Docker

Aim

The main aim of this assignment is to build a simple full-stack application where the **frontend is developed using Node.js (Express)** and the **backend is created using Flask**, and both services run together using **Docker and Docker Compose**.

Introduction

In modern web development, different technologies are often used together for frontend and backend development. In this project, I created a frontend using **Node.js** to collect user input, and a backend using **Flask** to process that input.

The frontend sends data to the backend through an API request, and the backend returns the result (such as grade calculation, file writing, or showing stored data). Both applications are containerized using **Docker**, which makes the setup portable and easy to run on any system.

How the System Works

1. Frontend (Node.js Application)

- The frontend displays a form where the user can:
    - Enter a score to check their grade.
    - Enter a student name and grade to store.
    - Request to view stored student records.
    - Enter text to save into a file.
- When the form is submitted, the data is sent to the backend through a POST API request.

2. Backend (Flask Application)

- The backend receives data from the frontend and performs the required task.

- It supports:
    1. Grade calculation
    2. Saving student details
    3. Showing all stored students
    4. Writing text to a file
    5. Reading text from the file
    6. A simple health check to verify backend is running

3. Docker and Docker Compose

- Both applications have their own Dockerfile.
- Docker Compose is used so both containers start together and share the same network.
- The frontend calls the backend container using the service name defined in `docker-compose.yml`.

How to Run the Project

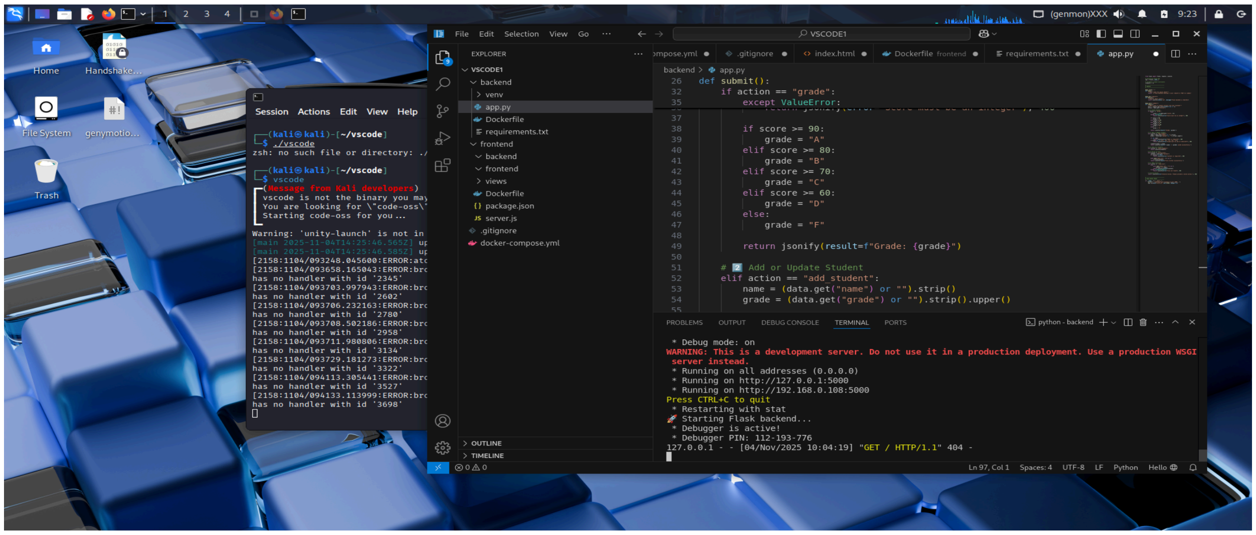
Once everything is set up:

```
docker compose up --build
```

- Frontend runs on: **http://localhost:3000**
- Backend runs on: **http://localhost:5000**

Screenshots :

1. Frontend Interface



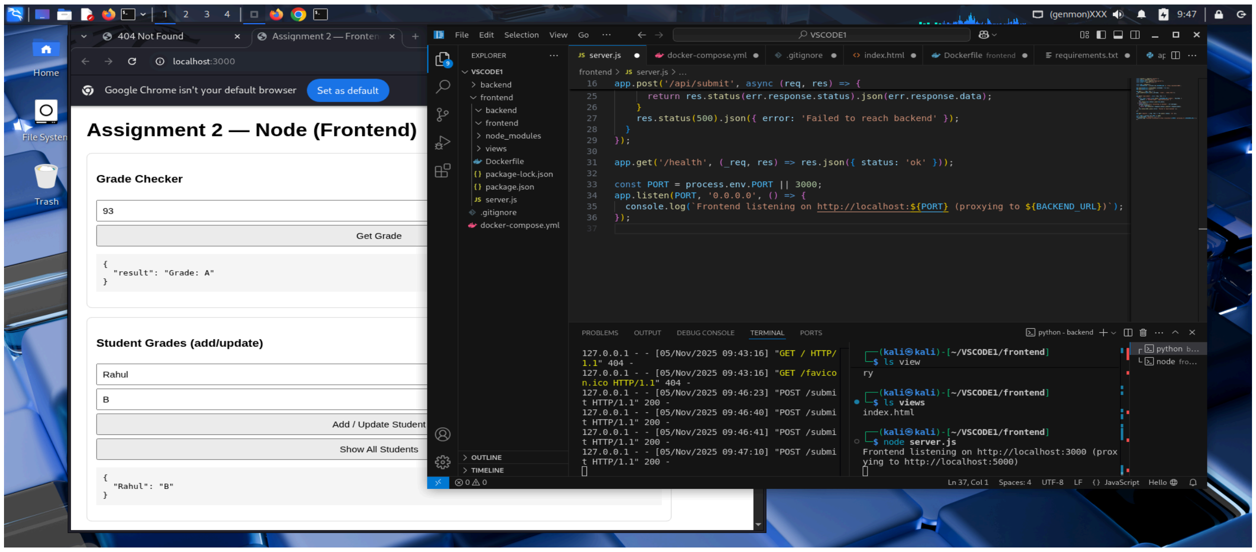
```
┌──(kali㉿kali)-[~/vscode]
└─$ ./vscode
zsh: no such file or directory: ./

┌──(kali㉿kali)-[~/vscode]
└─$ vscode
┌(Message from Kali developers)
│ vscode is not the binary you may
│ You are looking for \"code-oss\"
│ Starting code-oss for you...
└
Warning: 'unity-launch' is not in
[main 2025-11-04T14:25:46.565Z] up
[main 2025-11-04T14:25:46.585Z] up
[2158:1104/093248.045600:ERROR:atc
[2158:1104/093658.165043:ERROR:brc
has no handler with id '2345'
[2158:1104/093703.997943:ERROR:brc
has no handler with id '2602'
[2158:1104/093706.232163:ERROR:brc
has no handler with id '2780'
[2158:1104/093708.502186:ERROR:brc
has no handler with id '2958'
[2158:1104/093711.980806:ERROR:brc
has no handler with id '3134'
[2158:1104/093729.181273:ERROR:brc
has no handler with id '3322'
[2158:1104/094113.305441:ERROR:brc
has no handler with id '3527'
[2158:1104/094133.113999:ERROR:brc
has no handler with id '3698'
```

backend > app.py

```python
def submit():
    if action == "grade":
        except ValueError:

            if score >= 90:
                grade = "A"
            elif score >= 80:
                grade = "B"
            elif score >= 70:
                grade = "C"
            elif score >= 60:
                grade = "D"
            else:
                grade = "F"

            return jsonify(result=f"Grade: {grade}")

        # 2️⃣ Add or Update Student
        elif action == "add_student":
            name = (data.get("name") or "").strip()
            grade = (data.get("grade") or "").strip().upper()
```

```
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
 server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.0.108:5000
Press CTRL+C to quit
 * Restarting with stat
🚀 Starting Flask backend...
 * Debugger is active!
 * Debugger PIN: 112-193-776
127.0.0.1 - - [04/Nov/2025 10:04:19] "GET / HTTP/1.1" 404 -
```

2. Grade Calculation Output



Assignment 2 — Node (Frontend)

Grade Checker

93

Get Grade

```
{
  "result": "Grade: A"
}
```

Student Grades (add/update)

Rahul

B

Add / Update Student

Show All Students

```
{
  "Rahul": "B"
}
```

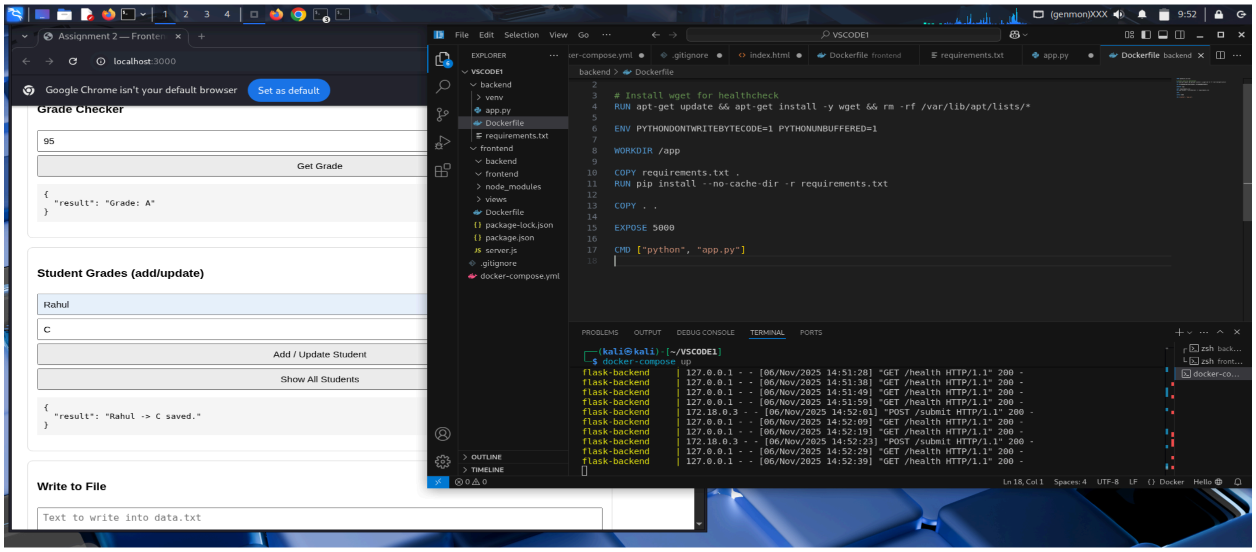
frontend > server.js > ...

```javascript
app.post('/api/submit', async (req, res) => {
      return res.status(err.response.status).json(err.response.data);
    }
    res.status(500).json({ error: 'Failed to reach backend' });
  }
});

app.get('/health', (_req, res) => res.json({ status: 'ok' }));

const PORT = process.env.PORT || 3000;
app.listen(PORT, '0.0.0.0', () => {
  console.log(`Frontend listening on http://localhost:${PORT} (proxying to ${BACKEND_URL})`);
});
```

```
127.0.0.1 - - [05/Nov/2025 09:43:16] "GET / HTTP/
1.1" 404 -
127.0.0.1 - - [05/Nov/2025 09:43:16] "GET /favico
n.ico HTTP/1.1" 404 -
127.0.0.1 - - [05/Nov/2025 09:46:23] "POST /submi
t HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2025 09:46:40] "POST /submi
t HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2025 09:46:41] "POST /submi
t HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2025 09:47:10] "POST /submi
t HTTP/1.1" 200 -
```

```
┌──(kali㉿kali)-[~/VSCODE1/frontend]
└─$ ls view
ry

┌──(kali㉿kali)-[~/VSCODE1/frontend]
└─$ ls views
index.html

┌──(kali㉿kali)-[~/VSCODE1/frontend]
└─$ node server.js
Frontend listening on http://localhost:3000 (prox
ying to http://localhost:5000)
```

3. Adding or Updating Student Record



Grade Checker

95

Get Grade

```
{
  "result": "Grade: A"
}
```

Student Grades (add/update)

Rahul

C

Add / Update Student

Show All Students

```
{
  "result": "Rahul -> C saved."
}
```

Write to File

Text to write into data.txt

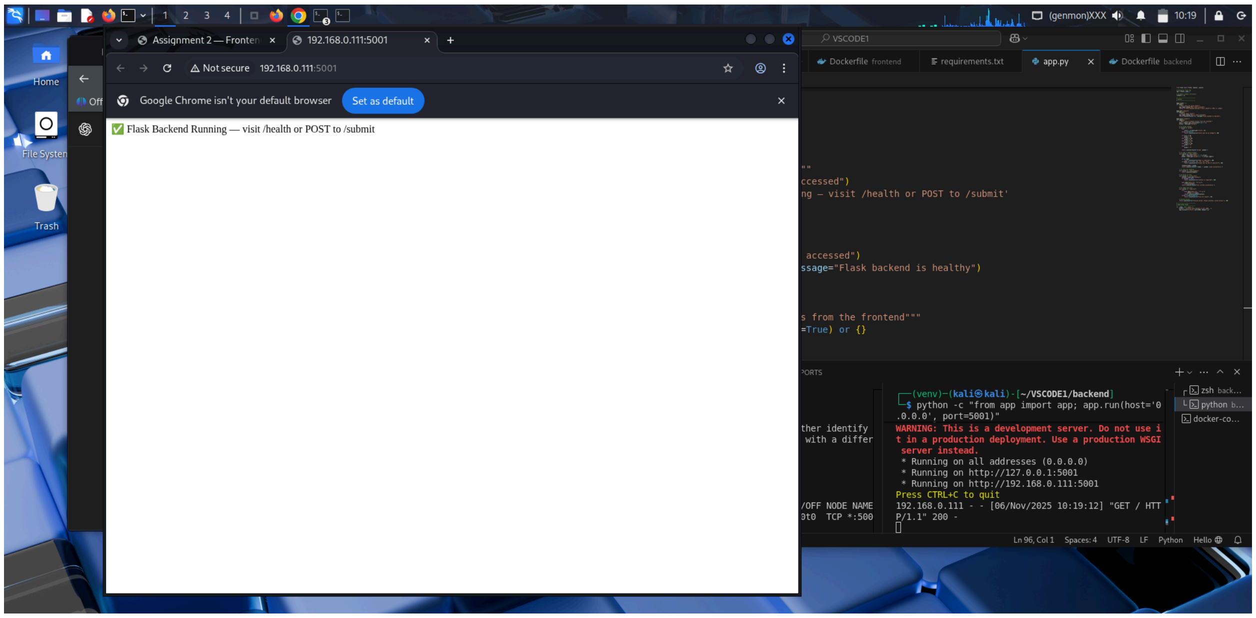
backend > Dockerfile

```dockerfile

# Install wget for healthcheck
RUN apt-get update && apt-get install -y wget && rm -rf /var/lib/apt/lists/*

ENV PYTHONDONTWRITEBYTECODE=1 PYTHONUNBUFFERED=1

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "app.py"]
```

```
┌──(kali㉿kali)-[~/VSCODE1]
└─$ docker-compose up
flask-backend    | 127.0.0.1 - - [06/Nov/2025 14:51:28] "GET /health HTTP/1.1" 200 -
flask-backend    | 127.0.0.1 - - [06/Nov/2025 14:51:38] "GET /health HTTP/1.1" 200 -
flask-backend    | 127.0.0.1 - - [06/Nov/2025 14:51:49] "GET /health HTTP/1.1" 200 -
flask-backend    | 127.0.0.1 - - [06/Nov/2025 14:51:59] "GET /health HTTP/1.1" 200 -
flask-backend    | 172.18.0.3 - - [06/Nov/2025 14:52:01] "POST /submit HTTP/1.1" 200 -
flask-backend    | 127.0.0.1 - - [06/Nov/2025 14:52:09] "GET /health HTTP/1.1" 200 -
flask-backend    | 127.0.0.1 - - [06/Nov/2025 14:52:19] "GET /health HTTP/1.1" 200 -
flask-backend    | 172.18.0.3 - - [06/Nov/2025 14:52:23] "POST /submit HTTP/1.1" 200 -
flask-backend    | 127.0.0.1 - - [06/Nov/2025 14:52:29] "GET /health HTTP/1.1" 200 -
flask-backend    | 127.0.0.1 - - [06/Nov/2025 14:52:39] "GET /health HTTP/1.1" 200 -
```

4. Viewing Stored Student Records

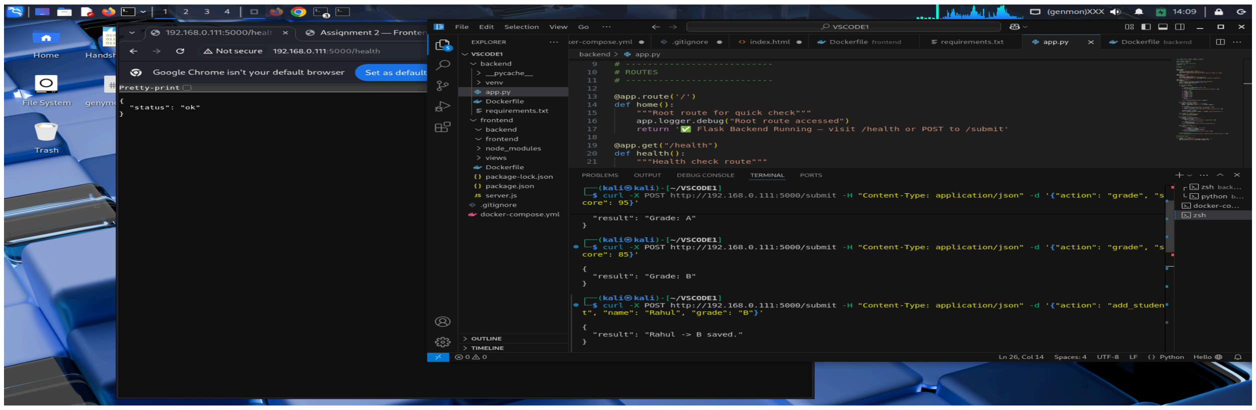
5. Writing and Reading from File



```
✅ Flask Backend Running — visit /health or POST to /submit
```

```
┌──(venv)─(kali㉿kali)-[~/VSCODE1/backend]
└─$ python -c "from app import app; app.run(host='0.0.0.0', port=5001)"
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5001
 * Running on http://192.168.0.111:5001
Press CTRL+C to quit
192.168.0.111 - - [06/Nov/2025 10:19:12] "GET / HTTP/1.1" 200 -
```

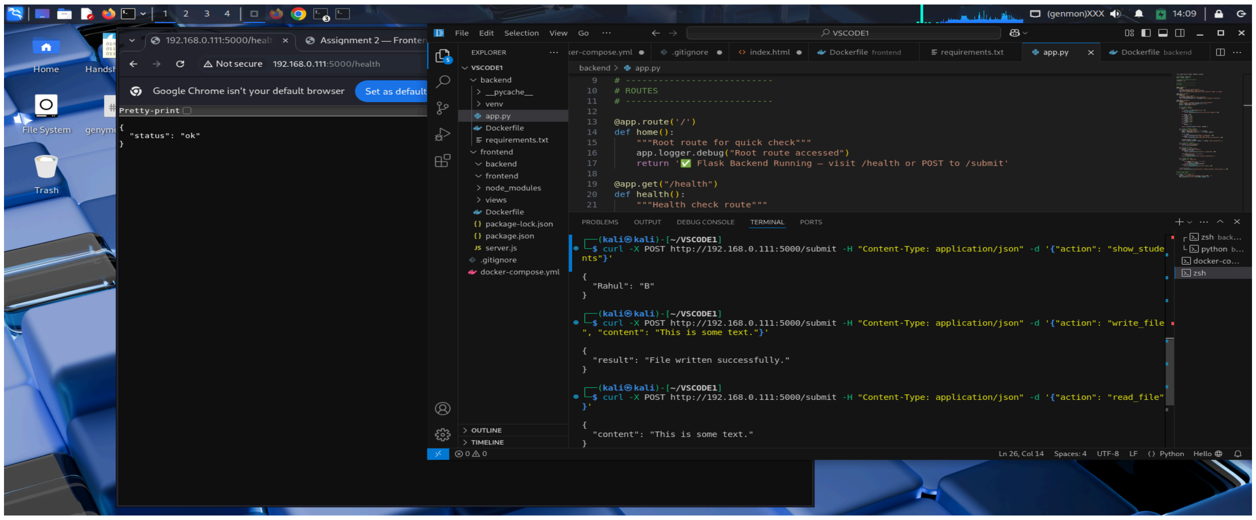
6. Docker Containers Running Successfully



```
{
  "status": "ok"
}
```

```python
# ---------------------------
# ROUTES
# ---------------------------

@app.route('/')
def home():
    """Root route for quick check"""
    app.logger.debug("Root route accessed")
    return '✅ Flask Backend Running — visit /health or POST to /submit'

@app.get("/health")
def health():
    """Health check route"""
```

```
┌──(kali㉿kali)-[~/VSCODE1]
└─$ curl -X POST http://192.168.0.111:5000/submit -H "Content-Type: application/json" -d '{"action": "grade", "score": 95}'
  "result": "Grade: A"
}

┌──(kali㉿kali)-[~/VSCODE1]
└─$ curl -X POST http://192.168.0.111:5000/submit -H "Content-Type: application/json" -d '{"action": "grade", "score": 85}'
{
  "result": "Grade: B"
}

┌──(kali㉿kali)-[~/VSCODE1]
└─$ curl -X POST http://192.168.0.111:5000/submit -H "Content-Type: application/json" -d '{"action": "add_student", "name": "Rahul", "grade": "B"}'
{
  "result": "Rahul -> B saved."
}
```



```
{
  "status": "ok"
}
```

```
┌──(kali㉿kali)-[~/VSCODE1]
└─$ curl -X POST http://192.168.0.111:5000/submit -H "Content-Type: application/json" -d '{"action": "show_students"}'
{
  "Rahul": "B"
}

┌──(kali㉿kali)-[~/VSCODE1]
└─$ curl -X POST http://192.168.0.111:5000/submit -H "Content-Type: application/json" -d '{"action": "write_file", "content": "This is some text."}'
{
  "result": "File written successfully."
}

┌──(kali㉿kali)-[~/VSCODE1]
└─$ curl -X POST http://192.168.0.111:5000/submit -H "Content-Type: application/json" -d '{"action": "read_file"}'
{
  "content": "This is some text."
}
```

Sample Output Summary

Operation	User Input	System Output
Grade Checking	Score: 93	Grade: A
Add Student	Rahul, B	Stored Successfully
View Students	—	{ Rahul: B }
Write to File	"Hello World"	Text Saved
Read File	—	"Hello World"
Check Backend	GET /health	{ status: "ok" }

Conclusion

This assignment helped me understand how different technologies can communicate with each other in a real application. I learned how to send data between a frontend and backend, handle responses, store information temporarily, and even read/write files.

Containerizing both the frontend and backend using Docker also made the deployment process much easier. With Docker Compose, both services could be started with a single command and worked together without additional configuration.

Overall, the assignment was useful for learning **full-stack integration**, **API communication**, and **Docker-based deployment** in a practical environment.