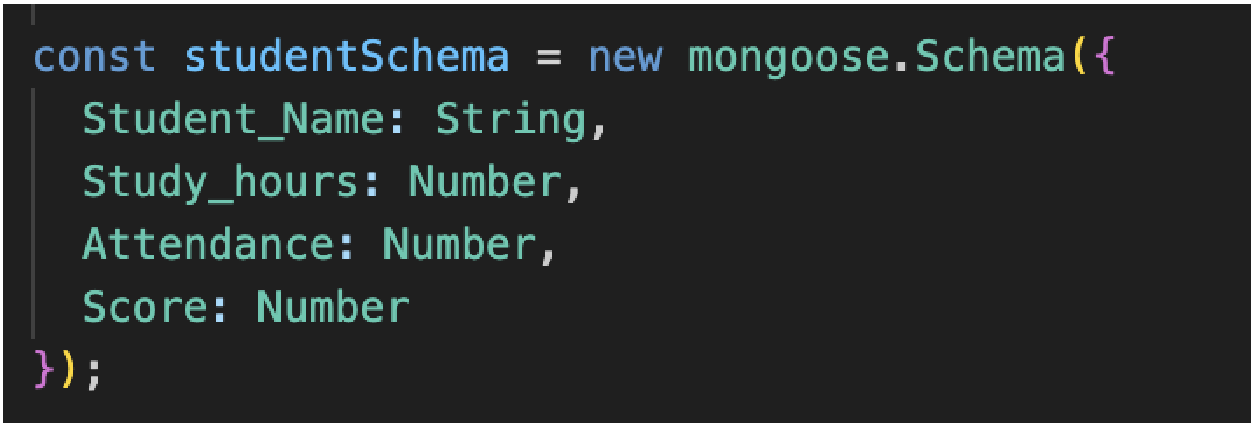
DATA-236 Sec 12 - Distributed Systems for Data Engineering
HOMEWORK 9
Nandhakumar Apparsamy
018190003

GitHub - https://github.com/Nandha951/DATA-236-HW9-Redis-FastAPI-MongoDB

Make a student schema consisting of the following fields

- Student Name (String)
- Study_hours (Number)
- Attendance (Number)
- Score (Number)

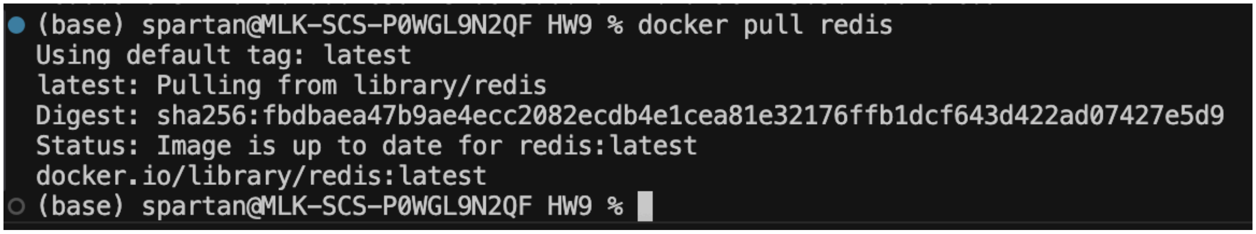


```
const studentSchema = new mongoose.Schema({
  Student_Name: String,
  Study_hours: Number,
  Attendance: Number,
  Score: Number
});
```

Redis Setup Instructions

1.Pull the official redis image from Docker Hub

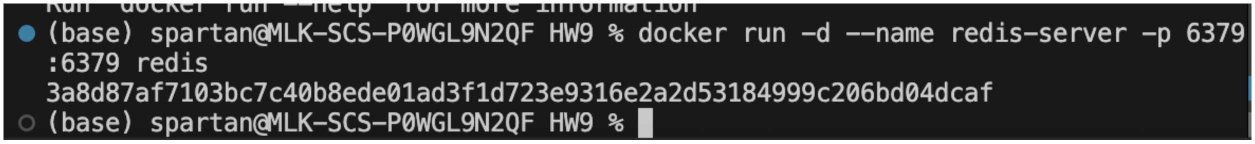
docker pull redis



```
● (base) spartan@MLK-SCS-P0WGL9N2QF HW9 % docker pull redis
  Using default tag: latest
  latest: Pulling from library/redis
  Digest: sha256:fbdbaea47b9ae4ecc2082ecdb4e1cea81e32176ffb1dcf643d422ad07427e5d9
  Status: Image is up to date for redis:latest
  docker.io/library/redis:latest
○ (base) spartan@MLK-SCS-P0WGL9N2QF HW9 % 
```

2.Run the redis container

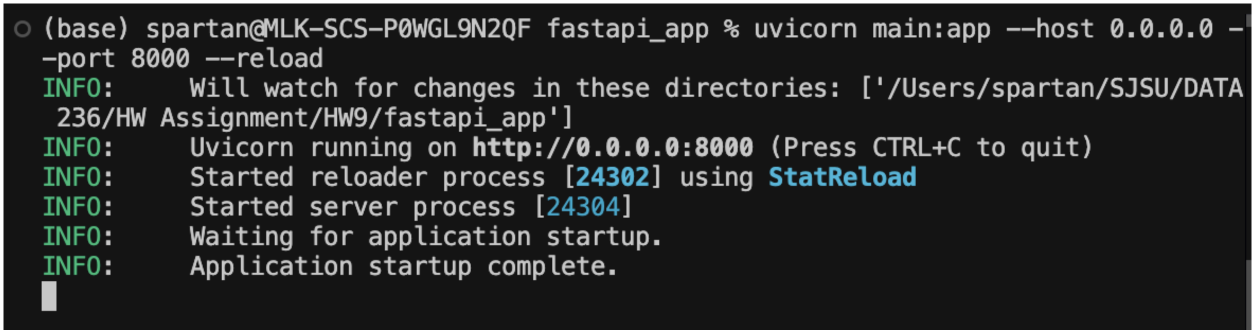
docker run -d --name redis-server -p 6379:6379 redis



```
Run  docker run --help  for more information
● (base) spartan@MLK-SCS-P0WGL9N2QF HW9 % docker run -d --name redis-server -p 6379
:6379 redis
3a8d87af7103bc7c40b8ede01ad3f1d723e9316e2a2d53184999c206bd04dcaf
○ (base) spartan@MLK-SCS-P0WGL9N2QF HW9 % █
```

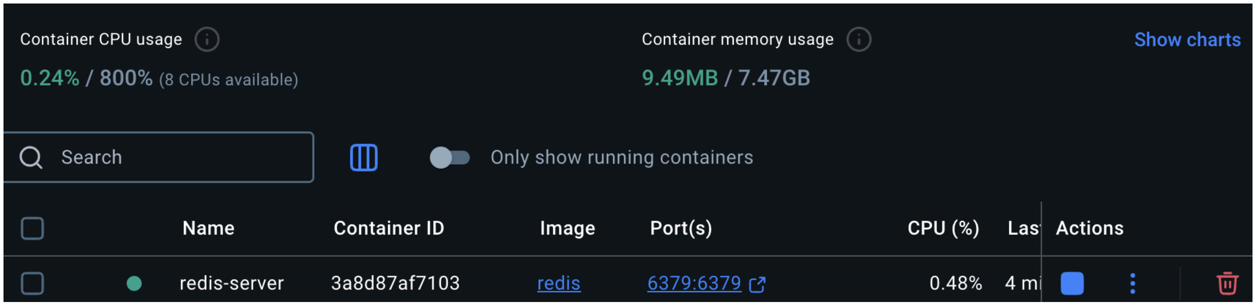
Command to Start FastAPI

uvicorn app:app --host 0.0.0.0 --port 8000 --reload



```
○ (base) spartan@MLK-SCS-P0WGL9N2QF fastapi_app % uvicorn main:app --host 0.0.0.0 -
-port 8000 --reload
INFO:     Will watch for changes in these directories: ['/Users/spartan/SJSU/DATA
 236/HW Assignment/HW9/fastapi_app']
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [24302] using StatReload
INFO:     Started server process [24304]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
█
```

Q1. Build a GET API using Node.js to fetch all student records from MongoDB(4 Points)



Container CPU usage

0.24% / 800% (8 CPUs available)

Container memory usage

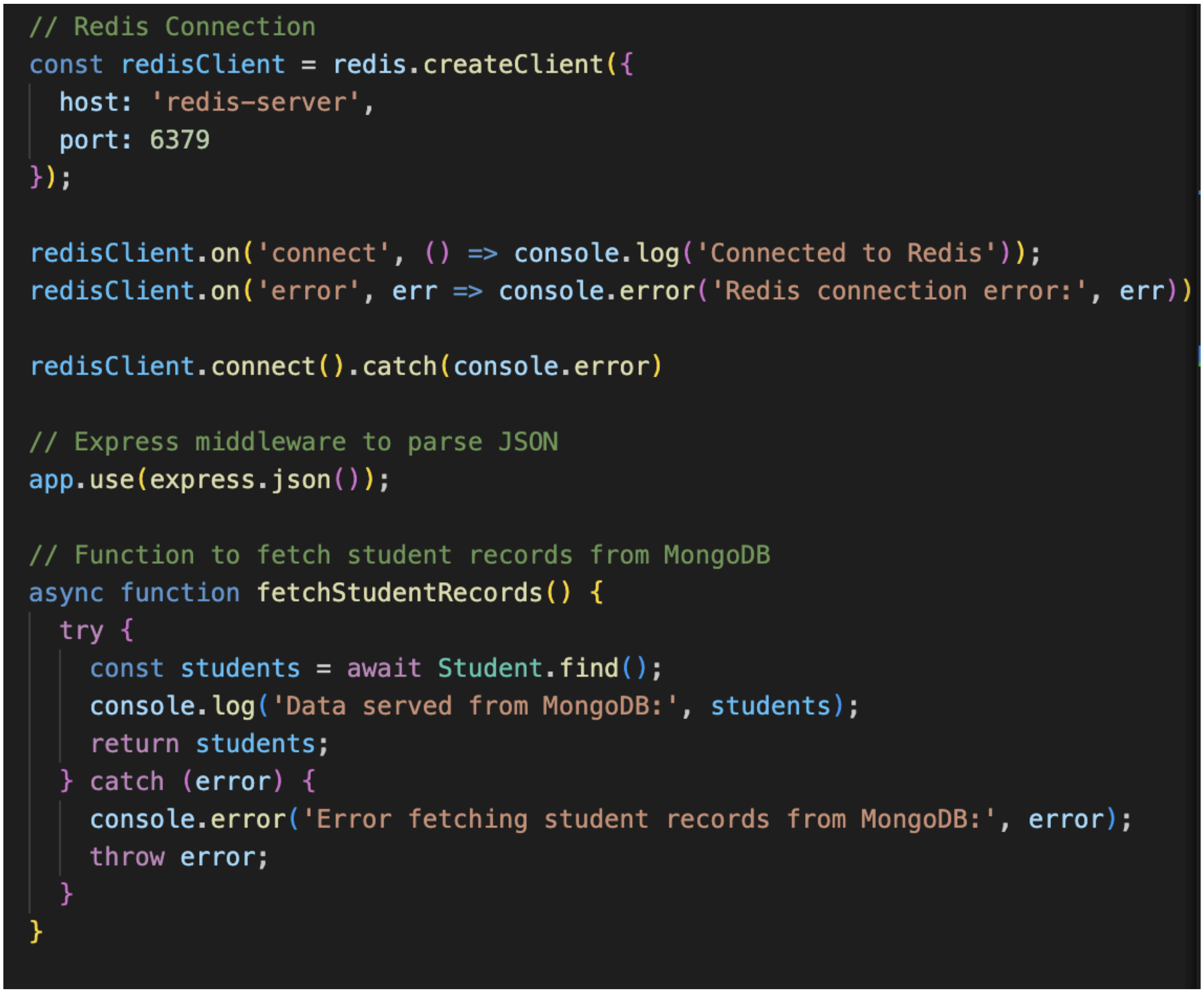
9.49MB / 7.47GB

☐		Name	Container ID	Image	Port(s)	CPU (%)	Las	Actions
☐	●	redis-server	3a8d87af7103	redis	6379:6379	0.48%	4 mi	

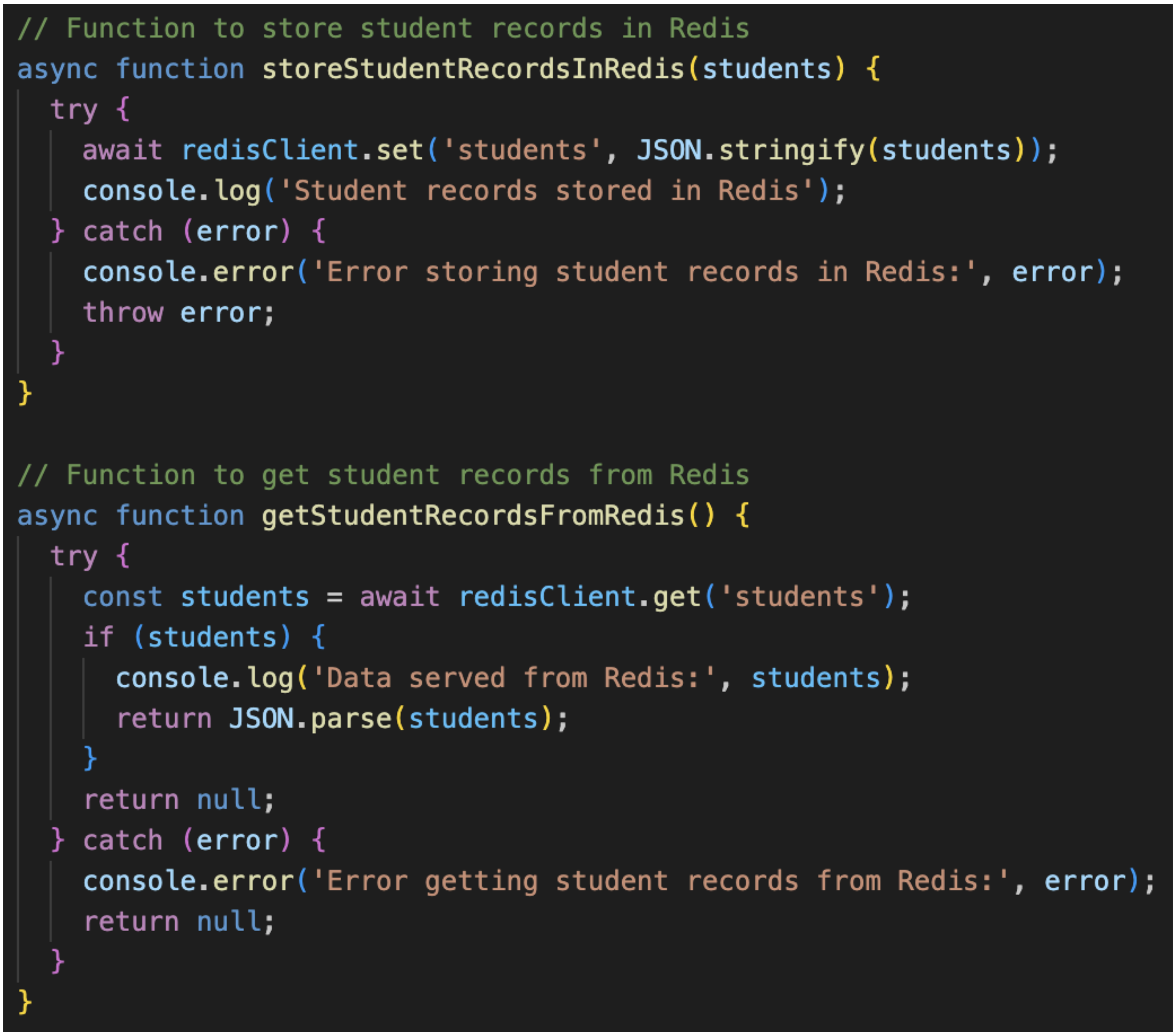
- ●Store the fetched result in Redis.

(Refer: https://redis.io/docs/latest/develop/clients/nodejs/)

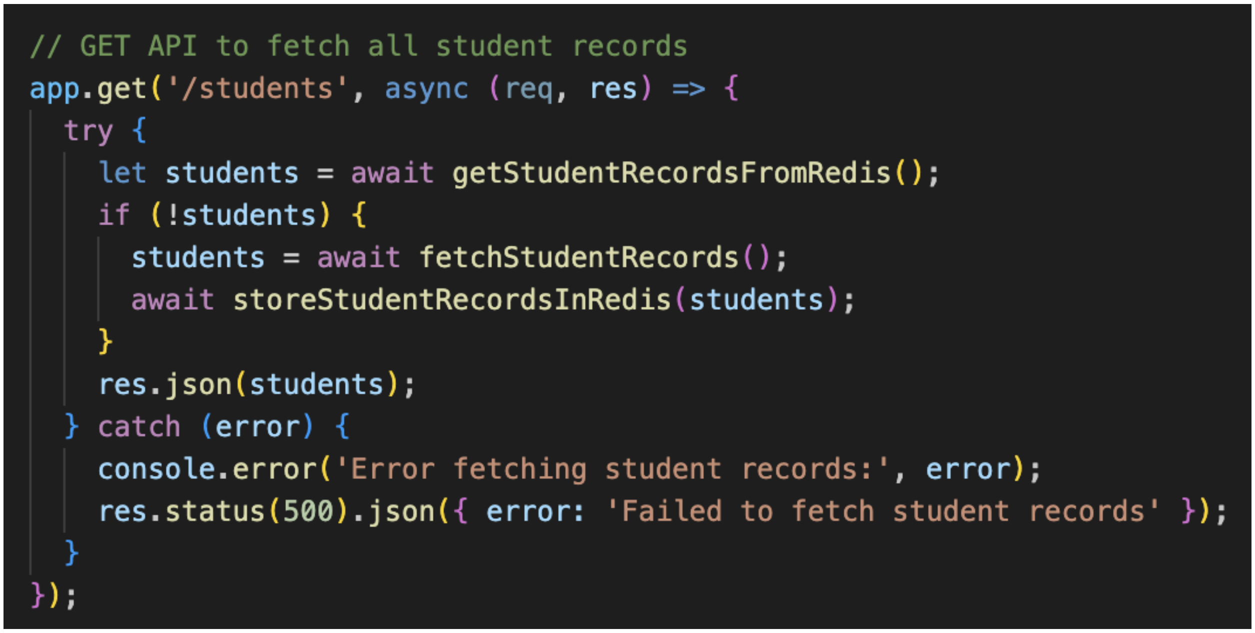
- ●If the data exists in Redis, return it from the cache.
- ●Log where the data is being served from (Redis or MongoDB).



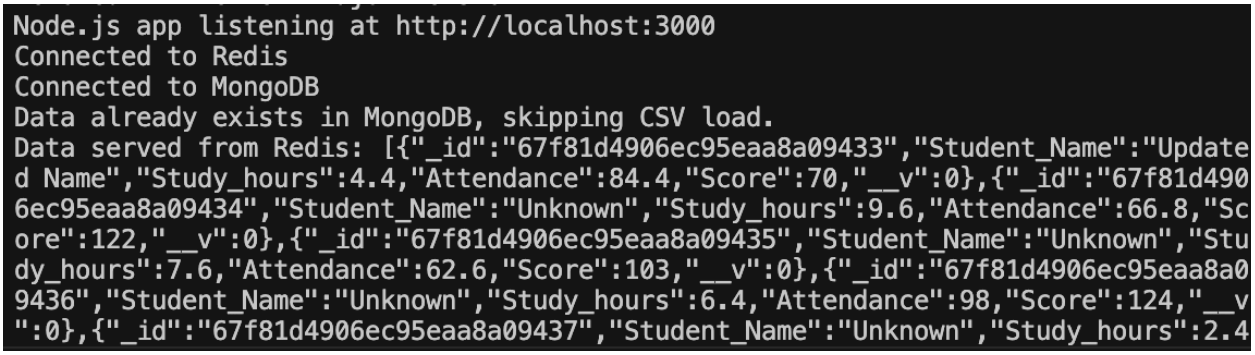
```
// Redis Connection
const redisClient = redis.createClient({
  host: 'redis-server',
  port: 6379
});

redisClient.on('connect', () => console.log('Connected to Redis'));
redisClient.on('error', err => console.error('Redis connection error:', err))

redisClient.connect().catch(console.error)

// Express middleware to parse JSON
app.use(express.json());

// Function to fetch student records from MongoDB
async function fetchStudentRecords() {
  try {
    const students = await Student.find();
    console.log('Data served from MongoDB:', students);
    return students;
  } catch (error) {
    console.error('Error fetching student records from MongoDB:', error);
    throw error;
  }
}
```



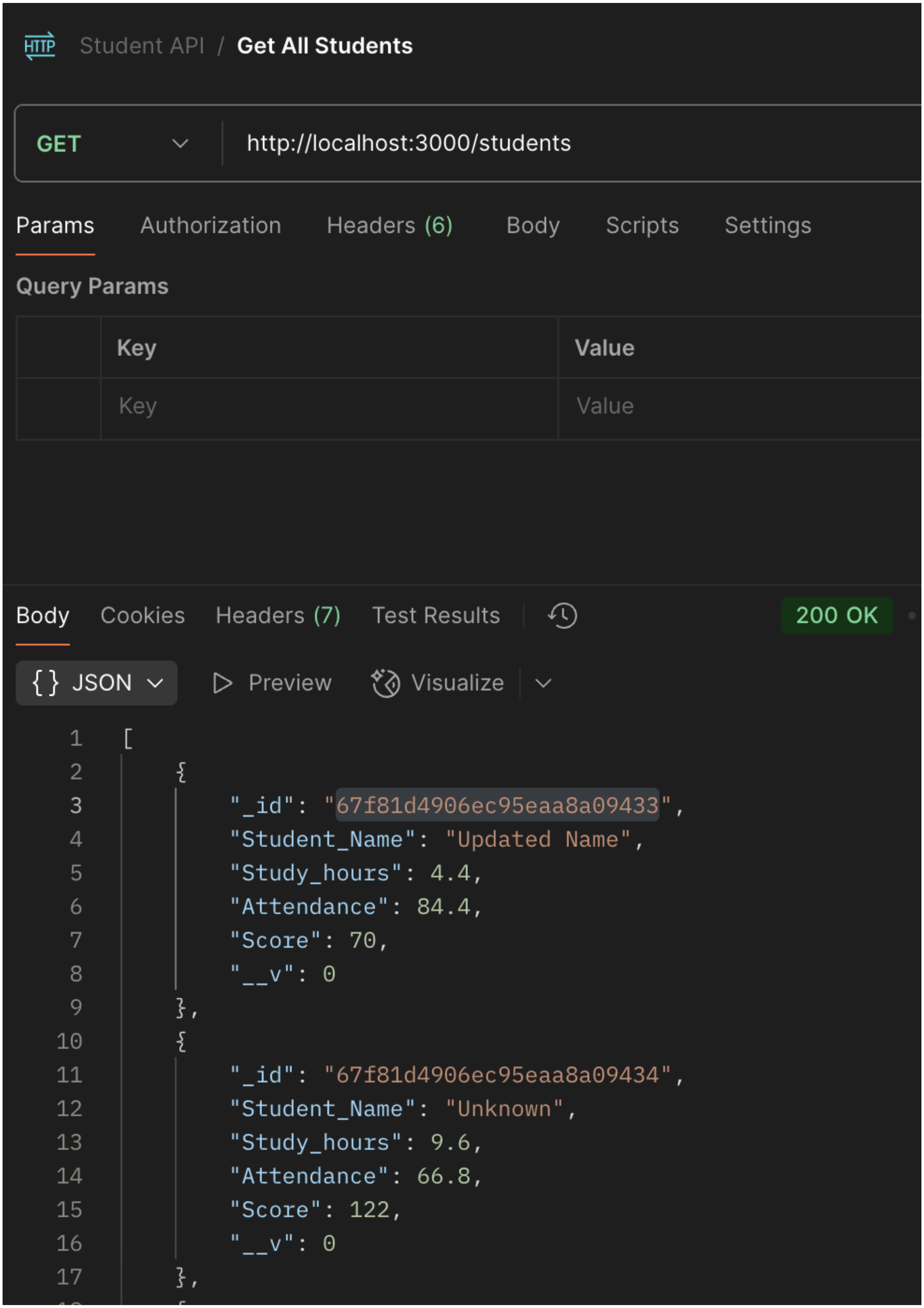
```javascript
// Function to store student records in Redis
async function storeStudentRecordsInRedis(students) {
  try {
    await redisClient.set('students', JSON.stringify(students));
    console.log('Student records stored in Redis');
  } catch (error) {
    console.error('Error storing student records in Redis:', error);
    throw error;
  }
}

// Function to get student records from Redis
async function getStudentRecordsFromRedis() {
  try {
    const students = await redisClient.get('students');
    if (students) {
      console.log('Data served from Redis:', students);
      return JSON.parse(students);
    }
    return null;
  } catch (error) {
    console.error('Error getting student records from Redis:', error);
    return null;
  }
}
```



```javascript
// GET API to fetch all student records
app.get('/students', async (req, res) => {
  try {
    let students = await getStudentRecordsFromRedis();
    if (!students) {
      students = await fetchStudentRecords();
      await storeStudentRecordsInRedis(students);
    }
    res.json(students);
  } catch (error) {
    console.error('Error fetching student records:', error);
    res.status(500).json({ error: 'Failed to fetch student records' });
  }
});
```



```
Node.js app listening at http://localhost:3000
Connected to Redis
Connected to MongoDB
Data already exists in MongoDB, skipping CSV load.
Data served from Redis: [{"_id":"67f81d4906ec95eaa8a09433","Student_Name":"Update
d Name","Study_hours":4.4,"Attendance":84.4,"Score":70,"__v":0},{"_id":"67f81d490
6ec95eaa8a09434","Student_Name":"Unknown","Study_hours":9.6,"Attendance":66.8,"Sc
ore":122,"__v":0},{"_id":"67f81d4906ec95eaa8a09435","Student_Name":"Unknown","Stu
dy_hours":7.6,"Attendance":62.6,"Score":103,"__v":0},{"_id":"67f81d4906ec95eaa8a0
9436","Student_Name":"Unknown","Study_hours":6.4,"Attendance":98,"Score":124,"__v
":0},{"_id":"67f81d4906ec95eaa8a09437","Student_Name":"Unknown","Study_hours":2.4
```

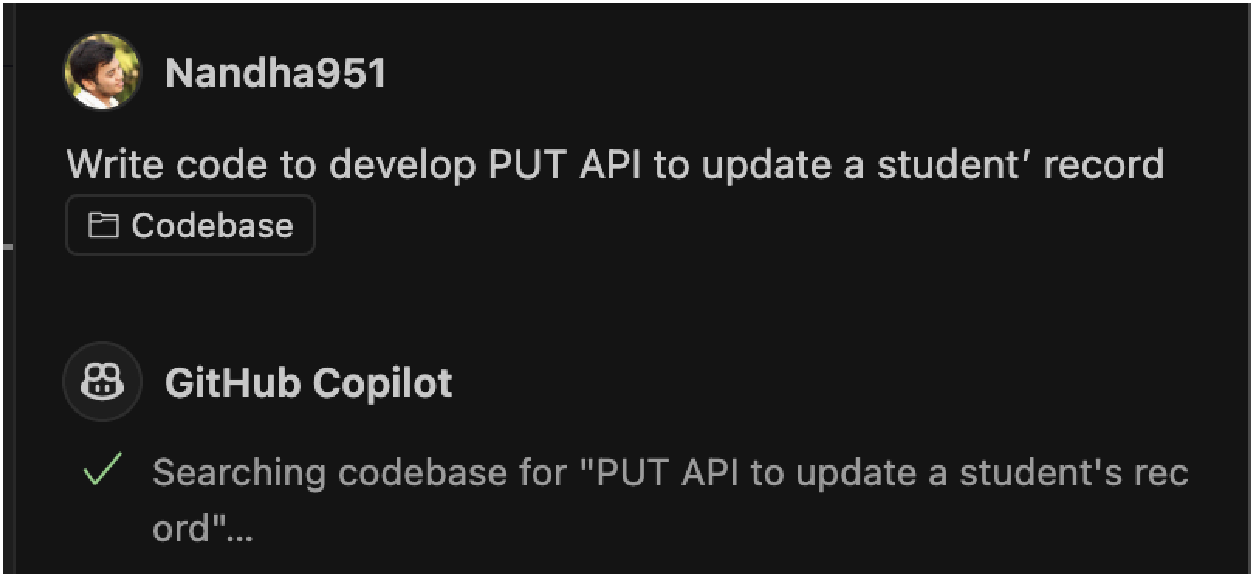


```
GET http://localhost:3000/students
```

200 OK

```
[
    {
        "_id": "67f81d4906ec95eaa8a09433",
        "Student_Name": "Updated Name",
        "Study_hours": 4.4,
        "Attendance": 84.4,
        "Score": 70,
        "__v": 0
    },
    {
        "_id": "67f81d4906ec95eaa8a09434",
        "Student_Name": "Unknown",
        "Study_hours": 9.6,
        "Attendance": 66.8,
        "Score": 122,
        "__v": 0
    },
```

Q2. Use GitHub Copilot to assist in writing the PUT API to update a student’s record.(2 Points)

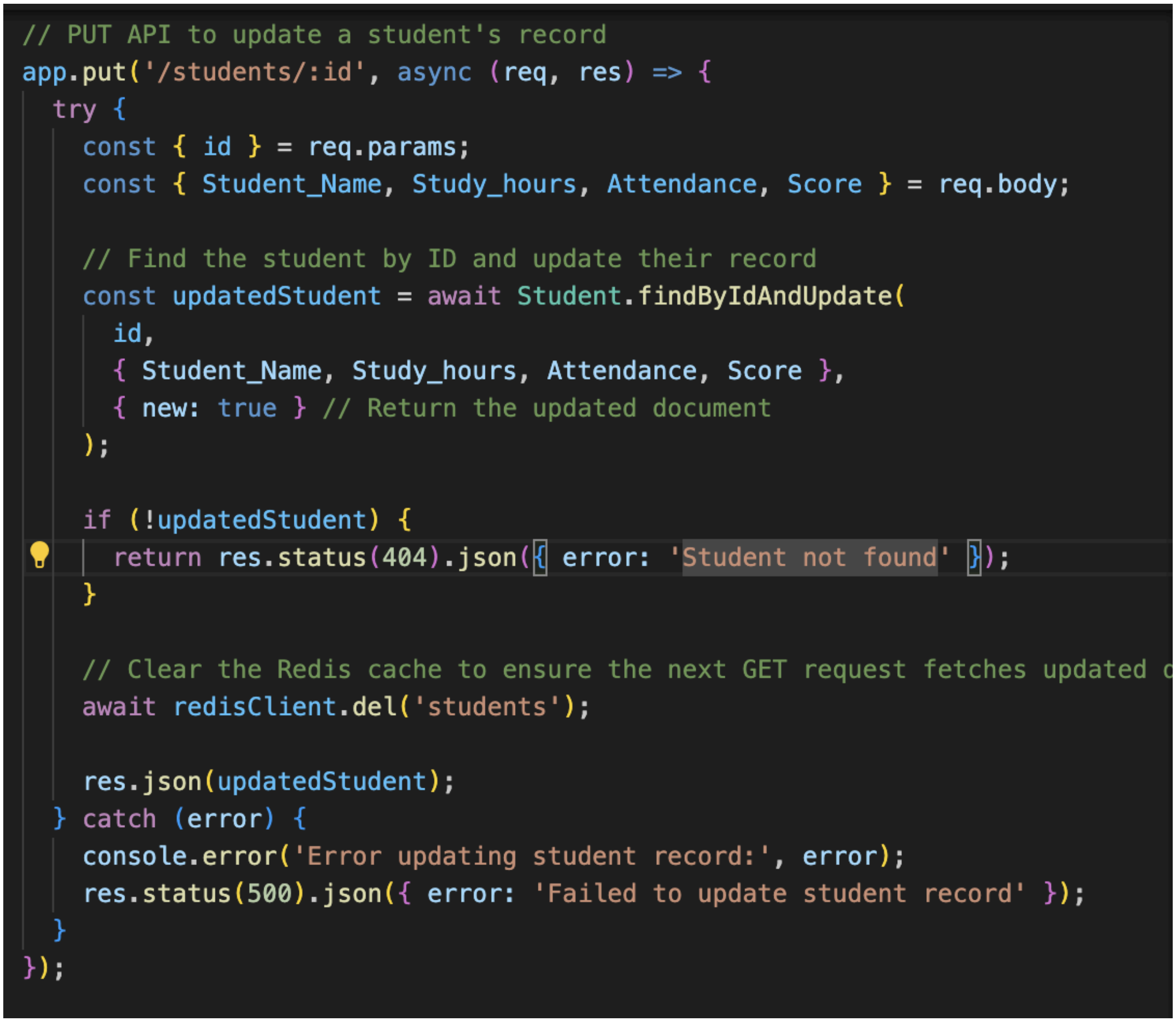


Nandha951

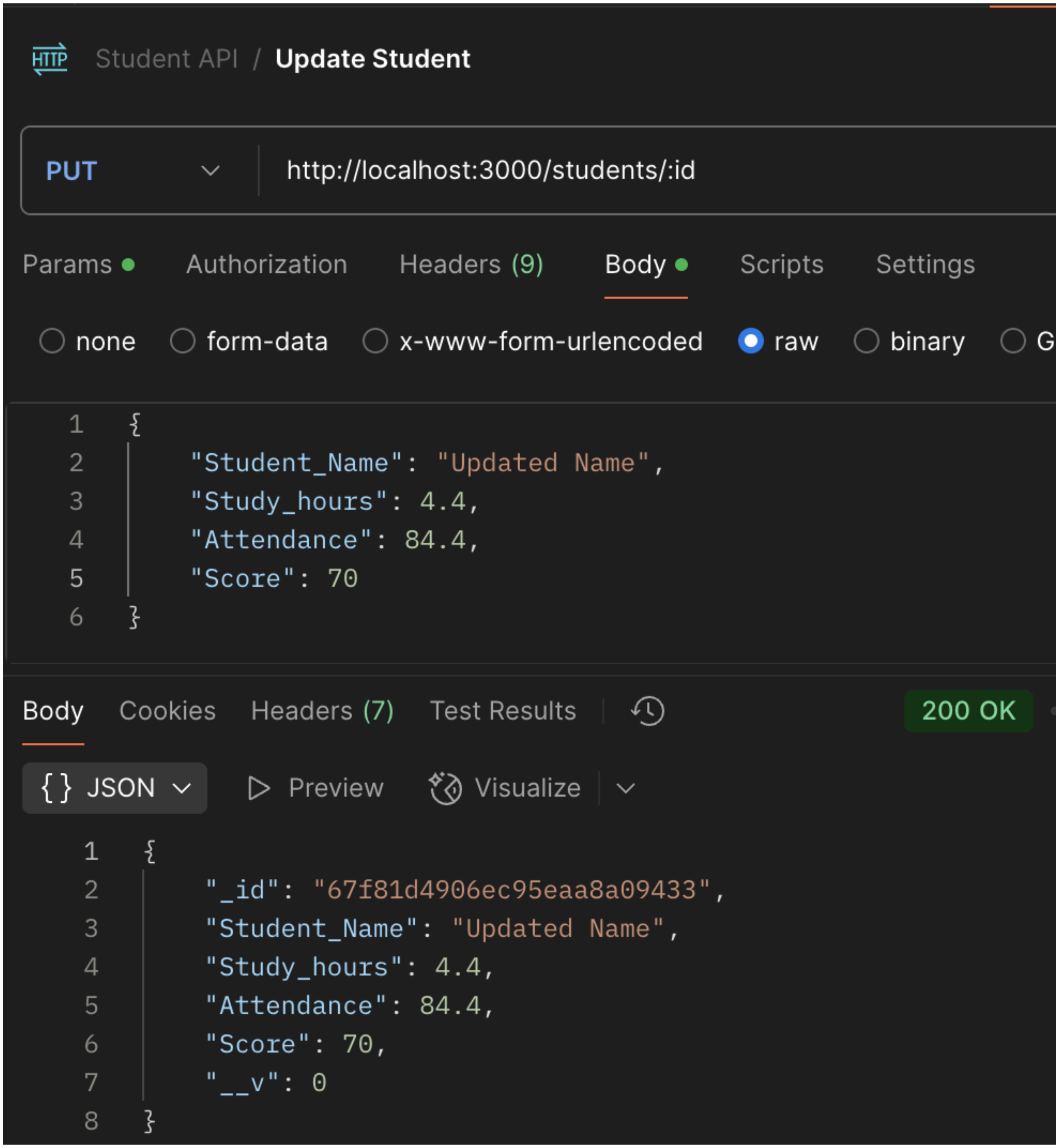
Write code to develop PUT API to update a student’ record

GitHub Copilot

✓ Searching codebase for "PUT API to update a student's record"...



```
// PUT API to update a student's record
app.put('/students/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const { Student_Name, Study_hours, Attendance, Score } = req.body;

    // Find the student by ID and update their record
    const updatedStudent = await Student.findByIdAndUpdate(
      id,
      { Student_Name, Study_hours, Attendance, Score },
      { new: true } // Return the updated document
    );

    if (!updatedStudent) {
      return res.status(404).json({ error: 'Student not found' });
    }

    // Clear the Redis cache to ensure the next GET request fetches updated c
    await redisClient.del('students');

    res.json(updatedStudent);
  } catch (error) {
    console.error('Error updating student record:', error);
    res.status(500).json({ error: 'Failed to update student record' });
  }
});
```



PUT http://localhost:3000/students/:id

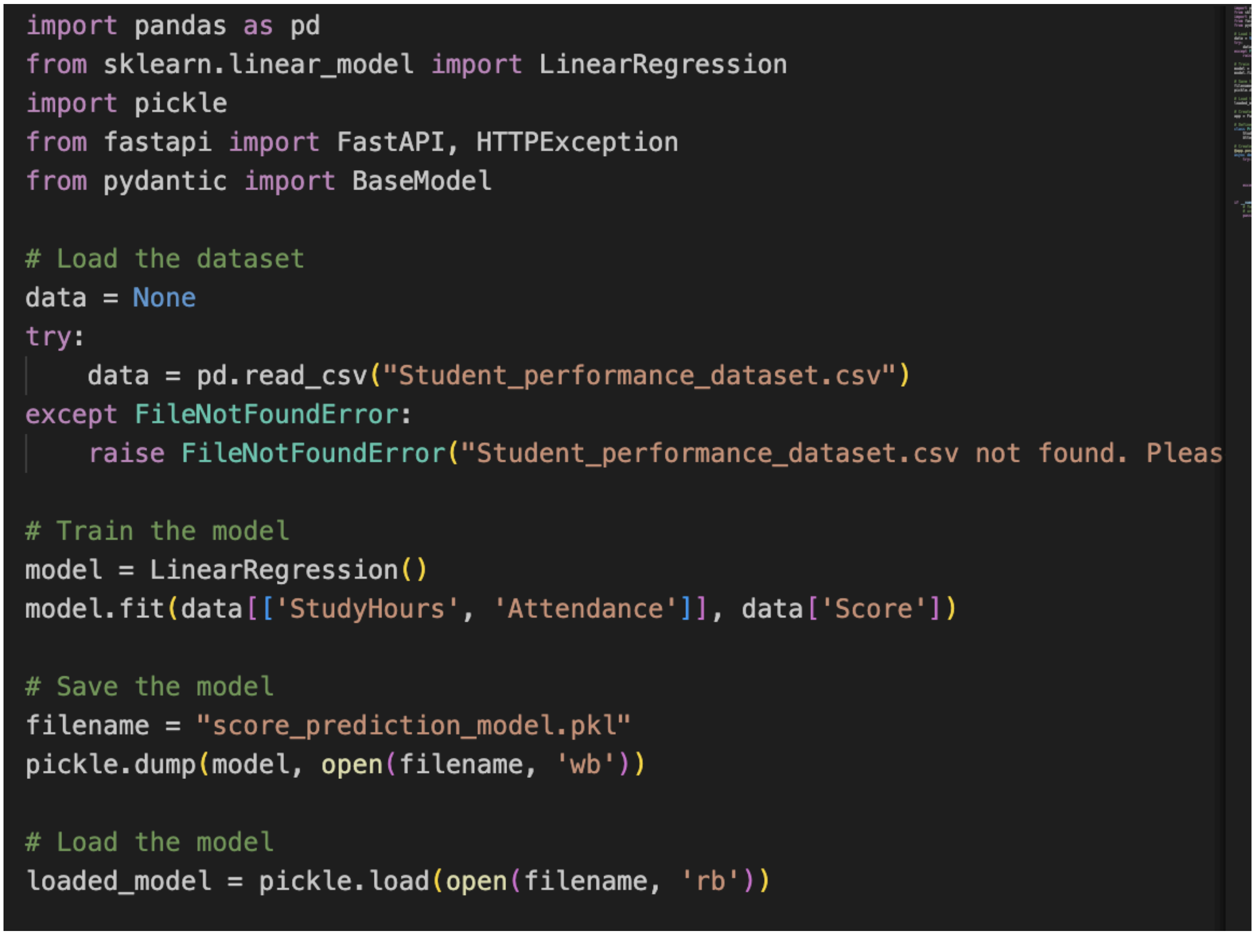
```
{
    "Student_Name": "Updated Name",
    "Study_hours": 4.4,
    "Attendance": 84.4,
    "Score": 70
}
```

200 OK

```
{
    "_id": "67f81d4906ec95eaa8a09433",
    "Student_Name": "Updated Name",
    "Study_hours": 4.4,
    "Attendance": 84.4,
    "Score": 70,
    "__v": 0
}
```

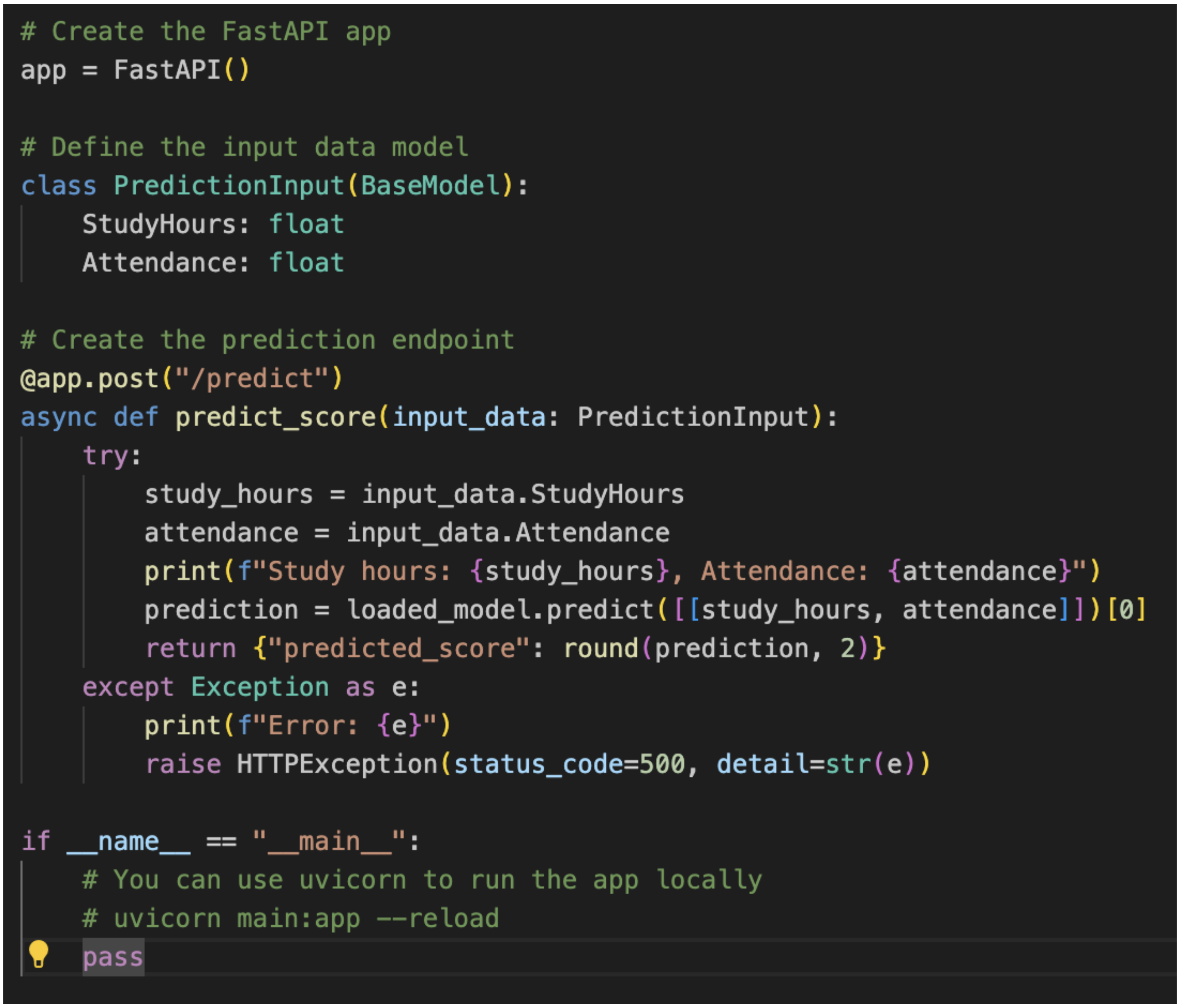
Q3. Build a score prediction system using FastAPI and a machine learning model.(4 Points)

- Use the provided dataset: Student_performance_dataset.csv
- Train a simple regression model to predict score based on study_hours and attendance.
- Save the model as a .pkl file.



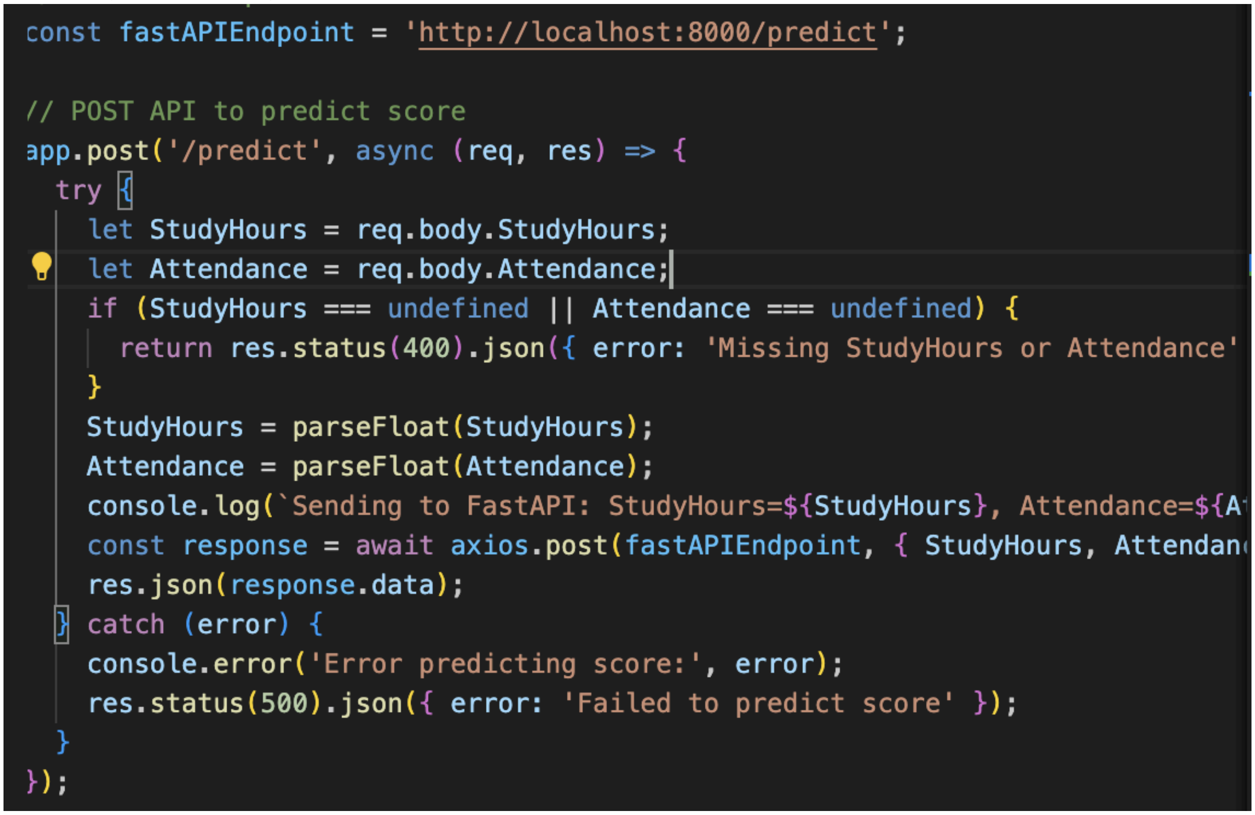
```
import pandas as pd
from sklearn.linear_model import LinearRegression
import pickle
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel

# Load the dataset
data = None
try:
    data = pd.read_csv("Student_performance_dataset.csv")
except FileNotFoundError:
    raise FileNotFoundError("Student_performance_dataset.csv not found. Pleas

# Train the model
model = LinearRegression()
model.fit(data[['StudyHours', 'Attendance']], data['Score'])

# Save the model
filename = "score_prediction_model.pkl"
pickle.dump(model, open(filename, 'wb'))

# Load the model
loaded_model = pickle.load(open(filename, 'rb'))
```

- Create a FastAPI endpoint (/predict) that:
- Accepts study hours and attendance
- Returns the predicted score



```python
# Create the FastAPI app
app = FastAPI()

# Define the input data model
class PredictionInput(BaseModel):
    StudyHours: float
    Attendance: float

# Create the prediction endpoint
@app.post("/predict")
async def predict_score(input_data: PredictionInput):
    try:
        study_hours = input_data.StudyHours
        attendance = input_data.Attendance
        print(f"Study hours: {study_hours}, Attendance: {attendance}")
        prediction = loaded_model.predict([[study_hours, attendance]])[0]
        return {"predicted_score": round(prediction, 2)}
    except Exception as e:
        print(f"Error: {e}")
        raise HTTPException(status_code=500, detail=str(e))

if __name__ == "__main__":
    # You can use uvicorn to run the app locally
    # uvicorn main:app --reload
    pass
```

- From your Node.js app, make a POST request to this FastAPI endpoint and return the prediction.



```
const fastAPIEndpoint = 'http://localhost:8000/predict';

// POST API to predict score
app.post('/predict', async (req, res) => {
  try {
    let StudyHours = req.body.StudyHours;
    let Attendance = req.body.Attendance;
    if (StudyHours === undefined || Attendance === undefined) {
      return res.status(400).json({ error: 'Missing StudyHours or Attendance'
    }
    StudyHours = parseFloat(StudyHours);
    Attendance = parseFloat(Attendance);
    console.log(`Sending to FastAPI: StudyHours=${StudyHours}, Attendance=${A
    const response = await axios.post(fastAPIEndpoint, { StudyHours, Attendan
    res.json(response.data);
  } catch (error) {
    console.error('Error predicting score:', error);
    res.status(500).json({ error: 'Failed to predict score' });
  }
});
```