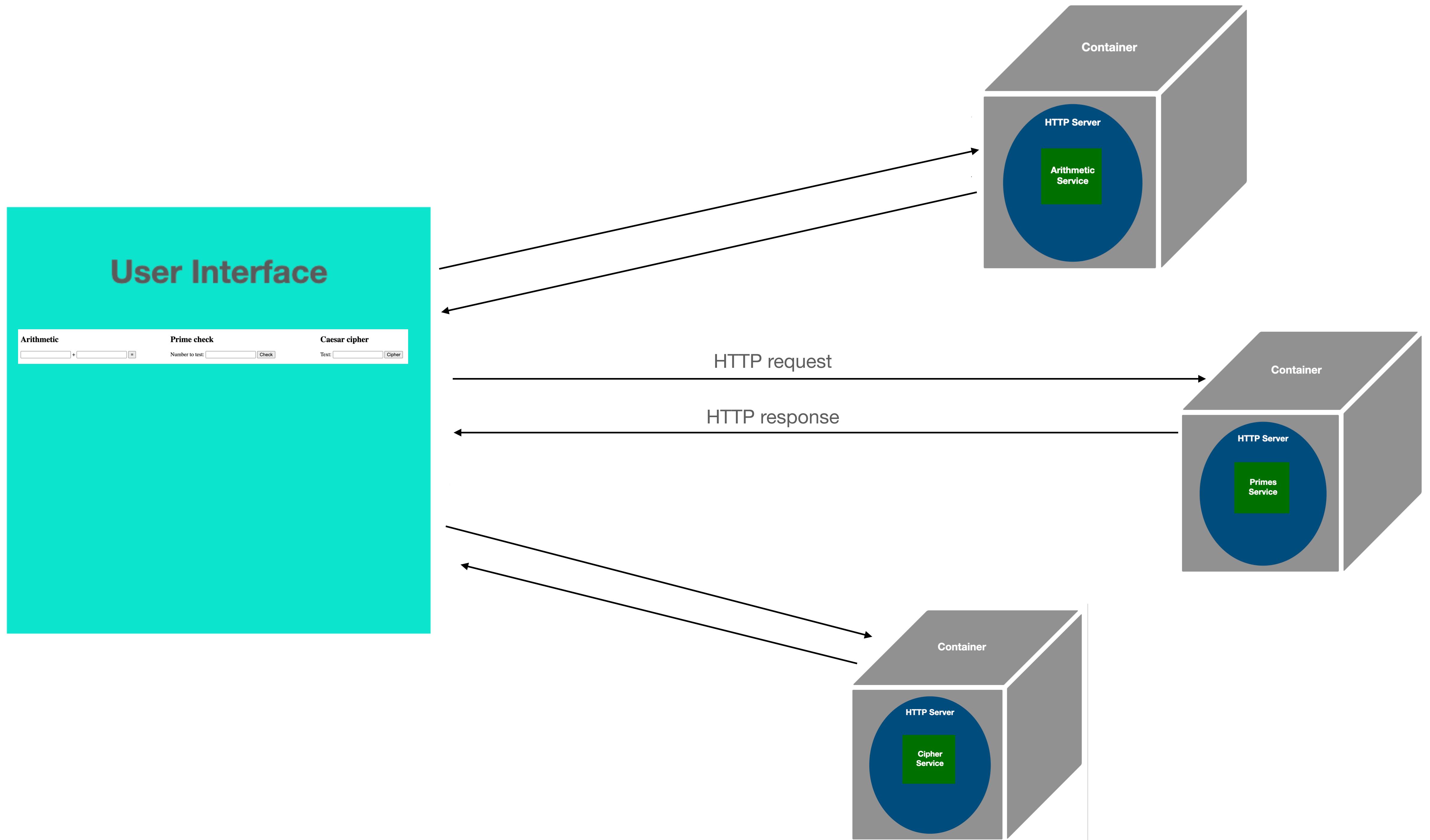


Practicum 10

CSCI-P 465/565 Software Engineering I

Indiana University Bloomington - Spring 2024

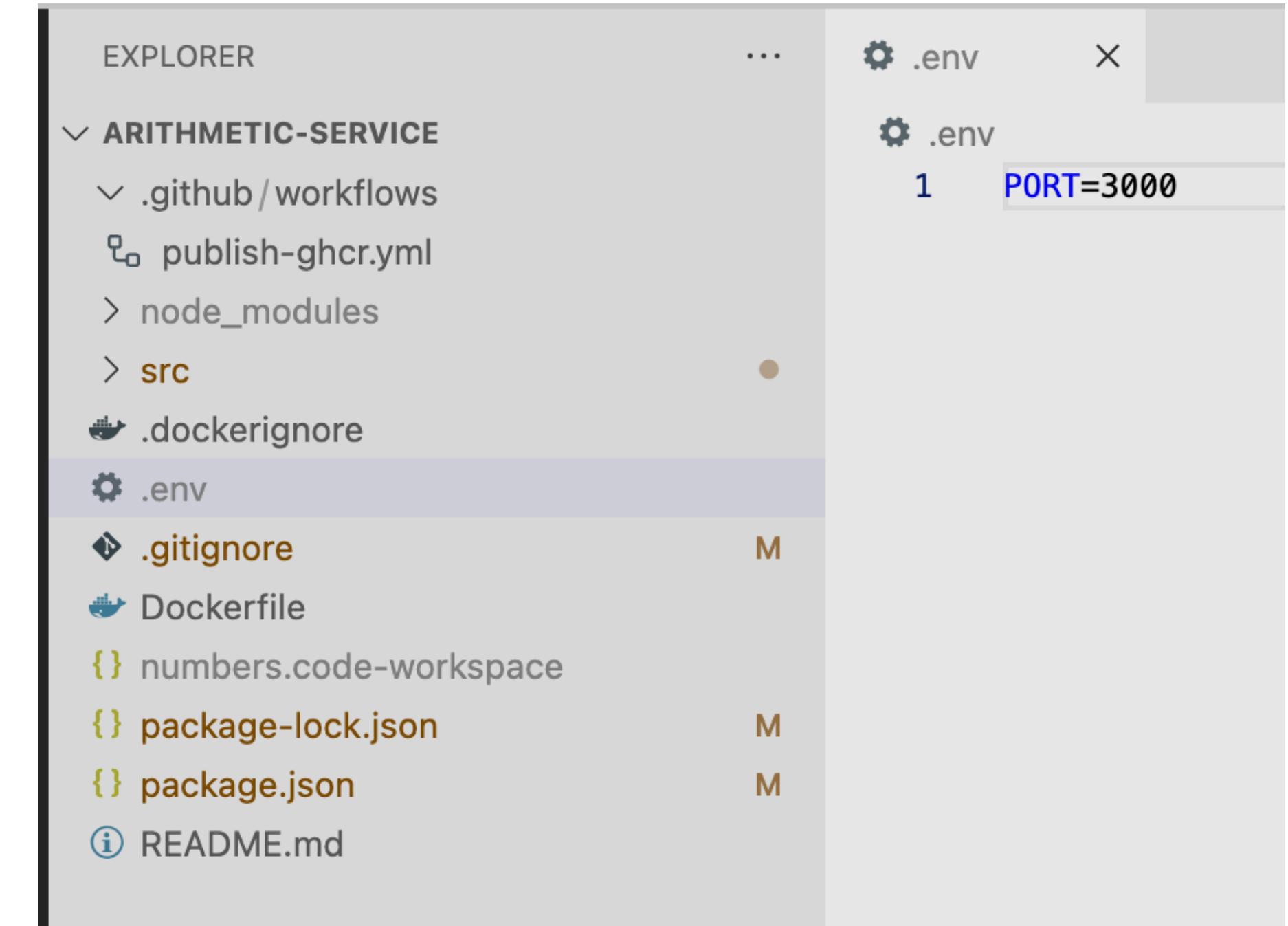


New tools

- Docker Compose
 - Docker Compose is included with Docker and allows us to configure, build, run, and manage multiple containers at the same time. It's useful for development, but we won't use it in production.
- Azure Storage
 - Azure Storage is a service to store files in the cloud. We can manage the assets through the Azure Portal, through the APIs, or from the command line. We'll upload a video through the Azure Portal and then use the Node.js Azure Storage SDK to read it back.
- MongoDB
 - MongoDB is a popular NoSQL type of database. It's lightweight, easy to set up and use, and it's convenient for microservices.

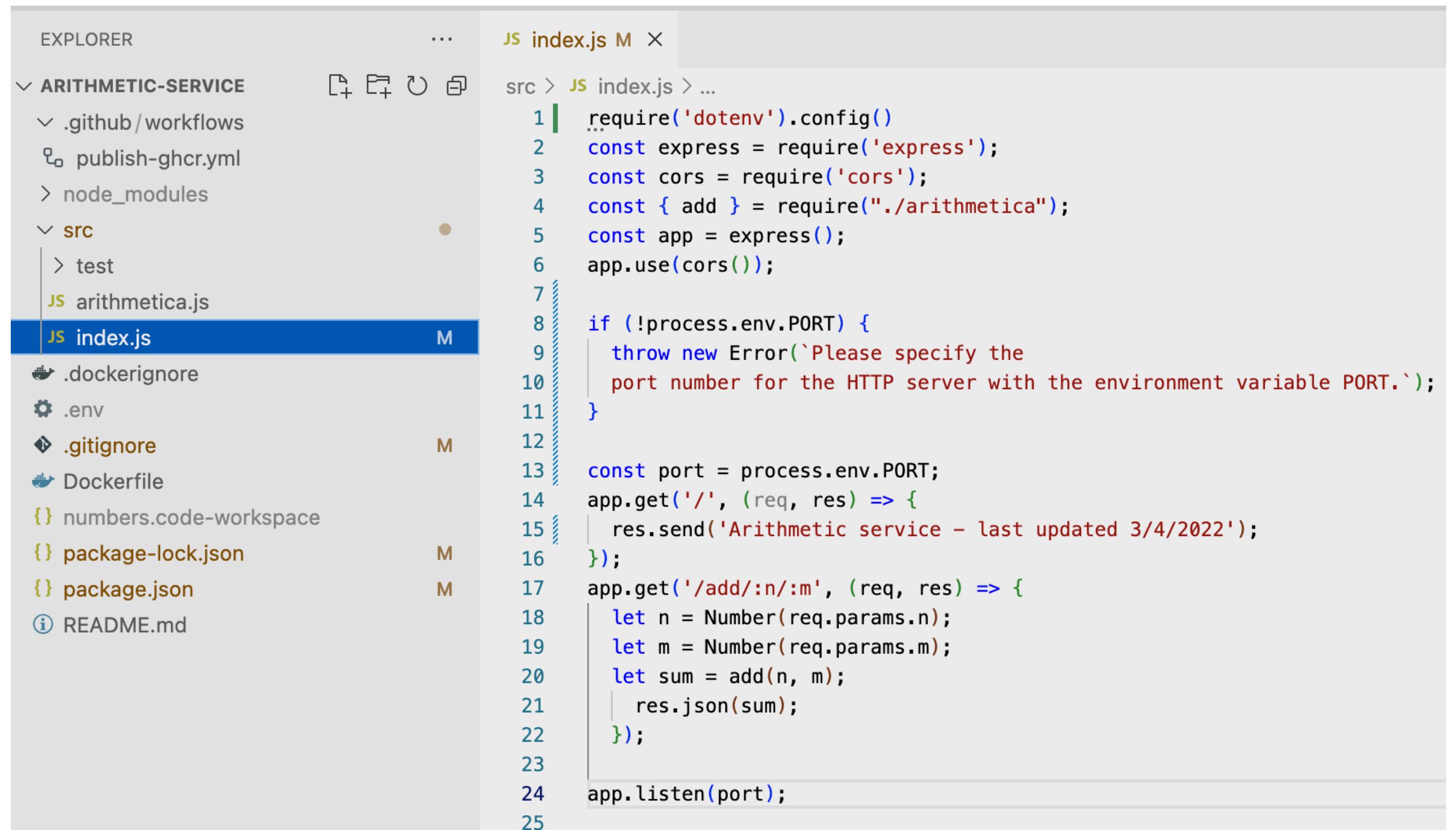
Steps

- Add a file called `.env` to your arithmetic-service folder.
- The `.env` file will store sensitive configuration data that you don't want to push to Github. Make sure you add the `.env` file to your `.gitignore`.



Steps

- Modify the index.js file to read the port number from the .env file.



The screenshot shows a code editor interface with two main sections: the Explorer view on the left and the code editor on the right.

EXPLORER pane:

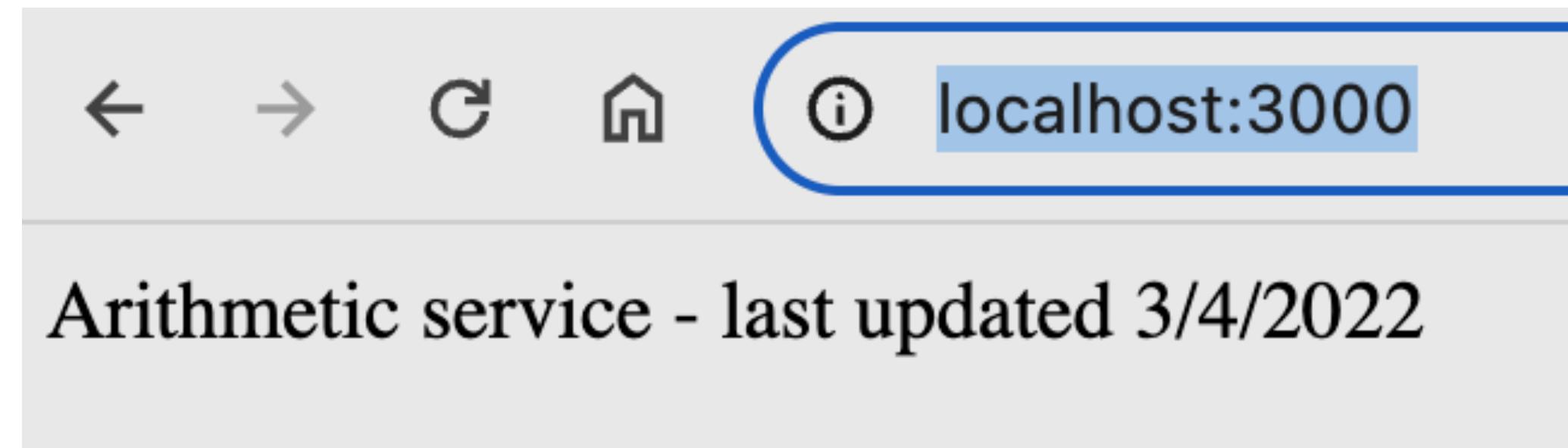
- ARITHMETIC-SERVICE folder
 - .github/workflows/publish-ghcr.yml
 - node_modules
 - src
 - test
 - arithmetica.js
 - index.js** (selected)
 - .dockerignore
 - .env
 - .gitignore
 - Dockerfile
 - numbers.code-workspace
 - package-lock.json
 - package.json
 - README.md

index.js code editor pane:

```
src > JS index.js > ...
1 | ...require('dotenv').config()
2 | const express = require('express');
3 | const cors = require('cors');
4 | const { add } = require("./arithmetica");
5 | const app = express();
6 | app.use(cors());
7 |
8 | if (!process.env.PORT) {
9 |   throw new Error(`Please specify the
10 |   port number for the HTTP server with the environment variable PORT.`);
11 |
12 |
13 | const port = process.env.PORT;
14 | app.get('/', (req, res) => {
15 |   res.send('Arithmetic service - last updated 3/4/2022');
16 | });
17 | app.get('/add/:n/:m', (req, res) => {
18 |   let n = Number(req.params.n);
19 |   let m = Number(req.params.m);
20 |   let sum = add(n, m);
21 |   res.json(sum);
22 | });
23 |
24 | app.listen(port);
25 |
```

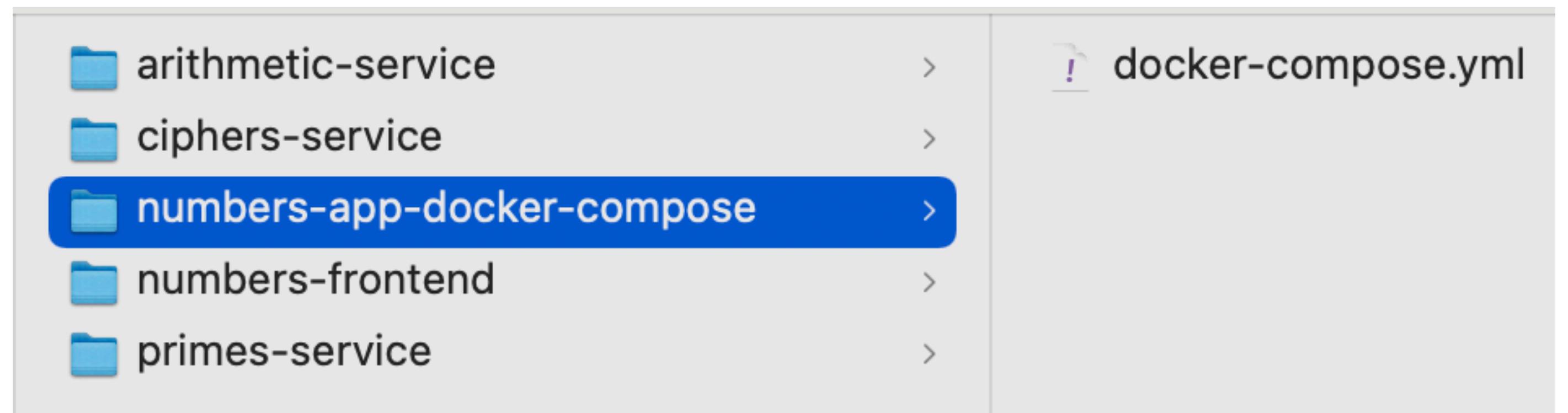
Steps

- Install the dotenv module:
npm install dotenv
- Run the service:
node ./src/index.js
- Verify that the server is working:
http://localhost:3000/



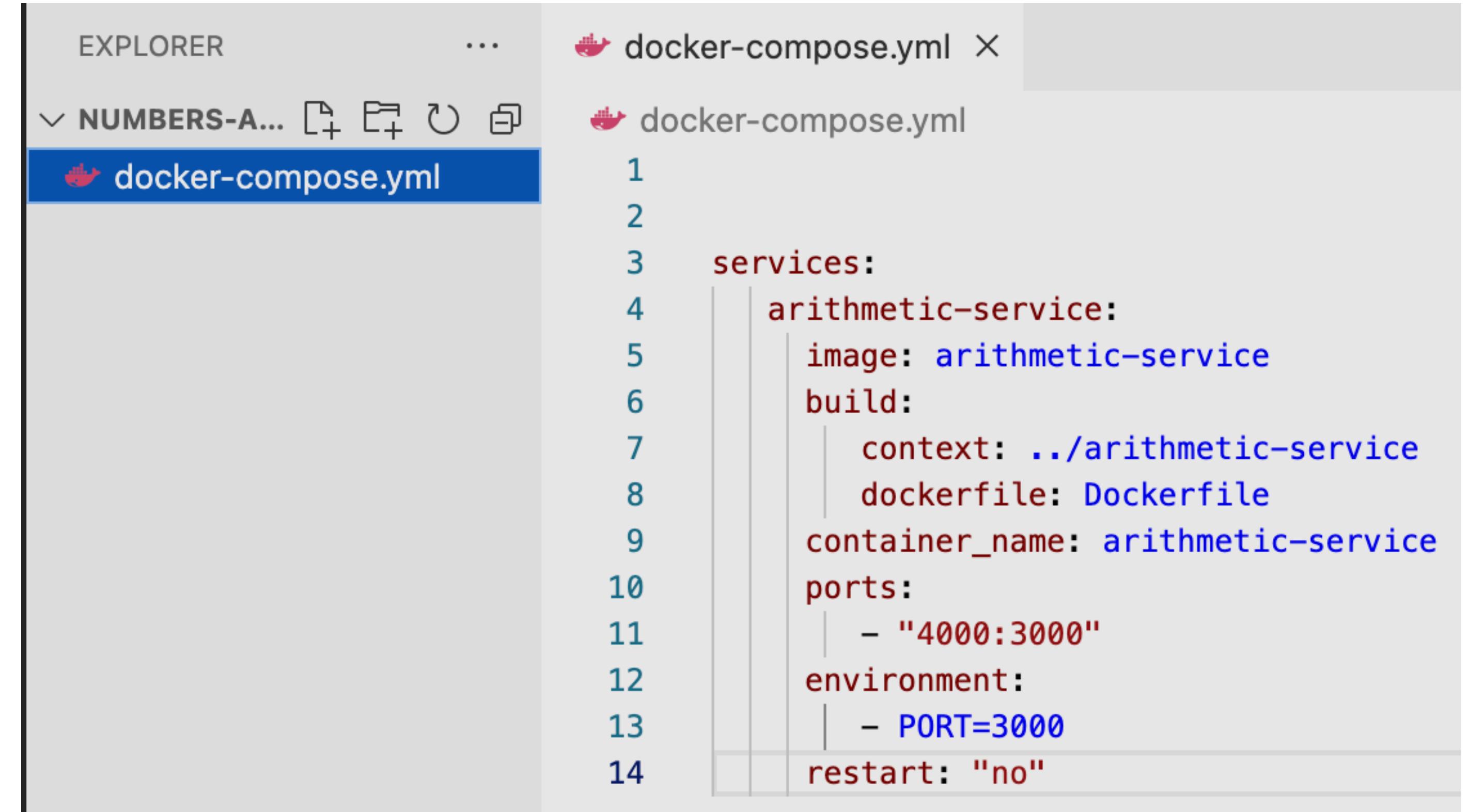
Steps

- Add a folder at the same level as your other backend folders and add a yaml file called docker-compose.yml to it.



Steps

- Add the following to the yaml file to describe the image and the container that we want to be created for our arithmetic-service.

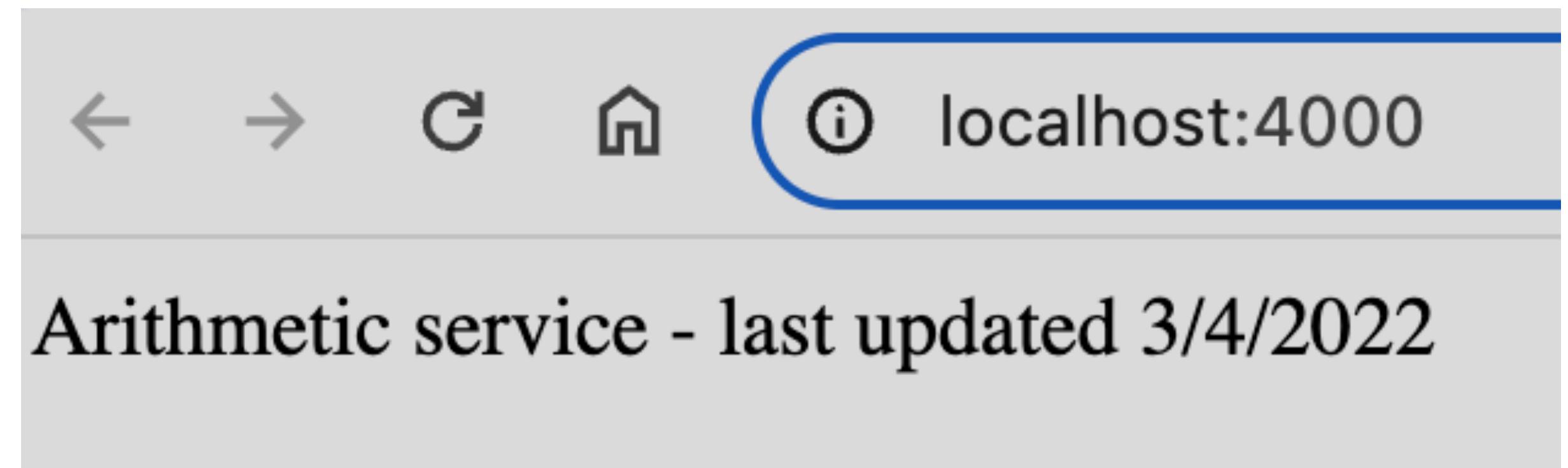


The screenshot shows a code editor interface with an 'EXPLORER' sidebar on the left. In the main area, there are two tabs: 'docker-compose.yml' (which is selected) and another 'docker-compose.yml' tab. The selected tab contains the following YAML code:

```
1
2
3 services:
4   arithmetic-service:
5     image: arithmetic-service
6     build:
7       context: ../arithmetic-service
8       dockerfile: Dockerfile
9     container_name: arithmetic-service
10    ports:
11      - "4000:3000"
12    environment:
13      - PORT=3000
14    restart: "no"
```

Steps

- Run the docker compose:
docker compose up --build
- Verify that the container is running:
<http://localhost:4000/>



Steps

- To delete the container(s), run the following in the same folder where your docker-compose is located:
docker compose down --volumes

Steps

- Or we can combine both to reboot the application when we update the code:
docker compose down --volumes && docker compose up –build
- You can create a bash file (or bat file for windows machines) containing the above command so that every time you need to stop-delete-create-start the docker images and containers, you can simply run the bash or bat file.

Steps

```
$ reboot.sh
1 #!/usr/bin/env bash
2
3 docker compose down --volumes && docker compose up --build
```

```
@echo off
2
3 docker "compose" "down" "--volumes" && docker "compose" "up" "--build"
```

Data Management

Tow types of data

- File storage
- Database
 - SQL databases
 - NOSQL databases

File storage using Azure

- *Azure Storage* is a Microsoft Azure service for hosting private or public files in the cloud. You upload your files to Azure Storage and can then access these through the Azure Storage API.

Steps

- Create a free Azure for students account: <https://azure.microsoft.com/en-us/free/students>
- Login to Azure portal: <https://portal.azure.com>

Steps

- Create a storage account called “practicums”:

Microsoft Azure Search resources, services, and docs (G+)

Home > Storage accounts > Create a storage account ...

Basics Advanced Networking Data protection Encryption Tags Review

Subscription * Azure for Students

Resource group * (New) practicums [Create new](#)

Instance details

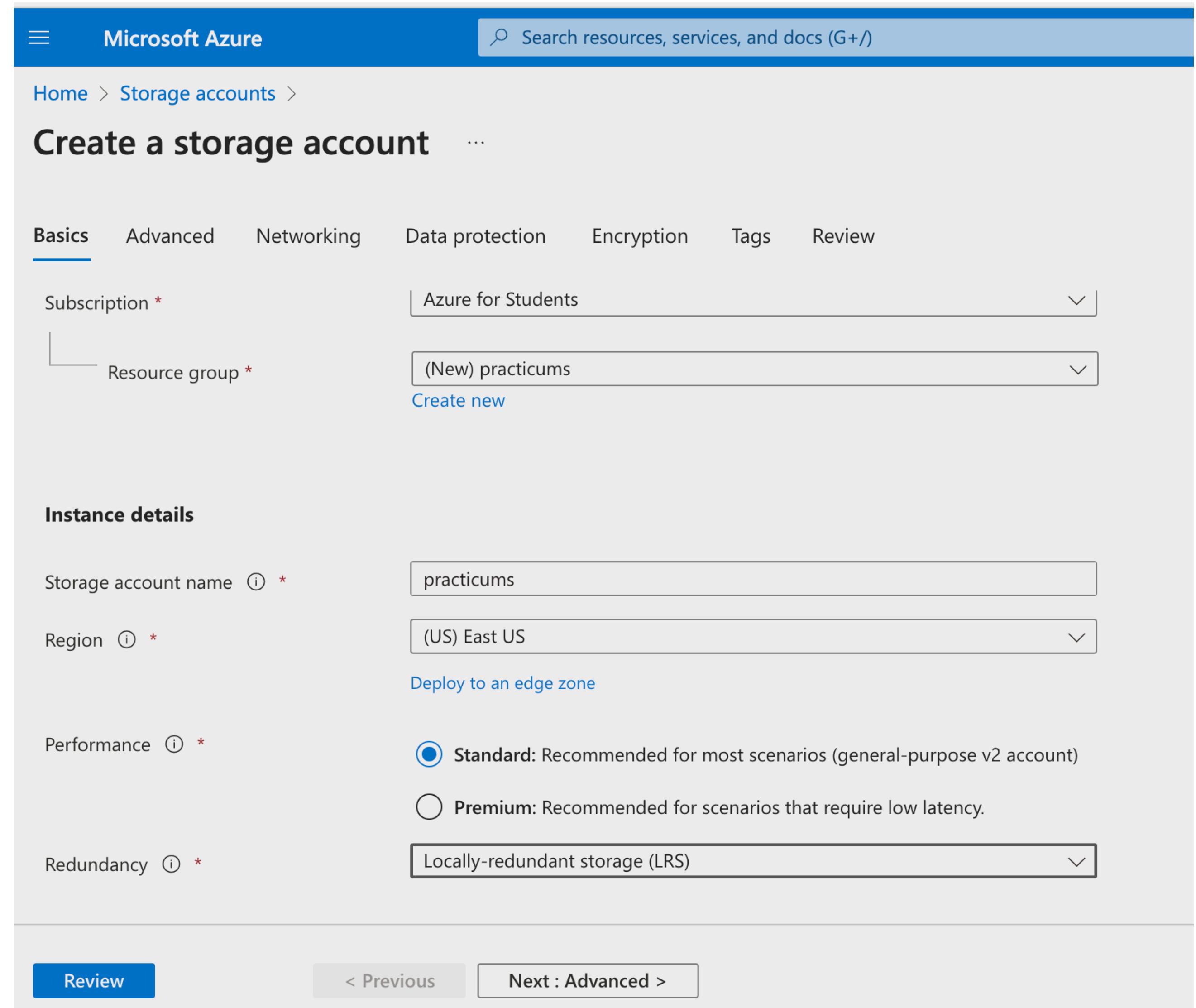
Storage account name ⓘ * practicums

Region ⓘ * (US) East US [Deploy to an edge zone](#)

Performance ⓘ * Standard: Recommended for most scenarios (general-purpose v2 account) Premium: Recommended for scenarios that require low latency.

Redundancy ⓘ * Locally-redundant storage (LRS)

[Review](#) [Next : Advanced >](#) [< Previous](#)



Steps

- Create a storage account called “practicums”:

Microsoft Azure Search resources, services, and docs (G+)

Home > Storage accounts > Create a storage account ...

Basics Advanced Networking Data protection Encryption Tags Review

Subscription * Azure for Students

Resource group * (New) practicums [Create new](#)

Instance details

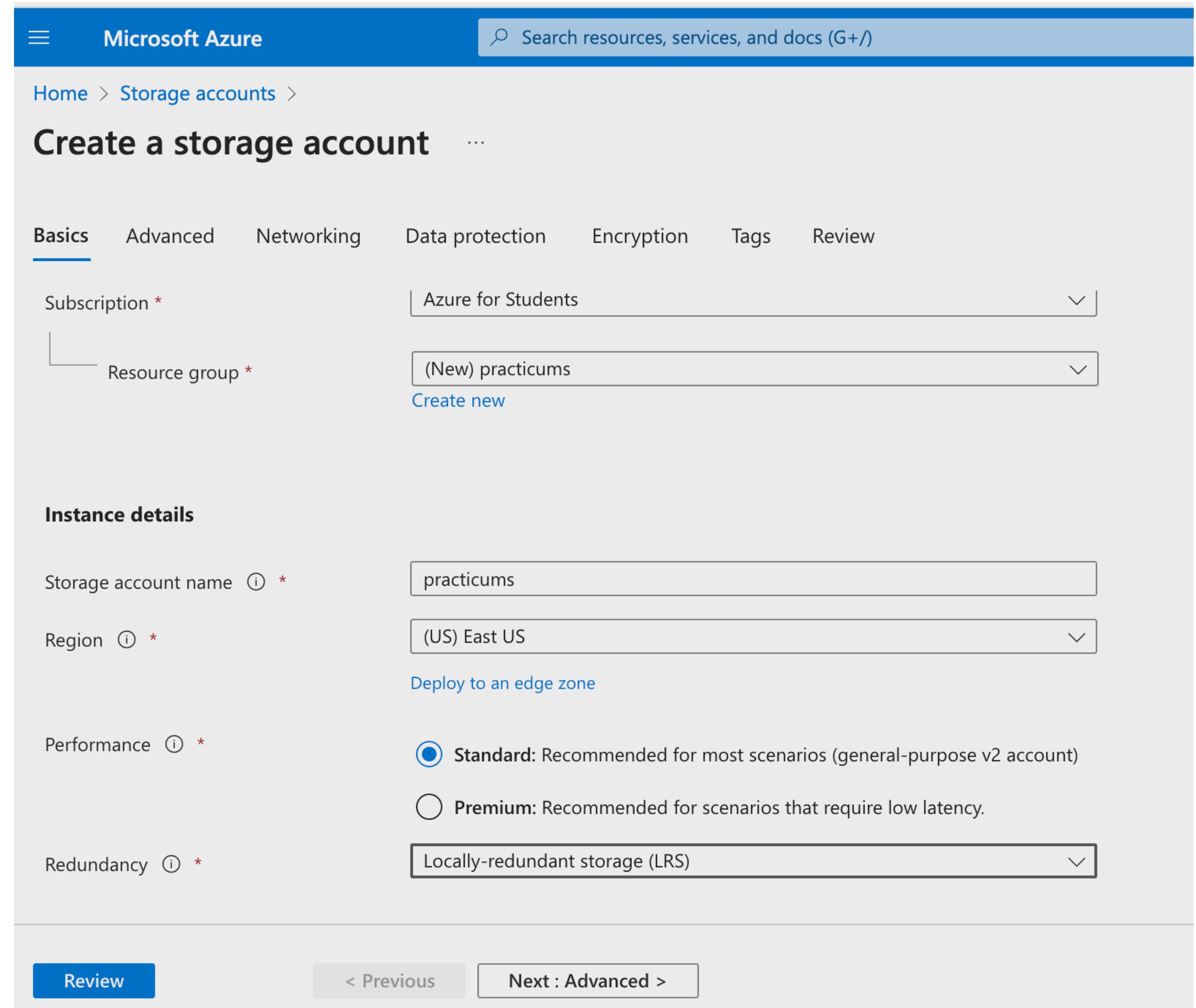
Storage account name ⓘ * practicums

Region ⓘ * (US) East US [Deploy to an edge zone](#)

Performance ⓘ * Standard: Recommended for most scenarios (general-purpose v2 account) Premium: Recommended for scenarios that require low latency.

Redundancy ⓘ * Locally-redundant storage (LRS)

[Review](#) [Next : Advanced >](#) [< Previous](#)



Steps

- Then create a folder called “philosophers” and add a picture file to it. In Azure storage system a folder is called a container! But that “container” has nothing to do with “docker container”. It is simply a folder.

The screenshot shows the Microsoft Azure Storage Accounts interface. The left sidebar lists 'Storage accounts' for Indiana University (indiana.onmicrosoft.com), with 'practicums' selected. The main pane displays the 'Containers' section for the 'practicums' storage account. It includes a search bar, a list of containers, and various management options. The table shows the following data:

Name	Last modified	Anonymous access le...	Lease state
\$logs	3/4/2024, 8:41:17 PM	Private	Available
philosophers	3/4/2024, 10:55:07 PM	Private	Available

The azure-storage-service

Steps

- Create a folder called “azure-storage-service” in the same folder where your other services are located.
- Change into the new directory:
 - **cd azure-storage-service**
- Create an empty nodejs project:
 - **npm init -y**
- Install Express:
 - **npm install express**

Steps

- Add the azure storage package to your service:
npm install --save @azure/storage-blob
- Install dotenv module: **npm install dotenv**
- Install nodemon to run the server in the development mode:
npm install --save-dev nodemon

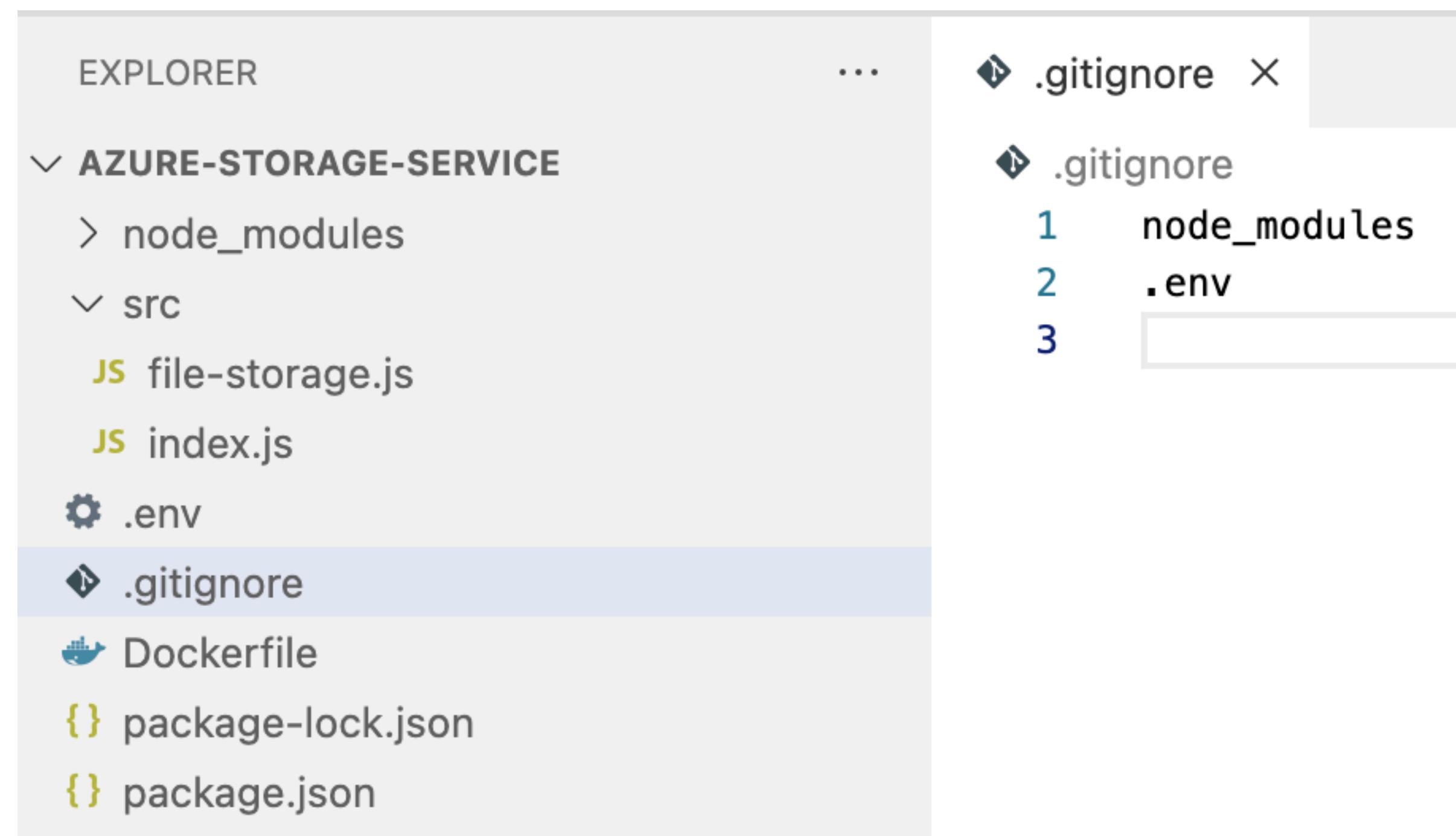
Steps

- Add the start commands to the package.json file:

```
"description": "",  
"main": "index.js",  
  ▷ Debug  
"scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1",  
    "start": "node ./src/index.js",  
    "start:development": "nodemon ./src/index.js"  
},  
"keywords": [],  
"author": "",  
"license": "ISC",  
"dependencies": {  
    "@azure/storage-blob": "^12.17.0",  
    "dotenv": "^16.4.5",  
    "express": "^4.18.3"  
},  
"devDependencies": {  
    "nodemon": "^3.1.0"  
}  
}
```

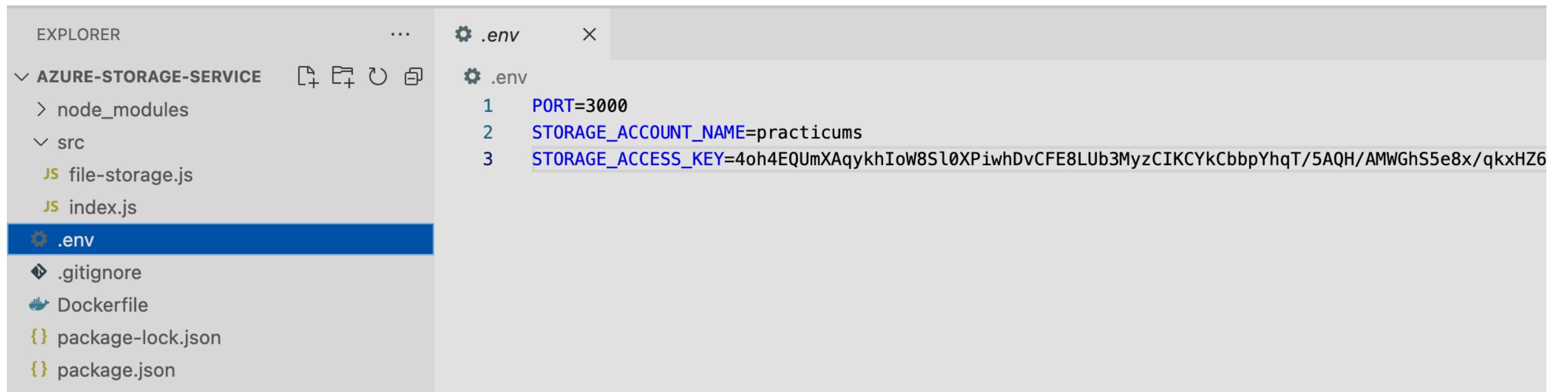
Steps

- Add a `.gitignore` file to the folder:



Steps

- Add a .env file to the folder.
- To find your access key go to your azure portal and navigate to your storage account. Under the Access Keys page you can find two keys. Use any of those.



```
PORT=3000
STORAGE_ACCOUNT_NAME=practicums
STORAGE_ACCESS_KEY=4oh4EQUmXAqykhIoW8Sl0XPiwhDvCFE8LUb3MyzCIKCYkCbbpYhqT/5AQH/AMWGhS5e8x/qkxHZ6
```

Steps

- Create a src folder and add a file called “file-storage.js”.

The screenshot shows a code editor interface with two main panes. On the left is the Explorer pane, which displays the project structure:

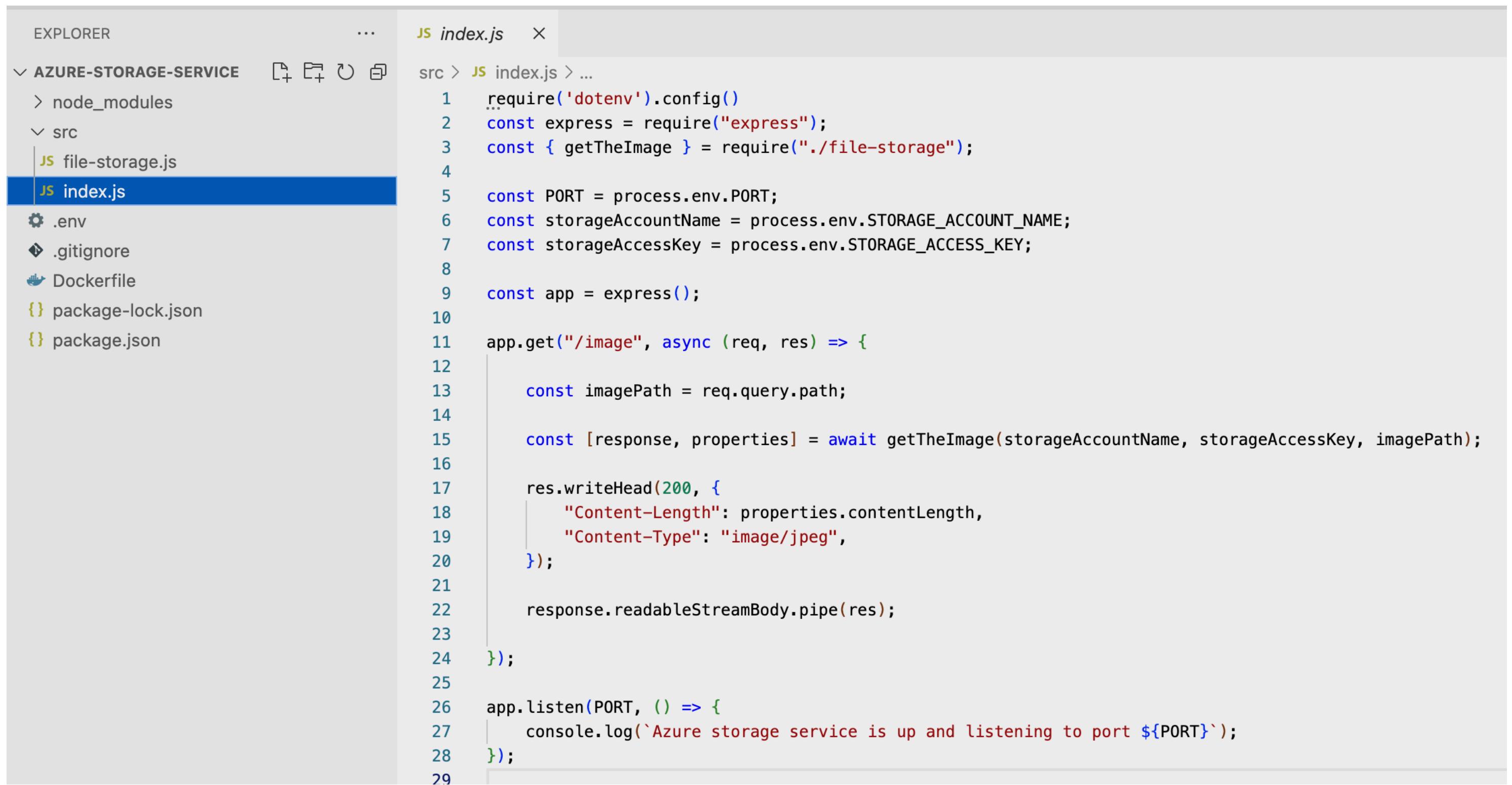
- AZURE-STORAGE-SERVICE
 - node_modules
 - src
 - file-storage.js
 - index.js
 - .env
 - .gitignore
 - Dockerfile
 - package-lock.json
 - package.json

The right pane is the Editor pane, showing the content of the file `file-storage.js`:

```
JS file-storage.js M X
src > JS file-storage.js > ...
1  const { BlobServiceClient, StorageSharedKeyCredential } = require("@azure/storage-blob");
2
3
4  function createBlobService(accountName, accessKey) {
5      const sharedKeyCredential = new StorageSharedKeyCredential(accountName, accessKey);
6      const blobService = new BlobServiceClient(
7          `https://${accountName}.blob.core.windows.net`,
8          sharedKeyCredential
9      );
10     return blobService;
11 }
12
13 async function getTheImage(accountName, accessKey, imagePath) {
14
15     const blobService = createBlobService(accountName, accessKey);
16
17     const containerName = "philosophers";
18     const containerClient = blobService.getContainerClient(containerName);
19     const blobClient = containerClient.getBlobClient(imagePath);
20
21     const properties = await blobClient.getProperties();
22     const response = await blobClient.download();
23     return [response, properties];
24 }
25
26
27 module.exports = { getTheImage }
```

Steps

- In the src folder add the index.js file:



The screenshot shows a code editor interface with two main sections: an Explorer sidebar on the left and a code editor on the right.

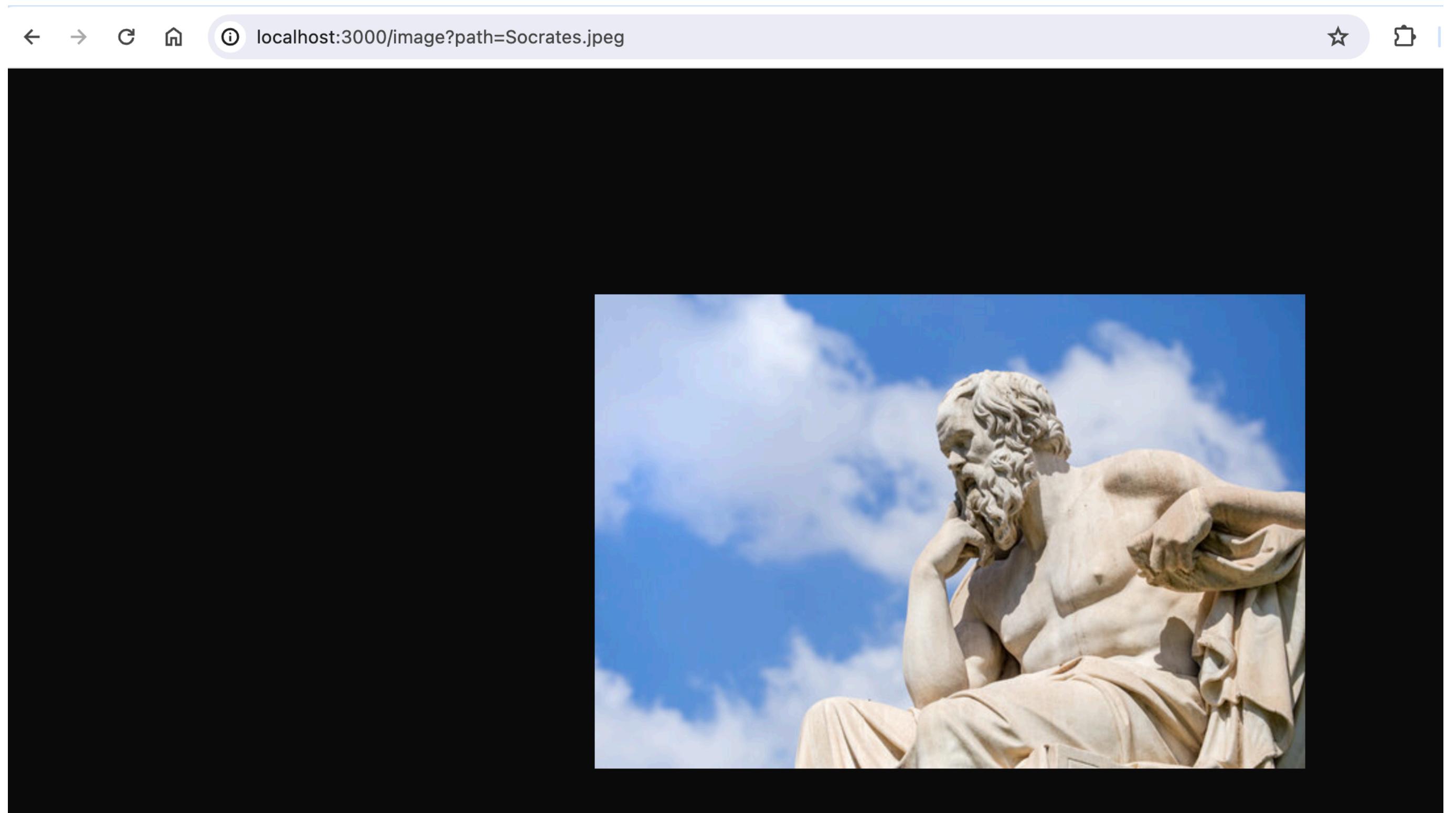
EXPLORER pane (left):

- AZURE-STORAGE-SERVICE
 - node_modules
 - src
 - file-storage.js
 - index.js** (highlighted with a blue selection bar)
- .env
- .gitignore
- Dockerfile
- package-lock.json
- package.json

```
JS index.js  X
src > JS index.js > ...
1  require('dotenv').config()
2  const express = require("express");
3  const { getTheImage } = require("./file-storage");
4
5  const PORT = process.env.PORT;
6  const storageAccountName = process.env.STORAGE_ACCOUNT_NAME;
7  const storageAccessKey = process.env.STORAGE_ACCESS_KEY;
8
9  const app = express();
10
11 app.get("/image", async (req, res) => {
12
13   const imagePath = req.query.path;
14
15   const [response, properties] = await getTheImage(storageAccountName, storageAccessKey, imagePath);
16
17   res.writeHead(200, {
18     "Content-Length": properties.contentLength,
19     "Content-Type": "image/jpeg",
20   });
21
22   response.readableStreamBody.pipe(res);
23
24 );
25
26 app.listen(PORT, () => {
27   console.log(`Azure storage service is up and listening to port ${PORT}`);
28 });
29
```

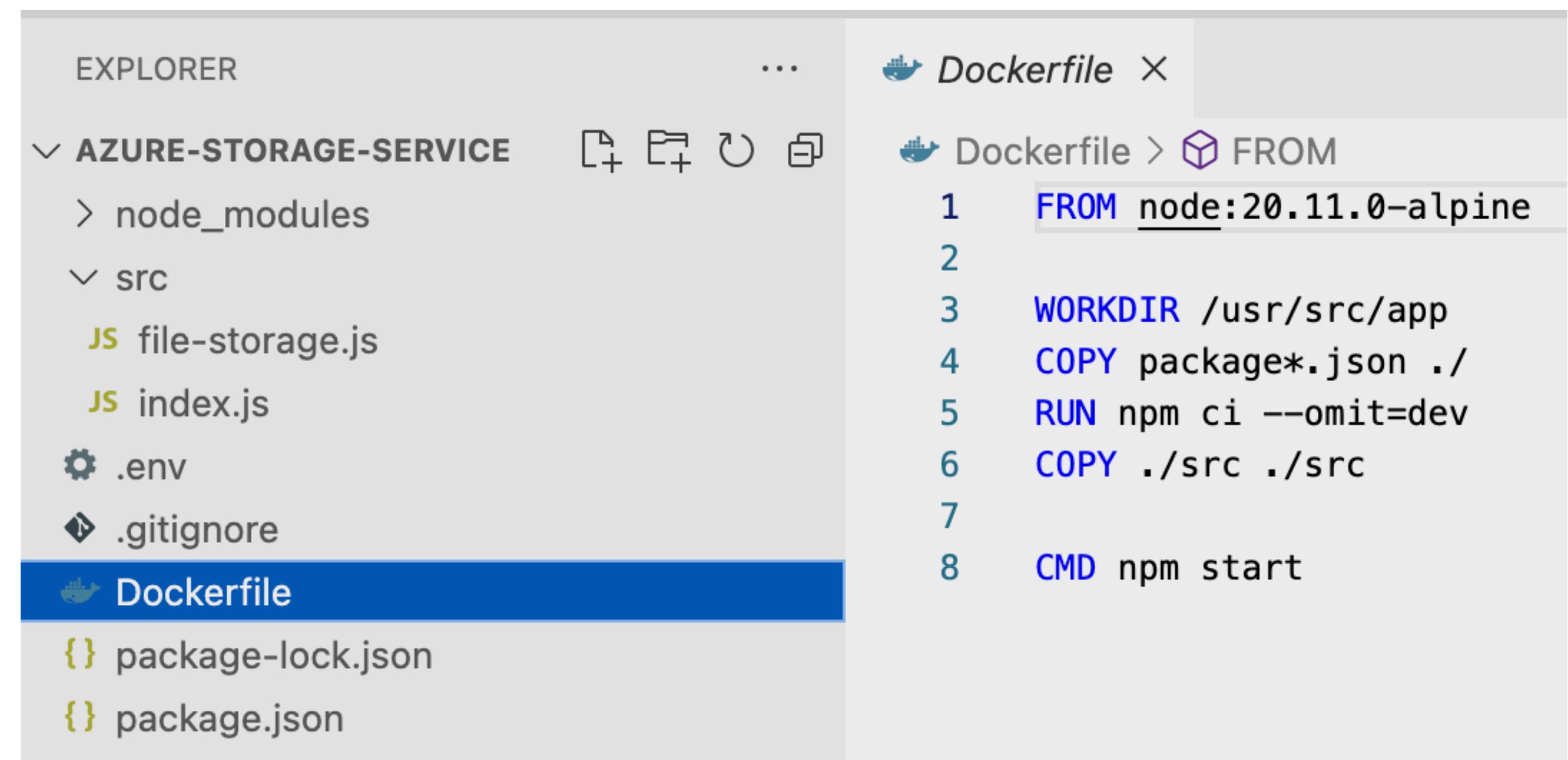
Steps

- Run the application in the development mode:
npm run start:development
- Verify that it is working.



Steps

- Add the docker file:



The screenshot shows the VS Code interface with two tabs open: 'EXPLORER' on the left and 'Dockerfile' on the right.

In the EXPLORER tab, the project structure is visible:

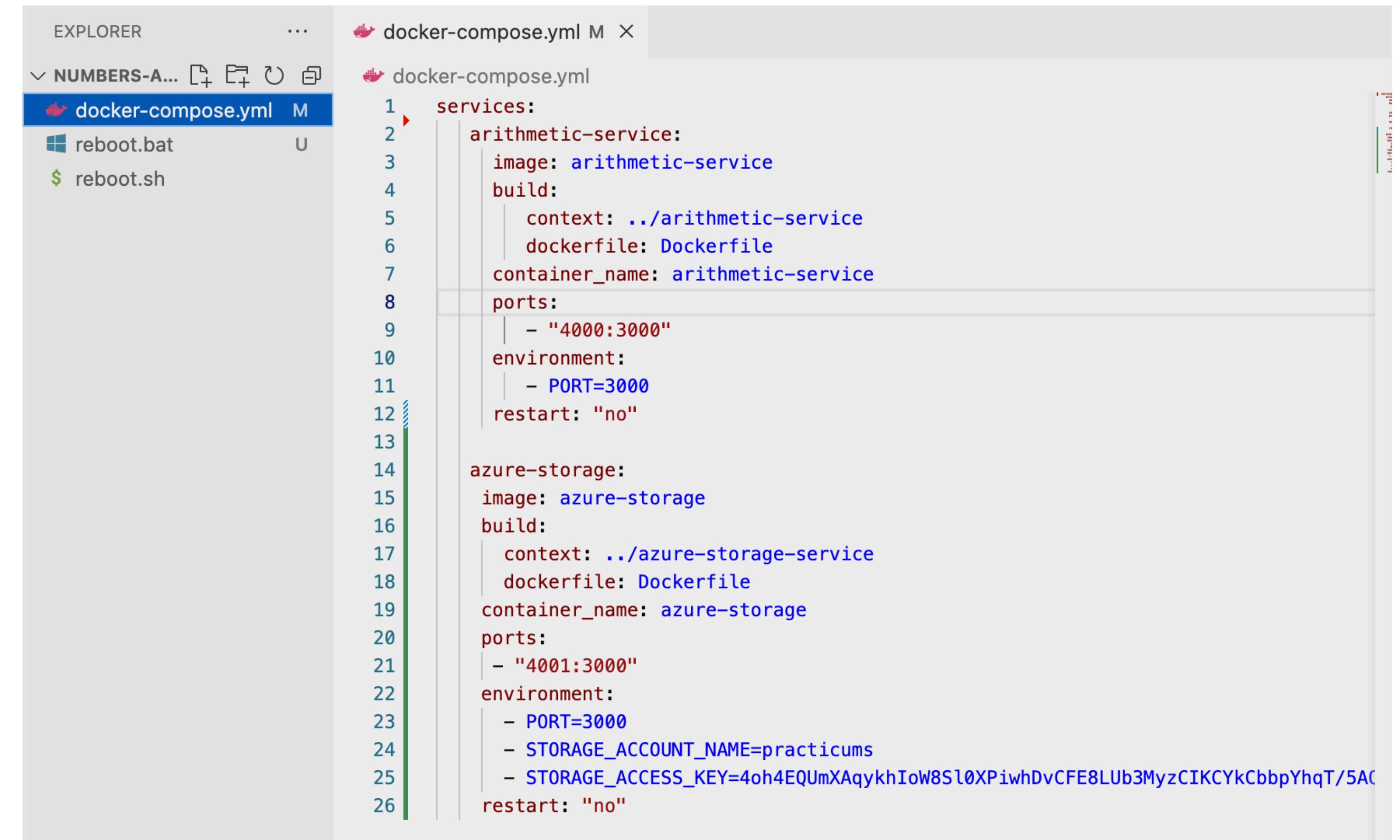
- AZURE-STORAGE-SERVICE
 - node_modules
 - src
 - file-storage.js
 - index.js
 - .env
 - .gitignore
- Dockerfile
- package-lock.json
- package.json

The 'Dockerfile' tab is active, showing the following Dockerfile content:

```
FROM node:20.11.0-alpine
WORKDIR /usr/src/app
COPY package*.json .
RUN npm ci --omit=dev
COPY ./src ./src
CMD npm start
```

Steps

- Add the service to your docker-compose file:



The screenshot shows a code editor interface with an 'EXPLORER' sidebar on the left containing files: 'docker-compose.yml' (selected), 'reboot.bat', and 'reboot.sh'. The main area displays the 'docker-compose.yml' file content, which defines two services: 'arithmetic-service' and 'azure-storage'. The 'arithmetic-service' is based on the 'arithmetic-service' image, built from the 'Dockerfile' in the 'arithmetic-service' context, and runs on port 3000. The 'azure-storage' service is based on the 'azure-storage' image, built from the 'Dockerfile' in the 'azure-storage-service' context, and runs on port 3000. Both services have their 'restart' setting set to 'no'.

```
services:
  arithmetic-service:
    image: arithmetic-service
    build:
      context: ../arithmetic-service
      dockerfile: Dockerfile
    container_name: arithmetic-service
  ports:
    - "4000:3000"
  environment:
    - PORT=3000
  restart: "no"

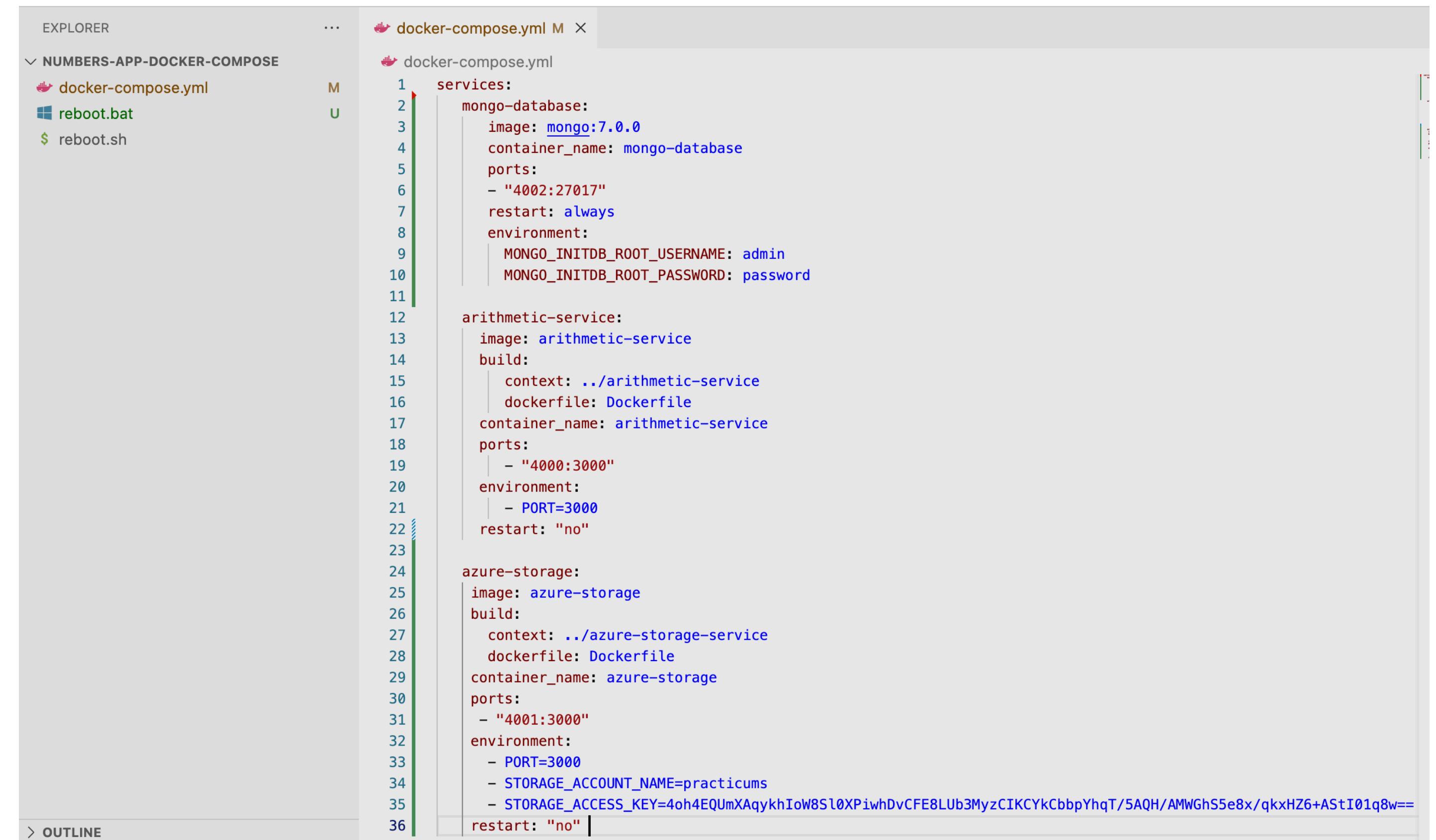
  azure-storage:
    image: azure-storage
    build:
      context: ../azure-storage-service
      dockerfile: Dockerfile
    container_name: azure-storage
  ports:
    - "4001:3000"
  environment:
    - PORT=3000
    - STORAGE_ACCOUNT_NAME=practicums
    - STORAGE_ACCESS_KEY=4oh4EQUmXaqykhIoW8Sl0XPiwhDvCFE8LUb3MyzCIKYkCbbpYhqT/5AC
  restart: "no"
```

- Run the reboot.sh file (reboot.bat file) and verify that both containers are up and running.

Adding a database

Steps

- Add a mongodb service to your docker-compose.yml file:

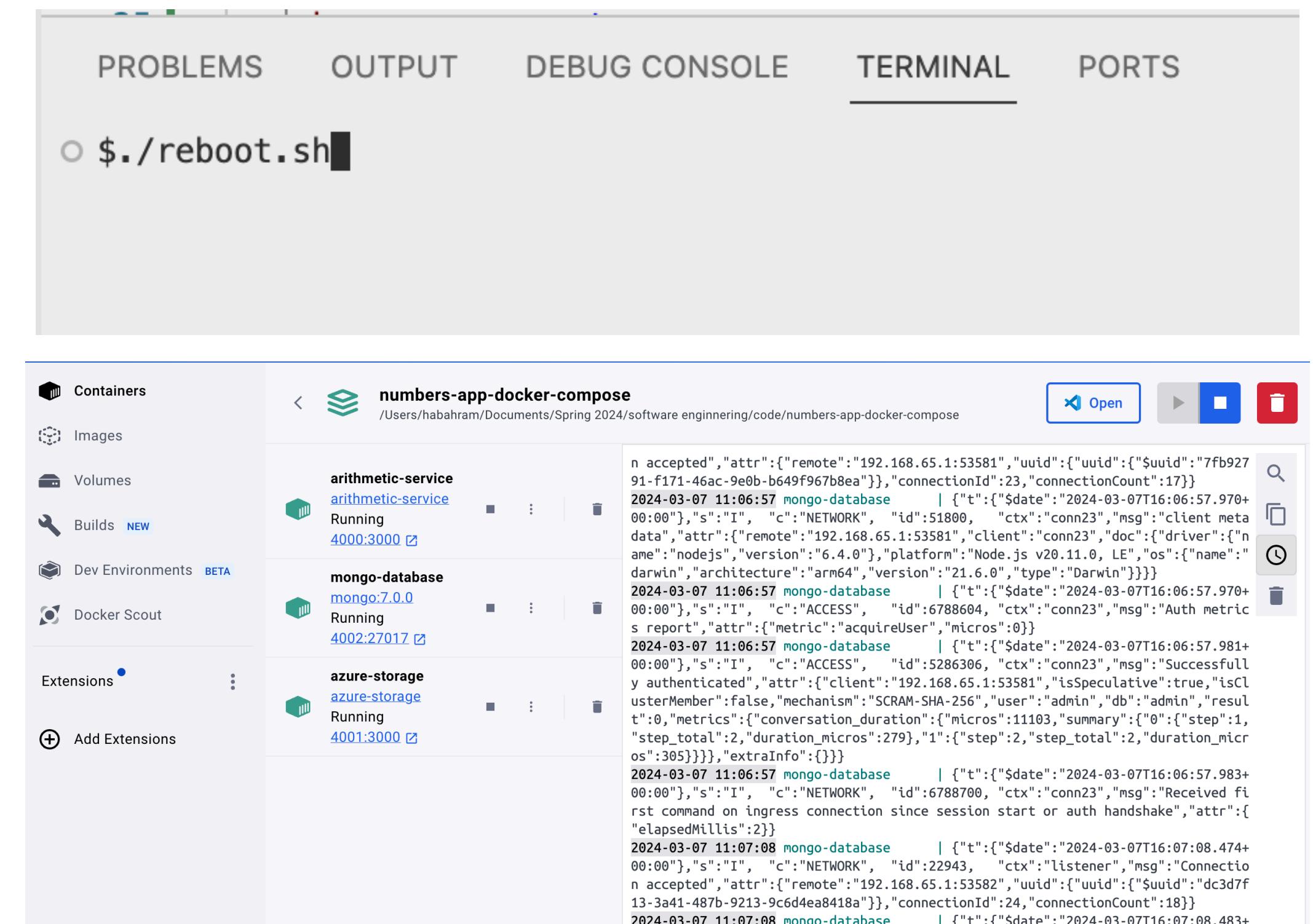


The screenshot shows a code editor with an 'EXPLORER' sidebar on the left containing files: 'docker-compose.yml' (marked with 'M'), 'reboot.bat', and 'reboot.sh'. The main area displays the 'docker-compose.yml' file content, which defines three services: 'mongo-database', 'arithmetic-service', and 'azure-storage'. The 'mongo-database' service uses the 'mongo:7.0.0' image, has port 4002 mapped to 27017, and runs with 'restart: always'. It also sets environment variables MONGO_INITDB_ROOT_USERNAME to 'admin' and MONGO_INITDB_ROOT_PASSWORD to 'password'. The 'arithmetic-service' uses the 'arithmetic-service' image, builds from the local directory, maps port 4000 to 3000, and runs with 'restart: "no"'. The 'azure-storage' service uses the 'azure-storage' image, builds from the local directory, maps port 4001 to 3000, and runs with 'restart: "no"'. Environment variables PORT=3000 and STORAGE_ACCOUNT_NAME=practicums are also defined.

```
version: '3.8'
services:
  mongo-database:
    image: mongo:7.0.0
    container_name: mongo-database
    ports:
      - "4002:27017"
    restart: always
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: password
  arithmetic-service:
    image: arithmetic-service
    build:
      context: ../arithmetic-service
      dockerfile: Dockerfile
    container_name: arithmetic-service
    ports:
      - "4000:3000"
    environment:
      - PORT=3000
    restart: "no"
  azure-storage:
    image: azure-storage
    build:
      context: ../azure-storage-service
      dockerfile: Dockerfile
    container_name: azure-storage
    ports:
      - "4001:3000"
    environment:
      - PORT=3000
      - STORAGE_ACCOUNT_NAME=practicums
      - STORAGE_ACCESS_KEY=4oh4EQUmXAqykhIoW8Sl0XPiwhDvCFE8LUb3MyzCIKCYkCbbpYhqT/5AQH/AMWGHs5e8x/qkxHZ6+ASTI01q8w==
    restart: "no"
```

Steps

- Restart the containers by running your reboot.sh (or reboot.bat on windows).



The screenshot shows the Docker extension interface in VS Code. At the top, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. In the TERMINAL tab, the command `./reboot.sh` is run, and the terminal output shows the containers being restarted.

Containers

- Images
- Volumes
- Builds NEW
- Dev Environments BETA
- Docker Scout
- Extensions •
- Add Extensions

numbers-app-docker-compose /Users/habahram/Documents/Spring 2024/software enginnering/code/numbers-app-docker-compose

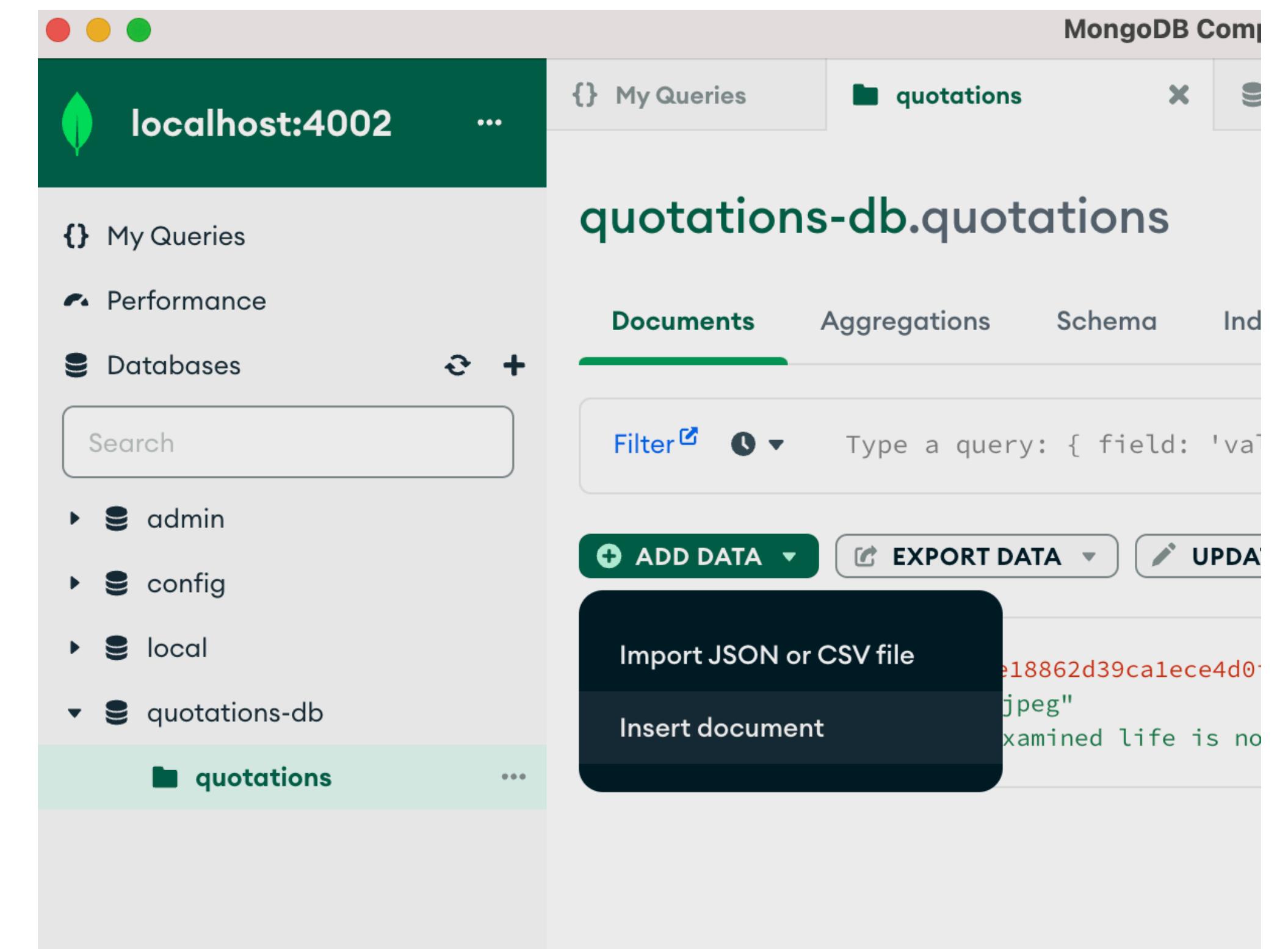
Service	Status	Port
arithmetic-service	Running	4000:3000
mongo-database	Running	4002:27017
azure-storage	Running	4001:3000

Terminal Output:

```
n accepted", "attr": {"remote": "192.168.65.1:53581", "uuid": {"$uuid": "7fb92791-f171-46ac-9e0b-b649f967b8ea"}}, "connectionId": 23, "connectionCount": 17}]}  
2024-03-07 11:06:57 mongo-database | {"t": {"$date": "2024-03-07T16:06:57.970+00:00"}, "s": "I", "c": "NETWORK", "id": 51800, "ctx": "conn23", "msg": "client metaData", "attr": {"remote": "192.168.65.1:53581", "client": "conn23", "doc": {"driver": {"name": "nodejs", "version": "6.4.0"}, "platform": "Node.js v20.11.0, LE", "os": {"name": "darwin", "architecture": "arm64", "version": "21.6.0", "type": "Darwin"}}}}  
2024-03-07 11:06:57 mongo-database | {"t": {"$date": "2024-03-07T16:06:57.970+00:00"}, "s": "I", "c": "ACCESS", "id": 6788604, "ctx": "conn23", "msg": "Auth metrics report", "attr": {"metric": "acquireUser", "micros": 0}}}  
2024-03-07 11:06:57 mongo-database | {"t": {"$date": "2024-03-07T16:06:57.981+00:00"}, "s": "I", "c": "ACCESS", "id": 5286306, "ctx": "conn23", "msg": "Successfully authenticated", "attr": {"client": "192.168.65.1:53581", "isSpeculative": true, "isClusterMember": false, "mechanism": "SCRAM-SHA-256", "user": "admin", "db": "admin", "result": 0, "metrics": {"conversation_duration": {"micros": 1103, "summary": {"0": {"step": 1, "step_total": 2, "duration_micros": 279}, "1": {"step": 2, "step_total": 2, "duration_micros": 305}}}}, "extraInfo": {}}}}  
2024-03-07 11:06:57 mongo-database | {"t": {"$date": "2024-03-07T16:06:57.983+00:00"}, "s": "I", "c": "NETWORK", "id": 6788700, "ctx": "conn23", "msg": "Received first command on ingress connection since session start or auth handshake", "attr": {"elapsedMillis": 2}}}  
2024-03-07 11:07:08 mongo-database | {"t": {"$date": "2024-03-07T16:07:08.474+00:00"}, "s": "I", "c": "NETWORK", "id": 22943, "ctx": "listener", "msg": "Connection accepted", "attr": {"remote": "192.168.65.1:53582", "uuid": {"$uuid": "dc3d7f13-3a41-487b-9213-9c6d4ea8418a"}}, "connectionId": 24, "connectionCount": 18}}  
2024-03-07 11:07:08 mongo-database | {"t": {"$date": "2024-03-07T16:07:08.483+00:00"}, "s": "I", "c": "ACCESS", "id": 6788701, "ctx": "conn24", "msg": "Auth metrics report", "attr": {"metric": "acquireUser", "micros": 0}}
```

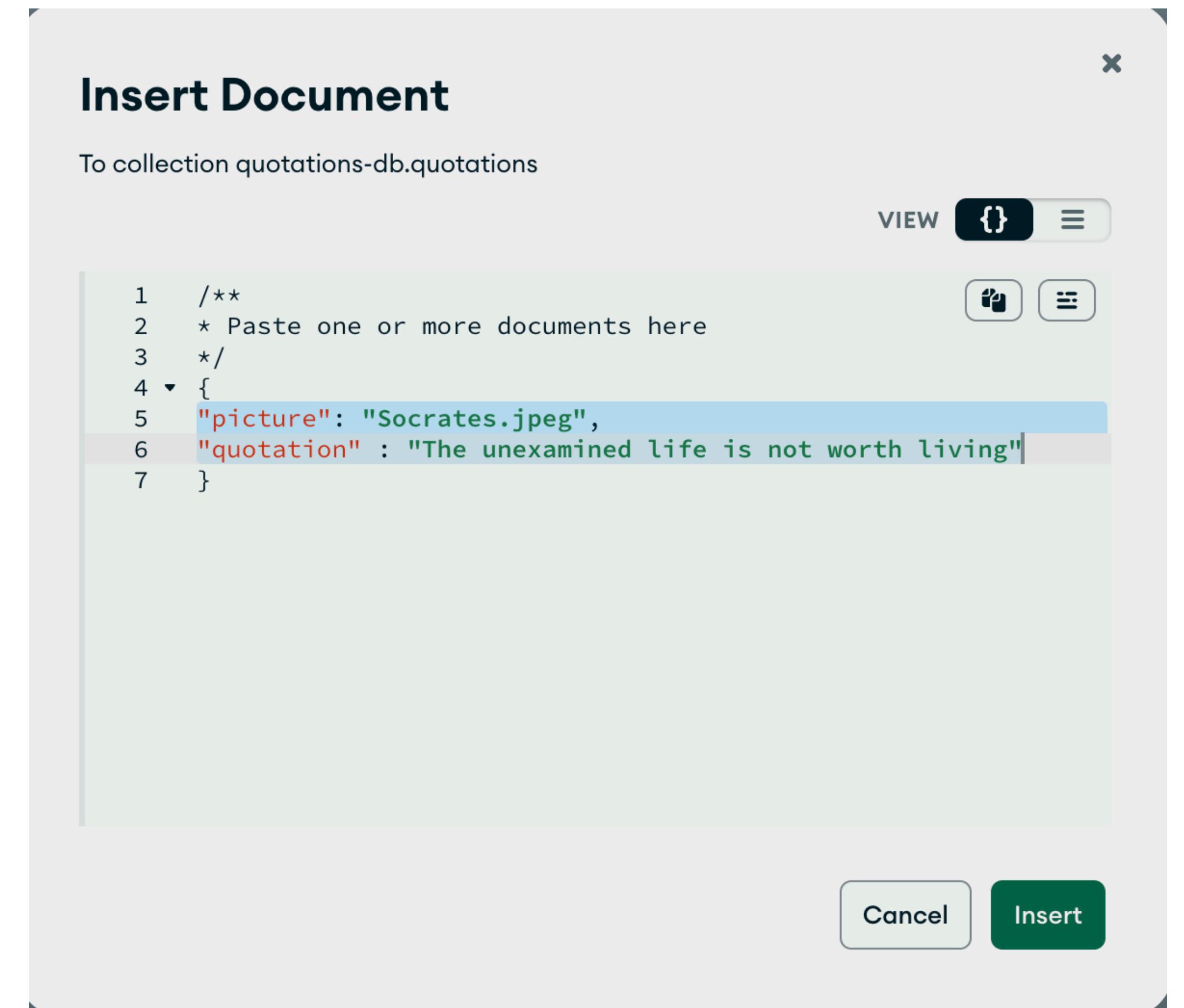
Steps

- Install MongoDB Compass tool:
<https://www.mongodb.com/products/tools/compass>
- Open the tool and connect to your mongo-database container.
- Create a database called “quotations-db” and in that database create a collection called “quotations”
- In the collection, create a document:



Steps

- Add the following to keys to the document:



Quotation service

Steps

- Create a folder called “quotation-service” in the same folder where your other services are located.
- Change into the new directory:
 - **cd quotation-service**
- Create an empty nodejs project:
 - **npm init -y**
- Install Express:
 - **npm install express**

Steps

- Install dotenv module: **npm install dotenv**
- Install nodemon to run the server in the development mode:
npm install --save-dev nodemon
- Install the mongodb package: **npm install mongodb**

Steps

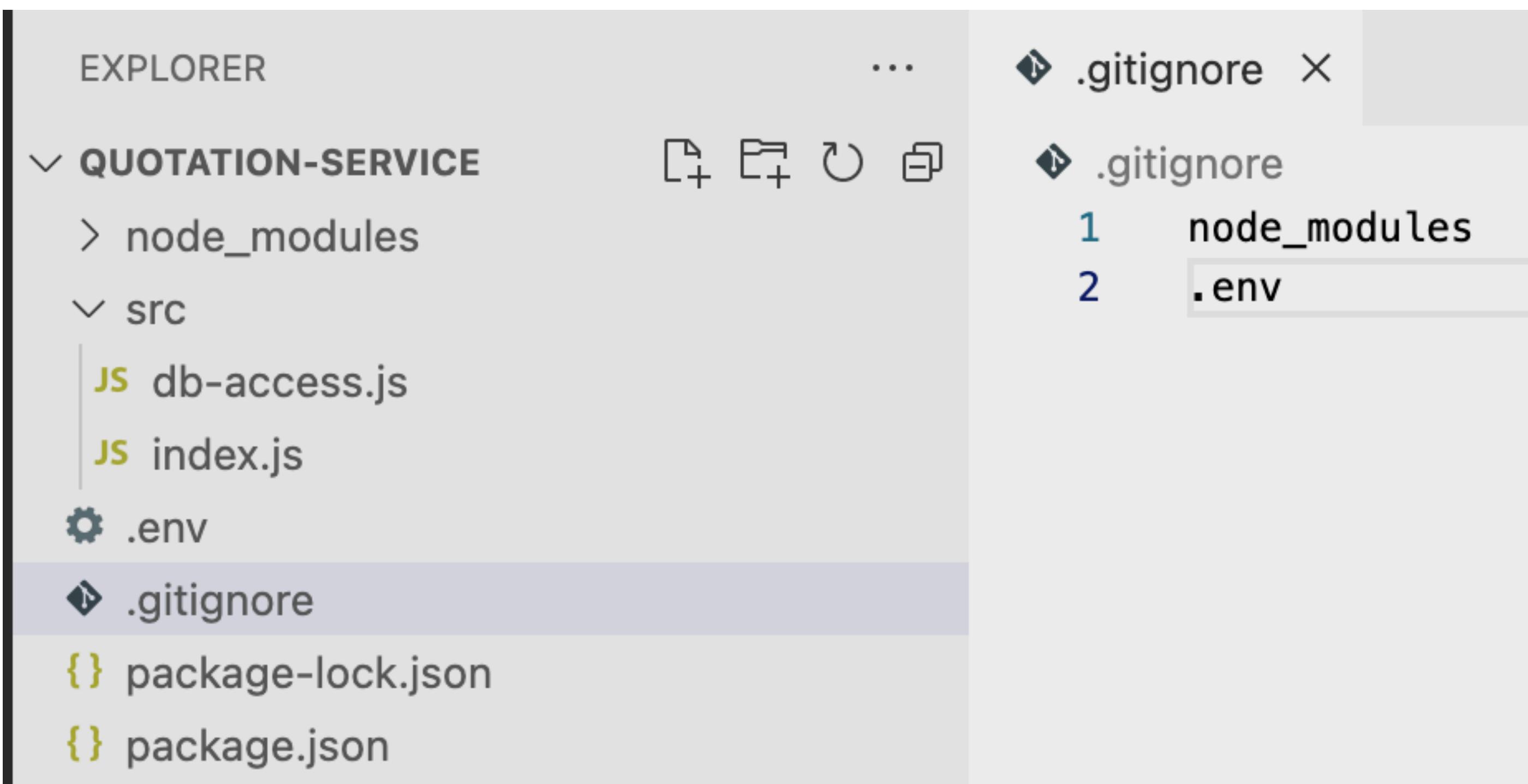
- Add the start commands to the package.json file:

The screenshot shows a code editor interface with two main panes. On the left is the Explorer pane, which displays the project structure under 'QUOTATION-SERVICE'. It includes 'node_modules', 'src' (containing 'db-access.js' and 'index.js'), '.env', '.gitignore', 'package-lock.json', and 'package.json'. The 'package.json' file is currently selected and highlighted with a light blue background. On the right is the main editor pane, showing the contents of the 'package.json' file. The file is a JSON object with several fields. The 'scripts' field contains three entries: 'test', 'start', and 'start:development'. The 'start:development' command is highlighted with a blue selection bar. The code is as follows:

```
{  
  "name": "quote-service",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1",  
    "start": "node ./src/index.js",  
    "start:development": "nodemon ./src/index.js"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "dotenv": "^16.4.5",  
    "express": "^4.18.3",  
    "http": "^0.0.1-security",  
    "mongodb": "^6.4.0"  
  },  
  "devDependencies": {  
    "nodemon": "^3.1.0"  
  }  
}
```

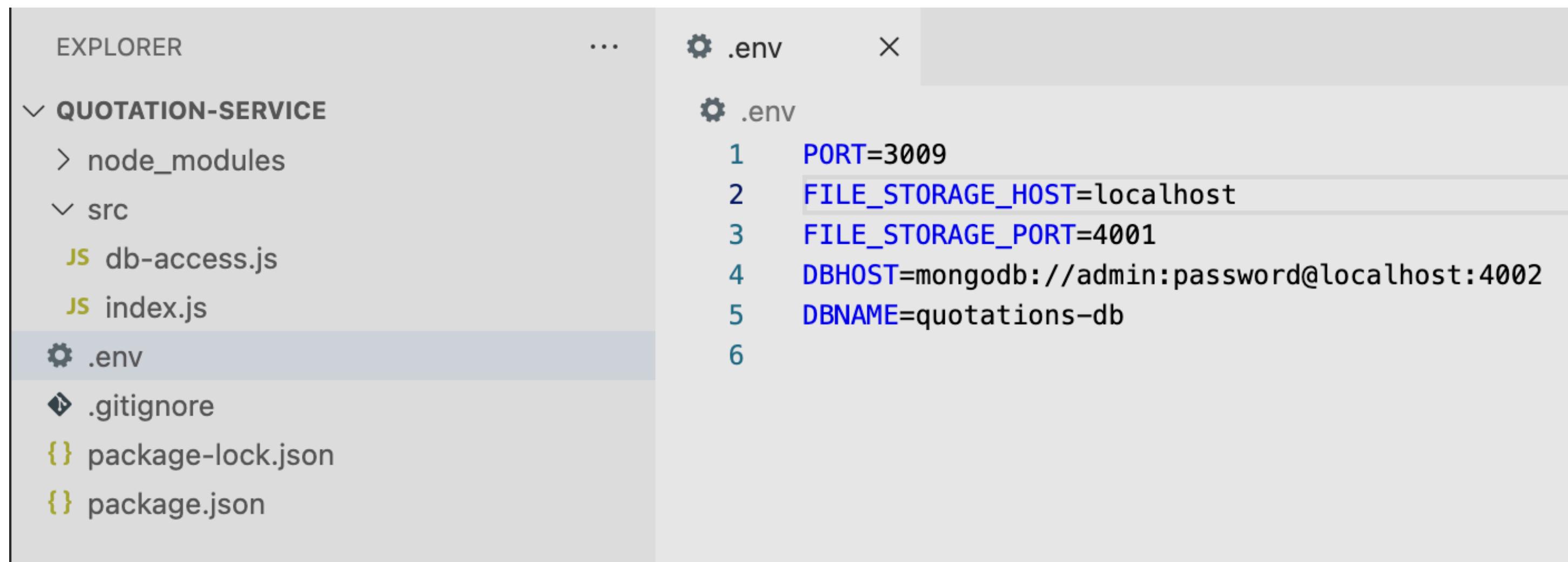
Steps

- Add a `.gitignore` file to the folder:



Steps

- Add a .env file to the folder.

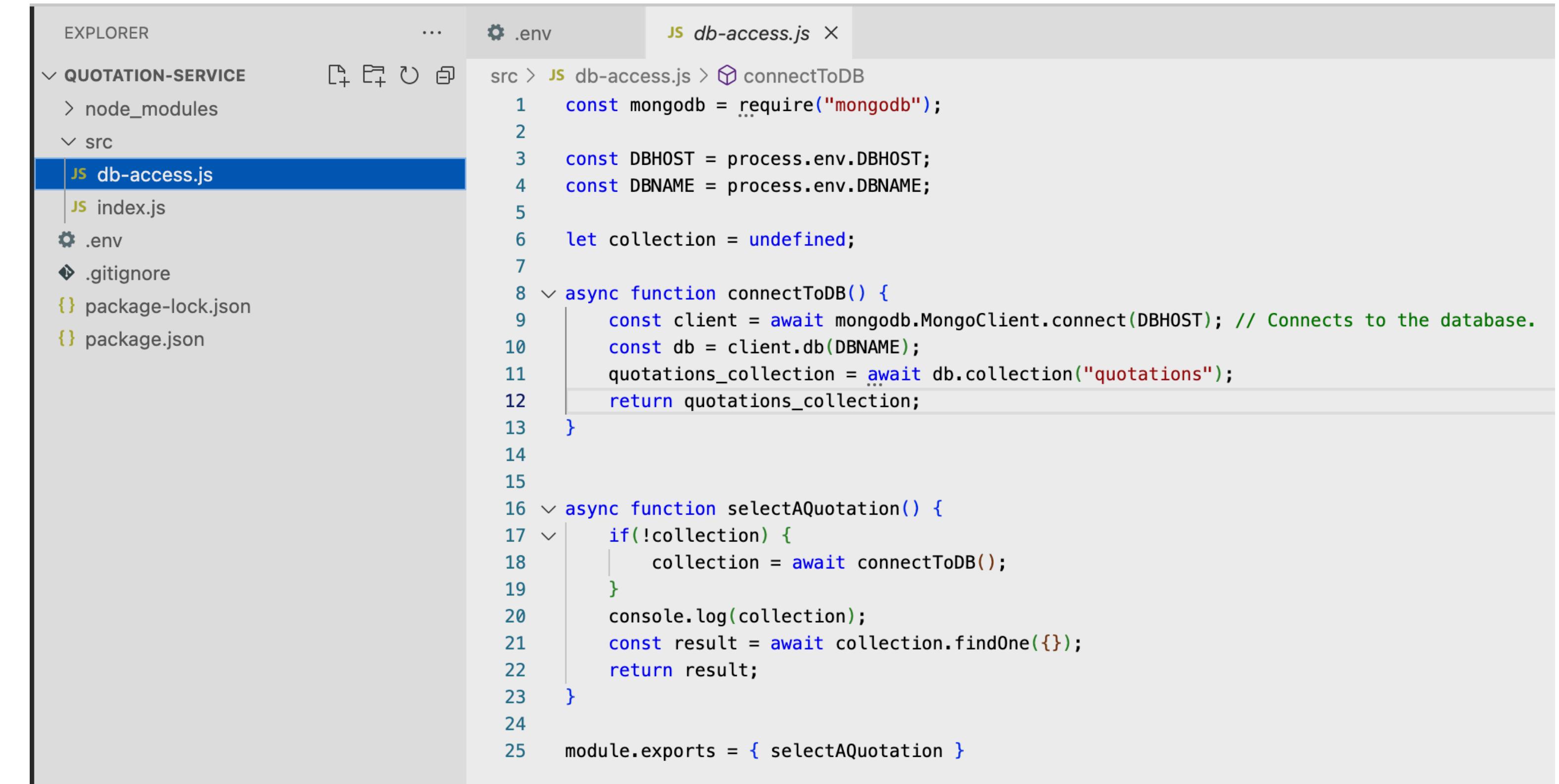


The image shows a file explorer interface with two panes. The left pane, titled 'EXPLORER', displays a project structure under 'QUOTATION-SERVICE'. It includes a 'node_modules' folder, a 'src' folder containing 'db-access.js' and 'index.js', a '.env' file (which is selected), a '.gitignore' file, and 'package-lock.json' and 'package.json' files. The right pane shows the contents of the selected '.env' file, which contains the following environment variables:

```
1 PORT=3009
2 FILE_STORAGE_HOST=localhost
3 FILE_STORAGE_PORT=4001
4 DBHOST=mongodb://admin:password@localhost:4002
5 DBNAME=quotations-db
6
```

Steps

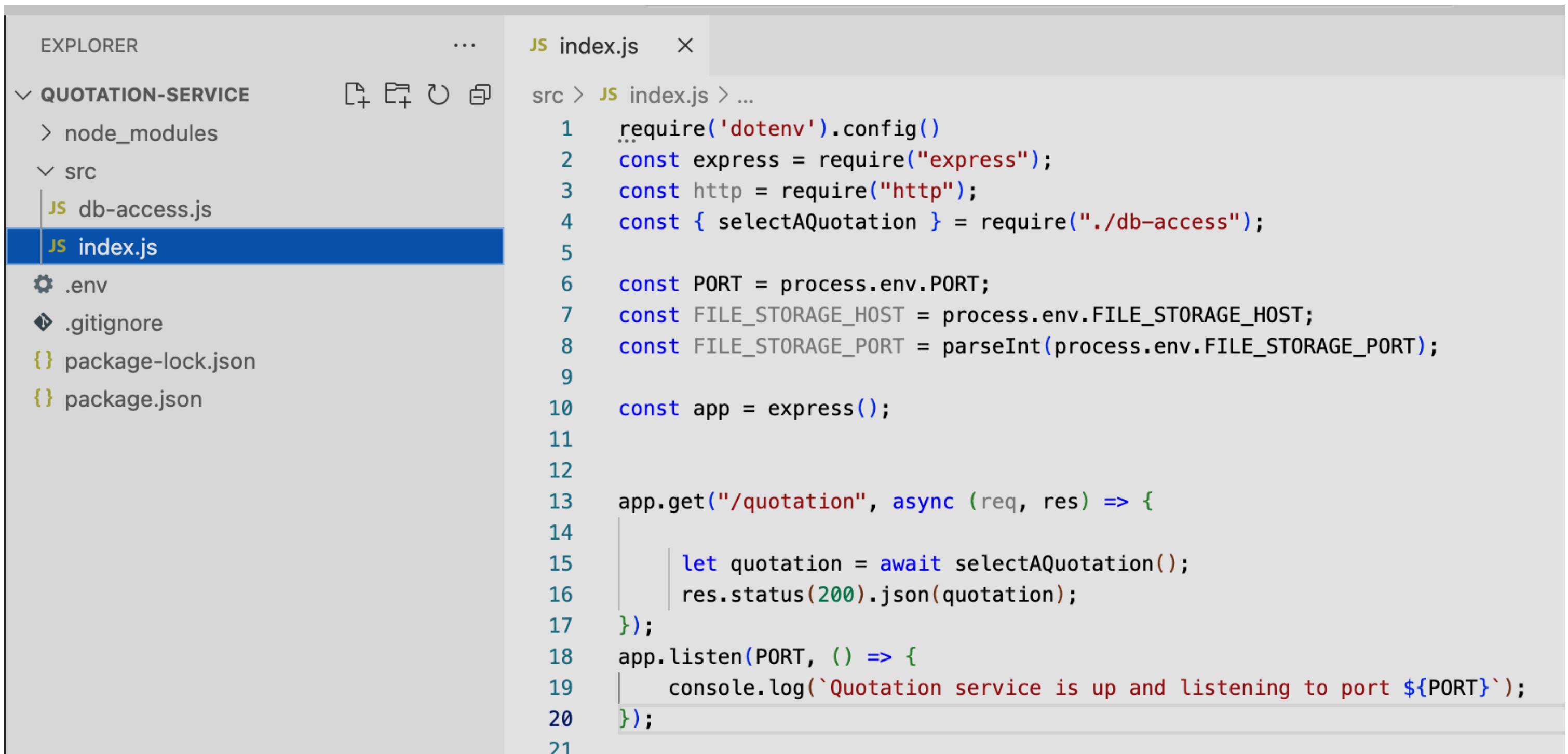
- Create a src folder and add two js files to it: index.js and db-access.js.
- Add the following to the db-access.js file:



```
src > JS db-access.js > connectToDB
1 const mongodb = require("mongodb");
2
3 const DBHOST = process.env.DBHOST;
4 const DBNAME = process.env.DBNAME;
5
6 let collection = undefined;
7
8 async function connectToDB() {
9     const client = await MongoClient.connect(DBHOST); // Connects to the database.
10    const db = client.db(DBNAME);
11    quotations_collection = await db.collection("quotations");
12    return quotations_collection;
13 }
14
15
16 async function selectAQuotation() {
17     if(!collection) {
18         collection = await connectToDB();
19     }
20     console.log(collection);
21     const result = await collection.findOne({});
22     return result;
23 }
24
25 module.exports = { selectAQuotation }
```

Steps

- Add the following to the index.js file:



The screenshot shows a code editor interface with two main sections: the Explorer and the Editor.

EXPLORER pane:

- Quotation-Service folder
 - node_modules
 - src
 - db-access.js
 - index.js** (highlighted with a blue selection bar)
 - .env
 - .gitignore
 - package-lock.json
 - package.json

Editor pane:

index.js

```
1  require('dotenv').config()
2  const express = require("express");
3  const http = require("http");
4  const { selectAQuotation } = require("./db-access");
5
6  const PORT = process.env.PORT;
7  const FILE_STORAGE_HOST = process.env.FILE_STORAGE_HOST;
8  const FILE_STORAGE_PORT = parseInt(process.env.FILE_STORAGE_PORT);
9
10 const app = express();
11
12
13 app.get("/quotation", async (req, res) => {
14
15   let quotation = await selectAQuotation();
16   res.status(200).json(quotation);
17 });
18 app.listen(PORT, () => {
19   console.log(`Quotation service is up and listening to port ${PORT}`);
20 });

21
```

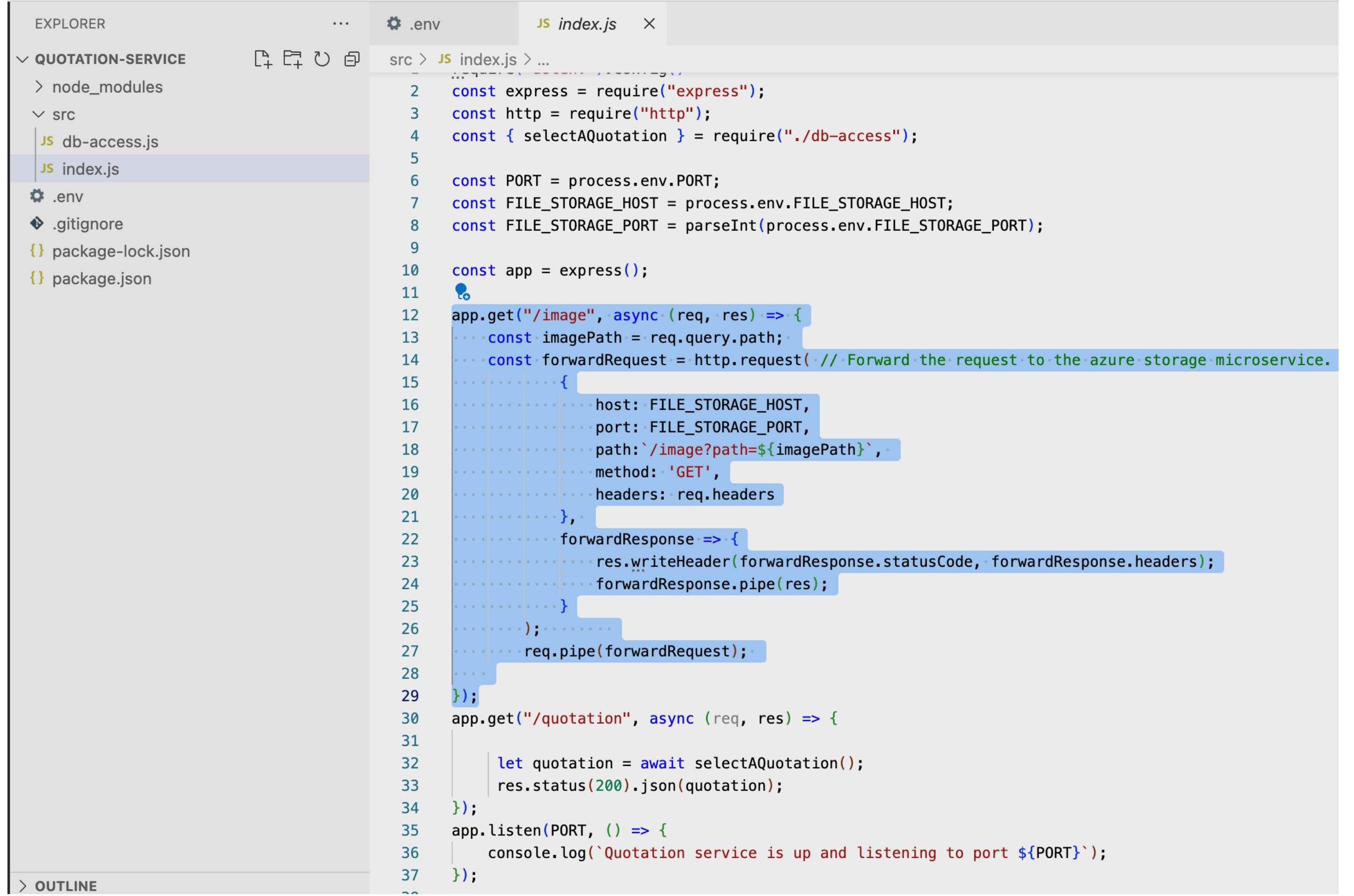
- Using postman, verify that the endpoint “/quotations” is working.

The screenshot shows the Postman application interface. At the top, the URL is `localhost:3009/quotations`. Below the URL, there are tabs for Params, Authorization, Headers (7), Body, Pre-request Script, Tests, and Settings. The Headers tab is selected, showing a table with one row: Key (empty) and Value (empty). In the bottom section, the Body tab is selected, displaying a JSON response:

```
1 {  
2   "_id": "65e9e18862d39ca1ece4d0f2",  
3   "picture": "Socrates.jpeg",  
4   "quotation": "The unexamined life is not worth living"  
5 }
```

The status bar at the bottom indicates `Status: 200 OK Time: 76 ms Size: 350 B`.

- Add the “/image” endpoint to your index.js file:



The screenshot shows the VS Code interface with the following details:

- EXPLORER** view: Shows the project structure under "QUOTATION-SERVICE". Files listed include node_modules, src (with db-access.js and index.js selected), .env, .gitignore, package-lock.json, and package.json.
- CODE EDITOR** view: The active file is index.js. The code implements an Express.js application with two routes: "/image" and "/quotation". The "/image" route uses the http module to forward requests to an Azure storage microservice. The "/quotation" route selects a quotation from a database and returns it as JSON.

```

1 // Import required modules
2 const express = require("express");
3 const http = require("http");
4 const { selectAQuotation } = require("./db-access");
5
6 const PORT = process.env.PORT;
7 const FILE_STORAGE_HOST = process.env.FILE_STORAGE_HOST;
8 const FILE_STORAGE_PORT = parseInt(process.env.FILE_STORAGE_PORT);
9
10 const app = express();
11
12 app.get("/image", async (req, res) => {
13   const imagePath = req.query.path;
14   const forwardRequest = http.request // Forward the request to the azure storage microservice.
15   {
16     host: FILE_STORAGE_HOST,
17     port: FILE_STORAGE_PORT,
18     path: `/image?path=${imagePath}`,
19     method: 'GET',
20     headers: req.headers
21   },
22   forwardResponse => {
23     res.writeHead(forwardResponse.statusCode, forwardResponse.headers);
24     forwardResponse.pipe(res);
25   }
26 );
27 req.pipe(forwardRequest);
28
29 });
30 app.get("/quotation", async (req, res) => {
31   let quotation = await selectAQuotation();
32   res.status(200).json(quotation);
33 });
34
35 app.listen(PORT, () => {
36   console.log(`Quotation service is up and listening to port ${PORT}`);
37 });
  
```

- Rerun the application.
- Using postman verify that the endpoint is working:

HTTP quotations-service / **get the picture**

GET localhost:3009/image?path=Socrates.jpeg Send

Params • Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

<input checked="" type="checkbox"/> Key	Value	Description	... Bulk Edit
<input checked="" type="checkbox"/> path	Socrates.jpeg		
	Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 499 ms Size: 41.41 KB Save as example ...

