MAKALAH PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2B
LABORATORIUM INFORMATIKA

Nama Praktikan: Nazril Bintang Pratama
NPM: 51423096
Kelas: 1IA21
PJ: Julius Wahyu W
Pertemuan: 7

UNIVERSITAS GUNADARMA
BEKASI
2024

Daftar Isi .....

Bab 1
Pendahuluan

1.1 Definisi API

API (Application Programming Interface) adalah antarmuka atau jembatan yang memungkinkan komunikasi antara dua aplikasi yang berbeda. API memungkinkan pengguna untuk meminta dan menerima sumber daya seperti data dari server. Contohnya dalam transaksi online, pengguna mengirim permintaan ke server, yang kemudian memproses respons dan mengirimkannya kembali ke klien.

1.2 Jenis-jenis API

- Public API: Dapat diakses oleh siapa saja, biasanya memiliki tingkat keamanan yang rendah.
- Private API: Hanya dapat diakses oleh developer atau organisasi tertentu untuk keperluan internal.
- Partner API: Diberikan kepada mitra oleh perusahaan untuk memungkinkan integrasi produk atau layanan mereka ke platform.
- Composite API: Gabungan dari beberapa API yang bekerja bersama untuk memberikan fitur yang lebih lengkap/

1.3 Arsitektur API

1. REST (Representational State Transfer): Arsitektur yang berbasis pada sumber daya, representasi, komponen, dan uniform interface untuk membangun API yang skalabel dan mudah diakses melalui internet.
2. SOAP (Simple Object Access Protocol): Arsitektur berbasis XML yang menggunakan pesan SOAP, XML, WSDL, dan UDDI.
3. RPC (Remote Procedure Call): Protokol yang digunakan untuk memanggil prosedur atau fungsi pada mesin di jaringan.
4. Falcor: Arsitektur yang mirip dengan GraphQL, menggunakan lapisan virtual untuk memetakan permintaan frontend ke layanan backend.
5. Websocket: API web modern yang menggunakan objek JSON untuk komunikasi dua arah antara aplikasi klien dan server.

1.4 API Endpoint

- API Endpoint adalah URL khusus yang digunakan oleh klien untuk mengakses API. Endpoint ini merupakan titik masuk untuk fungsi atau sumber daya tertentu yang disediakan oleh API.

1.5 Metode API

- GET: Digunakan untuk meminta data dari server tanpa mengubah data yang ada.
- POST: Digunakan untuk mengirim data ke server dan membuat sumber daya baru.
- PUT: Digunakan untuk memperbarui sumber data yang ada di server.
- DELETE: Digunakan untuk menghapus sumber daya dari server..

1.6 Metode Pengamanan API

- Otentikasi Dasar: Klien mengirimkan permintaan HTTP dengan header yang dibuat sebelumnya untuk memvalidasi kredensial.
- Otentikasi Berbasis Host: Hanya pengguna yang diautentikasi yang dapat mengakses sumber daya di server.
- OAuth: Protokol yang memverifikasi identitas pengguna dan menetapkan standar otorisasi.

- SAML: Proses otentikasi standar untuk menggunakan teknologi single sign-on.

1.7 Listing Program

Pertemuan 2.json > {} 1

```json
[
    {
    "nama": "Nazril Bintang Pratama",
    "kelas": "1IA21",
    "NPM": "51423096",
    "fakultas": "Teknologi Industri",
    "jurusan": "Informatika"
    },
    {
    "nama": "TIO",
    "kelas": "1IA21",
    "NPM": "51423096",
    "fakultas": "Teknologi Industri",
    "jurusan": "Informatika"
    },
    {
    "nama": "FAREL",
    "kelas": "1IA21",
    "NPM": "51423096",
    "fakultas": "Teknologi Industri",
    "jurusan": "Informatika"
    }
]
```

```python
import json

file_json = open("Pertemuan 2.json")

data = json.load(file_json)

print("--------------------\n")
for i in range(len(data)):
    print(f"Nama        : {data[i]['nama']}")
    print(f"Kelas       : {data[i]['kelas']}")
    print(f"NPM         : {data[i]['NPM']}")
    print(f"Fakultas    : {data[i]['fakultas']}")
    print(f"Jurusan    : {data[i]['jurusan']}")
    print("--------------------\n")
```

1.8 Logika Program

```
import json
```

Pertama" saya membuat modul json yang digunakan untuk parsing data JSON. Modul ini menyediakan metode untuk membaca dan menulis data JSON.

```
file_json = open("Pertemuan 2.json")

data = json.load(file_json)
```

Lalu saya membuat code file_json = open("Pertemuan 2.json") yang bertujuan untuk Membuka file Pertemuan 2.json dalam mode baca .Variabel file_json menjadi file object yang merujuk ke file tersebut.,Setelahnya saya membuat blok code data = json.load(file_json) yang bertujuan untuk membaca dan menguraikan (parse) konten dari file file_json menjadi struktur data Python, dalam hal ini, sebuah list dari dictionaries. Setiap dictionary merepresentasikan satu entri data mahasiswa.

```
print("--------------------\n")
for i in range(len(data)):
    print(f"Nama        : {data[i]['nama']}")
    print(f"Kelas       : {data[i]['kelas']}")
    print(f"NPM         : {data[i]['NPM']}")
    print(f"Fakultas    : {data[i]['fakultas']}")
    print(f"Jurusan    : {data[i]['jurusan']}")
    print("--------------------\n")
```

Lalu saya membuat print("--------------------\n") yang bertjuan untuk membuat garis pada output nanti,Lalu saya membuat for i in range(len(data)): yang digunakan untuk Melakukan iterasi (pengulangan) sebanyak jumlah elemen dalam list data,Selanjutnya kode print(f"Nama : {data[i]['nama']}") Sampe dengan Jurusan sesuai data yang ada di Json,Karna pada ouput nanti python akan memanggil data yg sesuai di file Json nya

1.9 Output Program

```
------------------

Nama        : Nazril Bintang Pratama
Kelas       : 1IA21
NPM         : 51423096
Fakultas    : Teknologi Industri
Jurusan    : Informatika
------------------

Nama        : TIO
Kelas       : 1IA21
NPM         : 51423096
Fakultas    : Teknologi Industri
Jurusan    : Informatika
------------------

Nama        : FAREL
Kelas       : 1IA21
NPM         : 51423096
Fakultas    : Teknologi Industri
Jurusan    : Informatika
------------------
```

Bab 2 Pembuatan API Endpoint dengan Flask

2.1 Definisi API Endpoint

API Endpoint adalah titik akhir atau URL tempat permintaan (request) dikirimkan untuk berinteraksi dengan aplikasi web. Endpoint ini memungkinkan pengguna untuk mengakses dan memanipulasi data atau sumber daya di aplikasi web melalui metode HTTP yang telah didefinisikan seperti GET, POST, PUT, dan DELETE!

2.2 Metode HTTP dalam Flask

- GET: Digunakan untuk mengambil data atau sumber daya dari API Endpoint tanpa mengubah data yang ada.
- POST: Digunakan untuk membuat data baru di dalam API Endpoint.
- PUT: Digunakan untuk memperbarui atau mengubah data yang sudah ada di dalam API Endpoint.
- DELETE: Digunakan untuk menghapus data yang sudah ada di dalam API Endpoint.

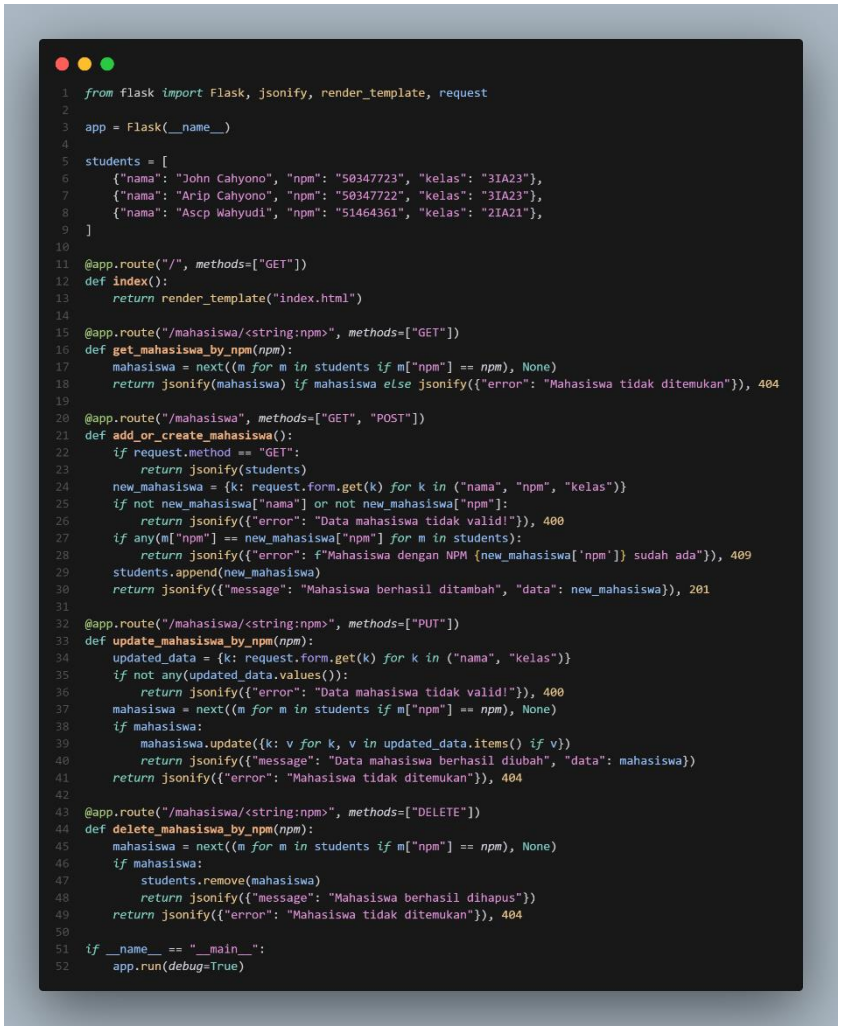
2.3 REST API Endpoint dengan Flask

RESTful API adalah sebuah arsitektur web yang digunakan untuk mempermudah interaksi antara klien dan server dalam memanipulasi data. Flask mendukung pembuatan RESTful API yang memungkinkan pengembang untuk menyediakan dan mengakses sumber daya atau data dalam aplikasi web menggunakan metode HTTP yang sesuai.

2.4 Flask-RESTful

Flask-RESTful adalah sebuah ekstensi Flask yang menyediakan framework untuk membangun RESTful API dengan cepat dan mudah menggunakan pendekatan berbasis kelas. Fitur utama dari Flask-RESTful termasuk menyediakan konsep resource-based API untuk mengelola sumber daya dan operasi CRUD (Create, Read, Update, Delete) secara terpisah serta memungkinkan penggunaan format data seperti JSON dan XML.

2.5 Listing Program



```python
from flask import Flask, jsonify, render_template, request

app = Flask(__name__)

students = [
    {"nama": "John Cahyono", "npm": "50347723", "kelas": "3IA23"},
    {"nama": "Arip Cahyono", "npm": "50347722", "kelas": "3IA23"},
    {"nama": "Ascp Wahyudi", "npm": "51464361", "kelas": "2IA21"},
]

@app.route("/", methods=["GET"])
def index():
    return render_template("index.html")

@app.route("/mahasiswa/<string:npm>", methods=["GET"])
def get_mahasiswa_by_npm(npm):
    mahasiswa = next((m for m in students if m["npm"] == npm), None)
    return jsonify(mahasiswa) if mahasiswa else jsonify({"error": "Mahasiswa tidak ditemukan"}), 404

@app.route("/mahasiswa", methods=["GET", "POST"])
def add_or_create_mahasiswa():
    if request.method == "GET":
        return jsonify(students)
    new_mahasiswa = {k: request.form.get(k) for k in ("nama", "npm", "kelas")}
    if not new_mahasiswa["nama"] or not new_mahasiswa["npm"]:
        return jsonify({"error": "Data mahasiswa tidak valid!"}), 400
    if any(m["npm"] == new_mahasiswa["npm"] for m in students):
        return jsonify({"error": f"Mahasiswa dengan NPM {new_mahasiswa['npm']} sudah ada"}), 409
    students.append(new_mahasiswa)
    return jsonify({"message": "Mahasiswa berhasil ditambah", "data": new_mahasiswa}), 201

@app.route("/mahasiswa/<string:npm>", methods=["PUT"])
def update_mahasiswa_by_npm(npm):
    updated_data = {k: request.form.get(k) for k in ("nama", "kelas")}
    if not any(updated_data.values()):
        return jsonify({"error": "Data mahasiswa tidak valid!"}), 400
    mahasiswa = next((m for m in students if m["npm"] == npm), None)
    if mahasiswa:
        mahasiswa.update({k: v for k, v in updated_data.items() if v})
        return jsonify({"message": "Data mahasiswa berhasil diubah", "data": mahasiswa})
    return jsonify({"error": "Mahasiswa tidak ditemukan"}), 404

@app.route("/mahasiswa/<string:npm>", methods=["DELETE"])
def delete_mahasiswa_by_npm(npm):
    mahasiswa = next((m for m in students if m["npm"] == npm), None)
    if mahasiswa:
        students.remove(mahasiswa)
        return jsonify({"message": "Mahasiswa berhasil dihapus"})
    return jsonify({"error": "Mahasiswa tidak ditemukan"}), 404

if __name__ == "__main__":
    app.run(debug=True)
```

2.6 Logika Pemrograman

Pertama, saya mengimpor modul Flask, jsonify, render_template, dan request dari paket flask. Modul ini digunakan untuk membuat aplikasi web, mengembalikan data dalam format JSON, merender template HTML, dan menangani permintaan HTTP.

```python
from flask import Flask, jsonify, render_template, request
```

Kedua, saya membuat instance dari aplikasi Flask dan mendefinisikan data mahasiswa dalam bentuk list of dictionaries.

```python
students = [
    {"nama": "John Cahyono", "npm": "50347723", "kelas": "3IA23"},
    {"nama": "Arip Cahyono", "npm": "50347722", "kelas": "3IA23"},
    {"nama": "Ascp Wahyudi", "npm": "51464361", "kelas": "2IA21"},
]
```

Ketiga, saya mendefinisikan route untuk halaman utama yang akan merender template index.html.

```python
@app.route("/", methods=["GET"])
def index():
    return render_template("index.html")
```

Keempat, saya mendefinisikan route untuk mendapatkan data mahasiswa berdasarkan NPM. Menggunakan next dengan default None untuk mencari mahasiswa.

```
@app.route("/mahasiswa/<string:npm>", methods=["GET"])
def get_mahasiswa_by_npm(npm):
    mahasiswa = next((m for m in students if m["npm"] == npm), None)
    return jsonify(mahasiswa) if mahasiswa else jsonify({"error": "Mahasiswa tidak ditemukan"}), 404
```

Kelima, saya mendefinisikan route untuk menambahkan atau mendapatkan daftar mahasiswa. Menggunakan dictionary comprehension untuk new_mahasiswa dan any untuk memeriksa validitas data dan keberadaan NPM.

```
@app.route("/mahasiswa", methods=["GET", "POST"])
def add_or_create_mahasiswa():
    if request.method == "GET":
        return jsonify(students)
    new_mahasiswa = {k: request.form.get(k) for k in ("nama", "npm", "kelas")}
    if not new_mahasiswa["nama"] or not new_mahasiswa["npm"]:
        return jsonify({"error": "Data mahasiswa tidak valid!"}), 400
    if any(m["npm"] == new_mahasiswa["npm"] for m in students):
        return jsonify({"error": f"Mahasiswa dengan NPM {new_mahasiswa['npm']} sudah ada"}), 409
    students.append(new_mahasiswa)
    return jsonify({"message": "Mahasiswa berhasil ditambah", "data": new_mahasiswa}), 201
```

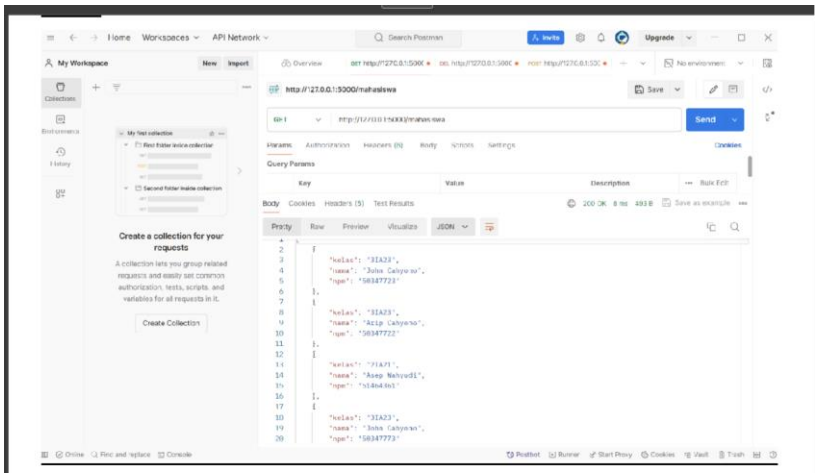
Keenam, saya mendefinisikan route untuk memperbarui data mahasiswa berdasarkan NPM. Menggunakan dictionary comprehension untuk updated_data dan next dengan default None untuk mencari mahasiswa.

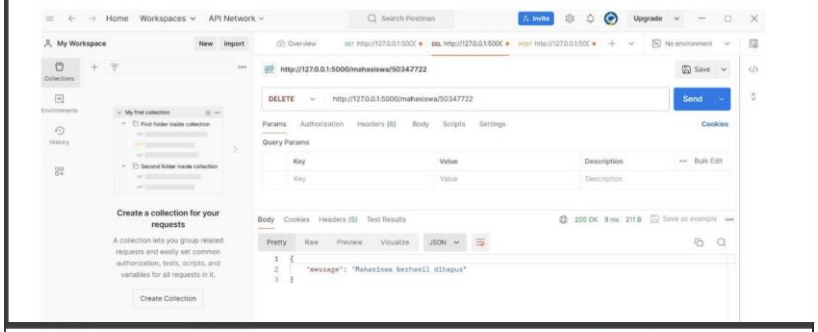
```
@app.route("/mahasiswa/<string:npm>", methods=["PUT"])
def update_mahasiswa_by_npm(npm):
    updated_data = {k: request.form.get(k) for k in ("nama", "kelas")}
    if not any(updated_data.values()):
        return jsonify({"error": "Data mahasiswa tidak valid!"}), 400
    mahasiswa = next((m for m in students if m["npm"] == npm), None)
    if mahasiswa:
        mahasiswa.update({k: v for k, v in updated_data.items() if v})
        return jsonify({"message": "Data mahasiswa berhasil diubah", "data": mahasiswa})
    return jsonify({"error": "Mahasiswa tidak ditemukan"}), 404
```

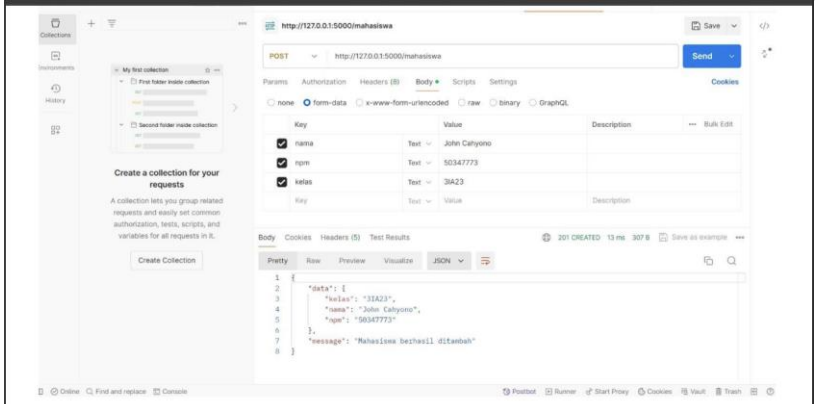
Ketujuh, saya mendefinisikan route untuk menghapus data mahasiswa berdasarkan NPM. Menggunakan next dengan default None untuk mencari mahasiswa.

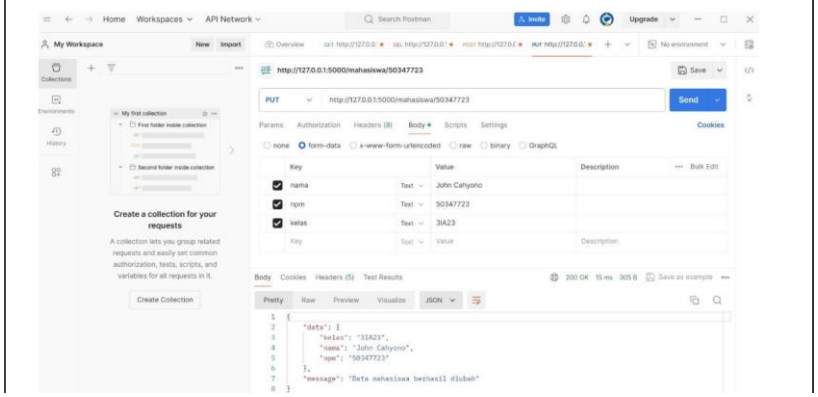
```
@app.route("/mahasiswa/<string:npm>", methods=["DELETE"])
def delete_mahasiswa_by_npm(npm):
    mahasiswa = next((m for m in students if m["npm"] == npm), None)
    if mahasiswa:
        students.remove(mahasiswa)
        return jsonify({"message": "Mahasiswa berhasil dihapus"})
    return jsonify({"error": "Mahasiswa tidak ditemukan"}), 404
```

2.7 Ouput Program









Bab 3 Pembuatan API Endpoint dengan Flask

3.1 Definisi PyQt

adalah metode untuk membuat aplikasi desktop dengan antarmuka pengguna grafis menggunakan bahasa pemrograman Python dan framework PyQt. PyQt adalah penghubung antara Python dan toolkit GUI Qt, yang dikembangkan oleh The Qt Company

3.2 Fitur Utama PyQt5

- Widget GUI: Komponen antarmuka yang beragam.
- Akses Database: Mendukung berbagai database seperti MySQL, PostgreSQL, dan SQLite.
- QScintilla: Widget editor teks berbasis Scintilla.
- SVG dan XML: Dukungan untuk SVG dan parsing XML.
- ActiveX: Integrasi kontrol ActiveX di Windows (komersial).

3.3 Komponen PyQt

- QtCore: Kelas inti non-GUI, termasuk pengulangan, sinyal, dan slot.
- QtGui: Kelas GUI.
- QtNetwork: Kelas untuk menulis klien dan server UDP dan TCP.
- QtOpenGL: Kelas untuk render grafis 3D.
- QtSql: Kelas untuk integrasi database SQL.
- QtSvg: Kelas untuk menampilkan file SVG.
- QtMultimedia: Fungsi multimedia tingkat rendah.
- QtDesigner: Memungkinkan ekstensi Qt Designer dengan PyQt.
- QtXml: SAX dan DOM untuk parsing XML.

3.4 Listing Program

```python
import sys
from PyQt5.QtWidgets import (
    QApplication,
    QWidget,
    QVBoxLayout,
    QHBoxLayout,
    QLabel,
    QLineEdit,
    QTextEdit,
    QPushButton,
    QListWidget,
    QMessageBox,
)

class AplikasiAktivitasPembelajaran(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle("Pengelola Aktivitas Pembelajaran")
        self.setGeometry(100, 100, 400, 300)

        self.layout = QVBoxLayout()

        self.label_nama_aktivitas = QLabel("Nama Aktivitas:")
        self.input_nama_aktivitas = QLineEdit()
        self.label_deskripsi_aktivitas = QLabel("Deskripsi:")
        self.input_deskripsi_aktivitas = QTextEdit()

        self.layout.addWidget(self.label_nama_aktivitas)
        self.layout.addWidget(self.input_nama_aktivitas)
        self.layout.addWidget(self.label_deskripsi_aktivitas)
        self.layout.addWidget(self.input_deskripsi_aktivitas)

        self.tombol_tambah = QPushButton("Tambah Aktivitas")
        self.tombol_edit = QPushButton("Edit Aktivitas")
        self.tombol_hapus = QPushButton("Hapus Aktivitas")

        self.layout_tombol = QHBoxLayout()
        self.layout_tombol.addWidget(self.tombol_tambah)
        self.layout_tombol.addWidget(self.tombol_edit)
        self.layout_tombol.addWidget(self.tombol_hapus)

        self.layout.addLayout(self.layout_tombol)

        self.daftar_aktivitas = QListWidget()
        self.layout.addWidget(self.daftar_aktivitas)

        self.setLayout(self.layout)

        self.tombol_tambah.clicked.connect(self.tambah_aktivitas)
        self.tombol_edit.clicked.connect(self.edit_aktivitas)
        self.tombol_hapus.clicked.connect(self.hapus_aktivitas)

    def tambah_aktivitas(self):
        nama = self.input_nama_aktivitas.text()
        deskripsi = self.input_deskripsi_aktivitas.toPlainText()
        if nama and deskripsi:
            self.daftar_aktivitas.addItem(f"{nama}: {deskripsi}")
            self.bersihkan_input()
        else:
            self.tampilkan_pesan_error("Harap isi kedua bidang")

    def edit_aktivitas(self):
        item_terpilih = self.daftar_aktivitas.currentItem()
        if item_terpilih:
            nama = self.input_nama_aktivitas.text()
            deskripsi = self.input_deskripsi_aktivitas.toPlainText()
            if nama and deskripsi:
                item_terpilih.setText(f"{nama}: {deskripsi}")
                self.bersihkan_input()
            else:
                self.tampilkan_pesan_error("Harap isi kedua bidang")
        else:
            self.tampilkan_pesan_error("Harap pilih aktivitas yang akan diedit")

    def hapus_aktivitas(self):
        item_terpilih = self.daftar_aktivitas.currentItem()
        if item_terpilih:
            self.daftar_aktivitas.takeItem(self.daftar_aktivitas.row(item_terpilih))
        else:
            self.tampilkan_pesan_error("Harap pilih aktivitas yang akan dihapus")

    def bersihkan_input(self):
        self.input_nama_aktivitas.clear()
        self.input_deskripsi_aktivitas.clear()

    def tampilkan_pesan_error(self, pesan):
        msg = QMessageBox()
        msg.setIcon(QMessageBox.Warning)
        msg.setText(pesan)
        msg.setWindowTitle("Error")
        msg.exec_()



if __name__ == "__main__":
    app = QApplication(sys.argv)
    ex = AplikasiAktivitasPembelajaran()
    ex.show()
    sys.exit(app.exec_())
```

3.5 Logika Program

```
import sys
from PyQt5.QtWidgets import (
    QApplication,
    QWidget,
    QVBoxLayout,
    QHBoxLayout,
    QLabel,
    QLineEdit,
    QTextEdit,
    QPushButton,
    QListWidget,
    QMessageBox,
)
```

Pertama, saya mengimpor modul sys dan beberapa komponen dari PyQt5.QtWidgets yang akan digunakan untuk membuat antarmuka pengguna grafis (GUI).

```
class AplikasiAktivitasPembelajaran(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
```

Saya mendefinisikan kelas AplikasiAktivitasPembelajaran yang merupakan subclass dari QWidget. Pada metode __init__, saya memanggil metode super().__init__() untuk menginisialisasi kelas induk dan kemudian memanggil metode initUI untuk mengatur antarmuka pengguna.

```
def initUI(self):
    self.setWindowTitle("Pengelola Aktivitas Pembelajaran")
    self.setGeometry(100, 100, 400, 300)
```

Pada metode initUI, saya mengatur judul jendela dan ukuran serta posisi jendela aplikasi.

```
self.layout = QVBoxLayout()

self.label_nama_aktivitas = QLabel("Nama Aktivitas:")
self.input_nama_aktivitas = QLineEdit()
self.label_deskripsi_aktivitas = QLabel("Deskripsi:")
self.input_deskripsi_aktivitas = QTextEdit()

self.layout.addWidget(self.label_nama_aktivitas)
self.layout.addWidget(self.input_nama_aktivitas)
self.layout.addWidget(self.label_deskripsi_aktivitas)
self.layout.addWidget(self.input_deskripsi_aktivitas)
```

Saya membuat layout vertikal (QVBoxLayout) dan menambahkan beberapa widget ke dalamnya, seperti QLabel, QLineEdit, dan QTextEdit untuk menginput nama dan deskripsi aktivitas.

```
self.tombol_tambah = QPushButton("Tambah Aktivitas")
self.tombol_edit = QPushButton("Edit Aktivitas")
self.tombol_hapus = QPushButton("Hapus Aktivitas")

self.layout_tombol = QHBoxLayout()
self.layout_tombol.addWidget(self.tombol_tambah)
self.layout_tombol.addWidget(self.tombol_edit)
self.layout_tombol.addWidget(self.tombol_hapus)

self.layout.addLayout(self.layout_tombol)
```

Saya membuat tiga tombol (QPushButton) untuk menambah, mengedit, dan menghapus aktivitas. Tombol-tombol ini ditambahkan ke dalam layout horizontal (QHBoxLayout), yang kemudian ditambahkan ke layout utama.

```
self.daftar_aktivitas = QListWidget()
self.layout.addWidget(self.daftar_aktivitas)

self.setLayout(self.layout)
```

Saya membuat widget QListWidget untuk menampilkan daftar aktivitas dan menambahkannya ke layout utama. Akhirnya, saya mengatur layout utama sebagai layout jendela.

```
self.tombol_tambah.clicked.connect(self.tambah_aktivitas)
self.tombol_edit.clicked.connect(self.edit_aktivitas)
self.tombol_hapus.clicked.connect(self.hapus_aktivitas)
```

Saya menghubungkan sinyal klik dari tombol-tombol ke metode yang sesuai (tambah_aktivitas, edit_aktivitas, hapus_aktivitas).

```
def tambah_aktivitas(self):
    nama = self.input_nama_aktivitas.text()
    deskripsi = self.input_deskripsi_aktivitas.toPlainText()
    if nama and deskripsi:
        self.daftar_aktivitas.addItem(f"{nama}: {deskripsi}")
        self.bersihkan_input()
    else:
        self.tampilkan_pesan_error("Harap isi kedua bidang")
```

Metode tambah_aktivitas digunakan untuk menambah aktivitas baru ke dalam daftar. Jika nama dan deskripsi diisi, aktivitas akan ditambahkan ke QListWidget dan input akan dibersihkan. Jika tidak, pesan error akan ditampilkan.

```
def edit_aktivitas(self):
    item_terpilih = self.daftar_aktivitas.currentItem()
    if item_terpilih:
        nama = self.input_nama_aktivitas.text()
        deskripsi = self.input_deskripsi_aktivitas.toPlainText()
        if nama and deskripsi:
            item_terpilih.setText(f"{nama}: {deskripsi}")
            self.bersihkan_input()
        else:
            self.tampilkan_pesan_error("Harap isi kedua bidang")
    else:
        self.tampilkan_pesan_error("Harap pilih aktivitas yang akan diedit")
```

Metode edit_aktivitas digunakan untuk mengedit aktivitas yang dipilih. Jika ada aktivitas yang dipilih dan input nama serta deskripsi diisi, aktivitas akan diperbarui. Jika tidak, pesan error akan ditampilkan.

```
def hapus_aktivitas(self):
    item_terpilih = self.daftar_aktivitas.currentItem()
    if item_terpilih:
        self.daftar_aktivitas.takeItem(self.daftar_aktivitas.row(item_terpilih))
    else:
        self.tampilkan_pesan_error("Harap pilih aktivitas yang akan dihapus")
```

Metode hapus_aktivitas digunakan untuk menghapus aktivitas yang dipilih dari daftar. Jika ada aktivitas yang dipilih, aktivitas tersebut akan dihapus. Jika tidak, pesan error akan ditampilkan.

```
def bersihkan_input(self):
    self.input_nama_aktivitas.clear()
    self.input_deskripsi_aktivitas.clear()
```

Metode bersihkan_input digunakan untuk membersihkan input nama dan deskripsi.

```
def tampilkan_pesan_error(self, pesan):
    msg = QMessageBox()
    msg.setIcon(QMessageBox.Warning)
    msg.setText(pesan)
    msg.setWindowTitle("Error")
    msg.exec_()
```

Metode tampilkan_pesan_error digunakan untuk menampilkan pesan error menggunakan QMessageBox.

```python
if __name__ == "__main__":
    app = QApplication(sys.argv)
    ex = AplikasiAktivitasPembelajaran()
    ex.show()
    sys.exit(app.exec_())
```

Bagian ini adalah titik masuk utama dari aplikasi. Saya membuat instance dari QApplication dan AplikasiAktivitasPembelajaran, menampilkan jendela aplikasi, dan menjalankan loop aplikasi.

Bab 4 Soal dan Jawaban

4.1 Soal

(BAB 1)

1. Jelaskan apa yang dimaksud dengan API dan bagaimana API digunakan dalam pengembangan perangkat lunak.
2. Apa itu JSON dan mengapa JSON sering digunakan dalam komunikasi data antara server dan klien?
3. Bagaimana cara mengirim permintaan (request) ke API menggunakan metode GET dan POST? Berikan contoh sederhana dalam bahasa pemrograman pilihan Anda.
4. Apa perbedaan antara API RESTful dan API SOAP? Sebutkan kelebihan dan kekurangan masing-masing.
5. Jelaskan bagaimana cara mengurai (parse) data JSON yang diterima dari API dan menggunakannya dalam aplikasi. Berikan contoh sederhana dalam bahasa pemrograman pilihan Anda.

Jawab

1. API (Application Programming Interface) adalah antarmuka yang memungkinkan aplikasi berkomunikasi satu sama lain. API digunakan untuk mengintegrasikan berbagai layanan atau aplikasi, seperti mengakses layanan web atau menghubungkan aplikasi mobile dengan server backend.
2. JSON (JavaScript Object Notation) adalah format data yang ringan dan mudah dibaca. JSON sering digunakan karena formatnya sederhana, mendukung struktur data kompleks, dan mudah diurai oleh banyak bahasa pemrograman.
3. Untuk mengirim permintaan GET dan POST, Anda dapat menggunakan pustaka HTTP di bahasa pemrograman pilihan Anda, seperti requests di Python.
4. RESTful API:

- Kelebihan: Sederhana, cepat, dan menggunakan HTTP standar.
- Kekurangan: Kurang cocok untuk operasi kompleks dan keamanan tinggi.

SOAP API:

- Kelebihan: Standar ketat, mendukung transaksi kompleks.
- Kekurangan: Kompleks, lebih lambat karena overhead tinggi.

5. Untuk mengurai data JSON, dapat menggunakan pustaka JSON di bahasa pemrograman pilihan Anda, seperti json di Python.

(BAB 2)
6. Jelaskan apa itu Flask dan mengapa Flask sering digunakan untuk membuat API endpoint.
7. Apa saja langkah-langkah dasar untuk membuat API endpoint menggunakan Flask?
8. Bagaimana cara menangani permintaan GET dan POST di Flask?
9. Apa itu JSONify dan mengapa penting dalam pembuatan API dengan Flask?
10. Bagaimana cara mengatur status kode HTTP dalam respons API Flask?

Jawab

1. Flask adalah framework web mikro untuk Python yang ringan dan mudah digunakan. Flask sering digunakan untuk membuat API endpoint karena kesederhanaannya, fleksibilitasnya, dan kemampuannya untuk menangani permintaan HTTP dengan mudah.
2. Langkah-langkah dasar meliputi:

- Menginstal Flask.
- Membuat aplikasi Flask.
- Mendefinisikan route untuk endpoint.
- Menjalankan aplikasi Flask.

3. Untuk menangani permintaan GET dan POST, Anda mendefinisikan route dengan metode yang sesuai dan menulis fungsi yang akan dijalankan saat permintaan diterima.
4. jsonify adalah fungsi Flask yang mengubah data Python menjadi format JSON. Ini penting karena API biasanya mengirim dan menerima data dalam format JSON, sehingga jsonify memudahkan proses ini.
5. Anda dapat mengatur status kode HTTP dengan mengembalikan tuple yang berisi data respons dan kode status dari fungsi route. Ini memungkinkan Anda untuk memberikan informasi status yang tepat kepada klien.

(BAB 3)
1. Jelaskan apa itu PyQt dan mengapa PyQt sering digunakan dalam pemrograman visual berbasis Python.
2. Apa saja langkah-langkah dasar untuk membuat aplikasi GUI sederhana menggunakan PyQt?
3. Bagaimana cara menambahkan widget seperti tombol dan label ke dalam aplikasi PyQt?
4. Apa itu layout dalam PyQt dan mengapa penting dalam pembuatan antarmuka pengguna?
5. Bagaimana cara menghubungkan sinyal dan slot dalam PyQt untuk menangani interaksi pengguna?

Jawab

1. PyQt adalah set binding Python untuk toolkit GUI Qt. PyQt sering digunakan karena menyediakan alat yang kuat dan fleksibel untuk membuat aplikasi desktop dengan antarmuka pengguna grafis yang kaya dan interaktif.
2. Langkah-langkah dasar meliputi:

- Menginstal PyQt.
- Membuat aplikasi PyQt.
- Membuat dan mengatur widget.
- Menyusun layout.

- Menjalankan aplikasi.

3. Anda menambahkan widget dengan membuat instance dari kelas widget (seperti QPushButton untuk tombol dan QLabel untuk label) dan menambahkannya ke layout atau parent widget.
4. Layout dalam PyQt adalah manajer tata letak yang mengatur posisi dan ukuran widget dalam aplikasi. Layout penting karena memastikan antarmuka pengguna terlihat rapi dan responsif terhadap perubahan ukuran jendela.
5. Anda menghubungkan sinyal dan slot dengan menggunakan metode connect. Sinyal adalah peristiwa yang dipicu oleh interaksi pengguna, dan slot adalah fungsi yang dijalankan sebagai respons terhadap sinyal tersebut.

Daftar Pustaka

Fahmi H., Jamhuri M., & Khudzaifah M. (2018). Pemrograman Komputer Dengan Python.

Summerfield M. (2007). Rapid GUI Programming With Python and Qt: The Definitive Guide to PyQt Programming.

Bodnar J. (2022). PyQt Tutorial: Learn PyQt Rapidly – From Beginner to Advanced.

Richardson, L., & Ruby, S. (2007). RESTful Web Services. O'Reilly Med