1. **Frontend (Node.js with Express)**:

- Create a frontend using Express and Node.js.
- Include a form similar to the one from the Flask Assignment 2.
- Configure the form to send a request to the Flask backend.
- **Backend (Flask)**:
- Use the Flask backend to handle the form submission and process the data.
- **Folder Structure**:
- Organize the project with separate folders for the frontend and backend.
- **Docker Configuration**:
- Create a `Dockerfile` for both the frontend and backend.
- Write a `.yaml` file (Docker Compose) to connect both services in the same network.
- Upload both images to docker hub and push your whole code to github and add the node_modules and other non required files(.vscode) in .gitignore

. /backend/app.py

```
from flask import Flask, request, jsonify
import os
import psycopg2
app = Flask(__name__)
# Root route
@app.route("/", methods=["GET"])
def home():
    return jsonify({"message": "Backend is running!"})
# API data route
@app.route("/api/data", methods=["GET"])
def get_data():
    return jsonify({"message": "Backend is working", "status": "success"})
# Submit data route
@app.route("/submit", methods=["POST"])
def submit_form():
```

```
    data = request.get_json()
    name = data.get("name")
    email = data.get("email")
    return jsonify({"message": f"Received {name} with email {email}"})
# Database health check
@app.route("/db-check", methods=["GET"])
def db_check():
    try:
        conn = psycopg2.connect(
            host="db",  # Docker service name
            database=os.getenv("POSTGRES_DB"),
            user=os.getenv("POSTGRES_USER"),
            password=os.getenv("POSTGRES_PASSWORD"),
            port=5432
        )
        cur = conn.cursor()
        cur.execute("SELECT version();")
        db_version = cur.fetchone()[0]
        cur.close()
        conn.close()
        return jsonify({"db_version": db_version})
    except Exception as e:
        return jsonify({"error": str(e)}), 500
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

- **Purpose**:
  This is the **Flask backend application**.
- **What it does**:
  - Listens on port 5000 (inside container).
  - Provides **API endpoints**:
    - /api/data → GET → Returns sample backend message.
    - /submit → POST → Accepts JSON data (name, email) and returns confirmation.
    - /db-check → GET → Checks PostgreSQL DB connection using psycopg2 and returns version info.
- **Why important**:
  This is the main backend logic. It processes requests from the frontend or tools like curl.

2. /backend/requirements.txt

flask

flask-cors

psycopg2-binary

- **Purpose**:
  Lists **Python dependencies** required for the backend.
- **What it contains**:

flask
psycopg2-binary

- **Why important**:
  Used by pip inside Docker to install all Python libraries for Flask and PostgreSQL connectivity.

3. /backend/Dockerfile

FROM python:3.11-slim

```
WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

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

- **Purpose**:
  Tells Docker **how to build** the backend image.
- **Steps inside**:
  - Use python:3.10 base image.
  - Set working directory to /app.
  - Copy requirements.txt and install packages with pip.
  - Copy all backend files into container.
  - Run python app.py when the container starts.
- **Why important**:
  Without it, Docker wouldn't know how to set up the backend environment.

4. /frontend/server.js

```
const express = require("express");

const axios = require("axios");

const cors = require("cors");

const app = express();

app.use(cors());

app.use(express.json()); // So we can handle JSON bodies


// Test route

app.get("/", (req, res) => {

  res.send("Frontend is running!");

});


// Fetch data from backend

app.get("/fetch-backend", async (req, res) => {
```

```
  try {
    const response = await axios.get("http://backend:5000/api/data");
    res.json(response.data);
  } catch (error) {
    res.status(500).json({ error: "Backend not reachable" });
  }
});

// Send data to backend
app.post("/send-to-backend", async (req, res) => {
  try {
    const response = await axios.post("http://backend:5000/submit", req.body);
    res.json(response.data);
  } catch (error) {
    res.status(500).json({ error: "Error sending data to backend" });
  }
});

app.listen(3000, () => {
  console.log("Frontend listening on port 3000");
});
```

- **Purpose**:
  This is the **Node.js + Express frontend server**.
- **What it does**:
  - Listens on port 3000.
  - / → Returns "Frontend is running!"
  - /fetch-backend → GET → Makes a request to backend:5000/api/data (container networking).
  - /send-data → POST → Sends JSON to backend /submit.

- **Why important**:
  Acts as the middle layer between the user and backend APIs.

5. /frontend/package.json

```
{
  "name": "frontend",
  "version": "1.0.0",
  "main": "server.js",
  "dependencies": {
    "express": "^4.18.2",
    "axios": "^1.6.0",
    "cors": "^2.8.5"
  }
}
```

- **Purpose**:
  Node.js project configuration file.
- **What it contains**:
  - App metadata (name, version).
  - dependencies like axios, cors, express.
  - Scripts (npm start runs server.js).
- **Why important**:
  Required by Node.js to install dependencies with npm install.

6. /frontend/Dockerfile

```
FROM node:18

WORKDIR /app
```

```
COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

- **Purpose**:
  Tells Docker **how to build** the frontend image.
- **Steps inside**:
  - Use node:18 base image.
  - Set working directory to /app.
  - Copy package*.json and run npm install.
  - Copy all frontend files.
  - Run npm start when the container starts.
- **Why important**:
  Defines the container environment for Node.js frontend.

7. /docker-compose.yml

```
version: "3.9"

services:

  backend:

    build: ./backend

    container_name: backend_container

    ports:

      - "5000:5000"

    env_file:

      - ./backend/.env

    volumes:

      - ./backend:/app

    depends_on:

      db:
```

```
      condition: service_healthy
  frontend:
    build: ./frontend
    container_name: frontend_container
    ports:
      - "3000:3000"
    volumes:
      - ./frontend:/app
      - /app/node_modules
    stdin_open: true
    tty: true
    depends_on:
      backend:
        condition: service_started
  db:
    image: postgres:15
    container_name: postgres_container
    restart: always
    environment:
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
      POSTGRES_DB: mydb
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U myuser -d mydb"]
```

```
    interval: 5s
    timeout: 5s
    retries: 5

volumes:
  postgres_data:
```

- **Purpose**:
  Orchestrates **multiple containers**: frontend, backend, PostgreSQL DB.
- **What it defines**:
  - **backend** → Builds from backend/Dockerfile, connects to DB.
  - **frontend** → Builds from frontend/Dockerfile, connects to backend.
  - **db** → Uses postgres:15 official image.
  - Networking between containers is automatic (Docker Compose default network).
- **Why important**:
  One command (docker-compose up) starts all services together.

8. /.gitignore

```
node_modules/
.vscode/
__pycache__/
*.pyc
.env
```

- **Purpose**:
  Lists files/folders Git should **not track**.
- **Example entries**:

    - .venv/ → Python virtual environment.
    - node_modules/ → Node.js dependencies.
    - .vscode/ → Editor configs.
    - .env → Environment secrets.
- **Why important**:
  Prevents uploading large/unnecessary files to GitHub.

9. /README.md

Flask + Node.js + Docker Project

Overview

This project is a **Dockerized Full Stack App** with:

- Backend: Flask + PostgreSQL
- Frontend: Node.js + Express
- Database: PostgreSQL

How to Run

docker-compose up --build

- **Purpose**:
  Documentation for the project.
- **Contents**:
    - Overview of the stack.
    - Setup instructions.
    - API endpoint descriptions.
    - Docker commands.
- **Why important**:
  Makes the project easy for others (and yourself later) to understand.