AI ASSISTED CODING – END SEMESTER LAB EXAM
Programme: B.Tech – CS & AI
Duration: 2 Hours

NAME : B.SRISHANTH **ROLL NO : 2403A510G3**

BATCH – CSB06 **DATE – 24 – 11 – 2025**

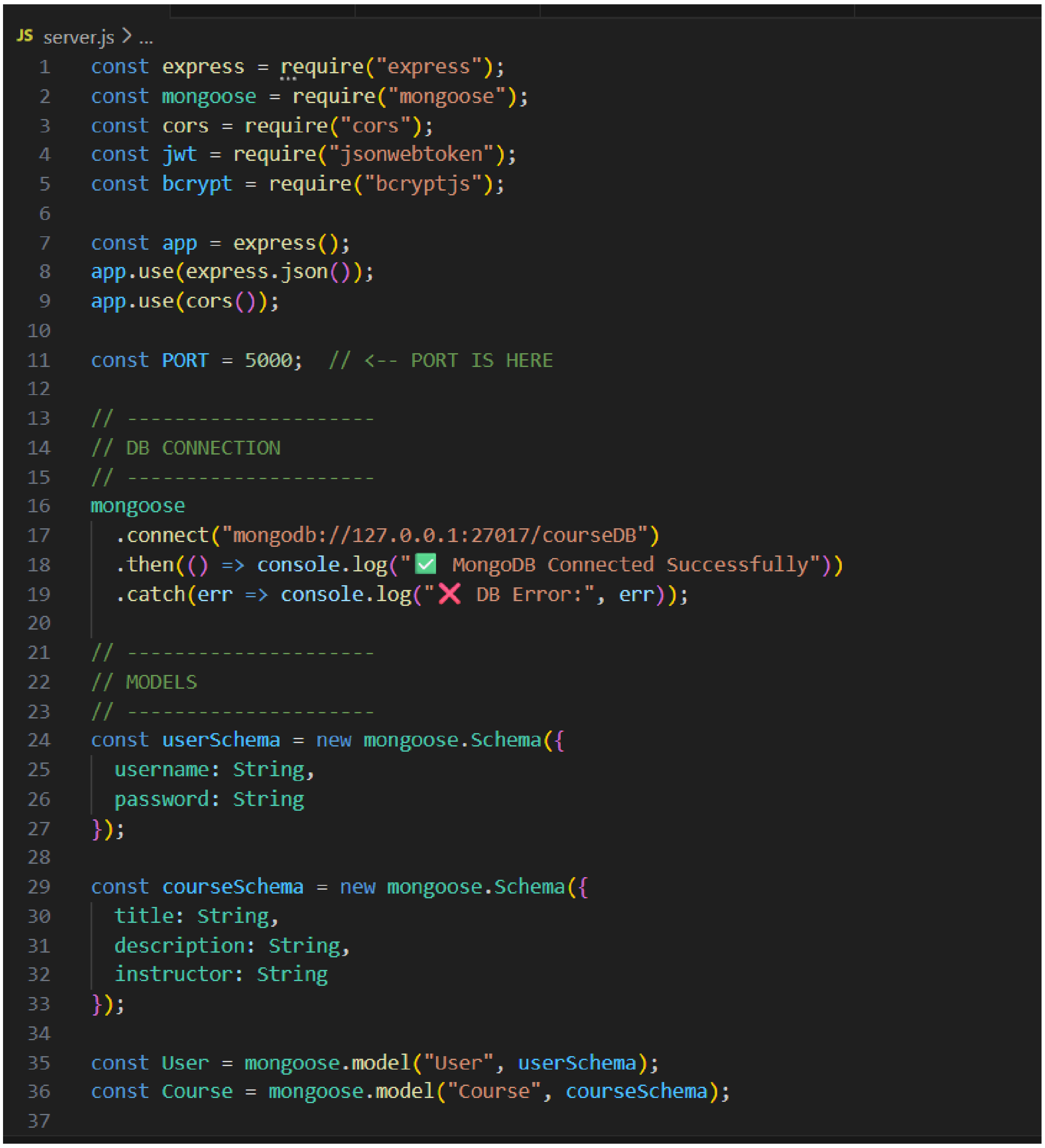
Subset 12 – Backend API Development

Q1 :-

PROMPT :

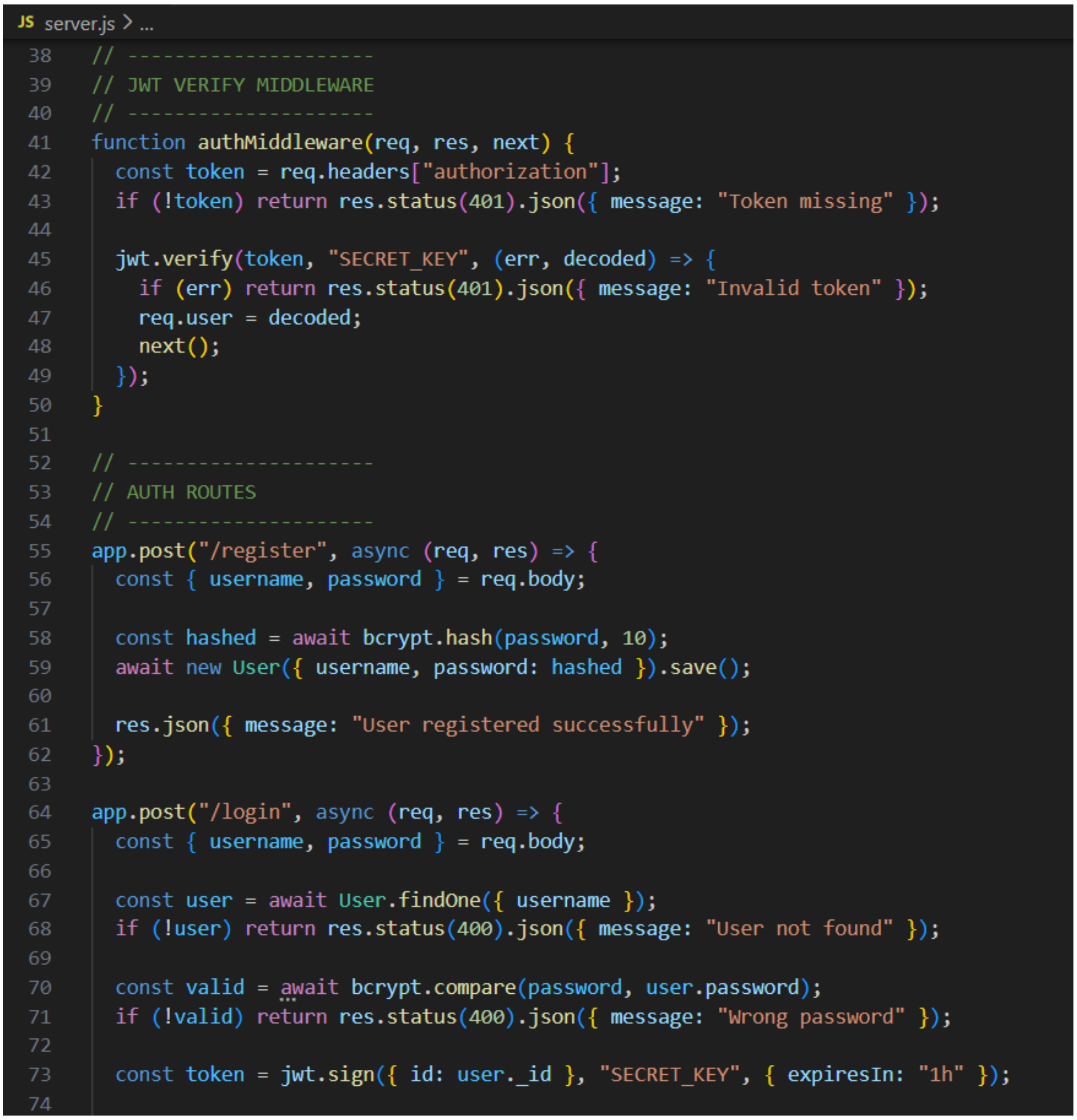
Create a backend REST API for a course management system using Node.js and Express.
Scaffold all CRUD endpoints (create, read, update, delete) for courses using AI.
Add user registration and login APIs.
Implement JWT authentication so that protected routes can be accessed only with a valid token

CODE :

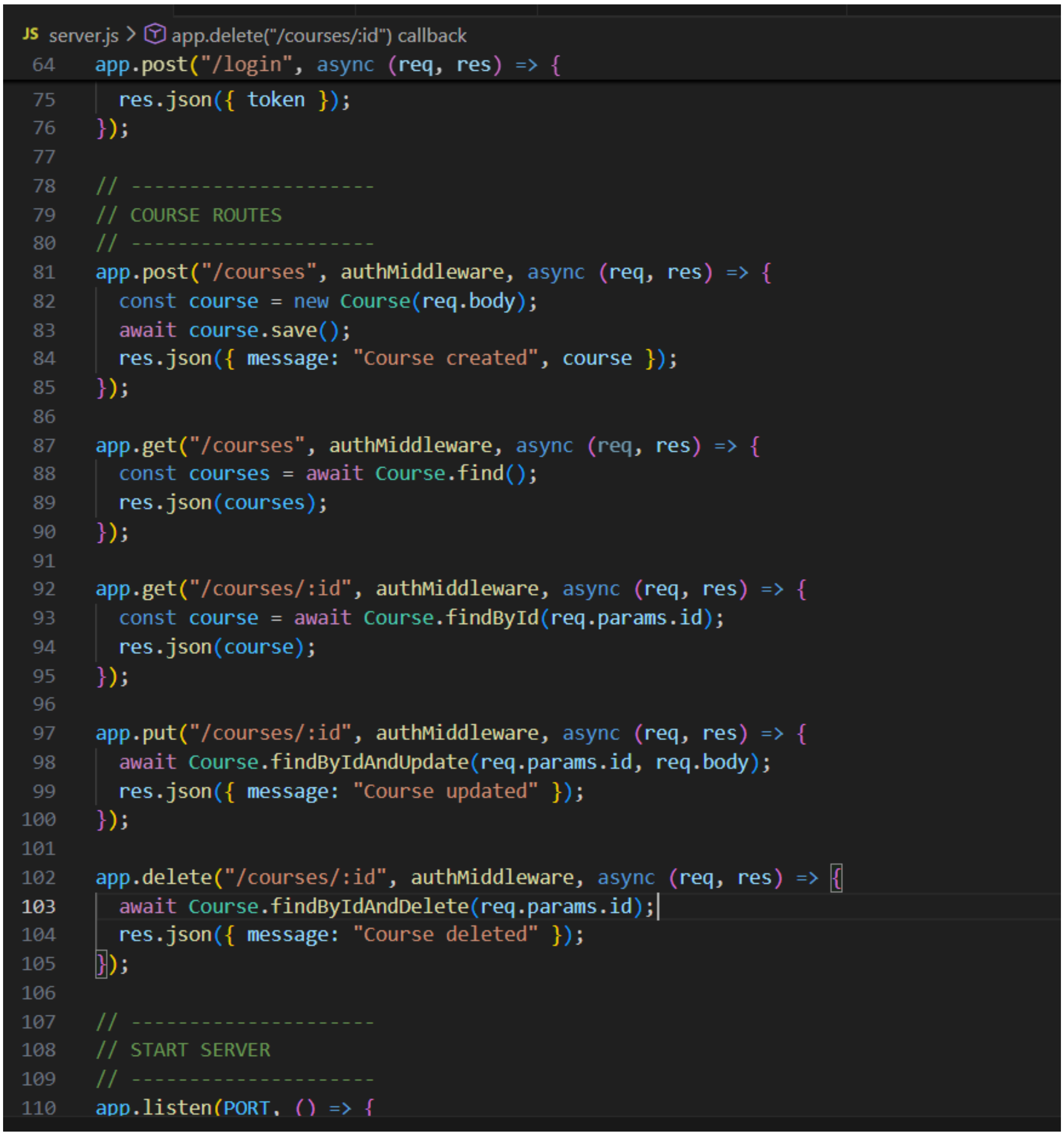


JS server.js > ...

```javascript
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const jwt = require("jsonwebtoken");
const bcrypt = require("bcryptjs");

const app = express();
app.use(express.json());
app.use(cors());

const PORT = 5000;  // <-- PORT IS HERE

// --------------------
// DB CONNECTION
// --------------------
mongoose
  .connect("mongodb://127.0.0.1:27017/courseDB")
  .then(() => console.log("✅ MongoDB Connected Successfully"))
  .catch(err => console.log("❌ DB Error:", err));

// --------------------
// MODELS
// --------------------
const userSchema = new mongoose.Schema({
  username: String,
  password: String
});

const courseSchema = new mongoose.Schema({
  title: String,
  description: String,
  instructor: String
});

const User = mongoose.model("User", userSchema);
const Course = mongoose.model("Course", courseSchema);

```


server.js > ...

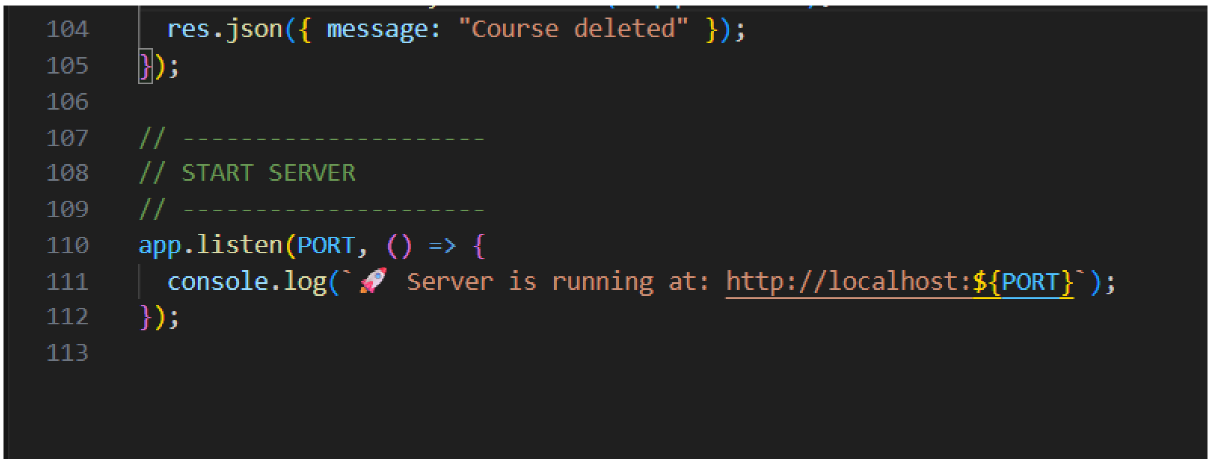
```javascript
// --------------------
// JWT VERIFY MIDDLEWARE
// --------------------
function authMiddleware(req, res, next) {
  const token = req.headers["authorization"];
  if (!token) return res.status(401).json({ message: "Token missing" });

  jwt.verify(token, "SECRET_KEY", (err, decoded) => {
    if (err) return res.status(401).json({ message: "Invalid token" });
    req.user = decoded;
    next();
  });
}

// --------------------
// AUTH ROUTES
// --------------------
app.post("/register", async (req, res) => {
  const { username, password } = req.body;

  const hashed = await bcrypt.hash(password, 10);
  await new User({ username, password: hashed }).save();

  res.json({ message: "User registered successfully" });
});

app.post("/login", async (req, res) => {
  const { username, password } = req.body;

  const user = await User.findOne({ username });
  if (!user) return res.status(400).json({ message: "User not found" });

  const valid = await bcrypt.compare(password, user.password);
  if (!valid) return res.status(400).json({ message: "Wrong password" });

  const token = jwt.sign({ id: user._id }, "SECRET_KEY", { expiresIn: "1h" });

```



JS server.js > app.delete("/courses/:id") callback

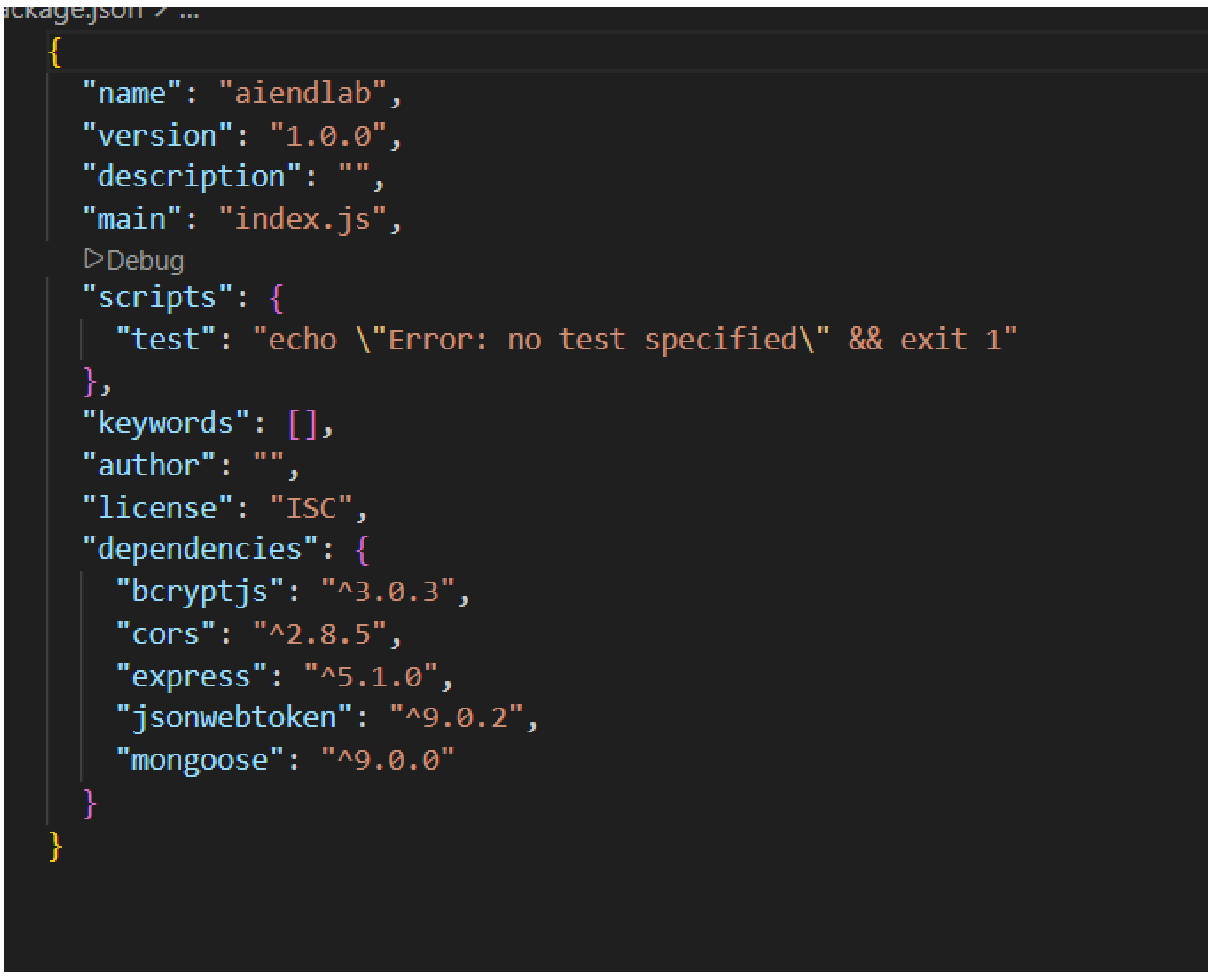
```javascript
app.post("/login", async (req, res) => {
```

```javascript
  res.json({ token });
});

// --------------------
// COURSE ROUTES
// --------------------
app.post("/courses", authMiddleware, async (req, res) => {
  const course = new Course(req.body);
  await course.save();
  res.json({ message: "Course created", course });
});

app.get("/courses", authMiddleware, async (req, res) => {
  const courses = await Course.find();
  res.json(courses);
});

app.get("/courses/:id", authMiddleware, async (req, res) => {
  const course = await Course.findById(req.params.id);
  res.json(course);
});

app.put("/courses/:id", authMiddleware, async (req, res) => {
  await Course.findByIdAndUpdate(req.params.id, req.body);
  res.json({ message: "Course updated" });
});

app.delete("/courses/:id", authMiddleware, async (req, res) => {
  await Course.findByIdAndDelete(req.params.id);
  res.json({ message: "Course deleted" });
});

// --------------------
// START SERVER
// --------------------
app.listen(PORT, () => {
```



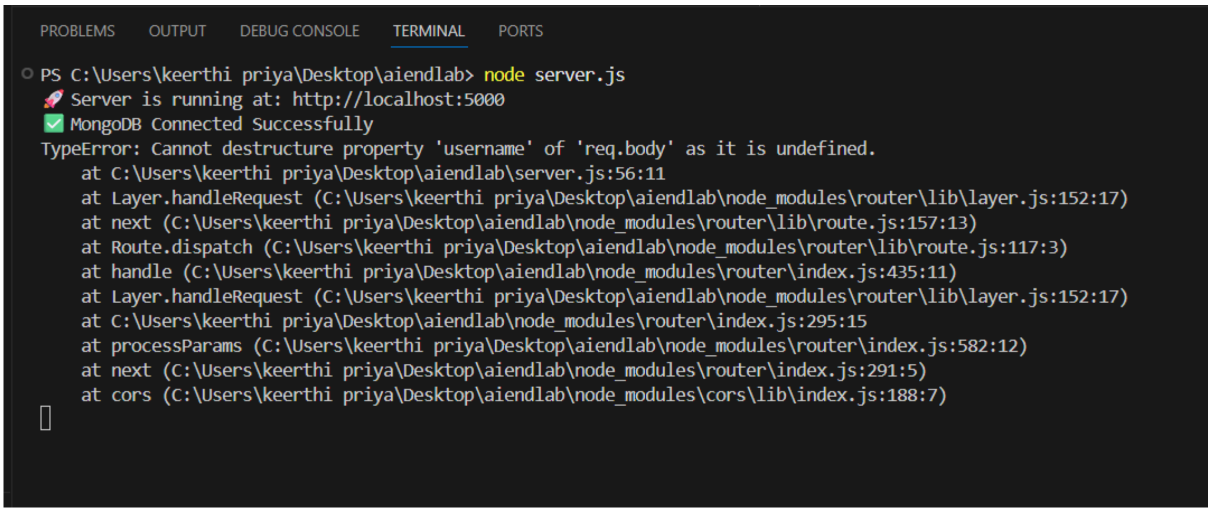
```javascript
  res.json({ message: "Course deleted" });
});

// --------------------
// START SERVER
// --------------------
app.listen(PORT, () => {
```

```javascript
  console.log(`🚀 Server is running at: http://localhost:${PORT}`);
});
```

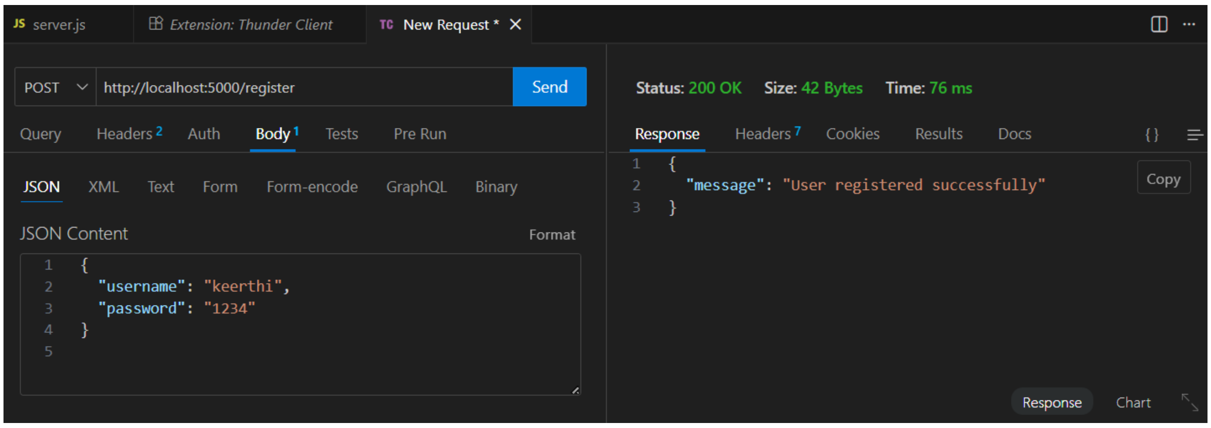


```
ckage.json > ...
{
  "name": "aiendlab",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^3.0.3",
    "cors": "^2.8.5",
    "express": "^5.1.0",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^9.0.0"
  }
}
```

OUTPUT :



```
PS C:\Users\keerthi priya\Desktop\aiendlab> node server.js
🚀 Server is running at: http://localhost:5000
✅ MongoDB Connected Successfully
TypeError: Cannot destructure property 'username' of 'req.body' as it is undefined.
    at C:\Users\keerthi priya\Desktop\aiendlab\server.js:56:11
    at Layer.handleRequest (C:\Users\keerthi priya\Desktop\aiendlab\node_modules\router\lib\layer.js:152:17)
    at next (C:\Users\keerthi priya\Desktop\aiendlab\node_modules\router\lib\route.js:157:13)
    at Route.dispatch (C:\Users\keerthi priya\Desktop\aiendlab\node_modules\router\lib\route.js:117:3)
    at handle (C:\Users\keerthi priya\Desktop\aiendlab\node_modules\router\index.js:435:11)
    at Layer.handleRequest (C:\Users\keerthi priya\Desktop\aiendlab\node_modules\router\lib\layer.js:152:17)
    at C:\Users\keerthi priya\Desktop\aiendlab\node_modules\router\index.js:295:15
    at processParams (C:\Users\keerthi priya\Desktop\aiendlab\node_modules\router\index.js:582:12)
    at next (C:\Users\keerthi priya\Desktop\aiendlab\node_modules\router\index.js:291:5)
    at cors (C:\Users\keerthi priya\Desktop\aiendlab\node_modules\cors\lib\index.js:188:7)
```

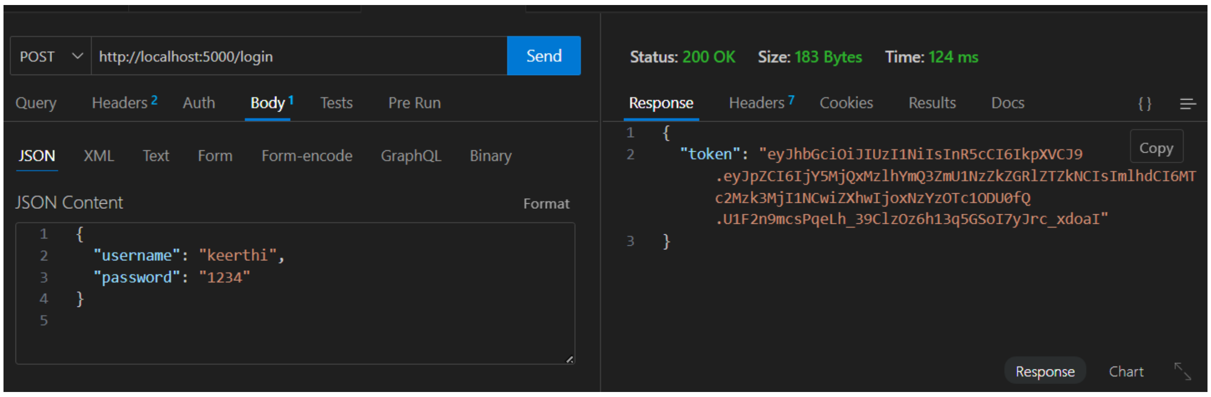


POST http://localhost:5000/register

```
{
  "username": "keerthi",
  "password": "1234"
}
```

Status: 200 OK Size: 42 Bytes Time: 76 ms

```
{
  "message": "User registered successfully"
}
```



POST http://localhost:5000/login

```
{
  "username": "keerthi",
  "password": "1234"
}
```

Status: 200 OK Size: 183 Bytes Time: 124 ms

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
      .eyJpZCI6IjY5MjQxMzlhYmQ3ZmU1NzZkZGRlZTZkNCIsImlhdCI6MT
      c2Mzk3MjI1NCwiZXhwIjoxNzYzOTc1ODU0fQ
      .U1F2n9mcsPqeLh_39ClzOz6h13q5GSoI7yJrc_xdoaI"
}
```

OBSERVATION :

In thunder client When we give correct username and password , the server generated a JWT token and returned it in the response.

Using that token in the Authorization header, the protected route was accessed successfully.

Without the token or with an invalid token, the server denied access.

This proves that JWT authentication is working correctly,

because only authenticated users can reach the protected APIs.

Q2 :

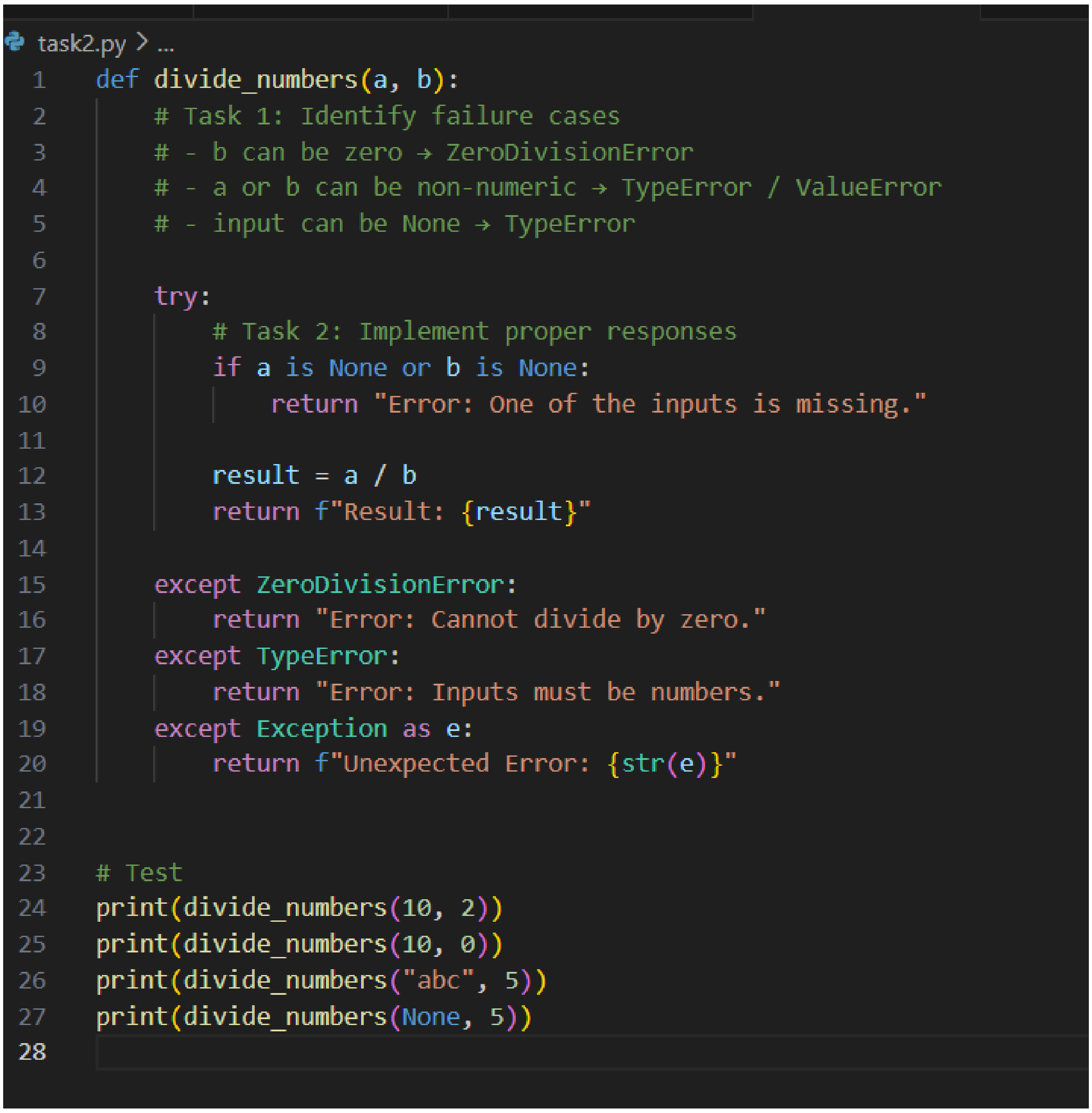
PROMPT :

Task 1 – Identify Failure Cases:
Analyze the given system/process/code snippet and list all possible failure cases or error scenarios. These may include invalid inputs,

missing data, exceptions, unexpected behavior, incorrect output, runtime failures, or edge-case breakdowns.

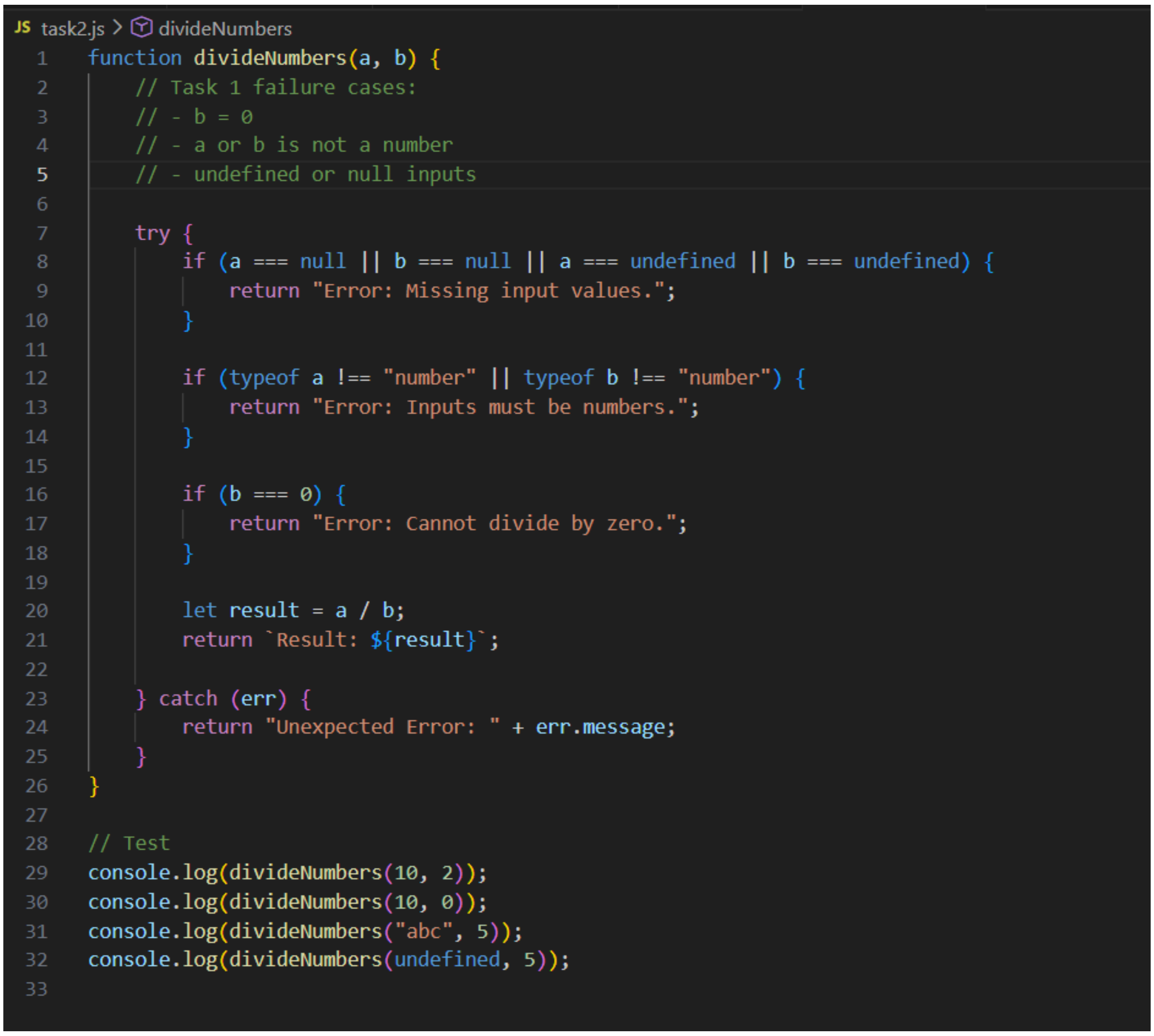
Task 2 – Implement Appropriate Responses:
For each failure case identified in Task 1, propose clear and effective error-handling responses.

CODE :



task2.py > ...

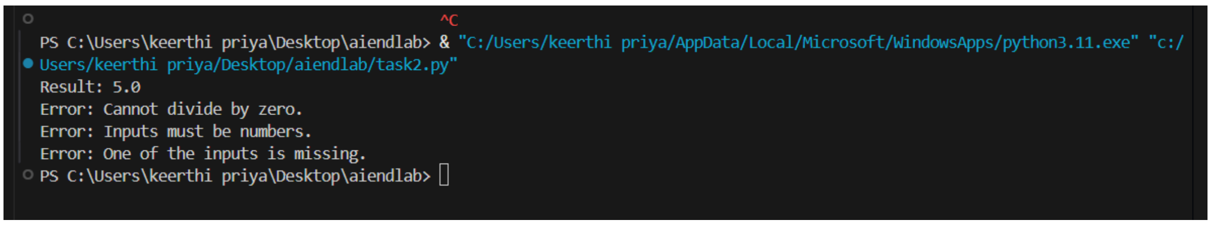
```python
def divide_numbers(a, b):
    # Task 1: Identify failure cases
    # - b can be zero → ZeroDivisionError
    # - a or b can be non-numeric → TypeError / ValueError
    # - input can be None → TypeError

    try:
        # Task 2: Implement proper responses
        if a is None or b is None:
            return "Error: One of the inputs is missing."

        result = a / b
        return f"Result: {result}"

    except ZeroDivisionError:
        return "Error: Cannot divide by zero."
    except TypeError:
        return "Error: Inputs must be numbers."
    except Exception as e:
        return f"Unexpected Error: {str(e)}"


# Test
print(divide_numbers(10, 2))
print(divide_numbers(10, 0))
print(divide_numbers("abc", 5))
print(divide_numbers(None, 5))

```



task2.js > divideNumbers

```javascript
function divideNumbers(a, b) {
    // Task 1 failure cases:
    // - b = 0
    // - a or b is not a number
    // - undefined or null inputs

    try {
        if (a === null || b === null || a === undefined || b === undefined) {
            return "Error: Missing input values.";
        }

        if (typeof a !== "number" || typeof b !== "number") {
            return "Error: Inputs must be numbers.";
        }

        if (b === 0) {
            return "Error: Cannot divide by zero.";
        }

        let result = a / b;
        return `Result: ${result}`;

    } catch (err) {
        return "Unexpected Error: " + err.message;
    }
}

// Test
console.log(divideNumbers(10, 2));
console.log(divideNumbers(10, 0));
console.log(divideNumbers("abc", 5));
console.log(divideNumbers(undefined, 5));
```

OUTPUT :



```
                                            ^C
PS C:\Users\keerthi priya\Desktop\aiendlab> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/
Users/keerthi priya/Desktop/aiendlab/task2.py"
Result: 5.0
Error: Cannot divide by zero.
Error: Inputs must be numbers.
Error: One of the inputs is missing.
PS C:\Users\keerthi priya\Desktop\aiendlab> 
```

OBSERVATION :

Error handling was implemented to detect invalid inputs and runtime failures. Instead of crashing, the system returned clear error messages. This made the API more stable and easier to use. Different exceptions were handled separately for better clarity. Overall, the application became more reliable under various input conditions.