## Overview

This document explains the steps taken to complete a Node.js project assignment. The tasks include creating a Node.js project, connecting to MongoDB, creating APIs for user management, adding proper logging, and writing unit tests to test the code.

### Step 1: Create a Node Project

- **Action:** Initialize a new Node.js project.
- **Tools Used:** Node.js, npm.
- **Description:** I created a project directory called express-mongodb-app and initialized it as a Node.js project using the npm init command. This generated the package.json file, which holds the project configuration.
- **Commands:**
    - ❖ mkdir express-mongodb-app
    - ❖ cd express-mongodb-app
    - ❖ npm init -y

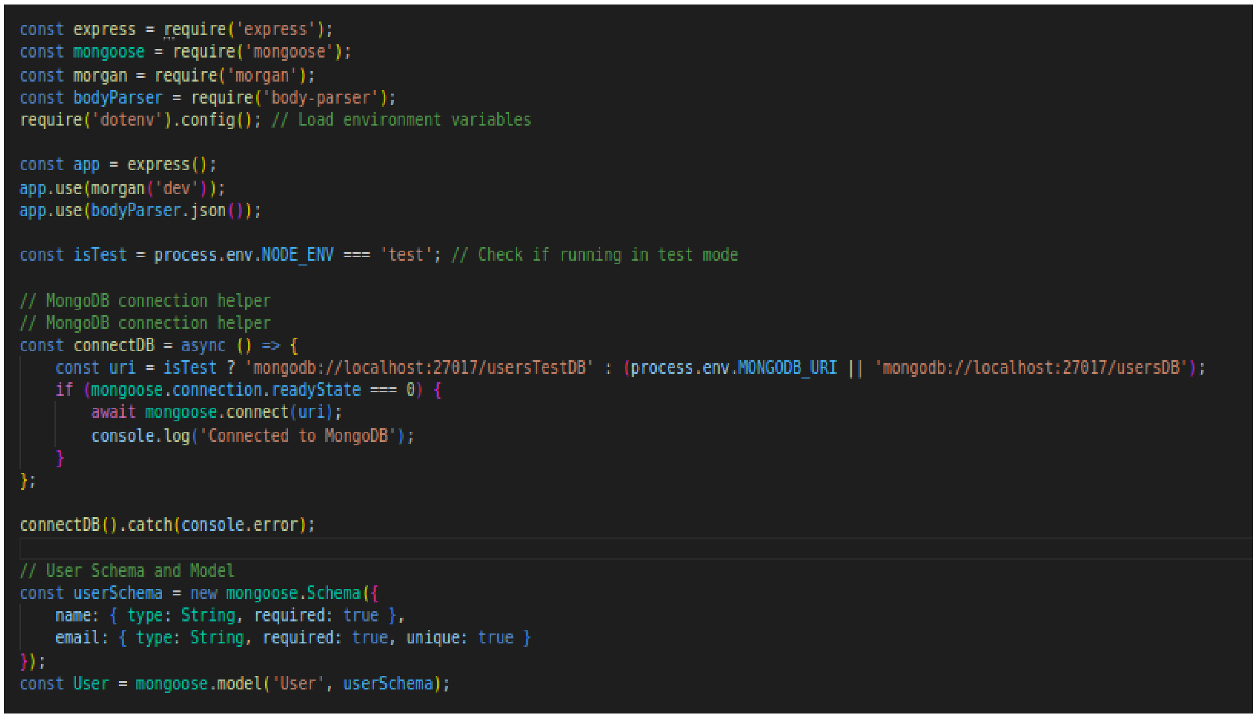
### Step 2: Install Required Packages

- **Action:** Install necessary packages for the project.
- **Tools Used:** npm.
- **Description:** I installed Express for building the server, Mongoose for MongoDB object modeling, Morgan for logging HTTP requests, and dotenv for environment variable management.
- **Command:**
    - ❖ npm install express mongoose morgan dotenv body-parser supertest --save

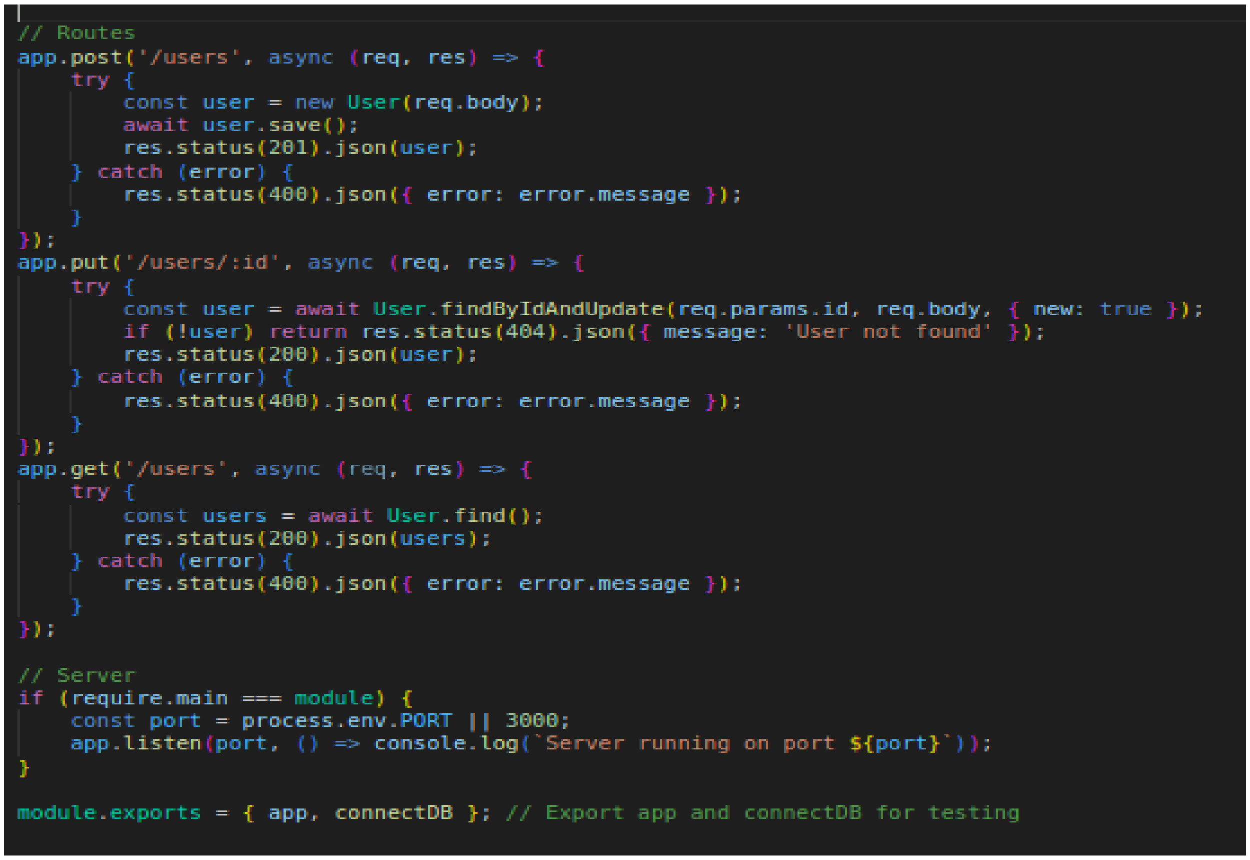
### Step 3: Create the Server and Connect to MongoDB

- **Action:** Set up an Express server and connect to MongoDB.
- **File Created:** app.js.
- **Description:** I wrote code in app.js to create an Express server, define a MongoDB connection using Mongoose, and set up middleware for logging and parsing JSON requests.

- **Code Snippet:**

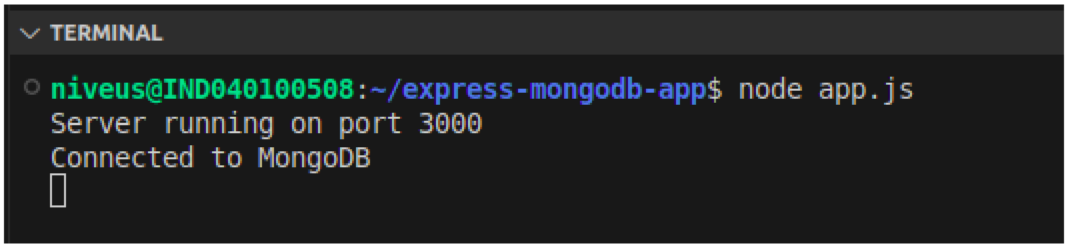


```
const express = require('express');
const mongoose = require('mongoose');
const morgan = require('morgan');
const bodyParser = require('body-parser');
require('dotenv').config(); // Load environment variables

const app = express();
app.use(morgan('dev'));
app.use(bodyParser.json());

const isTest = process.env.NODE_ENV === 'test'; // Check if running in test mode

// MongoDB connection helper
// MongoDB connection helper
const connectDB = async () => {
    const uri = isTest ? 'mongodb://localhost:27017/usersTestDB' : (process.env.MONGODB_URI || 'mongodb://localhost:27017/usersDB');
    if (mongoose.connection.readyState === 0) {
        await mongoose.connect(uri);
        console.log('Connected to MongoDB');
    }
};

connectDB().catch(console.error);

// User Schema and Model
const userSchema = new mongoose.Schema({
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true }
});
const User = mongoose.model('User', userSchema);
```



```
// Routes
app.post('/users', async (req, res) => {
    try {
        const user = new User(req.body);
        await user.save();
        res.status(201).json(user);
    } catch (error) {
        res.status(400).json({ error: error.message });
    }
});
app.put('/users/:id', async (req, res) => {
    try {
        const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });
        if (!user) return res.status(404).json({ message: 'User not found' });
        res.status(200).json(user);
    } catch (error) {
        res.status(400).json({ error: error.message });
    }
});
app.get('/users', async (req, res) => {
    try {
        const users = await User.find();
        res.status(200).json(users);
    } catch (error) {
        res.status(400).json({ error: error.message });
    }
});

// Server
if (require.main === module) {
    const port = process.env.PORT || 3000;
    app.listen(port, () => console.log(`Server running on port ${port}`));
}

module.exports = { app, connectDB }; // Export app and connectDB for testing
```

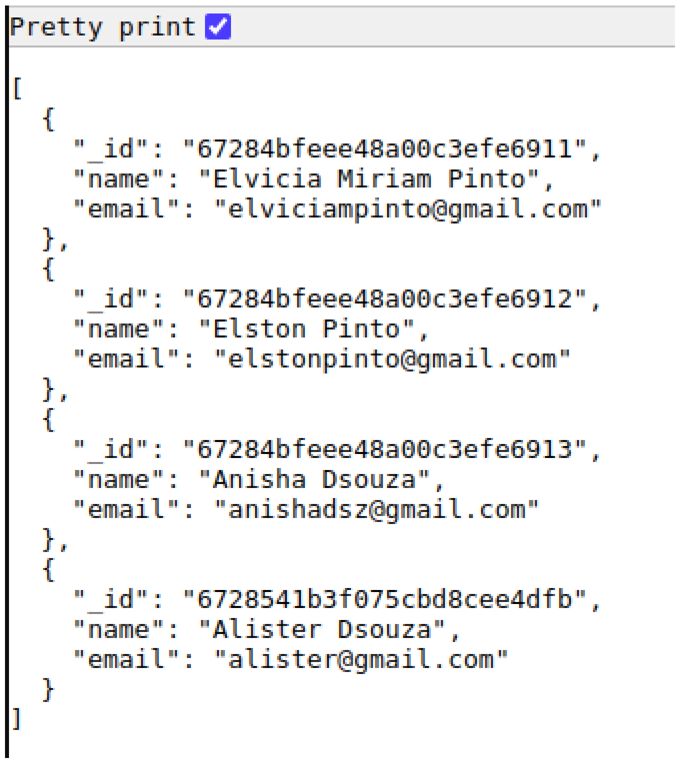
- **Output:**



```
TERMINAL
niveus@IND040100508:~/express-mongodb-app$ node app.js
Server running on port 3000
Connected to MongoDB
```

## Step 4: Create User APIs

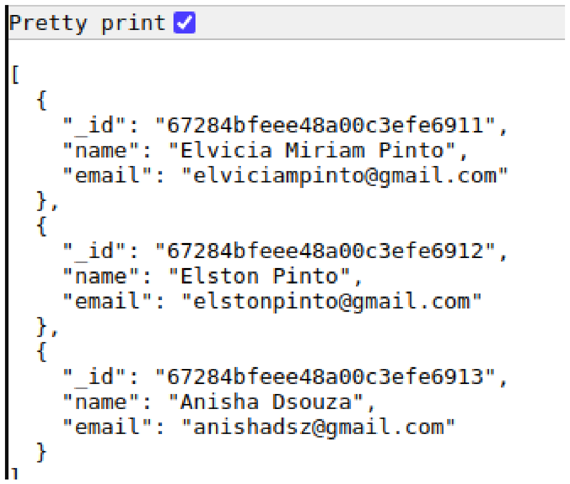
- **Action:** Implement APIs to create, update, and retrieve users.
- **Description:** I defined three endpoints in app.js for user management: POST /users to create a user, PUT /users/:id to update a user, and GET /users to retrieve a list of users.
- **Code Snippet:**
- **Output:**
  - ❖ Creating a user with the endpoint POST /users returns:



Pretty print ✓

```
[
  {
    "_id": "67284bfeee48a00c3efe6911",
    "name": "Elvicia Miriam Pinto",
    "email": "elviciampinto@gmail.com"
  },
  {
    "_id": "67284bfeee48a00c3efe6912",
    "name": "Elston Pinto",
    "email": "elstonpinto@gmail.com"
  },
  {
    "_id": "67284bfeee48a00c3efe6913",
    "name": "Anisha Dsouza",
    "email": "anishadsz@gmail.com"
  },
  {
    "_id": "6728541b3f075cbd8cee4dfb",
    "name": "Alister Dsouza",
    "email": "alister@gmail.com"
  }
]
```

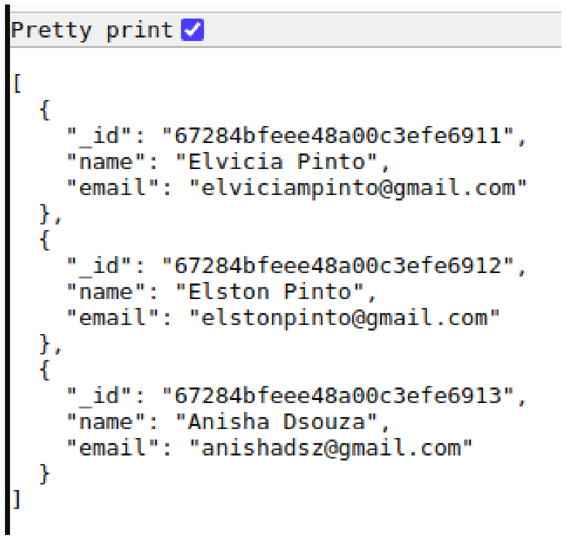
❖ Updating a user with the endpoint PUT /users/:id returns:



Pretty print ✓

```
[
  {
    "_id": "67284bfeee48a00c3efe6911",
    "name": "Elvicia Miriam Pinto",
    "email": "elviciampinto@gmail.com"
  },
  {
    "_id": "67284bfeee48a00c3efe6912",
    "name": "Elston Pinto",
    "email": "elstonpinto@gmail.com"
  },
  {
    "_id": "67284bfeee48a00c3efe6913",
    "name": "Anisha Dsouza",
    "email": "anishadsz@gmail.com"
  }
]
```

❖ Retrieving the list of users with GET /users returns:

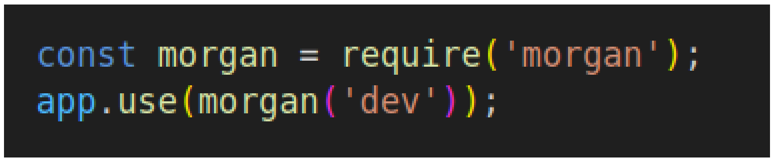


Pretty print ✓

```
[
  {
    "_id": "67284bfeee48a00c3efe6911",
    "name": "Elvicia Pinto",
    "email": "elviciampinto@gmail.com"
  },
  {
    "_id": "67284bfeee48a00c3efe6912",
    "name": "Elston Pinto",
    "email": "elstonpinto@gmail.com"
  },
  {
    "_id": "67284bfeee48a00c3efe6913",
    "name": "Anisha Dsouza",
    "email": "anishadsz@gmail.com"
  }
]
```

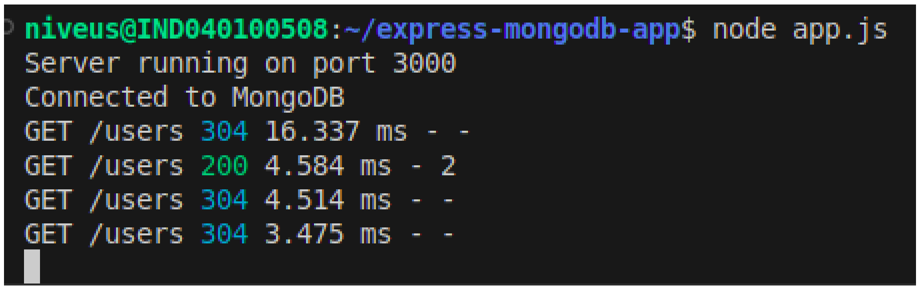
## Step 5: Add Logging

- **Action:** Implement logging for HTTP requests.
- **Description:** I used Morgan to log requests to the console, providing useful information for debugging and monitoring.
- **Code Snippet:**



```
const morgan = require('morgan');
app.use(morgan('dev'));
```

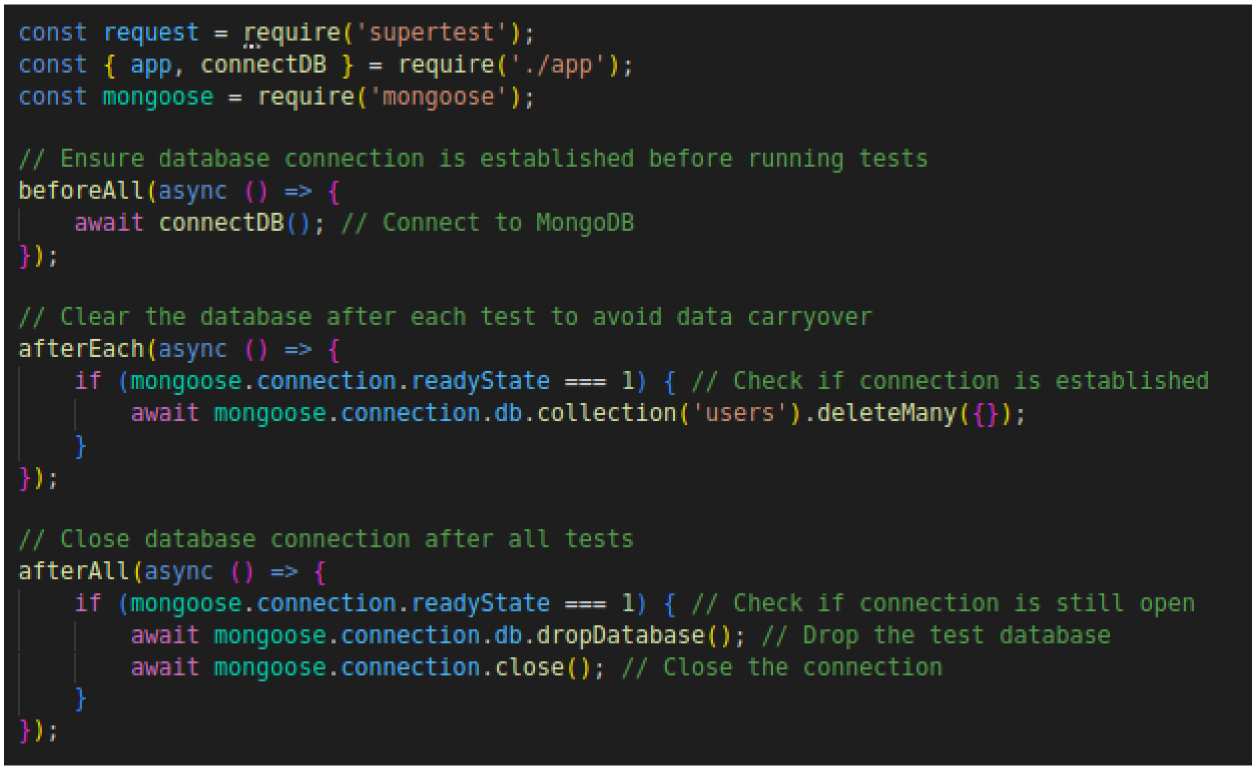
- **Output:**



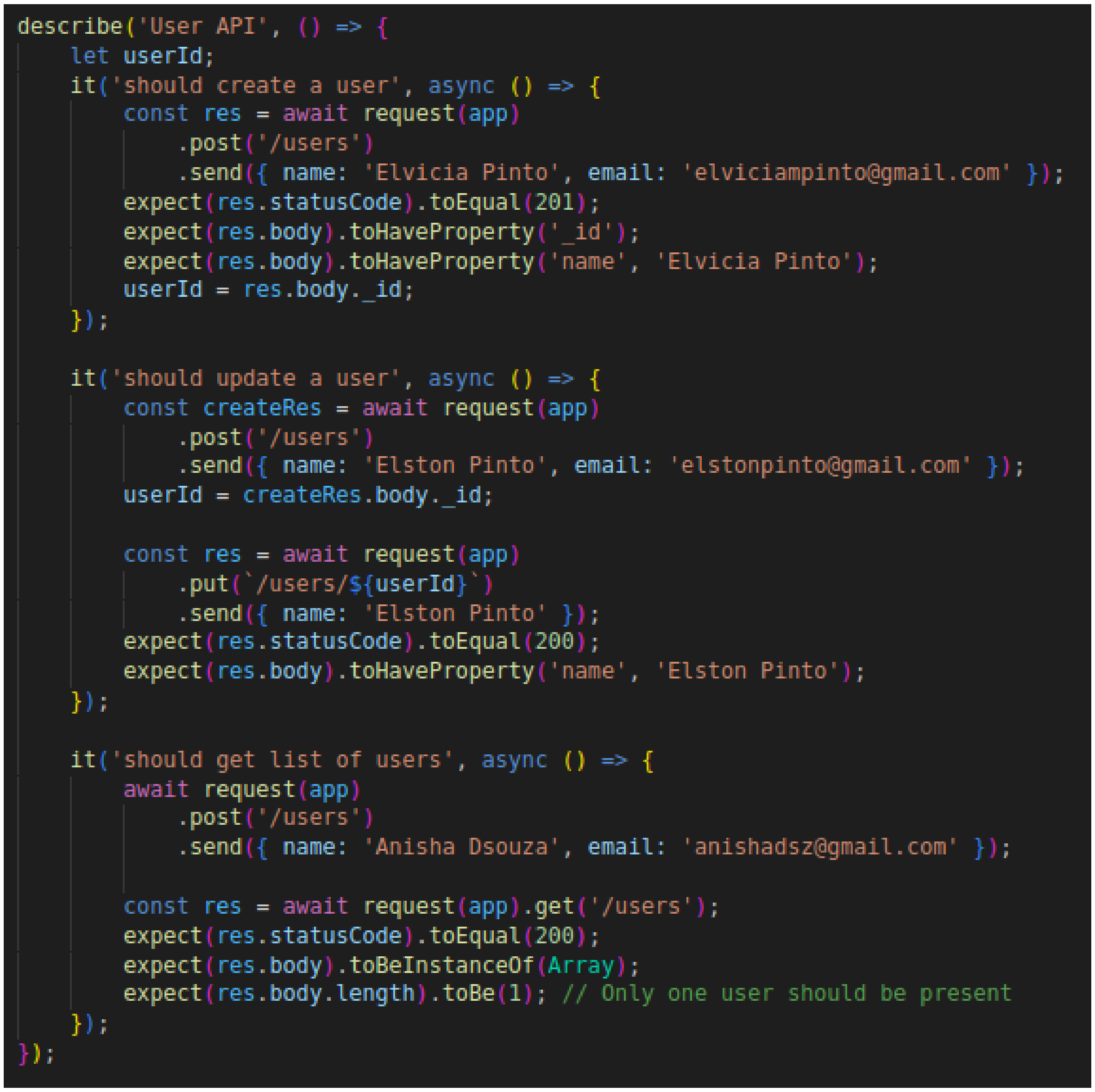
```
niveus@IND040100508:~/express-mongodb-app$ node app.js
Server running on port 3000
Connected to MongoDB
GET /users 304 16.337 ms - -
GET /users 200 4.584 ms - 2
GET /users 304 4.514 ms - -
GET /users 304 3.475 ms - -
```

## Step 6: Write Unit Tests

- **Action:** Create unit tests for the user API.
- **File Created:** app.test.js.
- **Description:** I wrote unit tests using Supertest to verify the functionality of the user creation, update, and retrieval endpoints.
- **Code Snippet:**

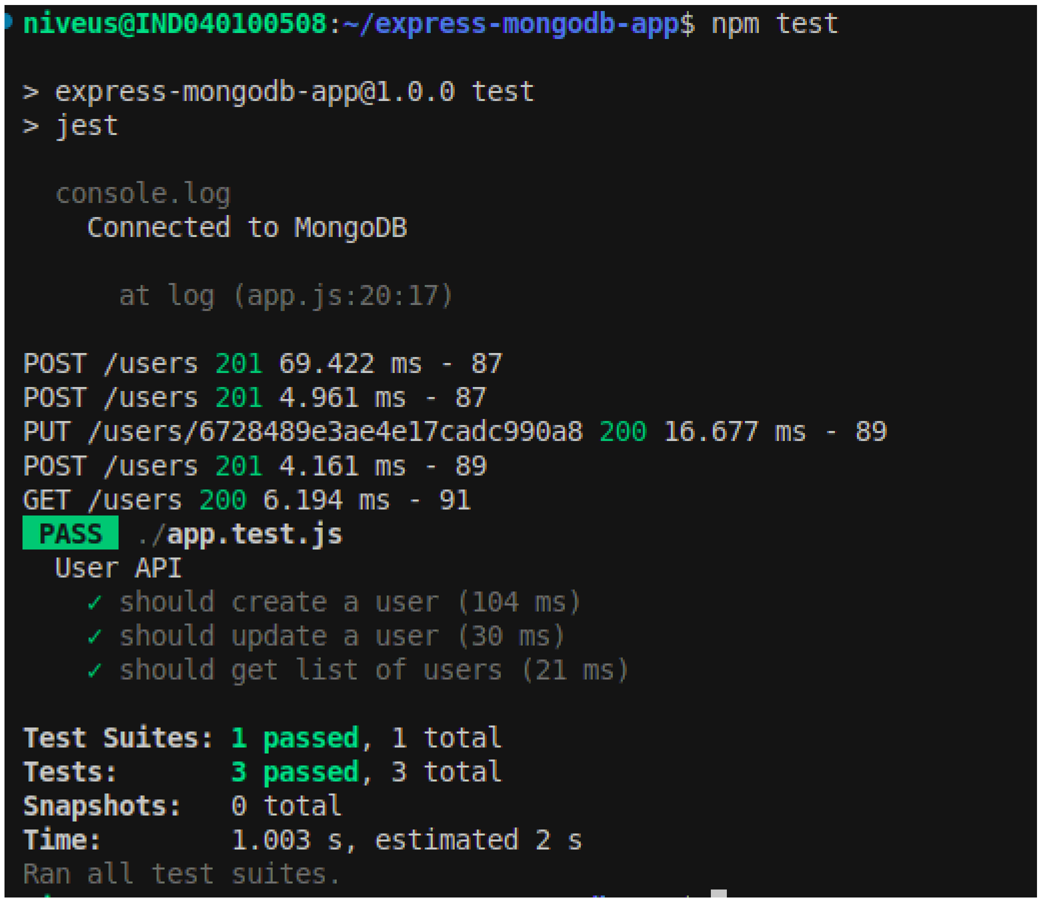


```javascript
const request = require('supertest');
const { app, connectDB } = require('./app');
const mongoose = require('mongoose');

// Ensure database connection is established before running tests
beforeAll(async () => {
    await connectDB(); // Connect to MongoDB
});

// Clear the database after each test to avoid data carryover
afterEach(async () => {
    if (mongoose.connection.readyState === 1) { // Check if connection is established
        await mongoose.connection.db.collection('users').deleteMany({});
    }
});

// Close database connection after all tests
afterAll(async () => {
    if (mongoose.connection.readyState === 1) { // Check if connection is still open
        await mongoose.connection.db.dropDatabase(); // Drop the test database
        await mongoose.connection.close(); // Close the connection
    }
});
```



```javascript
describe('User API', () => {
    let userId;
    it('should create a user', async () => {
        const res = await request(app)
            .post('/users')
            .send({ name: 'Elvicia Pinto', email: 'elviciampinto@gmail.com' });
        expect(res.statusCode).toEqual(201);
        expect(res.body).toHaveProperty('_id');
        expect(res.body).toHaveProperty('name', 'Elvicia Pinto');
        userId = res.body._id;
    });

    it('should update a user', async () => {
        const createRes = await request(app)
            .post('/users')
            .send({ name: 'Elston Pinto', email: 'elstonpinto@gmail.com' });
        userId = createRes.body._id;

        const res = await request(app)
            .put(`/users/${userId}`)
            .send({ name: 'Elston Pinto' });
        expect(res.statusCode).toEqual(200);
        expect(res.body).toHaveProperty('name', 'Elston Pinto');
    });

    it('should get list of users', async () => {
        await request(app)
            .post('/users')
            .send({ name: 'Anisha Dsouza', email: 'anishadsz@gmail.com' });

        const res = await request(app).get('/users');
        expect(res.statusCode).toEqual(200);
        expect(res.body).toBeInstanceOf(Array);
        expect(res.body.length).toBe(1); // Only one user should be present
    });
});
```

- **Output:**



```
niveus@IND040100508:~/express-mongodb-app$ npm test

> express-mongodb-app@1.0.0 test
> jest

  console.log
    Connected to MongoDB

      at log (app.js:20:17)

POST /users 201 69.422 ms - 87
POST /users 201 4.961 ms - 87
PUT /users/6728489e3ae4e17cadc990a8 200 16.677 ms - 89
POST /users 201 4.161 ms - 89
GET /users 200 6.194 ms - 91
 PASS  ./app.test.js
  User API
    ✓ should create a user (104 ms)
    ✓ should update a user (30 ms)
    ✓ should get list of users (21 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        1.003 s, estimated 2 s
Ran all test suites.
```

### Repository for Node.js Express and MongoDB Project

You can find the project files for the Node.js Express and MongoDB application in the GitHub repository linked below:

https://github.com/Elvicia13/express-mongodb-user-api

## Conclusion

This document outlines the steps taken to create a Node.js project, connect to MongoDB, implement user management APIs, add logging, and write unit tests. Each step showcases essential skills in Node.js development and prepares for further enhancements.