

PROJECT 5

BY 8DREAM (KELOMPOK 1)

Data Realm Engineers And Maestros



Anggota Kelompok

01

**Afroh
Fauziah**

02

**Althaf Nawadir
Taqiyyah**

03

**Andi
Rosilala**

04

**Andrew
Bintang
Pratama**

05

**Andrew
Fortino
Mahardika
Suadnya**



TUJUAN PROJEK

Project 5 - Create CRUD API Using Flask and Docker

Tools yang harus disiapkan:

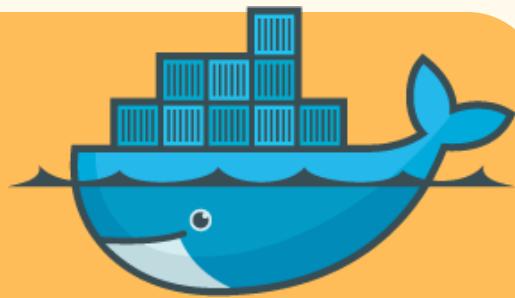
- docker
- postman
- visual code

API yang dibuat:

- create user
- update user
- get user
- delete user

Projek ini bertujuan untuk membuat sebuah API CRUD menggunakan Flask dan Docker. Alat yang dibutuhkan adalah Docker untuk manajemen kontainer, Postman untuk pengujian API, dan Visual Studio Code sebagai code editor. API yang akan dibuat mencakup operasi dasar CRUD untuk entitas user: create, update, get, dan delete.

STEP 1: Menyalakan Docker Menggunakan Docker Desktop

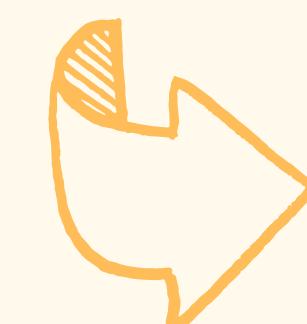


The screenshot shows the Docker Desktop application window. The left sidebar has icons for Containers, Images, Volumes, Builds, Dev Environments (BETA), and Docker Scout. Below these are sections for Extensions and Add Extensions. The main area is titled 'Containers' with a sub-instruction 'Give feedback'. It features a central graphic of three interconnected containers and the text 'Your running containers show up here'. A subtitle explains that 'A container is an isolated environment for your code'. At the bottom, there are two cards: 'What is a container?' (5 mins) and 'How do I run a container?' (6 mins), each with a preview of Docker commands. A link 'View more in the Learning center' is at the bottom of these cards. The status bar at the bottom shows 'Engine running', system icons, 'RAM 2.99 GB CPU 0.50%', 'Signed in', and 'v4.29.0'.



STEP 2: Membuat file docker-compose.yml

File `docker-compose.yml` ini digunakan untuk mendefinisikan layanan yang akan dijalankan dalam lingkungan Docker. Tujuannya untuk menyiapkan sebuah layanan PostgreSQL, database yang akan digunakan sebagai basis data untuk aplikasi Flask yang akan dibuat.



```
docker-compose.yml
1 version: "2"
2 services:
3   postgres-db:
4     image: postgres
5     environment:
6       POSTGRES_PASSWORD: admin
7       POSTGRES_USER: postgres
8       PGDATA: /var/lib/postgres/data
9       POSTGRES_DB: sample
10    volumes:
11      - pg-data:/var/lib/postgres/data
12    ports:
13      - 5432:5432/tcp
14    volumes:
15      pg-data: #docker volume create pg-data
16      | external: true
```

STEP 3: Menjalankan Kontainer

Command 'docker volume create pg-data' dijalankan untuk membuat volume Docker dengan nama **pg-data**. Tujuannya untuk menyiapkan tempat penyimpanan data yang akan digunakan oleh kontainer Docker (Postgres).

```
andrew@MSI:/mnt/d/Dokumen/TUGAS-TUGAS SIB DigitalSkola/TUGAS PROJEK (Individu + Kelompok)/PROJEK 5/PROJECT-5-SIB-DigitalSkola$ docker volume create pg-data
pg-data
andrew@MSI:/mnt/d/Dokumen/TUGAS-TUGAS SIB DigitalSkola/TUGAS PROJEK (Individu + Kelompok)/PROJEK 5/PROJECT-5-SIB-DigitalSkola$ docker-compose up -d
[+] Running 15/15
  ✓ postgres-db Pulled
  ✓ b0a0cf830b12 Pull complete
  ✓ dda3d8fdbd5ed Pull complete
  ✓ 283a477db7bb Pull complete
  ✓ 91d2729fa4d5 Pull complete
  ✓ 9739ced65621 Pull complete
  ✓ ae3bb1b347a4 Pull complete
  ✓ f8406d9c00ea Pull complete
  ✓ c199bff16b05 Pull complete
  ✓ e0d55fdb4d15 Pull complete
  ✓ c1cb13b19080 Pull complete
  ✓ 873532e5f8c7 Pull complete
  ✓ 050d9f8c3b1c Pull complete
  ✓ 710e142705f8 Pull complete
  ✓ cb628c265f09 Pull complete
[+] Running 2/2
  ✓ Network project-5-sib-digitalskola_default      Created
  ✓ Container project-5-sib-digitalskola-postgres-db-1 Started
```

Menjalankan kontainer Docker berdasarkan konfigurasi yang didefinisikan dalam file "docker-compose.yml" dengan command '**docker-compose up -d**'



Melihat status pada docker desktop:
tandanya sudah aktif

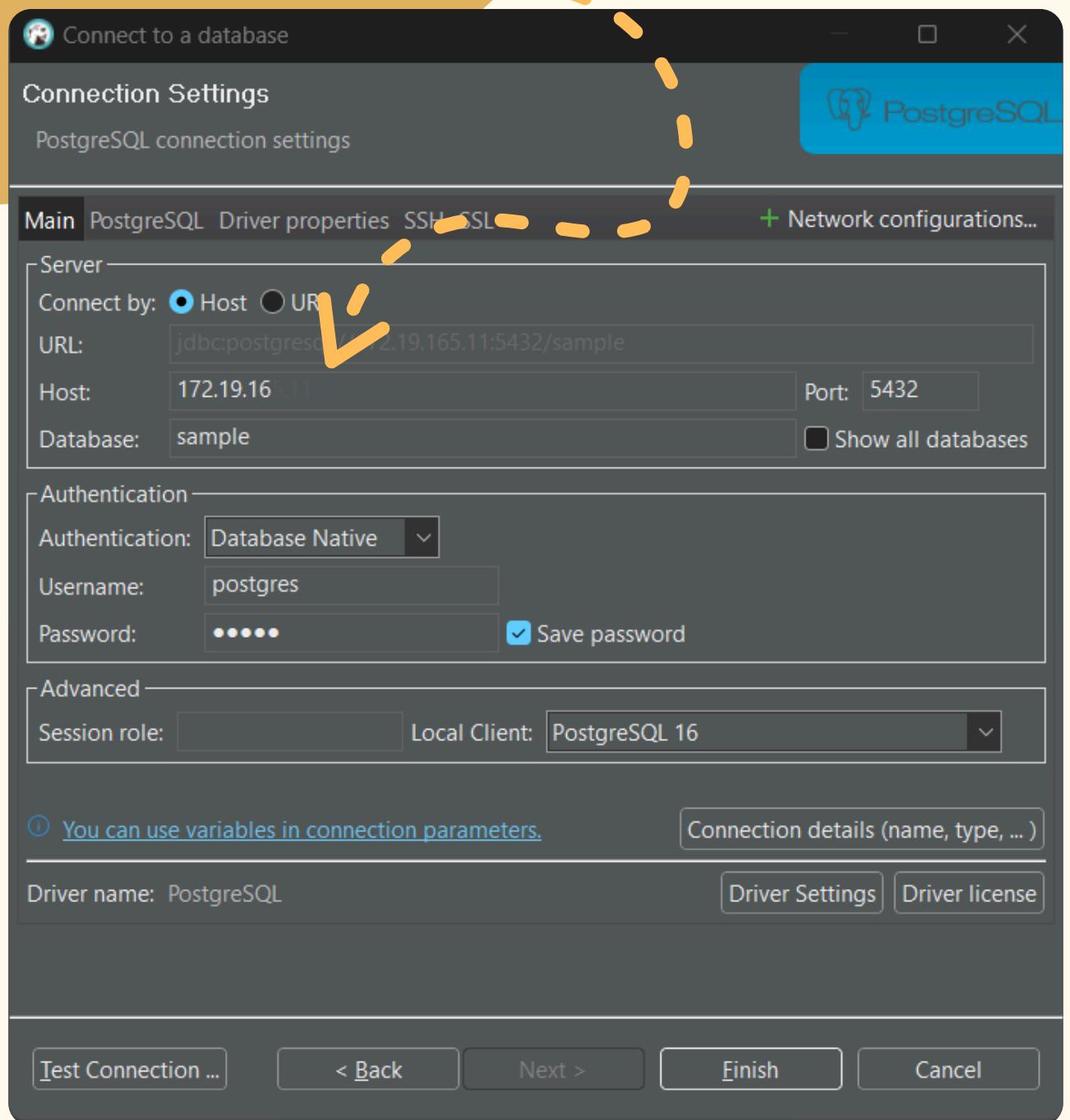
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
190e4331d792	postgres	"docker-entrypoint.s..."	28 seconds ago	Up 24 seconds	0.0.0.0:5432->5432/tcp	project-5-sib-digitalskola-postgres-db-1

Cek status dengan command '**docker ps**'

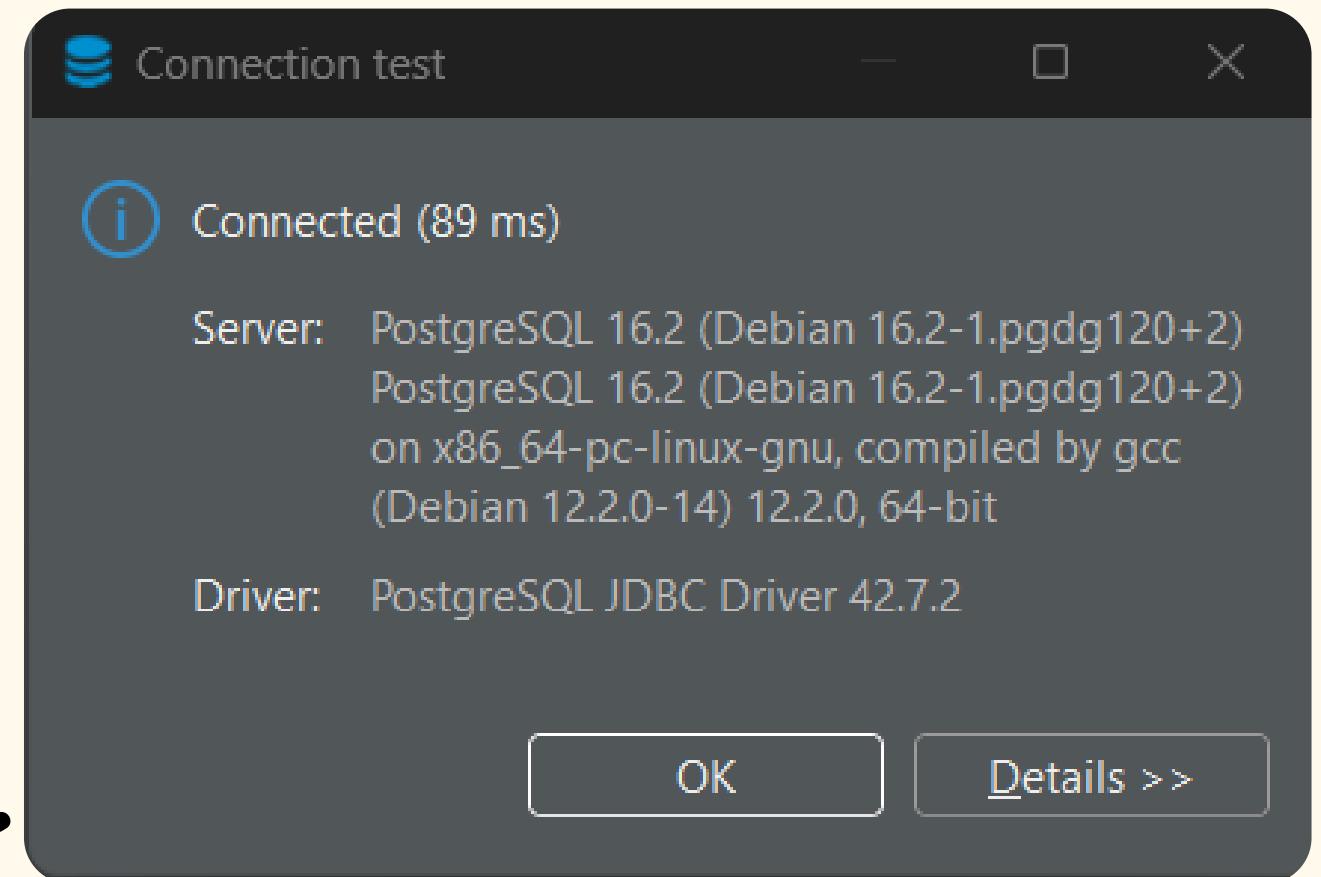
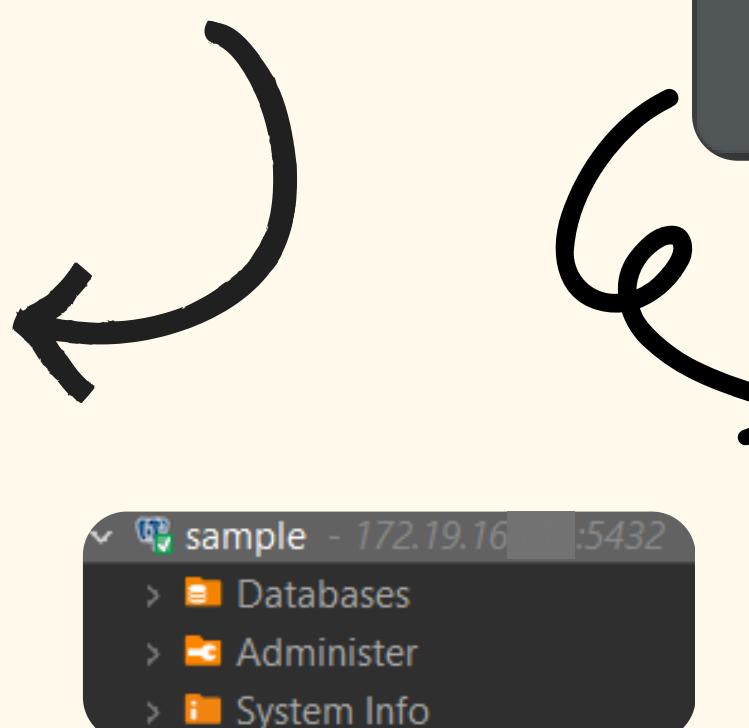
STEP 4: Test Connection Postgres



Host menggunakan
IP network



**Tes koneksi
ke database
Postgres
menggunakan
DBeaver**



**Connection tes “connected” artinya
sudah berhasil terkoneksi ke
Postgres dengan konfigurasi dari file
docker-compose.yml**



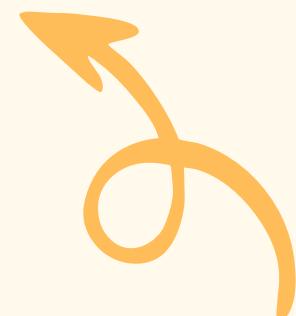
STEP 5: Membuat Endpoint untuk Memeriksa Kesehatan Aplikasi dan Koneksi Database pada Flask

```
requirements.txt
1 Flask
2 psycopg2
3 python-decouple
4 flask-sqlalchemy
```



File requirements.txt berisikan requirements yang diperlukan.

```
app > main.py > ...
1  from flask import Flask, jsonify
2  from decouple import config
3  import psycopg2
4
5
6  app = Flask(__name__)
7
8
9  @app.route("/health")
10 def health():
11     return jsonify({"status": "oke"})
12
13
14 @app.route("/db_check")
15 def db_check():
16     conn_pg = psycopg2.connect(
17         host=config('MB_DB_HOST'),
18         database=config('MB_DB_DBNAME'),
19         user=config('MB_DB_USER'),
20         password=config('MB_DB_PASS'),
21         port=int(config('MB_DB_PORT')),
22     )
23     cur = conn_pg.cursor()
24     return jsonify({"status": 200, "db": "connected"})
25
26
27 if __name__ == "__main__":
28     app.run(debug=True, host="0.0.0.0")
```



File main.py menyediakan endpoints yang dapat digunakan untuk memeriksa kesehatan aplikasi secara umum (/health) dan untuk memeriksa koneksi ke database (/db_check)

STEP 6: Konfigurasi Dockerfile untuk Membangun Image Docker Aplikasi Flask

.....

```
📄 Dockerfile > ...
1 FROM python:3.7
2
3 COPY requirements.txt /requirements.txt
4
5 COPY ./app /app
6
7 WORKDIR /
8
9 RUN pip install -r requirements.txt
10
11 ENTRYPOINT ["python"]
12
13 CMD ["app/main.py", "--reload"]
```



Dockerfile bertujuan untuk membangun image Docker yang berisi aplikasi Flask beserta dependensinya, dan mengatur agar aplikasi Flask dijalankan saat container Docker berjalan.

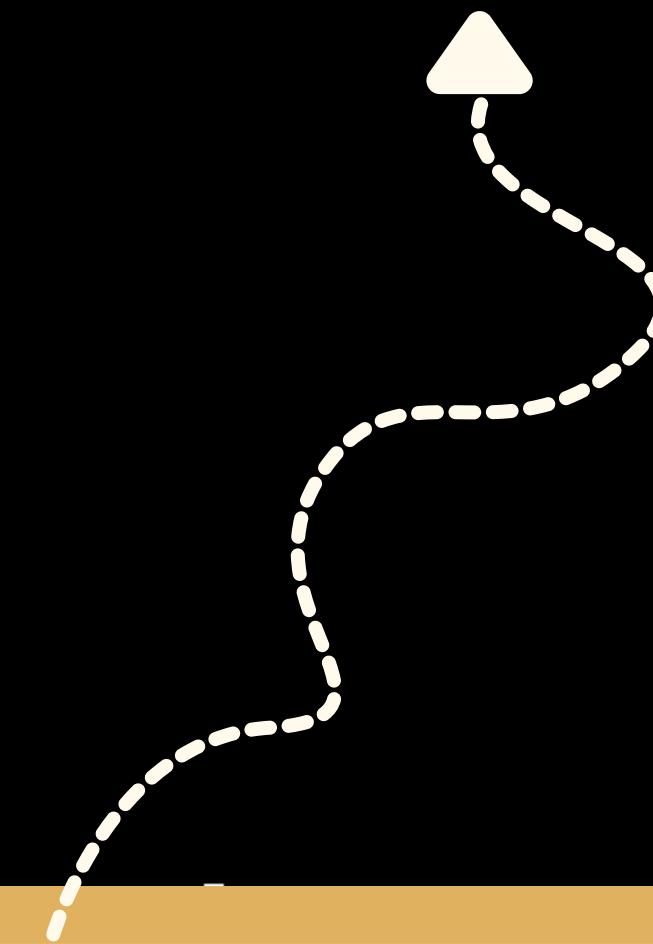


STEP 7: Build Image Docker

```
andrew@MSI:/mnt/d/Dokumen/TUGAS-TUGAS SIB DigitalSkola/TUGAS PROJEK (Individu + Kelompok)/PROJEK 5/PROJECT-5-SIB-Digitalskola$ docker build -t project-5-sib-digitalskola .  
[+] Building 30.8s (9/9) FINISHED  
=> [internal] load build definition from Dockerfile  
=> => transferring dockerfile: 228B  
=> [internal] load metadata for docker.io/library/python:3.7  
=> [internal] load .dockerignore  
=> => transferring context: 2B  
=> [1/5] FROM docker.io/library/python:3.7@sha256:eedf63967cdb57d8214db38ce21f105003ed4e4d0358f02bedc057341bcf92a0  
=> [internal] load build context  
=> => transferring context: 91B  
=> CACHED [2/5] COPY requirements.txt /requirements.txt  
=> CACHED [3/5] COPY ./app /app  
=> [4/5] RUN pip install -r requirements.txt  
=> exporting to image  
=> => exporting layers  
=> => writing image sha256:af9dc73b7c39fb6f873fe6fb983fdc1eddbb95a3f3792429fdd9b388d42b68da  
=> => naming to docker.io/library/project-5-sib-digitalskola
```

What's Next?

View a summary of image vulnerabilities and recommendations → [docker scout quickview](#)



Command 'docker build -t project-5-sib-digitalskola' digunakan untuk membangun image Docker dari Dockerfile yang ada di dalam direktori.

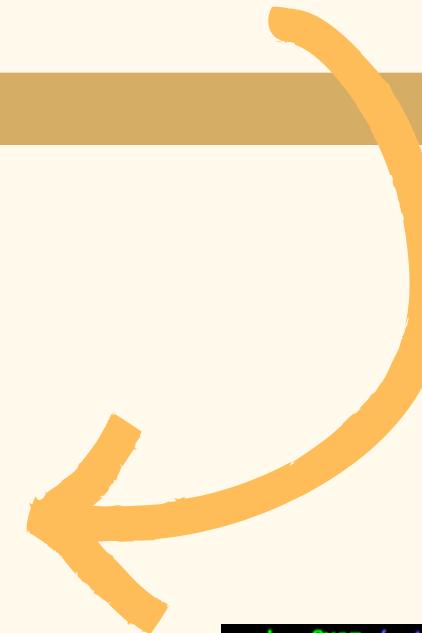
```
andrew@MSI:/mnt/d/Dokumen/TUGAS-TUGAS SIB DigitalSkola/TUGAS PROJEK (Individu + Kelompok)/PROJEK 5/PROJECT-5-SIB-Digitalskola$ docker images  
REPOSITORY          TAG      IMAGE ID   CREATED        SIZE  
project-5-sib-digitalskola  latest    af9dc73b7c39  52 minutes ago  1.03GB  
postgres            latest    8e4fc9e18489  2 months ago   431MB
```

Mengecek docker image yang sudah dibuat dengan command 'docker images'



STEP 8: Menambahkan layanan flask-app pada file docker-compose.yml

```
docker-compose.yml
1 services:
2   postgres-db:
3     image: postgres
4     environment:
5       POSTGRES_PASSWORD: admin
6       POSTGRES_USER: postgres
7       PGDATA: /var/lib/postgres/data
8       POSTGRES_DB: sample
9     volumes:
10    - pg-data:/var/lib/postgres/data
11   ports:
12    - "5432:5432/tcp"
13
14 flask-app:
15   image: project-5-sib-digitalskola
16   environment:
17     MB_DB_DBNAME: postgres
18     MB_DB_HOST: postgres-db
19     MB_DB_PASS: admin
20     MB_DB_PORT: 5432
21     MB_DB_TYPE: postgres
22     MB_DB_USER: postgres
23   volumes:
24    - ./app:/app
25   links:
26    - postgres-db:postgres-db
27   ports:
28    - "5000:5000/tcp"
29
30 volumes:
31   pg-data: #docker volume create pg-data
32   | external: true
```



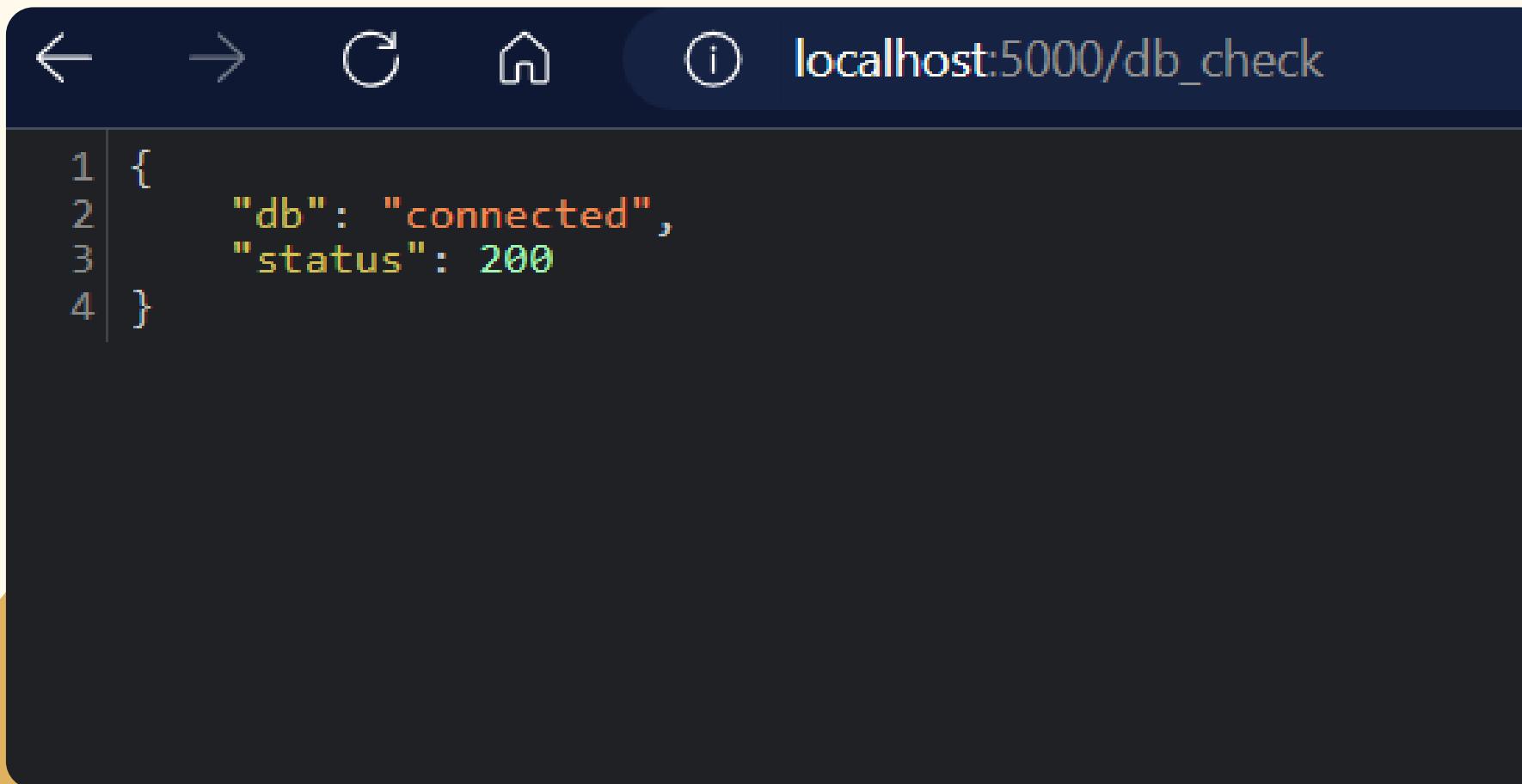
Mematikan docker compose 'docker-compose down' kemudian menjalankan kembali dengan command 'docker-compose up -d'

```
andrew@MSI:/mnt/d/Dokumen/TUGAS-TUGAS SIB DigitalSkola/TUGAS PROJEK (Individu + Kelompok)/PROJEK 5/PROJECT-5-SIB-DigitalSkola$ docker-compose down
[+] Running 3/3
  ✓ Container project-5-sib-digitalskola-flask-app-1 Removed
  ✓ Container project-5-sib-digitalskola-postgres-db-1 Removed
  ✓ Network project-5-sib-digitalskola default Removed
andrew@MSI:/mnt/d/Dokumen/TUGAS-TUGAS SIB DigitalSkola/TUGAS PROJEK (Individu + Kelompok)/PROJEK 5/PROJECT-5-SIB-DigitalSkola$ docker-compose up -d
[+] Running 3/3
  ✓ Network project-5-sib-digitalskola_default Created
  ✓ Container project-5-sib-digitalskola-postgres-db-1 Started
  ✓ Container project-5-sib-digitalskola-flask-app-1 Started
```

Cek status dengan command 'docker ps'

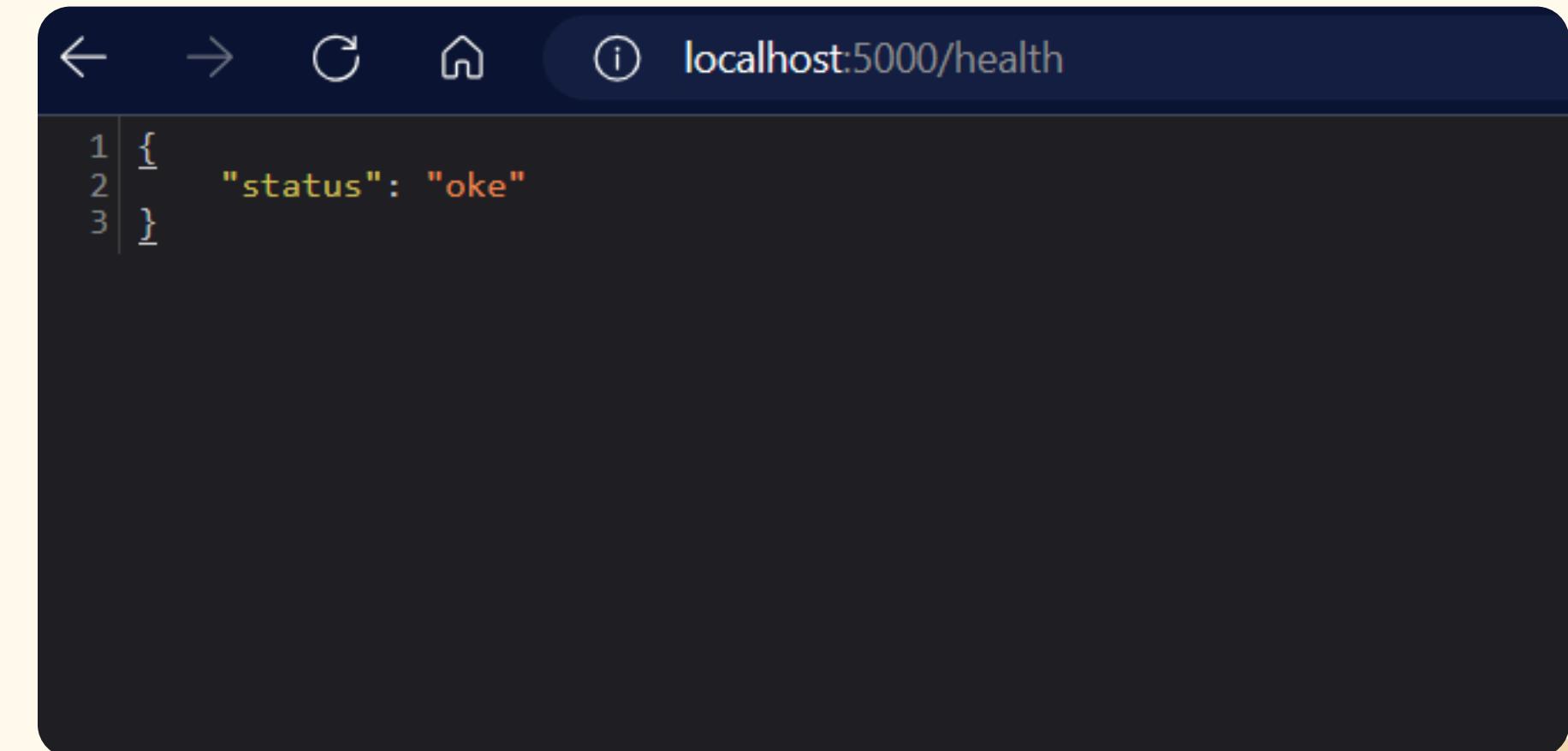
```
andrew@MSI:/mnt/d/Dokumen/TUGAS-TUGAS SIB DigitalSkola/TUGAS PROJEK (Individu + Kelompok)/PROJEK 5/PROJECT-5-SIB-DigitalSkola$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                 NAMES
5e59edf01ebf        project-5-sib-digitalskola   "python app/main.py ..."   12 seconds ago    Up 11 seconds   0.0.0.0:5000->5000/tcp   project-5-sib-digitalskola-flask-app-1
afb8131df8b5        postgres            "docker-entrypoint.s..."   12 seconds ago    Up 11 seconds   0.0.0.0:5432->5432/tcp   project-5-sib-digitalskola-postgres-db-1
```

STEP 9: Memeriksa Endpoint Koneksi /db_check dan /health pada localhost 5000



A screenshot of a browser window with the URL `localhost:5000/db_check`. The page displays a JSON response with the following content:

```
1 | {  
2 |     "db": "connected",  
3 |     "status": 200  
4 | }
```



A screenshot of a browser window with the URL `localhost:5000/health`. The page displays a JSON response with the following content:

```
1 | {  
2 |     "status": "oke"  
3 | }
```

STEP 10: Membuat DDL Pada Database Postgres di DBeaver

```
*<sample> Script-5 ×
CREATE SEQUENCE user_id_seq
    start with 1
    increment by 1
    no minvalue
    no maxvalue
    cache 1;
CREATE TABLE public.users (
    user_id int4 NOT NULL DEFAULT nextval('user_id_seq'::regclass),
    name varchar(100) NULL,
    city varchar(50) NULL,
    telp varchar(14) NULL,
    CONSTRAINT users_pkey PRIMARY KEY (user_id)
);
```

Membuat struktur database untuk menyimpan informasi pengguna dalam proyek Flask

Terdapat tabel baru dengan nama "users" yang berisikan kolom user_id, name, city, dan telp.

```
sample - 172.19.165.11:5432
Databases
  sample
    Schemas
      public
        Tables
          users
            Columns
              user_id (int4)
              name (varchar(100))
              city (varchar(50))
              telp (varchar(14))
```



STEP 11: Penambahan Endpoint CRUD dalam Aplikasi Flask

```
app > main.py > ...  
  
1  from flask import Flask, jsonify, request  
2  from decouple import config  
3  import psycopg2  
4  from flask_sqlalchemy import SQLAlchemy  
5  
6  
7  DB_URI = f"postgresql+psycopg2://{{config('MB_DB_USER')}}:{{config('MB_DB_PASS')}}@{{config('MB_DB_HOST')}}:{{str(config('MB_DB_PORT'))}}/{{config('MB_DB_DBNAME')}}"  
8  app = Flask(__name__)  
9  app.config["SQLALCHEMY_DATABASE_URI"] = DB_URI  
10 app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = True  
11 db = SQLAlchemy(app)  
12  
13  
14 class Users(db.Model):  
15     id = db.Column('user_id', db.Integer, primary_key=True)  
16     name = db.Column(db.String(100))  
17     city = db.Column(db.String(50))  
18     telp = db.Column(db.String(14))  
19  
20  
21 @app.route("/health")  
22 def health():  
23     return jsonify({"status": "oke"})  
24  
25  
26 @app.route("/db_check")  
27 def db_check():  
28     conn_pg = psycopg2.connect(  
29         host=config('MB_DB_HOST'),  
30         database=config('MB_DB_DBNAME'),  
31         user=config('MB_DB_USER'),  
32         password=config('MB_DB_PASS'),  
33         port=int(config('MB_DB_PORT')),  
34     )  
35     cur = conn_pg.cursor()  
36     return jsonify({"status": 200, "db": "connected"})  
37  
38  
39 @app.route("/user", methods=["GET", "POST", "PUT", "DELETE"])  
40 def user():  
41     if request.method == 'GET':  
42         users = Users.query.all()  
43         results = [{"id": u.id, "name": u.name, "city": u.city, "telp": u.telp} for u in users]  
44         return jsonify(results)  
45  
46  
47 if __name__ == "__main__":  
48     app.run(debug=True, host="0.0.0.0")
```

Menambahkan fitur-fitur CRUD yang memungkinkan aplikasi untuk melakukan operasi Create, Read, Update, dan Delete pada data pengguna dalam database PostgreSQL.

STEP 12: Menampilkan Hasil Permintaan API Flask pada Postman

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace', 'Collections', 'Environments', 'History', and a '+' button. The main area displays a collection named 'DigitalSkola: Basic API' with several endpoints: 'GET Get Users', 'GET Get User By Id', 'POST Add User', 'PUT Update User', and 'DEL Delete User'. A red arrow points from a callout bubble labeled 'Fungsi getUsers()' to the 'GET Get Users' endpoint. The 'GET Get Users' endpoint details are shown: Method 'GET', URL 'http://127.0.0.1:5000/user', and a 'Send' button. Below this, under the 'Body' tab, the 'form-data' option is selected, and a table shows two key-value pairs: 'Key' (id) and 'Value' (1). The 'Headers' tab shows five headers. At the bottom, the response status is '200 OK' with a response time of '8 ms' and a size of '356 B'. The response body is displayed in JSON format:

```
1 [  
2 {  
3   "city": "Solo",  
4   "id": 1,  
5   "name": "Kakang",  
6   "telp": "089762634343"  
7 },  
8 {  
9   "city": "Palembang",  
10  "id": 2,  
11  "name": "Althaf",  
12  "telp": "081234567890"  
13 }
```

At the very bottom, there are navigation icons for 'Postbot', 'Runner', 'Start Proxy', 'Cookies', 'Vault', 'Trash', and a question mark.

Fungsi addUsers()



Search Postman

Invite Upgrade

My Workspace New Import

Overview Getting sta GET Get User • POST Add User • GET Get User I • DEL Delete Use + No environment

Collections Environments History

DigitalSkola: Basic API

- GET Get Users
- GET Get User By Id
- POST Add User**
- PUT Update User
- DEL Delete User

HTTP DigitalSkola: Basic API / Add User

POST http://127.0.0.1:5000/user

Params Authorization Headers (8) Body Scripts Settings Cookies

Body (form-data)

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> name	Text	windah	
<input checked="" type="checkbox"/> city	Text	Bekasi	
<input checked="" type="checkbox"/> telp	Text	080987654321	

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "status": "ok"  
3 }
```

200 OK 28 ms 186 B Save as example

Online Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash ?

Fungsi getUserById()



The screenshot shows the Postman application interface. On the left, the sidebar displays 'My Workspace' with a collection named 'DigitalSkola: Basic API' containing five endpoints: 'Get Users', 'Get User By Id', 'Add User', 'Update User', and 'Delete User'. The 'Get User By Id' endpoint is selected. The main workspace shows an 'HTTP' request for 'DigitalSkola: Basic API / Get User By Id' with a 'GET' method and the URL 'http://127.0.0.1:5000/user/3'. Below the request, the 'Params' tab is active, showing a table for 'Query Params' with two columns: 'Key' and 'Value'. The 'Body' tab shows the JSON response from the API, which includes fields: 'city': 'Bekasi', 'id': 3, 'name': 'windah', and 'telp': '080987654321'. The status bar at the bottom indicates a 200 OK response with 15 ms latency and 245 B size.

User dengan
id = 3 di-get

200 OK 15 ms 245 B

```
1 {  
2   "city": "Bekasi",  
3   "id": 3,  
4   "name": "windah",  
5   "telp": "080987654321"  
6 }
```

Fungsi getUserById()



The screenshot shows the Postman interface with the following details:

- Collection:** DigitalSkola: Basic API
- Request:** GET /user/4
- Headers:** (6)
- Body:** (Pretty, Raw, Preview, Visualize, JSON)
- Response Status:** 200 OK
- Response Headers:** Content-Type: application/json; charset=utf-8
- Response Body:**

```
1 {  
2   "error": "id not found"  
3 }
```

A white arrow points from the text "User dengan id = 4 tidak ada sehingga sistem memberikan status error" to the "error" message in the response body.

User dengan id = 4
tidak ada sehingga
sistem memberikan
status error

Fungsi deleteUser()

The screenshot shows the Postman application interface. In the top left, there's a search bar labeled "Search Postman". The main workspace is titled "My Workspace" and contains a collection named "DigitalSkola: Basic API". This collection includes several endpoints: "GET Get Users", "GET Get User By Id", "POST Add User", "PUT Update User", and "DEL Delete User". The "DEL Delete User" endpoint is currently selected. The request details show a "DELETE" method and a URL "http://127.0.0.1:5000/user". The "Body" tab is selected, showing a single key-value pair: "user_id" with a value of "2". Below the body, the response status is "200 OK" with a response time of "26 ms" and a size of "186 B". The response body is displayed in JSON format: "status": "ok". A large white arrow points from the title "Fungsi deleteUser()" down to the "user_id" field in the Postman interface.

HTTP DigitalSkola: Basic API / Delete User

DELETE http://127.0.0.1:5000/user

Params Authorization Headers (8) Body Scripts Settings

None form-data x-www-form-urlencoded raw binary GraphQL

Key	Value	Description	Bulk Edit
user_id	2		

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "status": "ok"  
3 }
```

200 OK 26 ms 186 B Save as example

User dengan
id = 2 dihapus

The screenshot shows the Postman application interface. In the left sidebar, under 'My Workspace', there is a collection named 'DigitalSkola: Basic API' which contains several requests: 'Get Users', 'Get User By Id', 'Add User', 'Update User', and 'Delete User'. A tooltip with a curved arrow points from the text in the center to the 'Delete User' entry in the list.

The main workspace displays a 'Get Users' request. The method is 'GET' and the URL is 'http://127.0.0.1:5000/user'. The 'Body' tab is selected, showing the response body in JSON format:

```
1 [  
2 {  
3   "city": "Solo",  
4   "id": 1,  
5   "name": "Kakang",  
6   "telp": "089762634343"  
7 },  
8 {  
9   "city": "Bekasi",  
10  "id": 3,  
11  "name": "windah",  
12  "telp": "080987654321"  
13 }
```

The status bar at the bottom indicates a 200 OK response with 8 ms latency and 353 B size. There are also links for 'Postbot', 'Runner', 'Start Proxy', 'Cookies', 'Vault', 'Trash', and a help icon.

Ketika menjalankan fungsi getUsers(), user dengan id = 2 tidak ada karena telah dihapus

Fungsi updateUser()

The screenshot shows the Postman application interface. On the left, the sidebar displays 'My Workspace' with a collection named 'DigitalSkola: Basic API' containing five endpoints: 'GET Get Users', 'GET Get User By Id', 'POST Add User', 'PUT Update User' (which is selected), and 'DEL Delete User'. The main workspace shows a 'PUT DigitalSkola: Basic API / Update User' request. The 'Body' tab is selected, showing a form-data structure with four fields: 'name' (Text) set to 'Brando', 'city' (Text) set to 'Bekasi', 'telp' (Text) set to '089762634343', and 'user_id' (Text) set to '3'. Below the body, the response status is 200 OK with a response time of 33 ms and a size of 238 B. The response body is displayed in JSON format:

```
1 {  
2   "message": "User with ID 3 updated successfully",  
3   "status": "ok"  
4 }
```

A large white arrow points from the text 'User dengan id = 3 diupdate' at the bottom left towards the 'user_id' field in the Postman body table.

User dengan id = 3
diupdate

PUT DigitalSkola: Basic API / Update User

Params Authorization Headers (8) Body Scripts Settings Cookies

None form-data x-www-form-urlencoded raw binary GraphQL

	Key	Value	Description	... Bulk Edit
<input checked="" type="checkbox"/>	name	Text	Brando	
<input checked="" type="checkbox"/>	city	Text	Bekasi	
<input checked="" type="checkbox"/>	telp	Text	089762634343	
<input checked="" type="checkbox"/>	user_id	Text	3	

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

200 OK 33 ms 238 B Save as example

Online Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash ?

Ketika menjalankan fungsi getUserById() dengan user dengan id = 3, terlihat bahwa ada perubahan pada value kolom nama dan telp

HTTP DigitalSkola: Basic API / Get User By Id

GET http://127.0.0.1:5000/user/3

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	... Bulk Edit
	Key	Value	Description	

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "city": "Bekasi",  
3   "id": 3,  
4   "name": "Brando",  
5   "telp": "089762634343"  
6 }
```

200 OK 12 ms 245 B Save as example ...

Online Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash ?

Ketika menjalankan fungsi getUsers() terlihat perubahan pada value kolom nama dan telp user dengan id = 3

HTTP DigitalSkola: Basic API / Get Users

GET http://127.0.0.1:5000/user

Params Authorization Headers (6) Body Scripts Settings Cookies

None form-data x-www-form-urlencoded raw binary GraphQL

	Key	Value	Description	... Bulk Edit
	Key	Text	Value	Description

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 [  
2 {  
3   "city": "Solo",  
4   "id": 1,  
5   "name": "Kakang",  
6   "telp": "089762634343"  
7 },  
8 {  
9   "city": "Bekasi",  
10  "id": 3,  
11  "name": "Brando",  
12  "telp": "089762634343"  
13 }
```

Online Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash ?



Tampilan Endpoint Koneksi /user pada localhost 5000

The screenshot shows a web browser window with the URL `localhost:5000/user` in the address bar. The page content is a dark-themed JSON response:

```
1 [  
2   {  
3     "city": "Solo",  
4     "id": 1,  
5     "name": "Kakang",  
6     "telp": "089762634343"  
7   },  
8   {  
9     "city": "Bekasi",  
10    "id": 3,  
11    "name": "Brando",  
12    "telp": "089762634343"  
13  }  
14 ]
```

The browser interface includes a back button, forward button, refresh button, and a toolbar with various icons. The right side of the window features a vertical sidebar with icons for search, file, user, and other system functions.

TERIMAKASIH