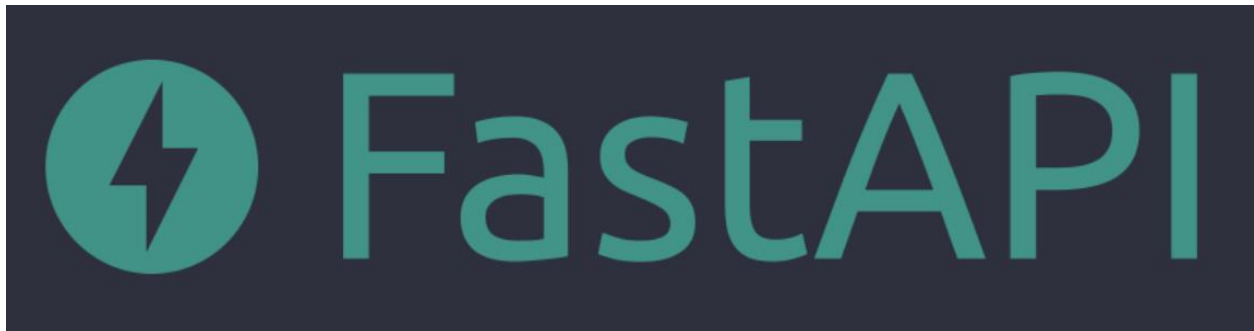


Webservice dengan FastAPI & SQLite



Mei 2023

Yudi Wibisono yudi@upi.edu

Ilmu Komputer UPI



<http://creativecommons.org/licenses/by-nc-sa/3.0/>

Modul ini bebas di-copy, didistribusikan, ditransmit dan diadaptasi/modifikasi/diremiks dengan syarat tidak untuk komersial, pembuat asal tetap dicantumkan dan hasil modifikasi dishare dengan lisensi yang sama.

Diasumsikan pembaca telah mengetahui tentang Python, jika belum berikut modul praktikum pengantar Python: https://docs.google.com/document/d/1hWVQ3ktx2k75zfhR_o-T0MkVFdiHxKQ0Vs2rH5YLSWQ/edit?usp=sharing

Rest API	3
Mengapa FAST API	3
Instalasi FastAPI dan Hello World	4
GET, POST, PUT, PATCH, DELETE	8
FastAPI GET	8
FastAPI POST.....	10
FastAPI PUT	15
FastAPI PATCH	17
FastAPI DELETE	20
POST dan GET file image	20
LAIN-LAIN	22

Rest API

API (Application Programming Interface) adalah spesifikasi komunikasi antara sistem software. Developer membuat API agar aplikasi lain dapat berkomunikasi dengan aplikasi mereka melalui software. Web service adalah API yang tersedia melalui internet. Tipe-tipe web service adalah SOAP, JSON-RPC dan REST.

REST (Representational State Transfer) adalah arsitektur software yang menggambarkan bagaimana API harus diimplementasikan. API yang mengikuti prinsip REST disebut RESTful APIs atau REST API.

Prinsip dari REST adalah:

1. Uniform interface

- a. Resource yang disediakan server memiliki identitas unik URI (Uniform Resource Identifier)
 - b. Resource dalam format yang mudah dipahami client seperti JSON, XML.
 - c. Setiap message mengandung semua informasi yang dibutuhkan (header/content type).
2. **Statelessness.** Server menyelesaikan request dari client independen dari request sebelumnya. Client dapat meminta request tanpa memerlukan keterurutan. Request terisolasi dengan request lainnya.
 3. **Layered system.** Client dapat meminta request melalui server mediator. Antar server dapat meneruskan request ke server lain. Dapat digunakan sistem berlapis. Misalnya lapisan keamanan, aplikasi dan business logic yang bekerjasama untuk melayani client. Layer-layer ini tidak terlihat client.
 4. **Cacheability** RESTful web services dapat melakukan caching, yaitu menyimpan response sehingga saat ada permintaan serupa, response tersebut yang diberikan sehingga mempersingkat waktu layanan. Client dapat meminta response yang non cacheable (selalu data yang fresh)
 5. **Code on demand** servers dapat menyediakan code yang akan dieksekusi oleh client. Misalnya javascript untuk proses bisnis yang kompleks. Tapi ini memiliki resiko dari sisi keamanan sehingga sudah jarang digunakan.

Mengapa FAST API

Kelebihan dari FastAPI (<https://fastapi.tiangolo.com/lo/>) adalah:

1. Menggunakan bahasa Python yang populer sehingga tidak perlu belajar bahasa baru.
2. Framework sederhana dan mudah dipelajari.

3. Cepat diantara framework python lainnya, walaupun dibandingkan framework lain seperti Express dan Fiber, FastAPI masih relatif lebih lambat (Benchmark lengkap berbagai alternatif framework dapat dilihat di: https://www.techempower.com/benchmarks/#section=data-r20&hw=ph&test=query)).

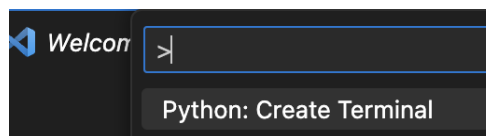
Instalasi FastAPI dan Hello World

Install Visual Studio Code

Install Python dan Conda (Conda dapat digantikan virtualenv)

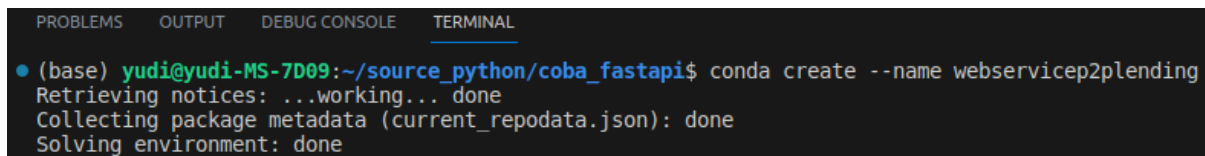
Install plugin python pada VS code.

Buka terminal di VSC (ctrl-shift-P untuk Windows lalu cari Python: Create Terminal seperti pada gambar di bawah)



Buat env conda di dalam terminal (nama dapat disesuaikan dengan keinginan)

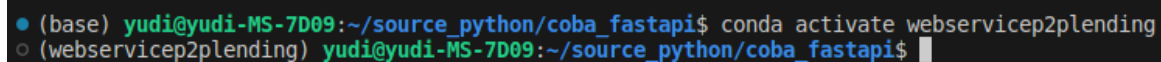
```
conda create --name webservicep2plending
```

A screenshot of a terminal window in Visual Studio Code. The terminal shows the output of the command 'conda create --name webservicep2plending'. The output is: '(base) yudi@yudi-MS-7D09:~/source_python/coba_fastapi\$ conda create --name webservicep2plending', 'Retrieving notices: ...working... done', 'Collecting package metadata (current_repodata.json): done', and 'Solving environment: done'. The terminal has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, with TERMINAL selected.

aktifkan environment tersebut

```
conda activate webservicep2plending
```

prompt akan berubah

A screenshot of a terminal window in Visual Studio Code. The terminal shows the output of the command 'conda activate webservicep2plending'. The output is: '(base) yudi@yudi-MS-7D09:~/source_python/coba_fastapi\$ conda activate webservicep2plending', '○ (webservicep2plending) yudi@yudi-MS-7D09:~/source_python/coba_fastapi\$'. The prompt has changed from '(base)' to '○ (webservicep2plending)'. The terminal has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, with TERMINAL selected.

install pip

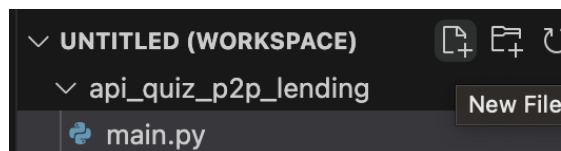
```
conda install pip
```

install fastapi dan uvicorn untuk web servernya.

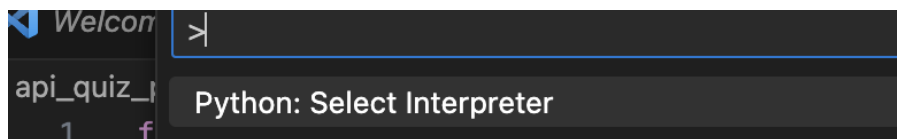
```
pip install fastapi
pip install "uvicorn[standard]"
```

Sekarang kita sudah siap untuk membuat app pertama, hello world

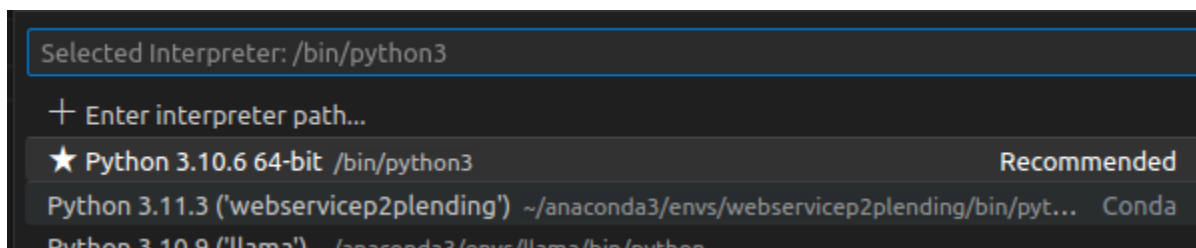
Buat directory baru, buat file main.py



Pilih python interpreter.



pilih environment yang telah kita buat tadi (webservicep2plending)



Tambahkan code berikut di main.py:

```
from typing import Union
from fastapi import FastAPI
```

```
app = FastAPI()
```

```
@app.get("/")
def read_root():
    return {"Hello": "World"}
```

Jalankan web server uvicorn di terminal:

```
uvicorn main:app --reload
```

Server akan berjalan di alamat 127.0.0.1:8000 (untuk stop server tekan ctrl-c)

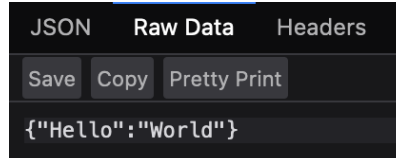
```
nding']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [27125] using WatchFiles
INFO:     Started server process [27129]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:57140 - "GET / HTTP/1.1" 200 OK
```

Opsi reload membuat webserver akan otomatis reload code terbaru (tidak perlu dimatikan lalu dinyalakan). Sangat berguna untuk proses debug, tapi bukan untuk produksi.

Buka browser di alamat

<http://127.0.0.1:8000/>

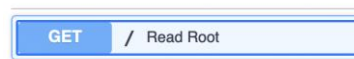
Maka hasilnya



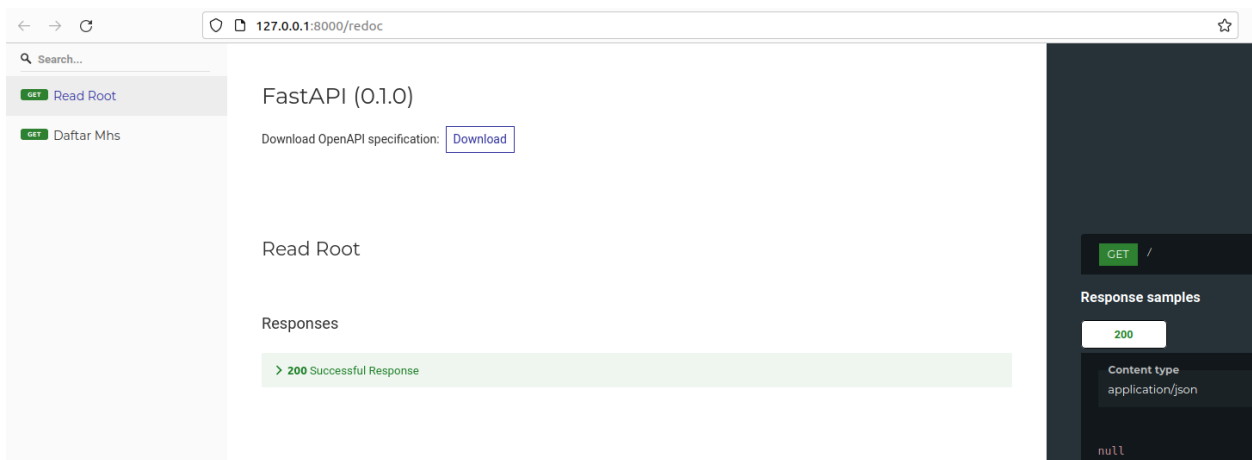
Dokumentasi juga otomatis dibangkitkan di <http://127.0.0.1:8000/docs>



default



Alternatif docs yang lain adalah <http://127.0.0.1:8000/redoc>



Catatan untuk Flutter: jika menggunakan web service lokal dan terjadi XMLHttpRequest error pada aplikasi Flutter, ini dapat disebabkan oleh CORS (Cross-Origin Resource Sharing) karena frontend flutter app yang dijalankan di Chrome menggunakan JScript yang berkomunikasi dengan backend, sehingga backend memiliki "origin" berbeda dengan frontend. Solusinya ada di sisi server, tambahkan code berikut saat pembuatan instance FastAPI:

```
from fastapi.middleware.cors import CORSMiddleware
from fastapi import FastAPI
app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

GET, POST, PUT, PATCH, DELETE

GET, POST dan PUT adalah method HTTP yang digunakan untuk berinteraksi dengan web resource. Sebenarnya ada method lain seperti HEAD, OPTION, TRACE tetapi tidak umum digunakan.

Fungsi dari setiap method adalah sebagai berikut:

GET: untuk mengambil resource.

POST: submit data untuk membuat resource baru (insert). Walaupun POST dapat digunakan untuk mengupdate, tetapi dianjurkan untuk menggunakan PUT.

PUT: mengupdate resource secara keseluruhan (replace).

PATCH: mirip seperti PUT tetapi hanya update sebagian

DELETE: menghapus resources

Perbedaan PUT vs POST adalah PUT sifatnya idempotent, artinya PUT yang sama walaupun dipanggil berkali-kali tidak akan mengubah state. POST sebaliknya, data yang sama jika dikirimkan berkali-kali melalui POST akan mengubah state.

Catatan, code lengkap untuk semua contoh di bawah: <https://pastebin.com/raw/7dweXC72>

FastAPI GET

Sebelumnya kita telah menggunakan method GET ini untuk mengirimkan "hello world".
"/" artinya root.

```
@app.get("/")
def read_root():
    return {"Hello": "World"}
```


Kita akan mencoba menggunakan GET dengan parameter. Parameter dapat ditambahkan dalam path, misalnya untuk mengambil data mahasiswa berdasarkan nim, path dapat berbentuk (asumsikan NIM=13594022:

```
http://127.0.0.1:8000/mahasiswa/13594022
```

atau alternatif lain dengan query

```
http://127.0.0.1:8000/mahasiswa/?nim=13594022
```

Berikut contoh implementasi model path untuk endpoint:

```
http://127.0.0.1:8000/mahasiswa/1234
```

Tambahkan code berikut di dalam main.py, simpan maka akan dilakukan autoreload oleh uvicorn.

```
@app.get("/mahasiswa/{nim}")
def ambil_mhs(nim:str):
    return {"nama": "Budi Martami"}
```

Sedangkan contoh implementasi model query (bukan path) yang sama.

```
http://127.0.0.1:8000/mahasiswa2/?nim=1234
```

Tambahkan

```
@app.get("/mahasiswa2/")
def ambil_mhs2(nim:str):
    return {"nama": "Budi Martami 2"}
```

Model query cocok jika parameter lebih dari satu. Sebagai contoh jika kita mengambil data mahasiswa berdasarkan id provinsi dan angkatan

```
http://127.0.0.1:8000/daftar_mhs/?id_prov=22&angkatan=2019
```

```
@app.get("/daftar_mhs/")
def daftar_mhs(id_prov:str, angkatan:str):
    return {"query": "idprov: {} ; angkatan: {}"
            .format(id_prov, angkatan), "data": [{"nim": "1234"}, {"nim": "1235"}]}
```

Jika dijalankan hasilnya

```
{"query": "idprov: 22 ; angkatan: 2019", "data": [{"nim": "1234"}, {"nim": "1235"}]}
```

Kita dapat melakukan test API secara lebih mudah melalui `http://127.0.0.1:8000/docs`

GET /daftar_mhs/ Daftar Mhs

Parameters

Name	Description
id_prov * required string (query)	22
angkatan * required string (query)	2009

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/daftar_mhs/?id_prov=22&angkatan=2009' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/daftar_mhs/?id_prov=22&angkatan=2009
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "query": " idprov: 22 ; angkatan: 2009 ", "data": [{ "nim": "1234" }] }</pre>

FastAPI POST

Berbeda dengan GET yang menyimpan parameter pada URL, POST menyimpan informasi pada body. Gunakan POST untuk menambahkan data baru.

Kita akan membuat database SQLite sederhana dengan satu tabel, lalu POST untuk insert satu mahasiswa dan GET untuk menampilkan data. SQLite tidak memerlukan instalasi terpisah karena sudah terintegrasi dengan Python.

Pertama untuk inisialisasi database. End point ini dipanggil sekali saja untuk meng-create database dan tabel yang diperlukan.

```
# panggil sekali saja
@app.get("/init/")
def init_db():
    try:
        DB_NAME = "upi.db"
        con = sqlite3.connect(DB_NAME)
        cur = con.cursor()
```

```

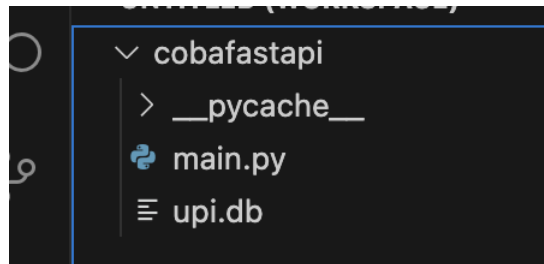
create_table = """ CREATE TABLE mahasiswa(
    ID      INTEGER PRIMARY KEY      AUTOINCREMENT,
    nim     TEXT          NOT NULL,
    nama    TEXT          NOT NULL,
    id_prov TEXT          NOT NULL,
    angkatan TEXT        NOT NULL,
    tinggi_badan INTEGER
)
"""

cur.execute(create_table)
con.commit
except:
    return ({"status": "terjadi error"})
finally:
    con.close()

return ({"status": "ok, db dan tabel berhasil dicreate"})

```

Panggil endpoint ini sekali saja karena database cukup sekali dibuat. Selanjutnya dapat dilihat di VSC, ada satu file yang telah di-create yaitu upi.db.



Catatan:

Seluruh database SQLite disimpan dalam satu file. Gunakan tools seperti <https://sqlitebrowser.org/> untuk membuat tabel, melihat isi dan melakukan SQL query

Berikutnya kita akan buat POST untuk menginsert satu record mahasiswa. Buat class Mhs yang merupakan turunan dari BaseModel dari pydantic. Untuk proses insert ke tabel, cara low level ini sebaiknya dihindari karena rentan terkena SQL injection, gunakan library seperti SQLAlchemy.

```

from pydantic import BaseModel

```

```
class Mhs(BaseModel):
    nim: str
    nama: str
    id_prov: str
    angkatan: str
    tinggi_badan: int | None = None

@app.post("/tambah_mhs/")
def tambah_mhs(m: Mhs):
    try:
        DB_NAME = "upi.db"
        con = sqlite3.connect(DB_NAME)
        cur = con.cursor()
        # hanya untuk test, rawan sql injection, gunakan SQLAlchemy
        cur.execute("""insert into mahasiswa (nim,nama,id_prov,angkatan,tinggi_badan)
values
( "{", "{", "{", "{", "}" )""".format(m.nim,m.nama,m.id_prov,m.angkatan,m.tinggi_badan))
        con.commit()
    except:
        return ({"status": "terjadi error"})
    finally:
        con.close()
    return {"status": "ok berhasil insert satu record"}
```

Karena jika menggunakan POST parameter diletakkan di body, maka gunakan 127.0.0.1/docs untuk mencoba endpoint ini (gambar di bawah).

POST

/tambah_mhs/

Tambah Mhs

Parameters

No parameters

Request body

required

application/json

```
{
  "nim": "111",
  "nama": "budi",
  "id_prov": "23",
  "angkatan": "2019",
  "tinggi_badan": 170
}
```

Coba lakukan insert beberapa data, setelah itu kita akan buat endpoint untuk menampilkan semua data (tidak direkomendasikan dalam kondisi sebenarnya, karena berbahaya jika jumlah data sudah sangat banyak).

Catatan

Menurut standar protokol HTTP, pemanggilan POST harus me-return data yang di-create dan menghasilkan status 201. Fungsi POST juga harus mengembalikan header "Location" yang berisi alamat objek yang baru discrete tersebut.

Jadi jika ingin sesuai standar, fungsi POST sebelumnya perlu dimodifikasi sebagai berikut:

```
#status code 201 standard return creation
#return objek yang baru dicreate (response_model tipenya Mhs)
@app.post("/tambah_mhs/", response_model=Mhs,status_code=201)
def tambah_mhs(m: Mhs,response: Response, request: Request):
    try:
        DB_NAME = "upi.db"
        con = sqlite3.connect(DB_NAME)
        cur = con.cursor()
        # hanya untuk test, rawan sql injection, gunakan spt SQLAlchemy
        cur.execute("""insert into mahasiswa (nim,nama,id_prov,angkatan,tinggi_badan)
values
( "{}", "{}", "{}", "{}", {})""".format(m.nim,m.nama,m.id_prov,m.angkatan,m.tinggi_badan
))
        con.commit()
    except:
        return ({"status":"terjadi error"})
    finally:
        con.close()
    # tambah location data yg baru di-add di header
    response.headers["location"] = "/mahasiswa/{}".format(m.nim)
    return m
```

Dibagian parameter function ada tambahan response_model = Mhs yang artinya mewajibkan return bertipe Mhs. status_code=201 artinya method ini menghasilkan return 201 jika berhasil. response_header['Location'] diisi dengan alamat objek yang baru di-add ini.

Jika dijalankan returnnya adalah sebagai berikut. Perhatikan returns 201 dan header 'location'.

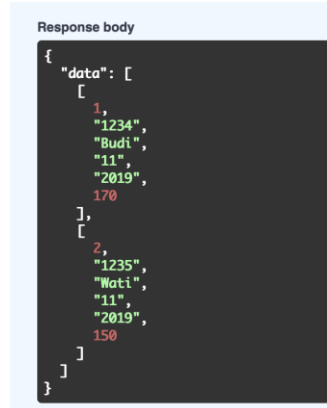
Code	Details
201 <i>Undocumented</i>	Response body <pre>{ "nim": "5573", "nama": "ahmad7", "id_prov": "9", "angkatan": "2017", "tinggi_badan": 160 }</pre> Response headers <pre>content-length: 81 content-type: application/json date: Mon, 15 May 2023 22:28:12 GMT location: /mahasiswa/5573 server: uvicorn</pre>

Menampilkan semua mahasiswa

Untuk memeriksa apakah data sudah masuk ke database, tambahkan endpoint berikut untuk menampilkan semua record di dalam database.

```
@app.get("/tampilkan_semua_mhs/")
def tampil_semua_mhs():
    try:
        DB_NAME = "upi.db"
        con = sqlite3.connect(DB_NAME)
        cur = con.cursor()
        recs = []
        for row in cur.execute("select * from mahasiswa"):
            recs.append(row)
    except:
        return ({"status": "terjadi error"})
    finally:
        con.close()
    return {"data": recs}
```

Jika dipanggil hasilnya



FastAPI PUT

Put digunakan untuk mengupdate data. Jangan lupa sifat PUT yang idempoten (dipanggil berkali-kali hasilnya sama, tidak mengubah state). PUT digunakan untuk menggantikan secara **keseluruhan** data. Untuk penggantian secara parsial, gunakan PATCH (dibahas berikutnya).

Berikut adalah code-nya

```
@app.put("/update_mhs_put/{nim}", response_model=Mhs)
def update_mhs_put(response: Response, nim: str, m: Mhs):
    #update keseluruhan record
    try:
        DB_NAME = "upi.db"
        con = sqlite3.connect(DB_NAME)
        cur = con.cursor()
        sqlstr = "update mahasiswa set nama = '{}',id_prov = '{}', angkatan='{}',
tinggi_badan={} where
nim='{}'".format(m.nama,m.id_prov,m.angkatan,m.tinggi_badan,m.nim)
        cur.execute(sqlstr)
        con.commit()
        response.headers["location"] = "/mahasiswa/{}".format(m.nim)
    except:
        return ({ "status": "terjadi error" })
    finally:
        con.close()

    return m
```

Jika item tidak ditemukan, idealnya PUT menghasilkan error 404 (item not found). Jika terjadi exception (misal ada masalah di database), maka error yang umum adalah 500 (internal server error).

Code di atas dapat dapat diupdate sehingga mengembalikan status code yang lebih akurat sebagai berikut:

```
raise HTTPException(status_code=500, detail="Terjadi exception")
raise HTTPException(status_code=404, detail="Item not found")
```

Code lengkapnya yang menangani record mahasiswa tidak ditemukan

```
from fastapi import FastAPI, Response, Request, HTTPException

FastAPI PATCH@app.put("/update_mhs_put/{nim}", response_model=Mhs)
def update_mhs_put(response: Response, nim: str, m: Mhs ):
    #update keseluruhan
    #karena primary key, nim tidak diupdate
    #data mhs disimpan di "m", nim yang akan diupdate disimpan di "nim"
    try:
        DB_NAME = "upi.db"
        con = sqlite3.connect(DB_NAME)
        cur = con.cursor()
        cur.execute("select * from mahasiswa where nim = ?", (nim,) ) #tambah koma untuk
menandakan tuple
        existing_item = cur.fetchone()
    except Exception as e:
        raise HTTPException(status_code=500, detail="Terjadi exception:
{}".format(str(e))) # misal database down

    if existing_item: #data ada, lakukan update
        cur.execute("update mahasiswa set nama = ?, id_prov = ?, angkatan=?,
tinggi_badan=? where nim=?", (m.nama,m.id_prov,m.angkatan,m.tinggi_badan,nim))
        con.commit()
        response.headers["location"] = "/mahasiswa/{}".format(m.nim)
    else: # data tidak ada 404, item not found
        raise HTTPException(status_code=404, detail="Item Not Found")

    con.close()
    return m
```

FastAPI PATCH

PATCH digunakan untuk mengupdate data sebagian. Untuk mengganti data keseluruhan gunakan PUT.

Jika kita menggunakan struktur Mhs maka seluruh field harus ada, padahal kita ingin agar hanya field yang ingin di-update yang dikirim.

Solusinya adalah membuat struktur Mhs yang baru yang memungkinkan nilai Null. Penggunaan "kosong" agar kita dapat membedakan apakah user tidak ingin mengupdate field tersebut atau mau mengisi field dengan None.

```
# khusus untuk patch, jadi boleh tidak ada
class MhsPatch(BaseModel):
    nama: str | None = "kosong"
    id_prov: str | None = "kosong"
    angkatan: str | None = "kosong"
    tinggi_badan: Optional[int] | None = -9999
```

Dilakukan pengecekan apakah mahasiswa dengan nim tersebut ada, jika tidak ada return 401

```
@app.patch("/update_mhs_patch/{nim}", response_model = MhsPatch)
def update_mhs_patch(response: Response, nim: str, m: MhsPatch ):
    try:
        print(str(m))
        DB_NAME = "upi.db"
        con = sqlite3.connect(DB_NAME)
        cur = con.cursor()
        cur.execute("select * from mahasiswa where nim = ?", (nim,) ) #tambah koma untuk
# menandakan tuple
        existing_item = cur.fetchone()
    except Exception as e:
        raise HTTPException(status_code=500, detail="Terjadi exception: {}".format(str(e)))
# misal database down

    if existing_item: #data ada, lakukan update
        sqlstr = "update mahasiswa set " #asumsi minimal ada satu field update
        # todo: bisa di-refactor dan dirapikan
        if m.nama!="kosong":
            if m.nama!=None:
                sqlstr = sqlstr + " nama = '{}' ,".format(m.nama)
            else:
                sqlstr = sqlstr + " nama = null ,"

        if m.angkatan!="kosong":
            if m.angkatan!=None:
```

```

        sqlstr = sqlstr + " angkatan = '{}' ,".format(m.angkatan)
    else:
        sqlstr = sqlstr + " angkatan = null ,"

    if m.id_prov!="kosong":
        if m.id_prov!=None:
            sqlstr = sqlstr + " id_prov = '{}' ,".format(m.id_prov)
        else:
            sqlstr = sqlstr + " id_prov = null, "

    if m.tinggi_badan!=-9999:
        if m.tinggi_badan!=None:
            sqlstr = sqlstr + " tinggi_badan = {} ,".format(m.tinggi_badan)
        else:
            sqlstr = sqlstr + " tinggi_badan = null ,"

    sqlstr = sqlstr[:-1] + " where nim='{}' ".format(nim) #buang koma yang trakhir
    print(sqlstr)
    try:
        cur.execute(sqlstr)
        con.commit()
        response.headers["location"] = "/mahasixswa/{}".format(nim)
    except Exception as e:
        raise HTTPException(status_code=500, detail="Terjadi exception:
        {}".format(str(e)))

    else: # data tidak ada 404, item not found
        raise HTTPException(status_code=404, detail="Item Not Found")

con.close()
return m

```

Todo: dirapikan dan disederhanakan.

Contoh penggunaan untuk mahasiswa dengan nim=13594022

Angkatan tidak diubah sehingga tidak masuk ke JSON, tinggi badan diupdate dengan null dan nama diganti.

PATCH

/update_mhs_patch/{nim} Update Mhs Patch

Parameters

Name	Description
------	-------------

nim * required

string

(path)

13594022

Request body required

```
{
  "nama": "Asep Furqon",
  "id_prov": "21",
  "tinggi_badan": null
}
```

FastAPI DELETE

DELETE digunakan untuk menghapus resource. Sebagai contoh:

```
@app.delete("/delete_mhs/{nim}")
def delete_mhs(nim: str):
    try:
        DB_NAME = "upi.db"
        con = sqlite3.connect(DB_NAME)
        cur = con.cursor()
        sqlstr = "delete from mahasiswa where nim='{}'.format(nim)
        print(sqlstr) # debug
        cur.execute(sqlstr)
        con.commit()
    except:
        return ({ "status": "terjadi error" })
    finally:
        con.close()

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

Code lengkap: <https://pastebin.com/raw/7dweXC72>

POST dan GET file image

```
# pip install python-multipart
# buat terlebih dulu direktori /data_file untuk menyimpan file

from fastapi import File, UploadFile
from fastapi.responses import FileResponse

#upload image
@app.post("/uploadimage")
def upload(file: UploadFile = File(...)):
    try:
        print("mulai upload")
        print(file.filename)
        contents = file.file.read()
        with open("./data_file/"+file.filename, 'wb') as f:
            f.write(contents)
    except Exception:
```

```
        return {"message": "Error upload file"}
    finally:
        file.file.close()

    return {"message": "Upload berhasil: {file.filename}"}

# ambil image berdasarkan nama file

@app.get("/getimage/{nama_file}")
async def getImage(nama_file:str):
    return FileResponse("./data_file/"+nama_file)
```

Todo: token Oauth

LAIN-LAIN

Best Practice untuk perancangan RESTful API:

<https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api>

Contoh tutorial deploy FAST API ke Google Clouds:

<https://tutlinks.com/deploy-fastapi-app-on-google-cloud-platform/>

Todo:
Penjelasan tentang BaseModel Pydantic untuk verifikasi.