Testing

Tugas 1:

```
Docker
Docker merupakan sebuah platform yang berfungsi dalam pembangunan sebuah
software yang mampu mengemas dan menjalankan beberapa instance sekaligus.

Container
Container memungkinkan developer dalam mengemas komponen komponen software
seperti kode dependansi dan pustaka agar dapat berjalan pada mesin.
```

- Melakukan Instalasi Python
- Melakukan Instalasi Node JS
- Melakukan Instalasi Docker
- Melakukan Koneksi ke Database
-

Tugas 2

- Menjalankan perintah *vnev\Script\active* dan melakukan instalasi Flask *pip install flask*
- melakukan run *python app.py*
- Menampilkan tampilan JSON pada browser dengan menggunakan kode

```
from flask import Flask, jsonify

app = Flask(__name__)  # Baris ini membuat instance baru dari aplikasi
Flask.

@app.route('/')
def home():
    return jsonify({"message": "Hello from Flask!"})  # Fungsi ini
mendefinisikan rute untuk URL root ('/'). Ketika URL ini diakses, fungsi
ini
mengembalikan respon JSON dengan pesan.

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)  # Baris ini
menjalankan
server pengembangan Flask dengan debugging diaktifkan, mendengarkan pada
semua
alamat IP yang tersedia (host='0.0.0.0') di port 5000.
```

- Melakukan instalasi dengan masuk kedalam virtual enviroment *pip freeze > requirements.txt* dan *pip install -r* untuk menginstall dependesi yang berisi library

```
Flask==3.1.0
Jinja2==3.1.5
Werkzeug==3.1.3
click==8.1.8
itsdangerous==2.2.0
```

Tugas 3

Membuat menjalankan proyek react sederhana pada browser

- melakukan instalasi react dengan kode berikut:

```
cd ../frontend
npm create vite@latest my-react-app -- --template react
cd my-react-app
npm install
npm run dev
```

- Melakukan run *npm run dev* untuk melihat react berjalan dengan benar
- Melakukan perubahan isi konten pada *src/App.jsx/* dengan kode berikut:

```
src/App/jsx dengan menggunakan kode
import React from 'react';
function App() {
return (
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
    <h1>Hello from React + Vite!</h1>
    <p>This is a simple React app built with Vite.</p>
    </div>
);
}
export default App;
```

Tugas 4

Menghubungkan React ke Flask

- Melakukan penambahan baris kode pada *App.py* dengan kode berikut:

```
@app.route('/api/data')
```

def get_data():
return jsonify({"data": "Hello from Flask API"})
```

- Masuk kedalam virtual enviroment dengan .\*venv*\*Scripts*\*activate* dan Melakukan run *python app.py*
- Merubah isi konten pada file *App.jsx* pada folder *src* dengan kode berikut:

```
    import React, { useState, useEffect } from 'react';

function App() {
  const [apiData, setApiData] = useState(null);

  useEffect(() => {
    fetch('http://localhost:5000/api/data')
      .then(response => response.json())
      .then(data => {
        setApiData(data.data);
      })
      .catch(error => console.error(error));
  }, []);

  return (
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
      <h1>React & Flask Integration</h1>
      {apiData ? apiData : "Loading data..."}

  );
}
```

- Melakukan run *npm run dev* untuk menjalankan React pada web.

Tugas 5

Integrasi Flask dengan PostgreSQL

- Menginstall PostgreSQL pada program dengan *pip install psycopg2-binary*
- Membuat koneksi dengan Database pada *app.py* dengan kode berikut:

```
    import psycopg2

    # Tambahkan di bagian atas file sebelum deklarasi route
    def get_db_connection():
        conn = psycopg2.connect(
            host="localhost",
```
```

```
        database="test_db",
        user="student",
        password="password"
    )
    return conn

from flask import Flask, jsonify
app = Flask(__name__)

@app.route('/')
def home():
    return jsonify({"message": "Hello from Flask!"})

@app.route('/api/data')
def get_data():
    return jsonify({"data": "Hello from Flask API"})

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

- Menyiapkan Database dengan software pgAdmin versi Desktop
- Gunakan kode berikut untuk membuat tabel pada database

```
CREATE TABLE IF NOT EXISTS items (
id SERIAL PRIMARY KEY,
name VARCHAR(100),
description TEXT
);
```

- Menambahkan fungsi CRUD pada *app.py*
  POST

```
@app.route('/api/items', methods=['POST'])
def create_item():
data = request.json
name = data['name']
description = data['description']
conn = get_db_connection()
cur = conn.cursor()
cur.execute("INSERT INTO items (name, description) VALUES (%s, %s)
RETURNING id;",
            (name, description))
new_id = cur.fetchone()[0]
conn.commit()
cur.close()
conn.close()
return jsonify({"id": new_id, "name": name, "description":
description}), 201
```

GET

```
@app.route('/api/items', methods=['GET'])
def get_items():
conn = get_db_connection()
cur = conn.cursor()
cur.execute("SELECT id, name, description FROM items;")
rows = cur.fetchall()
cur.close()
conn.close()

items = []
for row in rows:
items.append({"id": row[0], "name": row[1], "description": row[2]})
return jsonify(items)
```

PUT

```
@app.route('/api/items/<int:id>', methods=['PUT']) def update_item(id):
data = request.json
name = data['name']
description = data['description']

conn = get_db_connection()
cur = conn.cursor()
cur.execute("UPDATE items SET name = %s, description = %s WHERE id = %s;",
(name, description, id))
conn.commit()
cur.close()
conn.close()

return jsonify({"id": id, "name": name, "description": description}), 200
```

DELETE

```
@app.route('/api/items/<int:id>', methods=['DELETE']) def delete_item(id):
conn = get_db_connection()
cur = conn.cursor()
cur.execute("DELETE FROM items WHERE id = %s;", (id,))
conn.commit()
cur.close()
conn.close()

return jsonify({"message": f"Item with id {id} has been deleted."}), 200
```

- Lakukan Testing pada program dengan curl atau postman menggunakan kode berikut:

```
    curl http://localhost:5000/api/items ### Berfungsi Menampilkan isi Tabel

    curl -X POST -H "Content-Type: application/json" \
        -d '{"name": "Test Item", "description": "Test Description"}' \
        http://localhost:5000/api/items ### Berfungsi Menambahkan isi Tabel

    curl -X DELETE http://localhost:5000/api/items/4 ### Berfungsi Menghapus
isi Tabel sesuai nomor id

    curl -X PUT -H "Content-Type: application/json" \
        -d '{"name": "Updated Item","description":"new description"}' \
        http://localhost:5000/api/items/6 ### Berfungsi Mengedit isi Tabel
sesuai nomor id
```

Tugas 6

Membuat Dockerfile untuk Flask

- Membuat *Dockerfile* dan menambahkan kode pada file

```
    # backend/Dockerfile
    FROM python:3.9-slim

    WORKDIR /app

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

    COPY . .

    EXPOSE 5000
    CMD ["python", "app.py"]
```

- Menambahkan kode library pada *requirements.txt*

```
    flask
    flask-cors
    psycopg2-binary
```

- Menjalankan perintah *docker build -t flask-backend:1.0 .* untuk membuat image
- Menjalankan container dengan perintah *docker run -d -p 5000:5000 --name flask-container flask-backend:1.0*

Tugas 7

Membuat Dockerfile untuk React dengan Vite

- Buat file *Dockerfile* baru pada direktori *frontend\my-react-app* dan tambahkan kode pada file

```
# frontend/my-react-app/Dockerfile
FROM node:14-alpine

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

# Build untuk production menggunakan Vite
RUN npm run build

# Gunakan Nginx untuk serve static file
FROM nginx:stable-alpine
COPY --from=0 /app/dist /usr/share/nginx/html

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

- Menjalankan perintah *npm run build* pada terminal
- jalankan perintah *docker build -t react-frontend-vite:1.0 .* untuk membuat image
- jalankan perintah *docker run -d -p 3000:80 --name react-container-vite react-frontend-vite:1.0* untuk mengaktifkan container
- Gunakan ink *http://localhost:3000* untuk melihat program berjalan

Tugas 8 UTS

Integrasi Full Stack dengan Docker Compose

- Membuat file docker-compose.yml

```
version: '3.7'
services:
  backend:
    build: ./backend
    container_name: flask_container
    ports: ["5000:5000"]
    environment:
      - DB_HOST=db
      - DB_NAME=test_db
      - DB_USER=student
      - DB_PASSWORD=password
    depends_on: [db]
```

```
  frontend:
    build: ./frontend/my-react-app
    container_name: react_container
    ports: ["3000:80"]
    depends_on: [backend]

  db:
    image: postgres:12-alpine
    container_name: postgres_container
    environment:
      - POSTGRES_DB=test_db
      - POSTGRES_USER=student
      - POSTGRES_PASSWORD=password
    ports: ["5432:5432"]
    volumes:
      - db_data:/var/lib/postgresql/data
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
volumes:
  db_data:
```

- Inisialisasi database PostgreSQL

```
CREATE TABLE IF NOT EXISTS items (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  description TEXT
);
INSERT INTO items (name, description)
VALUES ('Test Item', 'This is a test description');
```

- Konfigurasi Flask untuk koneksi database

```
# ... (koneksi DB menggunakan environment variables)
@app.route('/api/items', methods=['GET'])
def get_items():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT * FROM items;")
    # ... (return data sebagai JSON)
```

- Konfigurasi React untuk akses API Flask

```
useEffect(() => {
  fetch("http://localhost:5000/api/items")
    .then(response => response.json())
```

```
    .then(data => setItems(data));
}, []);
```

- Build dan jalankan seluruh stack *docker compose up -d --build*
- Cek program berjalan apa pada browser *http://localhost:3000