|  |
| --- |
| Faculty of Applied Sciences and Technology |
| **NODE/EXPRESS WEB API** |
| ITE5315 - Project |
|  |
| **Kartik Nain, Hardeep Singh**  **Submission Date** |
| **4/10/2023** |

|  |
| --- |
| This document explains how to build Node/Express Web API …………………………. |

Table of Contents

[Question 1: 2](#_Toc100081673)

[Question 2: 3](#_Toc100081674)

[Question 3: 4](#_Toc100081675)

[Question 4: 5](#_Toc100081676)

[Question 5: 6](#_Toc100081677)

[Question 6: 7](#_Toc100081678)

[Summary 8](#_Toc100081679)

# Question 1:

(Describe the major steps for implementing the MongoDB database in Atlas )

1. Logged into the account on MongoDB Atlas.
2. Used the existing cluster and the existing database ‘sample\_supplies’ and collection ‘sales’.
3. Connected to the MongoDB cluster from my Node.js application by specifying the connection string provided by MongoDB Atlas. The connection string is saved into the .env file of the application. ‘Dotenv’ library is used to retrieve the connection string from the .env file.
4. Configured the Node.js application to use the Object-Document Mapping library -Mongoose.

Text

Description automatically generated

# Question 2:

(Describe the major steps for implementing Routes in the API, how you test this program, add some screenshots of the output)

Using the Router() method of the express framework, we define all the sales route in a separate file called ‘sales.js’. All the requests starting with ‘/api/sales’ will be redirected to this module where they will be handled. The sales module is then imported in the index.js file. Text

Description automatically generated

Following are the different routes defined in the sales module:

1.

Text

Description automatically generated

This route handles POST requests to the root endpoint / of the API. When a POST request is made, it creates a new Sale document using the Sale model defined by Mongoose, saves it to the database, and returns a response with a status code of 201 if successful or a status code of 500 if there was an error. A screenshot of a computer

Description automatically generated with medium confidence

2.

Text

Description automatically generated

This route handles GET requests to the root endpoint / of the API. It uses the celebrate middleware to validate the request query parameters, which include page, perPage, and storeLocation. If the parameters are valid, it uses Mongoose to query the database for Sale documents that match the specified storeLocation (if provided), skips over skip documents based on the page and perPage parameters, and returns a response with a status code of 200 and the Sale documents if successful, or a status code of 404 if no documents were found. router.use(errors()) handles validation errors generated by the celebrate middleware. A screenshot of a computer

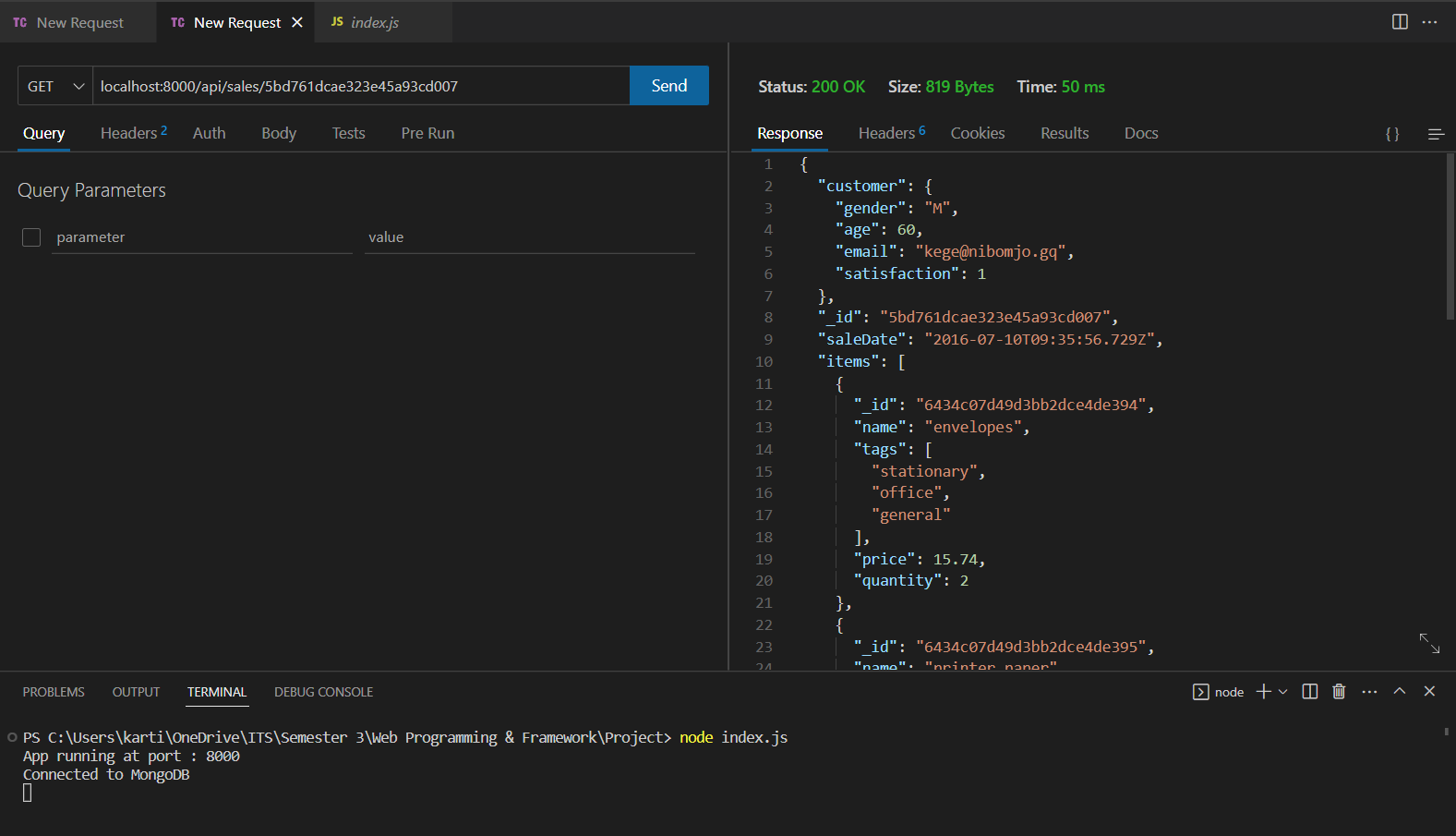
Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence

3.

A screenshot of a computer

Description automatically generated with medium confidence

This route handles GET requests to a specific Sale document using its \_id parameter. It uses Mongoose to find the Sale document by \_id and returns a response with a status code of 200 and the Sale document if successful, or a status code of 404 if the document was not found. 

4.

A screenshot of a computer

Description automatically generated with medium confidence

This route handles PUT requests to update a specific Sale document using its \_id parameter. It uses Mongoose to update the document with the specified \_id using the data in the request body, and returns a response with a status code of 201 if successful, or a status code of 500 if there was an error. A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

This route handles DELETE requests to delete a specific Sale document using its \_id parameter. It uses Mongoose to delete the document with the specified \_id and returns a response with a status code of 201 if successful, or a status code of 500 if there was an error. A screenshot of a computer

Description automatically generated with medium confidence

# Question 3:

(Describe the major steps for designing the FORM/UI, how you test this program, add some screenshots of the output)

Graphical user interface, website

Description automatically generated

First the user enters the token generated using login route in the field and clicks on submit, passport.authenticate method on the backend authenticates the token and grant user the access to this below page. The token from the field is sent to the passport using request field which is extracted using this method - ExtractJwt.fromBodyField('token') . All these pages are made using template engine – hbs.

Graphical user interface, application, website

Description automatically generated

This is the UI page that will take input from user as per fields like Page No, Number of record per page and store location. Store location is optional.

And then it will fetch sales data from mongodb. A screenshot of a computer

Description automatically generated

# Question 4:

(Describe the major steps for setting up environment variables.)

First, a module – ‘dotenv’ is installed which is then used in the application to load the .env file in the application.



The config() method is a function provided by the dotenv package that reads the contents of the .env file, parses it, and adds the key-value pairs to the environment variables of the Node.js process. These environment variables are then available to the Node.js application through the process.env object.

# Question 5:

(Describe the major steps for implementing security features, how you test this program, add some screenshots of the output.)

For All Routes:

In all route we added security by using users collection login data.

When user will log in to the page it will generate token and after giving bearer in header we authorize users to access all routes.

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

# Question 6:

(Describe the major steps for deployment)

1. Created a new account on Cyclic.sh and logged in to the dashboard.
2. Created a new project and selected Node.js as the application type.
3. Created a local git repository in the project root folder.
4. Connected my code repository to Cyclic.sh by providing the repository URL and authentication credentials.
5. Configured the deployment settings by specifying the branch to deploy.
6. Set up necessary environment variables like the connection string using the Cyclic.sh dashboard.
7. Deployed the application by initiating a new build and configuring automatic deployment when changes are pushed to the repository.
8. Monitored the application logs and added the 0.0.0.0/0 IP address whitelist to MongoDB account so that the cyclic platform can access the database.
9. Updated the application by pushing new code changes to the connected repository or by manually updating the application files in the Cyclic.sh dashboard.

# Summary

(Describe how did you divide the work, share your feedback about this project like new points that you learn, challenges, …)

We divided the work on bases of how front UI looks like and how our backend code should work. Kartik developed routes related code and authenticated using login collection data that we created in previous classes and Hardeep developed the front UI page and authentication page to restrict the user access to sales data by using token with ID as one of the documents of sales and private key.

We faced challenge in authentication as we were confused about should we use the new collection login or should we use cookies or token for that while creating User Interface.

Overall, We applied all concepts that we learnt in class in that project.