Question1: -

Step1: -

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Step 3: -

A) A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

B) A screenshot of a computer

Description automatically generated

C) A white screen with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

Step 4:-

1. How does the Step3: A, B, C work? Explain the workflow, route, and the way the query executed.

Step A: -

**Method of POST (/api/employees):**

Express parses the incoming form data when the request reaches the server.

The POST request is processed via the app.post('/api/employees') route.

Using Employee.create(), it adds a new employee record to the database depending on the received data (req.body).

Following record creation, Employee.find() is used to retrieve every employee, and the updated list is returned in the response.

**Response:**

The list of all employees, including the recently added employee data, would be included in the response.

Step B: -

**Route: GET workers at /api/employees**

To retrieve every employee from the database, the server receives a GET request to fetch all data from the database (Employee.find()).

**Execution of Query:**

The server-side code expects to use Employee.find() to access all employee data from the database because the route /api/employees is made to support GET requests.

Then, it makes an attempt to deliver a JSON response (res.json(employees)) with all of the database's employee data in it.

Step C :-

**Get /api/employees/:employee\_id**:

When an employee's ID is used to retrieve their data, the server expects a GET request to fetch employee data by their ID (Employee.findById()).

Using their distinct \_id, the employees can be uniquely identified by using the :employee\_id dynamic parameter in the route.

**Execution of Query:**

The server-side code uses the \_id provided in the URL to search the database for an employee record in accordance with the route that has been provided.

The employee with the supplied \_id is found using Employee.findById().

When it finds that employee, it returns a JSON object with the employee's information in it.

**b) What is the role of:**

**a. module.exports = mongoose.model('Employee', EmpSchema);**

The line module.exports = mongoose.model('Employee', EmpSchema); in Node.js is used to export the Mongoose model named 'Employee' along with its defined schema (EmpSchema) to other parts of the application.

**b. Employee.findByIdAndUpdate**

Employer.findByIdAndUpdate is used to update the contents of a particular employee in the MongoDB collection by locating it using its \_id field.

It requires multiple parameters.

id: The value of the \_id field, which is used to identify and update the employee document.

data: An object with the updated values for the fields that need to be updated. It contains fields for name, salary, and age in the context that is provided.

A callback function to handle the result of the update operation.

**c) Using the idea of Step3:C, try to update one of the records in the employee table. Find related route ☺ in the code and explain how it works.**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Route Defined:**

path for processing /api/employees/:employee\_id HTTP PUT requests to update employee data.

**Extracting Data:**

gathers name, salary, and age from the request body and extracts employee\_id from the request URL.

**Preparing Data:**

depending on the extracted values, prepares an object (data) with updated employee information.

**Update Operation:**

Using Employee.findByIdAndUpdate() to find and update the employee record based on the provided employee\_id with the new data (data object).

**Response Handling:**

if the update process is successful, returns a success message to the client stating that the employee was updated.

if the update fails, handles errors (invalid ID, for example) and may give an error response.

**d) Using the idea of Step3:C, try to delete one of the record in the employee table. Find related route ☺ in the code and explain how it works.**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Route Defined:**

path for handling /api/employees/:employee\_id HTTP DELETE requests to remove an employee record.

**Extracting Data:**

obtains the request URL's employee\_id parameter.

**Delete:**

based on the \_id given in the URL parameters, finds and deletes an employee record from the database using Employee.remove().

The req.params.employee\_id is used to specify the \_id that is to be destroyed.

**Response Handling:**

An error answer is sent if there is a problem with the delete operation.

The employee is deleted, and a success message is sent if the deletion is successful.

**Question 2:-**

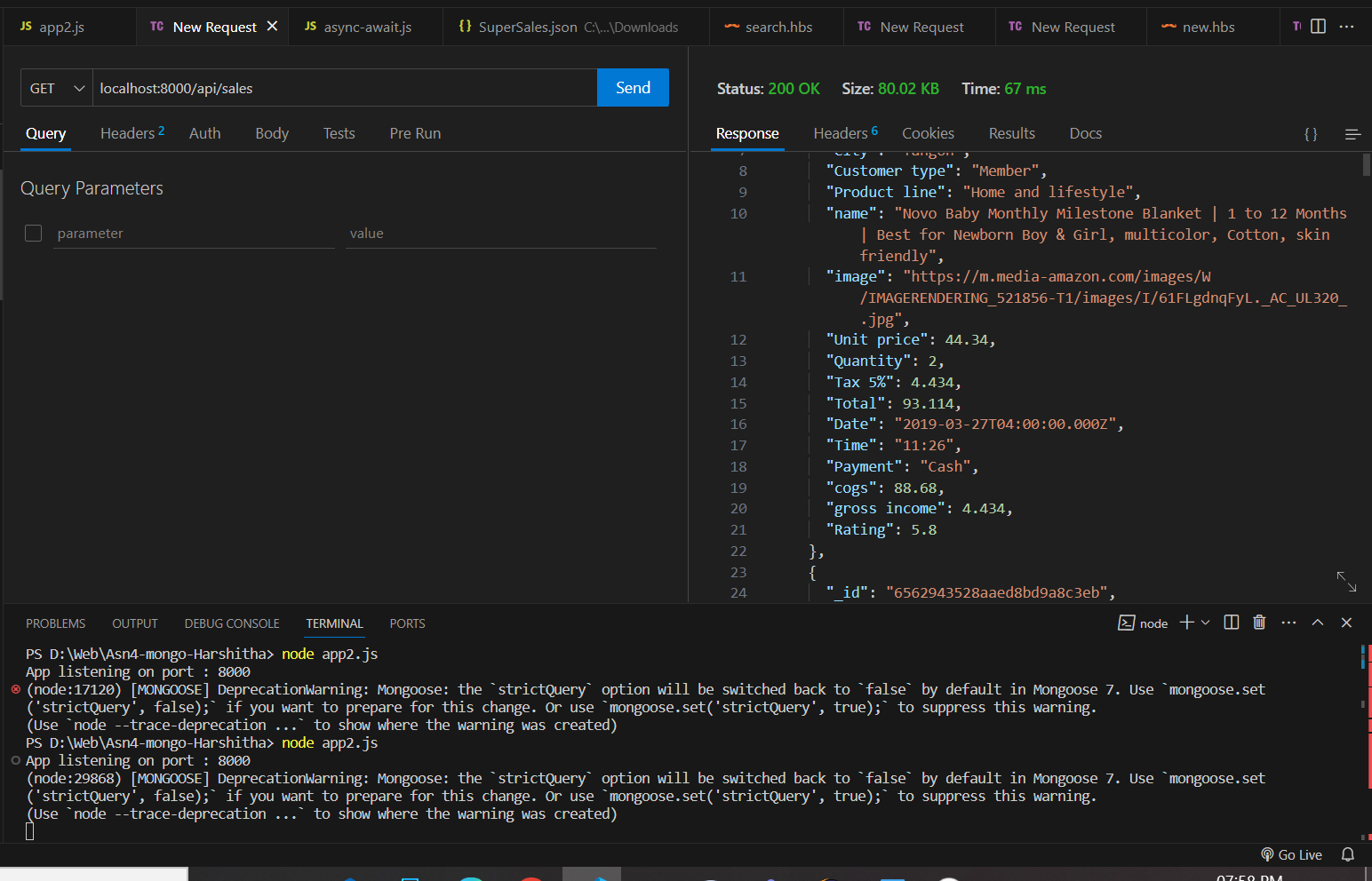
**Step 1:-** : Create a new MongoDB database in Atlas based on the given dataset of Assignment2.

A screenshot of a computer

Description automatically generated

**Step 2:-**

Show all invoice-info



Show a specific invoice (based on the \_id or invoiceID)

A screenshot of a computer

Description automatically generated

Insert a new invoice

A screenshot of a computer

Description automatically generated

Delete an existing invoice (based on the \_id or invoiceID)

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

Update “Customer type” & “unit price” of an existing invoice (based on the \_id or invoiceID)

**Before**

A screenshot of a computer

Description automatically generated

A computer screen shot of a program

Description automatically generated

**After**

A screenshot of a computer

Description automatically generated

Using Handlebar and Form complete the followings (hint:use ideas from Assingment2):

▪ Show all invoice-info

A screenshot of a sales data

Description automatically generated

▪ Insert a new invoice

A screenshot of a computer screen

Description automatically generated

**Newly added item in the list**

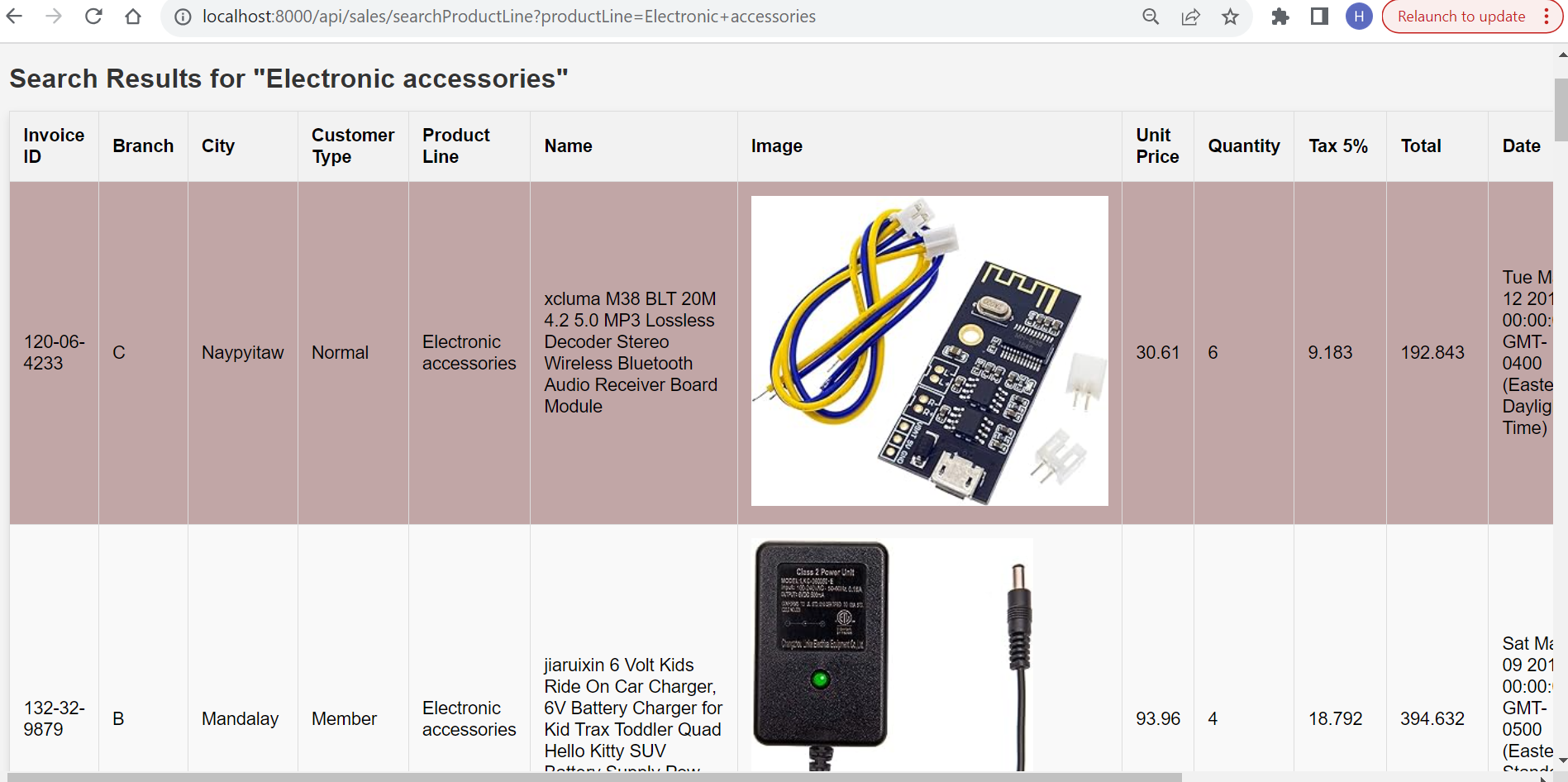
A screenshot of a video game

Description automatically generated

Step 3:Using your creativity, Add a new functionality to this app.

A computer screen shot of a computer

Description automatically generated



**Question 3:-**

Step 1: and run it using nodemon. Look at the output of the program.

The output of the program is

A screen shot of a computer program

Description automatically generated

Step 2: What if you remove wait from Task1, any error? Explain what you have learned.

If we remove await from task1, then I see TypeError: client.db is not a function and TypeError: client.close is not a function. This is because code doesn't wait for the connection to be established and proceeds to the next lines.

**client** variable does not hold the MongoDB client instance. As a result, methods like .db() or .close() cannot be called on this incomplete object, leading to TypeError as observed.

1 is logged to the console within the try block because it occurs before any asynchronous operations. Despite encountering errors within the findAll() function, the timer from setTimeout() triggers after 5 seconds, calling findAll() again and logging 'iter' to the console.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Step 3: What if you remove all await/async from the task1 and 2. How do you explain changes in the output compare to Step 1?

The difference in the output from previous step is that iter is not displaying because Errors within the try block (like client.db()) can be synchronous which is causing the code to jump to the catch block or finally block without completing the scheduled tasks, including 'iter'.

A screenshot of a computer program

Description automatically generated

Step 4: Bonus: Can you design the given functionality/program using Promise?

Using the MongoClient.connect() method, the connectDB() function establishes a connection to the MongoDB database and provides a promise. promise is encased in a custom promise that, in the event that the connection is unsuccessful, rejects with an error or resolves with the client object

findAll() function utilizes the connectDB() promise. Upon successful connection it proceeds with the operation.

