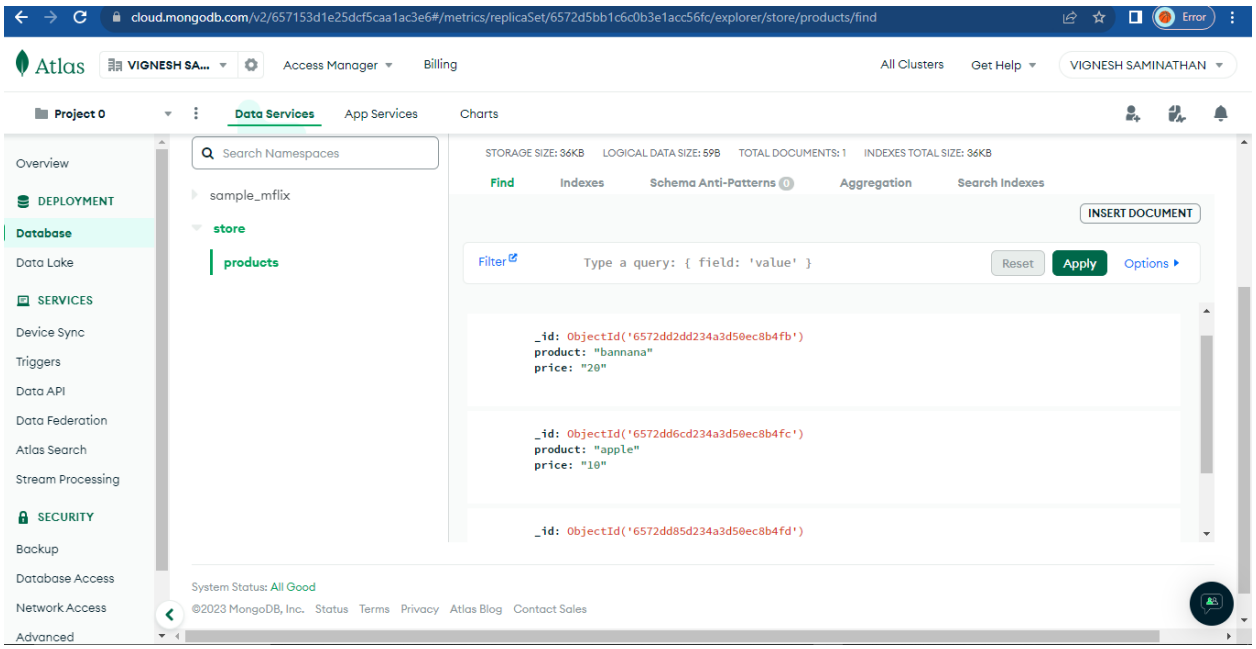
Question 1: Connecting MongoDB and CRUD Operations
How would you establish a connection between an AWS Lambda function and a MongoDB database? Provide the necessary configuration steps and code snippets. Assume you have a MongoDB database with a collection named "test-data." Write a sample AWS Lambda function (using Node.js) that performs CRUD operations (Create, Read, Update, Delete) on the "test-data" collection. Include error handling in your code

Created user and database and connection was established and successfully created a Crud Application.



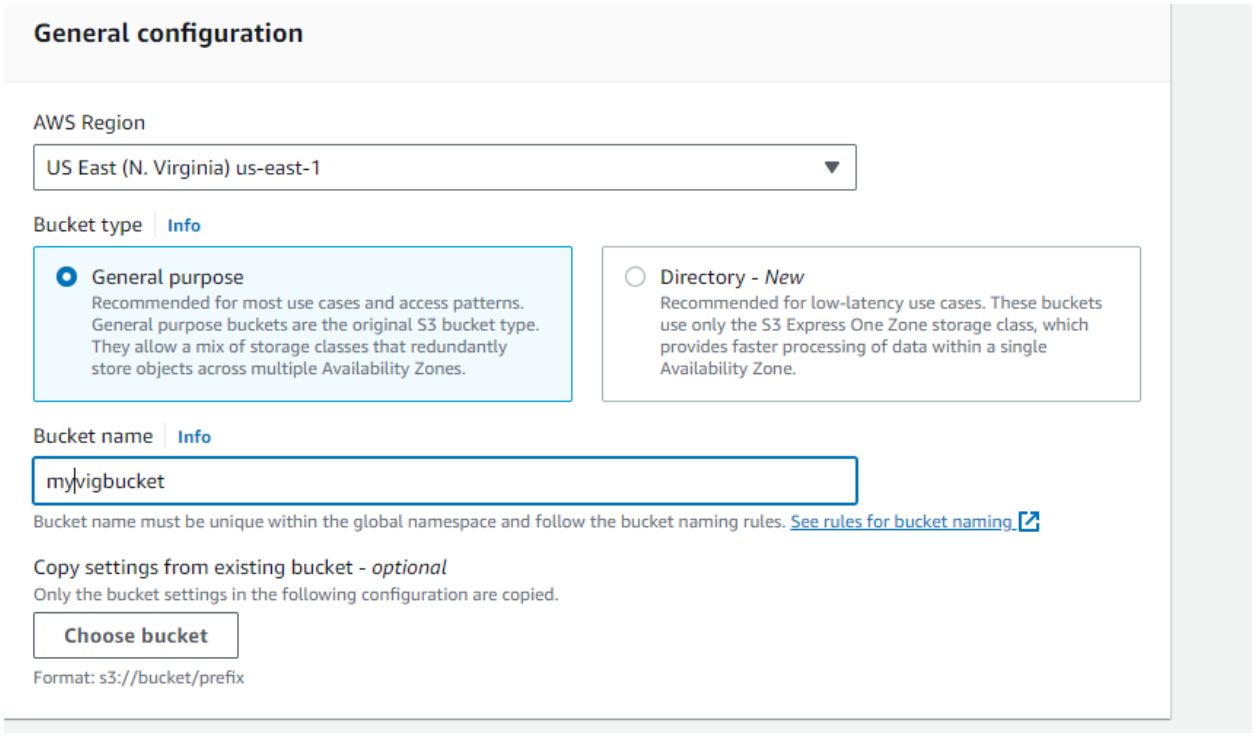
```
Node.js v20.10.0
PS C:\Users\Vignesh\Desktop\awslambda> node dbManager.js

--- Product Management ---
1. View Products
2. Add Product
3. Update Product
4. Delete Product
5. Exit

Select an option: 1
Products:
{
  _id: new ObjectId('6572dd2dd234a3d50ec8b4fb'),
  product: 'bannana',
  price: '20'
}
{
  _id: new ObjectId('6572dd6cd234a3d50ec8b4fc'),
  product: 'apple',
  price: '10'
}
{
  _id: new ObjectId('6572dd85d234a3d50ec8b4fd'),
  product: 'mango',
  price: '10'
}
{
  _id: new ObjectId('6572ec13585c9b1df11e98b0'),
  name: 'Lichie',
  price: 30
}
>
```

```
const { MongoClient, ObjectId } = require("mongodb");
const readline = require("readline");

const url =

"mongodb+srv://vigneshmongo:mongo@vigneshmongo.itudsgr.mongodb.net/?retryW
rites=true&w=majority";
const dbName = "store";
const collectionName = "products";

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
```

```
});

async function mainMenu() {
  console.log("\n--- Product Management ---");
  console.log("1. View Products");
  console.log("2. Add Product");
  console.log("3. Update Product");
  console.log("4. Delete Product");
  console.log("5. Exit\n");

  const option = await prompt("Select an option: ");
  switch (option) {
    case "1":
      await getProducts();
      break;
    case "2":
      await createProduct();
      break;
    case "3":
      await updateProduct();
      break;
    case "4":
      await deleteProduct();
      break;
    case "5":
      console.log("Exiting...");
      process.exit(0);
    default:
      console.error("Invalid option. Please try again.");
      await mainMenu();
  }
}

async function prompt(question) {
  return new Promise((resolve) => {
    rl.question(question, (answer) => resolve(answer));
  });
}

async function connectToDb() {
```

```
  try {
    const client = await MongoClient.connect(url);
    const db = client.db(dbName);
    const collection = db.collection(collectionName);
    return { client, collection };
  } catch (error) {
    console.error("Error connecting to MongoDB:", error);
    throw error;
  }
}
```

Question 2: Creating Pre-Signed URLs for S3 Operations
How would you generate a pre-signed URL for both uploading and downloading files from an S3 bucket using the AWS SDK in a serverless environment? Include relevant code snippets for generating these pre-signed URLs and highlight any security considerations.




```
const {
```

```
  S3Client,
  PutObjectCommand,
  GetObjectCommand,
  HeadObjectCommand,
} = require("@aws-sdk/client-s3");

const bucketName = "myvigbucket";
const region = "US East(N.Virginia)us-east-1";
const fileName = "file.txt";
const expirationTime = 3600; // URL expires in 1 hour

const s3Client = new S3Client({ region });

async function generateUploadURL() {
  const params = {
    Bucket: bucketName,
    Key: fileName,
    Expires: expirationTime,
  };

  const command = new PutObjectCommand(params);
  const url = await s3Client.getSignedUrl(command);

  console.log("Upload pre-signed URL:", url);
}

async function getDownloadURL() {
  const params = { Bucket: bucketName, Key: fileName };

  const headObjectCommand = new HeadObjectCommand(params);
  const objectExists = await s3Client.headObject(headObjectCommand);

  if (objectExists) {
    const command = new GetObjectCommand(params);
    const url = await s3Client.getSignedUrl(command);

    console.log("Download pre-signed URL:", url);
  } else {
    console.error("File does not exist in the S3 bucket.");
  }
```

```
}

generateUploadURL();
getDownloadURL();
```

Question 3: Writing serverless.yml for Deployment
Write a sample serverless.yml file for deploying the AWS Lambda functions created in Question 1 and Question 2 via AWS API Gateway. Include the necessary configuration for integrating the Lambda functions with the API Gateway, specifying the HTTP methods, and defining the resource paths. Ensure that the deployment also includes any required IAM roles or permissions for accessing MongoDB and S3.

```
service: product-management
provider:
  name: aws
  runtime: nodejs16.x
  region: us-east-1

functions:

  createProduct:
    handler: main.createProduct
    events:

      - http:
          path: /products
          method: post
          authorizer: aws_iam

    role: arn:aws:iam::281604526131:role/product-management-role


  getProducts:
    handler: main.getProducts
    events:

      - http:
          path: /products
          method: get
```

```
        authorizer: aws_iam

  role: arn:aws:iam::281604526131:role/product-management-read-role


updateProduct:
  handler: main.updateProduct
  events:

    - http:
        path: /products/{id}
        method: patch
        authorizer: aws_iam

  role: arn:aws:iam::281604526131:role/product-management-write-role


deleteProduct:
  handler: main.deleteProduct
  events:

    - http:
        path: /products/{id}
        method: delete
        authorizer: aws_iam

  role: arn:aws:iam::281604526131:role/product-management-write-role
```