**Part 4. REST APIs & Database Documentation**

**Task 1.** Create and call REST APIs

Joe Black has heard about REST API is an architectural style that supports HTTP requests from applications to access data in MongoDB. He would like you to illustrate how the API layer which provides the decoupling of client applications from server through this uniform interface. The illustration must use a client application with HTTP access the URI resources to perform the database actions on JSON data with MongoDB. You are required to build the REST APIs (create, read, update, and delete i.e. CRUD) using Express.js, Node.js, nodemon, and Mongoose. The illustration must also include codes with error trappings to show whether the database operations are executed successfully or not. To complete these tasks, no need to write a full web application, for the purpose of illustrating the REST API calls, use Postman as the development tool without writing the html pages for simplifying the build, test and modify the APIs.

Node.js

Express

MongoDB Driver

APIs

APIs



Postman

MongoDB

The REST API architectural style with MongoDB

1. Login the MongoDB Atlas cluster in the cloud, obtain a connection string for application and the Node.js driver. Show the connection parameters and the code of Node.js driver (in full).

(i.e. no need to show your password.)

|  |
| --- |
| const { MongoClient, ServerApiVersion } = require('mongodb');  const uri = "mongodb+srv://andrealexandrov:<password>@shoppingworld0.ld75ke0.mongodb.net/?retryWrites=true&w=majority";  // Create a MongoClient with a MongoClientOptions object to set the Stable API version  const client = new MongoClient(uri, {  serverApi: {  version: ServerApiVersion.v1,  strict: true,  deprecationErrors: true,  }  });  async function run() {  try {  // Connect the client to the server (optional starting in v4.7)  await client.connect();  // Send a ping to confirm a successful connection  await client.db("admin").command({ ping: 1 });  console.log("Pinged your deployment. You successfully connected to MongoDB!");  } finally {  // Ensures that the client will close when you finish/error  await client.close();  }  }  run().catch(console.dir); |

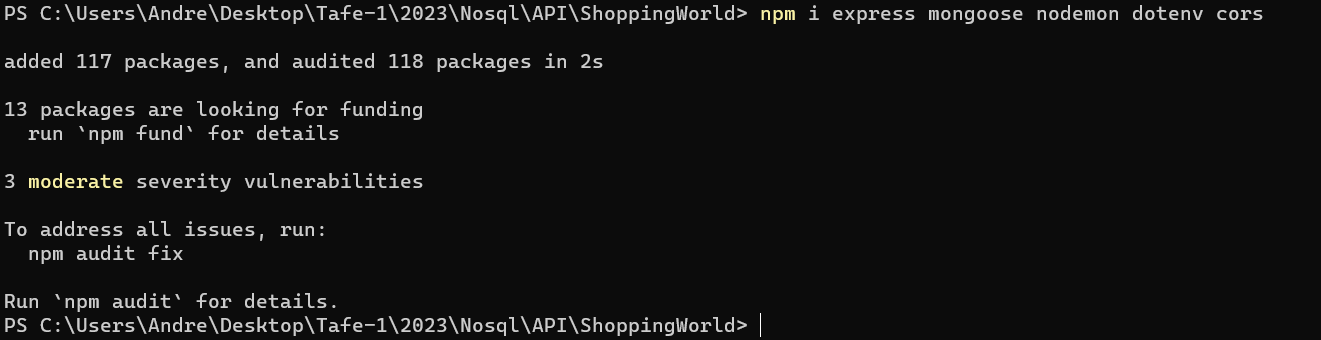
1. Create a directory called “shoppingWorld” in the c: drive. Use Visual Studio Code to create the package.json file and run the **npm init** to initialise the project such as name of the package, version, author and description and the dependent packages used etc. Enter/Edit this information with the context of ShoppingWorld database development. You must include the following information:
   * Name of the package
   * Version of the package
   * Description of the package
   * Author name
   * License information
   * List of the dependency packages and their version used.

For submission: In the Visual Studio codes include the **package.json** file which specified the above details.

Note: There are many packages in the market that you may use to develop the REST API, however you may need to install the mongoose library to translate the objects between the code and MongoDB. Note: whichever packages that you have used for the development, you must include those packages in your submission of the codes.

A screen shot of a computer program

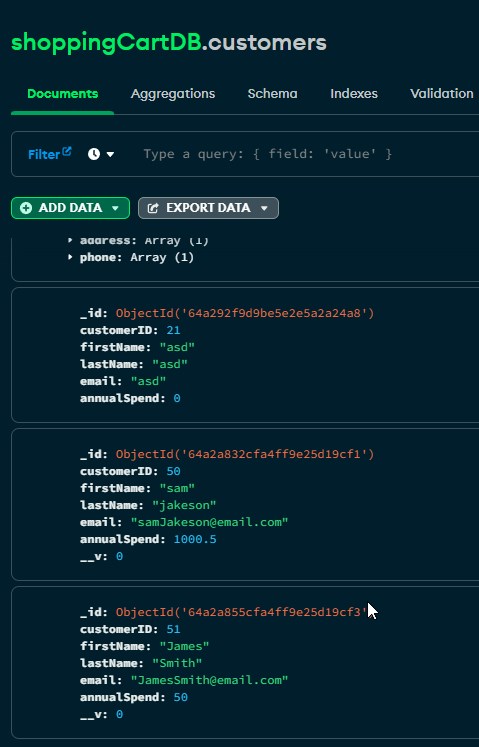
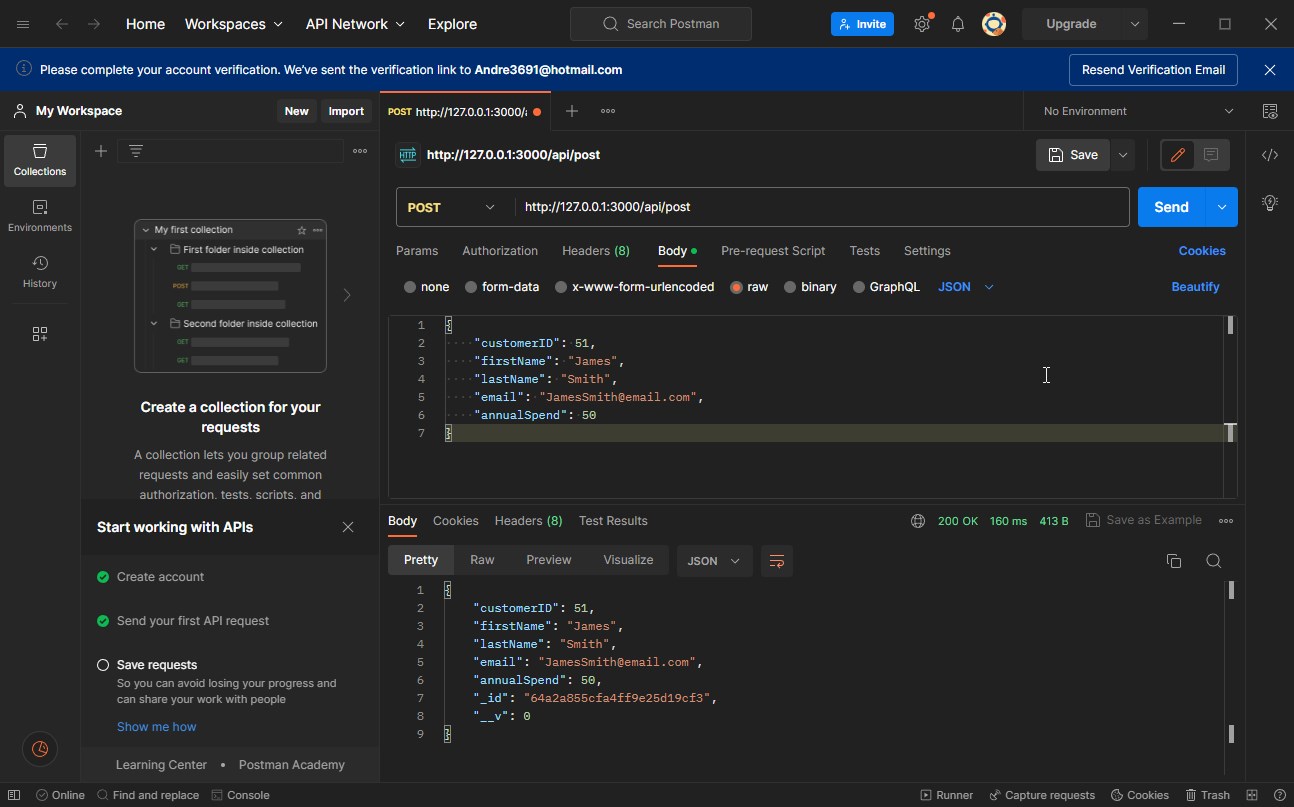
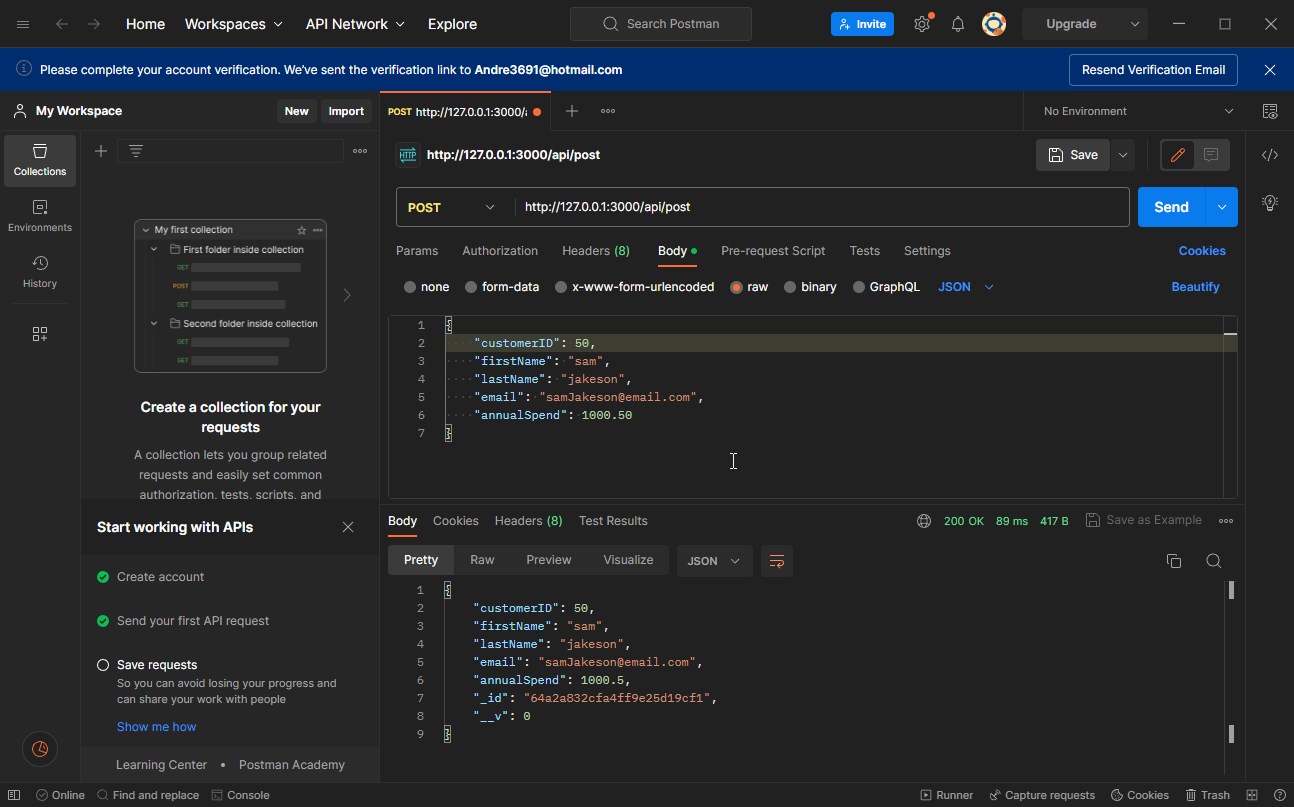
Description automatically generated with low confidence



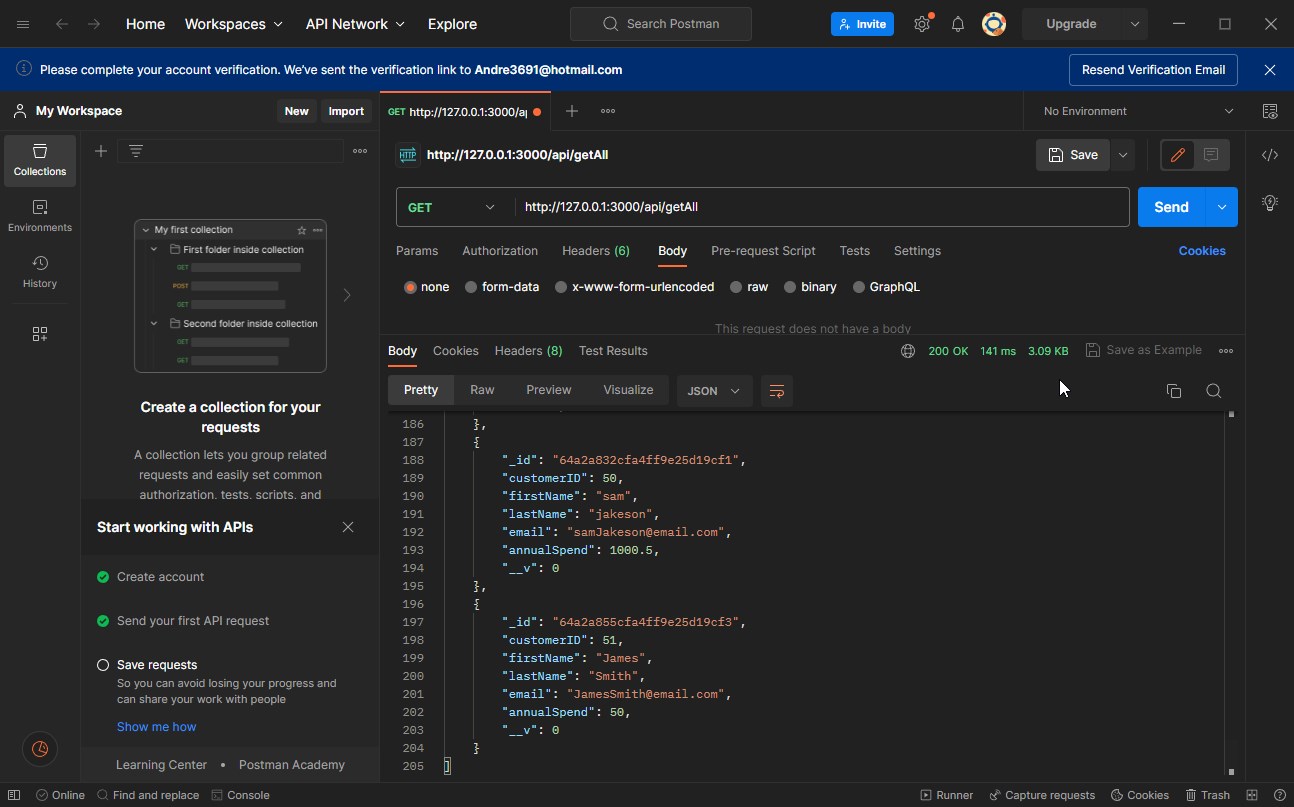
1. Use Node.js to write a REST API route for the purpose of performing operations to create, read, update, delete **(CRUD)** customer document into the customers collection in MongoDB at the end. Test the execution by sending the requests through the Postman as the front-end application. For this task, create a file called **index.js** under the node\_modules to include the following features:
2. At the top of the file, define constants for the Express and Mongoose libraries.
3. Provide a connection function to connect the MongoDB Atlas database using Mongoose. Use the .env file for the database connection string.
4. Define the end point for the routes.

1. Define the listening port for the Express server.

1. Write a REST API route to perform insert a customer document into the customers collection. To simplify the illustration to Joe Black, no need to include the sub-documents in this task. The customer information should have customerID, firstName, LastName, email and annualSpend. Test the execution by sending a post request through Postman with your example data. You are required to insert two customers into the customers collection. For submission, other than the project code, show the screen shots of the Postman request and response from the API call.



1. Write a REST API route to perform read operation to read all customer documents from the customer collections. Test the execution by sending a get request through Postman to retrieve the customers that have been inserted in **Part** **d**. For submission, show the screen shots of the Postman request and response from the API call.

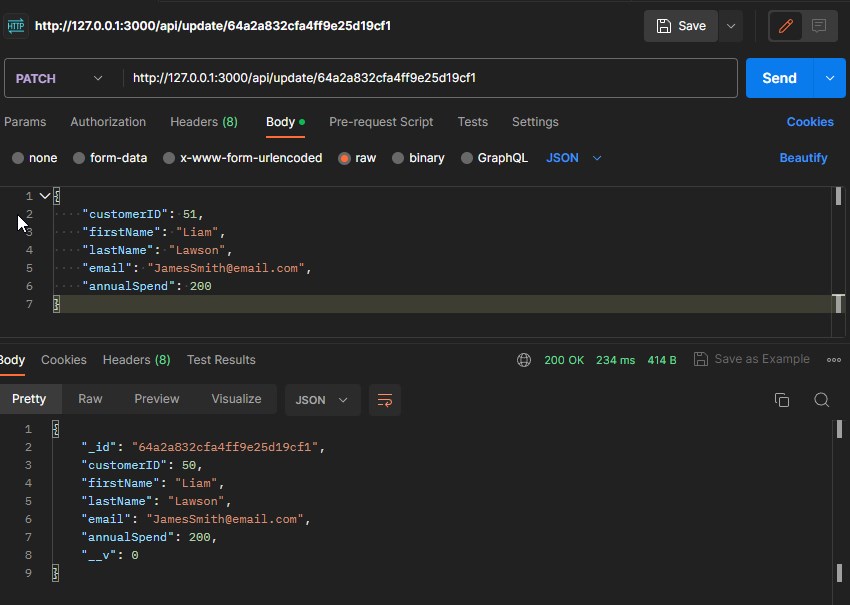


1. Write a REST API route to perform a read operation to find one customer document from the customer collections. Test the execution by sending a get request through Postman using one of the customer object\_id that has been inserted in **Part** **d** as the search criteria. For submission, show the screen shots of the Postman request and response from the API call.

A screenshot of a computer

Description automatically generated with medium confidence

1. Write a REST API route to perform an update operation for a customer document in the customer collections. Test the execution by sending a patch request through Postman for the customer object \_id that has been inserted in **Part** **d** with a new information firstName, lastName and annualSpend. For submission, show the screen shots of the Postman request and response from the API call.



1. Write a REST API route to perform a delete operation to delete a customer document in the customer collections. Test the execution by sending a delete request through Postman to delete the customer document that you have created in **Part d** above. For submission, show the screen shots of the Postman request and response from the API call.

A screenshot of a computer

Description automatically generated

1. This task is to use the debugger in the Visual Studio Code. While you are developing the REST API route for **Task d** (i.e. insert customer document into the customers collection) with the .js code in Visual Studio Code, add or modify your code with the following error trapping if you have not got one:

e.g.

try {

Add a break point at this statement.

const dataToSave = await customers.save();

res.status(200).json(dataToSave);

}

catch(error) {

var myMessage = error.message;

res.status(400).json({message: myMessage});

}

* Add a break point at the first try statement.
* Run npm start with debug mode.

You are required to provide the screen shots for the following tasks:

* Capture the first screen shot that shows the program execution stops at the break point.
* When the program stops at the break point, use Postman to post a customer with your own data so that it failed to insert the customer due to data error.
* Use the debugger to Step over to those lines

**var myMessage = error.message;**

res.status(400).json({message: myMessage});

* Capture the 2nd screen shot that shows the value of the myMessage local variable after the line has been executed.

For submission:

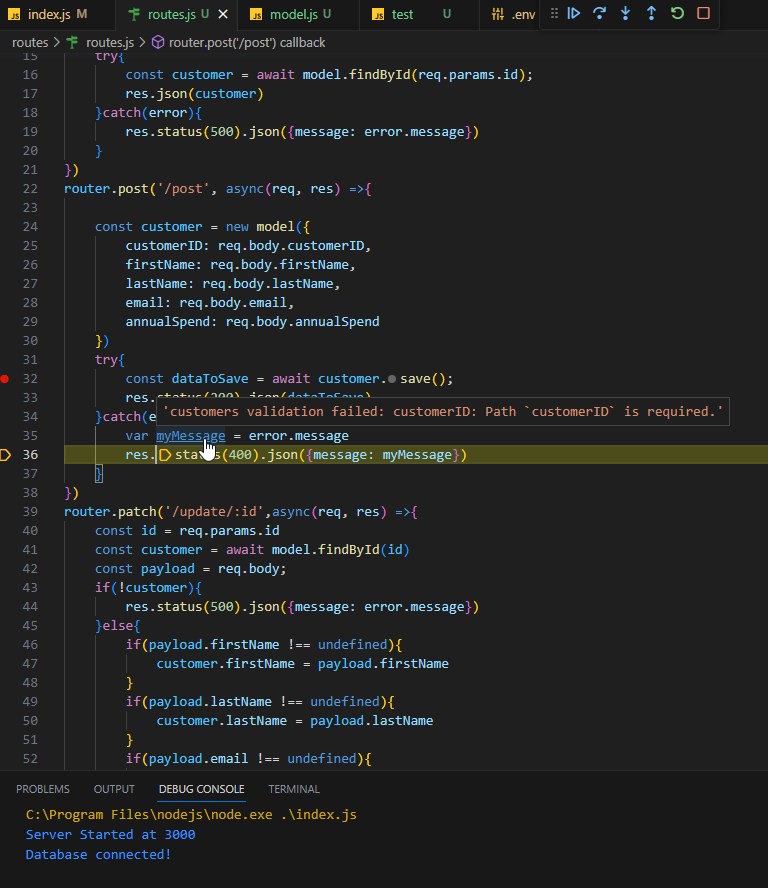
Submit the two screen shots in the word document. Label the question no. clearly.

Screen Shot 1

A screenshot of a computer program

Description automatically generated with medium confidence

Screen Shot 2



**Submission for Task 1 sub-tasks (a) to (i):**

* Submit the Visual Studio Codes project including all .js files such as package.js, index.js, .env, model.js and routes.js files.
* Submit screen shots for all REST API operations tested with the Postman requests for sub-tasks (d) ~ (i).

**Task 2.** Database Trigger

The Client Database Administrator (Joe Black) has concern with the change stream handled by the client application. The consistency and integrity of data relies on the client application are always running so that it would not miss any of the change events. MongoDB have the feature of server-side database triggers. You are required to create a database trigger and show the notification of the event has happened.

a. Use MongoDB Atlas web UI, create an update database trigger called “customerChangeTrigger” **change** event to respond when a document customer is changed (e.g. the value of the annualBalance is changed).

For submission:

* + Submit a screen shot with the name of the update event trigger has been created.

A screenshot of a computer

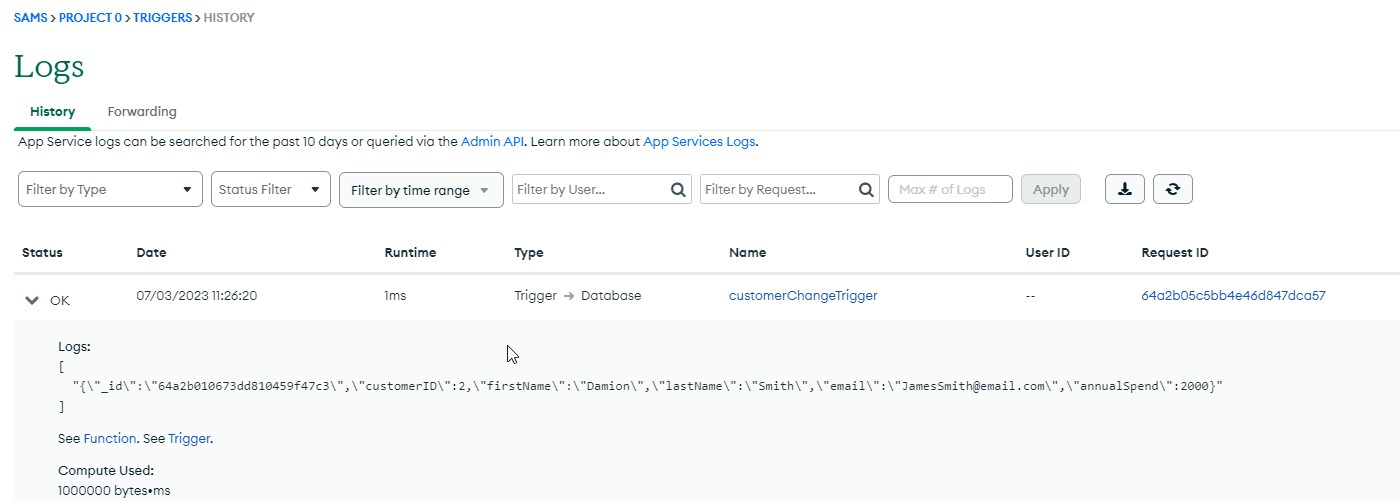
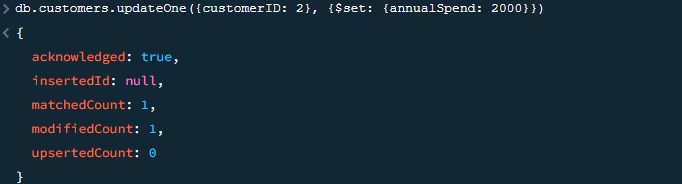
Description automatically generated with medium confidence

* + Submit a screen shot of the notification function to display the update event details through the RealmApp.

A screenshot of a computer

Description automatically generated with low confidence

b. Write an updateOne statement to test the change event trigger. The update statement can be run either through the REST API call or any client tools like MongoDB Compass or even command line. The update statement must change the customer document for customerID = 2 with the annualSpend of $2000.00. Display the notification of the change event has happened through the Triggers\_RealmApp.



**Task 3.** MongoDB data Encryption

MongoDB supports three types of data encryptions: Encryption in Transit/Transport (TLS), Client-Side Field Level Encryption, and encryption at REST. These features need to be turned on and these will increase the daily cluster pricing. Describe each of the following encryption options: (Approx. 40 words each).

1. Encryption in Transit/Transport layer security (TLS)

Transport Layer Security is a widely used cryptographic protocol for securing web applications. It encrypts data during transmission, ensuring secure communication between clients and servers. TLS/SSL relies on SSL certificates, containing domain and public key information, to validate server identity and enable encryption.

1. Client-Side Field Level Encryption

Data is encrypted in your app before being sent to MongoDB. CSFLE ensures MongoDB can't access unencrypted data. Keys used for encryption/decryption are encrypted with a master key. KMS is needed for CMK access. Deleting CMK makes encrypted data permanently unreadable. Supports GCP, Azure, AWS, local keys.

1. Encryption at REST

Encryption at rest protects data by encrypting it on disk, making it inaccessible to attackers without the encryption keys. MongoDB's default encryption mode uses AES256-CBC via OpenSSL, ensuring secure encryption and decryption. All data files are fully encrypted in the storage layer, remaining unencrypted only in memory and during transmission.

**Task 4.** Document the database

Document the database by completing the **IT Works MongoDB Database Documentation Template.docx** which can be found in the Assignment ASDS Part 4 student file. All sections must be filled. The answers must be in the context of the Shopping World. (Note: rename the file name **without** the word “**Template**”.

For submission:

Submit the completed **IT Works MongoDB database documentation.docx** document.

**Part 4 Submission:**

* + A world document includes answers of Task 1 to Task 3. Label each task and sub tasks clearly with the answers, commands and screen shots as required. The screen shots must cover the script and the output of the execution.
  + For Task 1, also submit the Visual Studio Codes project including all .js files such as package.js, index.js, .env, model.js and routes.js files. Put these files under the **Part4\_Task1** folder.
  + The completed IT Works MongoDB Database Document.docx.
  + Zip the above files/folders into a single zip file for submission to LEARN.