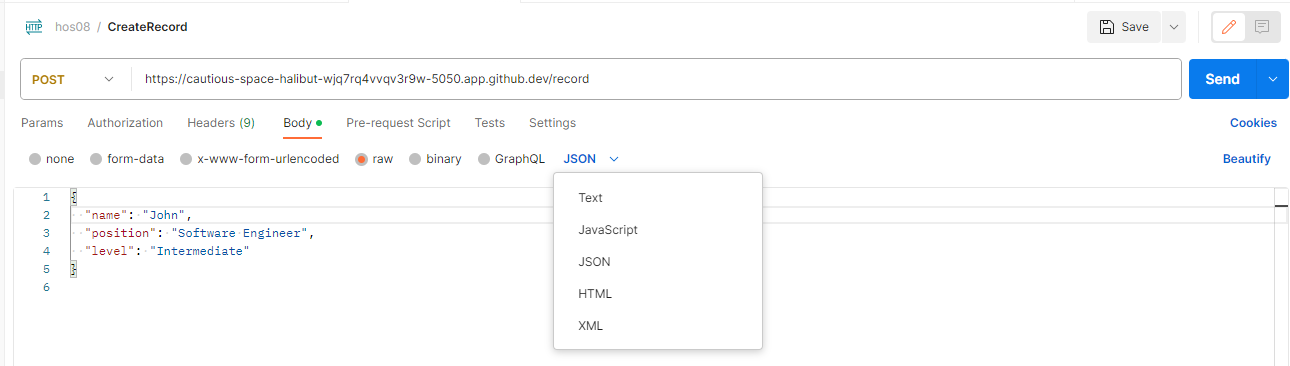
**CreateRecord**:

Let us start by inserting few records to database. Name your request as CreateRecord. We will make a POST request to the URL we used in the browser earlier.

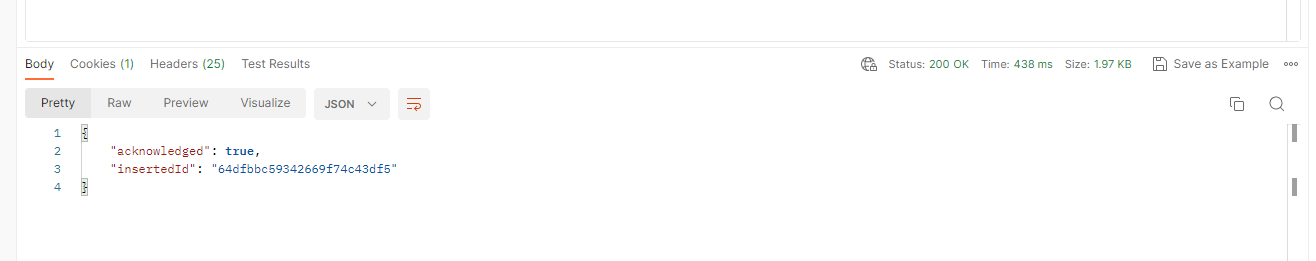
Under Headers useà Content-Type: application/json



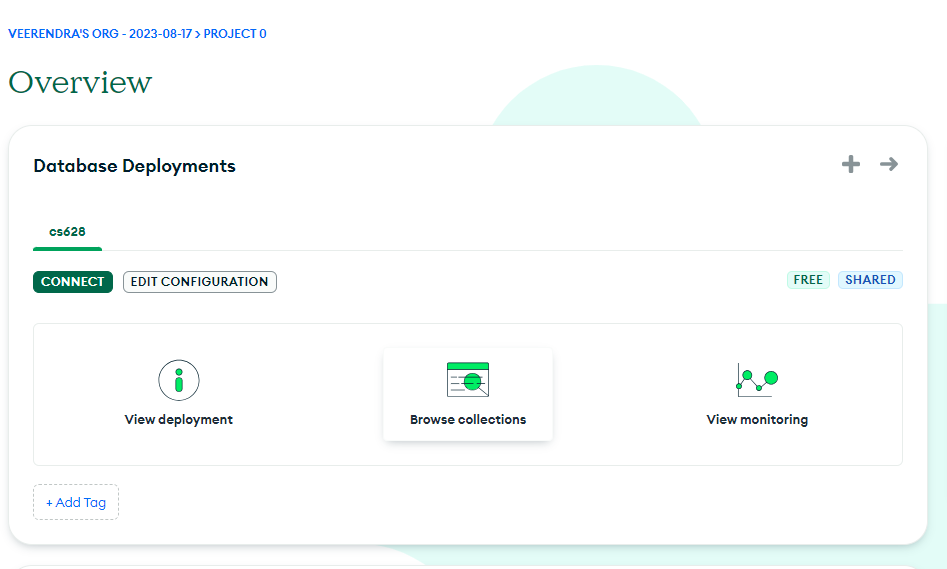
Under Body à Raw, use the request body as shown below to insert the first record and select JSON.



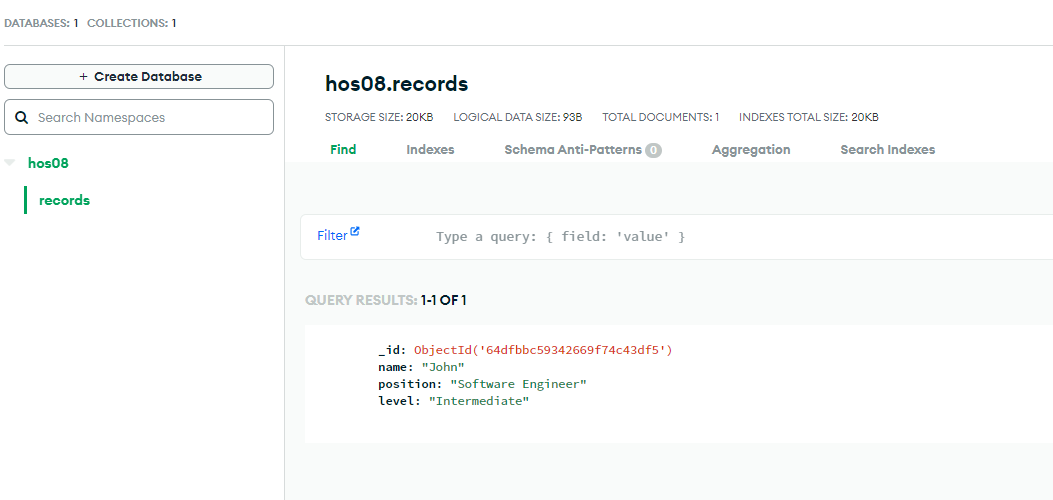
Click on Send and you will see the response body as shown below. Our first record is inserted.



Verify the data on the MongoDB side. Go to the cluster and click on browse collections.



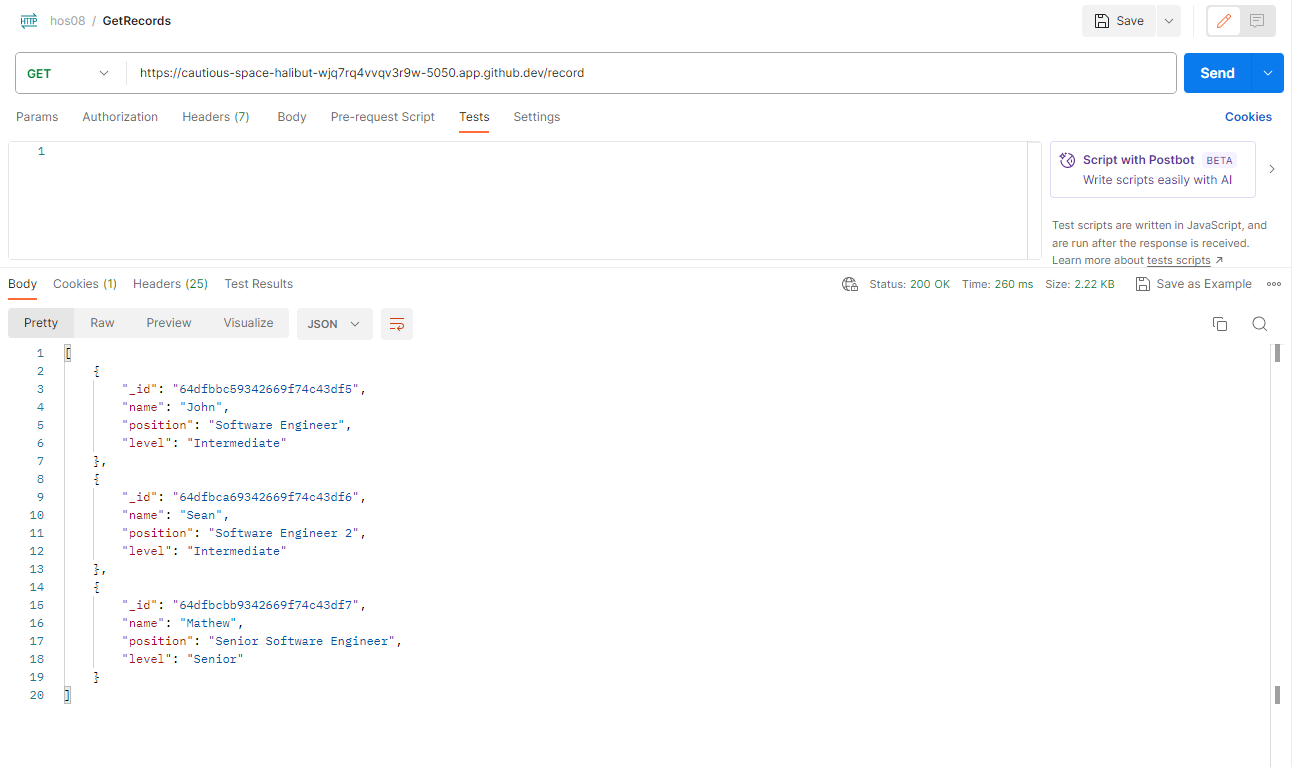
You will see the record we just inserted.



You can now change the request body in the postman tool and insert multiple records in similar fashion.

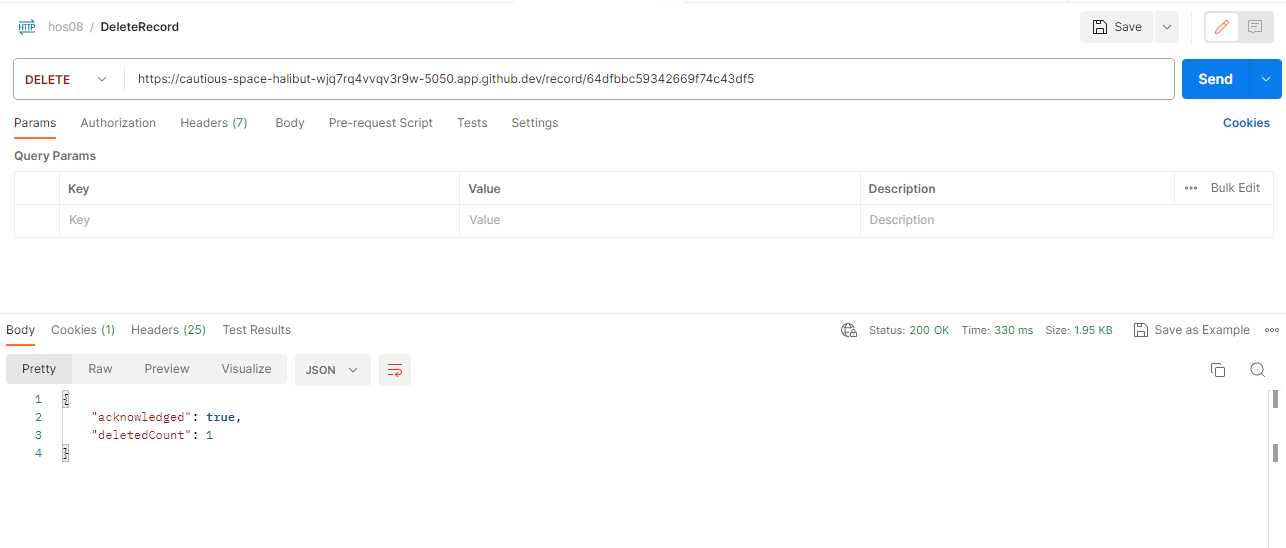
**GetRecords**:

Create another request under the collections and name it GetRecords. Use GET method and the URL obtained earlier to make the request as shown below. Observe the records returned in the response.



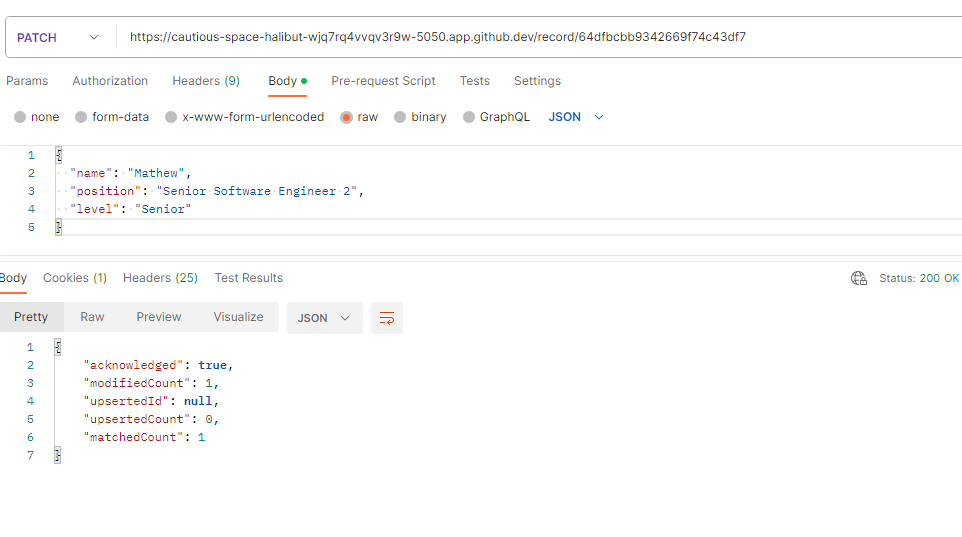
**DeleteRecord**:

Similarly, we can create another request to delete a record. Use DELETE HTTP method this time and you need to append the id in the URL after **/record.** You can get the id to delete from earlier GetRecords request.



**UpdateRecord**:

UpdateRecord request is like create request but with two differences. We will use PATCH Http method and pass the id in the URL for the record we are trying to update. Pease see below.



**Section 6: Pushing your work to GitHub**

* 1. Go to Source Control on your GitHub codespace and observe the pending changes.

Graphical user interface, text, application

Description automatically generated

* 1. Type the Message for your changes in the Message box on the top. For example,” **Submission for Module08 – Your Name**”
  2. Click on the dropdown beside the commit button and select **Commit & Push** to update the changes to your repository main branch.
  3. Select **Yes** when prompted.

Graphical user interface, application

Description automatically generated