

# 102- FASTAPI E2E

<input checked="" type="checkbox"/> Status	Not started
<input type="checkbox"/> Assign	
<input type="checkbox"/> Date	
@ Property	

## Create a Simple API

We are going to create a new api for posting an item , creating users and deleting them and authentication. we also allow user to like an post

## Lets Get Started

### Creating Virtual Environment

```
python3 -m venv venv

Activate the Env
source venv/bin/activate

Install fastapi
pip install fastapi[all]

#it will all the required packages
```

```
source /home/esak/esak_2022/learnapi_beginners/venv/bin/activate
esak@esakpc:~/esak_2022/learnapi_beginners$ source /home/esak/esak_2022/learnapi_beginners/venv/bin/activate
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ ls
LICENSE main.py README.md venv
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ source venv/bin/activate
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$
```

### Creating a Postgres user

```
sudo -u postgres psql
postgres=# create database mydb;
postgres=# create user myuser with encrypted password 'mypass';
postgres=# grant all privileges on database mydb to myuser;
```

Creating user, database and adding access on PostgreSQL

NOTE : Right off the bat - this is valid as on March 2017, running on Ubuntu 16.04.2, with PostgreSQL 9.6  
TL;DR; version sudo -u postgres psql|postgres=# create database mydb;postgres=# create user myuser with encrypted password 'mypass';postgres=# grant all privileges on database mydb to myuser; One nice thing about

<https://medium.com/coding-blocks/creating-user-database-and-adding-access-on-postgresql-8bfcd2f4a91e>



## Create a Simple FASTAPI Code

```
from fastapi import FastAPI

app = FastAPI() # create fastapi Instance

# create a Route
@app.get("/")
async def root():
    return {"message": "welcome to fastAPI"}

# here we are returning a Dictionary and fastapi will automatically convert it into an JSON
# You can also return Pydantic models (you'll see more about that later).
# There are many other objects and models that will be automatically converted to JSON (including ORMs, etc).
# when user hit the above / path , above code will be executed
# ORDER always matter
# if we have same path operation First one will be used
```

## Lets Run the Code

```
uvicorn main:app --reload

# main is the FileName
# app is the fastapi Object , if we declare the fasApi object in a different name we have to use that as well
```

We can view the Output using a tool called postman

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, APIs, Environments, Mock Servers, Monitors, and History. The main area has a 'Scratch Pad' tab selected. A central panel shows a 'GET' request to 'http://localhost:8000'. The 'Params' tab is active, showing a single parameter 'Key' with value 'Value'. Below the request, the 'Body' tab is selected, showing the JSON response: { "message": "welcome to fastAPI v2" }. The status bar at the bottom indicates a 200 OK response.

## Lets Create a Post

we are going to send some data to the web server using the HTTP method called POST

## Extracting Data from the body of a request

```
from fastapi import FastAPI
from fastapi.param_functions import Body

@app.post("/createposts")
def create_post(payload: dict = Body(...)):
    print(payload)
    return {"message": "Sucessfully created Post"}
```

```

Here
payload: dict = Body(...) Body will get all the Json value we passed through the Postman Tool and it will be converted back to Dictionary
and assigned to variable named payload
so when we print payload we can see {'title': 'Top 10 Languages in 2022', 'content': 'Check these Langiages'}

```

we can see the output of the Body

```

13
14     @app.post("/createposts")
15     def create_post(payload: dict = Body(...)):
16         print(payload)
17         return {"message": "Sucessfully created Post"}

```

```

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE    python3 + □ ^ ×
y']. Reloading...
INFO: Shutting down
INFO: Waiting for application shutdown.
INFO: Application shutdown complete.
INFO: Finished server process [13281]
INFO: Started server process [13283]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:36186 - "POST /createposts HTTP/1.1" 200 OK
INFO: 127.0.0.1:36194 - "GET /posts HTTP/1.1" 200 OK
WARNING: WatchGodReload detected file change in ['/home/esak/esak_2022/learnapi_beginners/main.p
y']. Reloading...
INFO: Shutting down
INFO: Waiting for application shutdown.
INFO: Application shutdown complete.
INFO: Finished server process [13283]
INFO: Started server process [13527]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:36226 - "POST /createposts HTTP/1.1" 422 Unprocessable Entity
{'title': 'Top 10 Languages in 2022', 'content': 'Check these Langiages'}
INFO: 127.0.0.1:36238 - "POST /createposts HTTP/1.1" 200 OK

```

Client can send any kind of data he wants we need to validate these data . we ultimately want the client to send data in a schema that we expect

All the data validation is performed under the hood by [Pydantic](#), so you get all the benefits from it. And you know you are in good hands.

# we can create an enum for our path parameters as well

```

from enum import Enum

from fastapi import FastAPI

class ModelName(str, Enum):
    alexnet = "alexnet"
    resnet = "resnet"
    lenet = "lenet"

app = FastAPI()

@app.get("/models/{model_name}")
async def get_model(model_name: ModelName):
    if model_name == ModelName.alexnet:
        return {"model_name": model_name, "message": "Deep Learning FTW!"}

```

```

if model_name.value == "lenet":
    return {"model_name": model_name, "message": "LeCNN all the images"}

return {"model_name": model_name, "message": "Have some residuals"}

```

When you declare other function parameters that are not part of the path parameters, they are automatically interpreted as "query" parameters.

```

@app.get("/items/")
async def read_item(skip: int = 0, limit: int = 10):
    return fake_items_db[skip : skip + limit]

in the URL it will be like
http://127.0.0.1:8000/items/?skip=0&limit=10

## The query is the set of key-value pairs that go after the ? in a URL, separated by & characters.

```

Lets Take another Example

```

@app.get("/items/{item_id}")
async def read_item(item_id: str, q: Optional[str] = None):
    if q:
        return {"item_id": item_id, "q": q}
    return {"item_id": item_id}

# Here item_id is a path parameter
# q is query parameter

```

we can perform additional validation in the Query Parameter

```

We are going to enforce that even though q is optional, whenever it is provided, its length doesn't exceed 50 characters.

from fastapi import FastAPI, Query
@app.get("/items/")
async def read_items(q: Optional[str] = Query(None, max_length=50)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results

# q: Optional[str] = None === q: Optional[str] = Query(None)
# Addition to the default value we have added a parameter to the query to check the max length < 50 or not

# if we want to declare the default value to others
async def read_items(q: str = Query("fixedquery", min_length=3)):

So, when you need to declare a value as required while using Query, you can use ... as the first argument:
async def read_items(q: str = Query(..., min_length=3)):

# if we use deprecated=True, in the Query , it will be reflected in the Docs

```

Like using Query Parameter we can also use the Path Parameter

```

@app.get("/items/{item_id}")
async def read_items(
    *,
    item_id: int = Path(..., title="The ID of the item to get", gt=0, le=1000),
    q: str,
):
    # here we are declaring the Path Parameter , which is required and its greater than 0 and less than 1000
    # we declare the query parameter

```

```
# Python will complain if you put a value with a "default" before a value that doesn't have a "default".
# To Overcome this Limitation we are using '*' which makes everything as kwargs after *
```

And you can also declare numeric validations:

```
gt: greater than
ge: greater than or equal
lt: less than
le: less than or equal
```

A **request** body is data sent by the client to your API. A **response** body is the data your API sends to the client.

To declare a **request** body, you use [Pydantic](#) models with all their power and benefits.

we can created Doc string while Creating the Function Itself

```
@app.post("/items/", response_model=Item, summary="Create an item")
async def create_item(item: Item):
    """
    Create an item with all the information:

    - **name**: each item must have a name
    - **description**: a long description
    - **price**: required
    - **tax**: if the item doesn't have tax, you can omit this
    - **tags**: a set of unique tag strings for this item
    """
    return item
```

## To make a Path as deprecated

```
@app.get("/elements/", tags=["items"], deprecated=True)
async def read_elements():
    return [{"item_id": "Foo"}]
```

## Schema validation with pydantic

We want the user to send only title : str , and content : str , anything else should be rejected

Lets Create a Pydantic Class , it is from the BaseModel Class

```
from pydantic import BaseModel

# Creating a Pydantic Class for Data Validation
class Post(BaseModel):
    title: str
    content: str
```

Lets Update the Route Code

```
@app.post("/createposts")
def create_post(new_post:Post):  # we are providing a parameter called new_post which is of type Post Pydantic Class
    print(new_post)
    print(type(new_post))
    return {"data": new_post} # we are returning the data in the Dict Format ,
when we check type of the data , it will be Post Class
title='Top 10 Languages in 2022' content='Check these Languages'
<class 'main.Post'>
```

To Extract Values from new\_post we have access the attributes using the dot operator

```
print(new_post.title)
```

Lets Check if we dont send the value using the postman , it will through the Error Message as below . it performs the validation automatically and throws the error

The screenshot shows a Postman interface with a POST request to `http://localhost:8000/createposts`. The Body tab is selected, showing a JSON payload:

```
1
2
3   "content": "Check these Languages"
4
```

The response tab shows the following details:

- Status: 422 Unprocessable Entity
- Time: 24 ms
- Size: 232 B

The error message in the response body is:

```
4   "loc": [
5     "body",
6     "title"
7   ],
8   "msg": "field required",
9   "type": "value_error.missing"
10 }
```

Lets Update our pydantic Class to include a optional Field , if the user choose the value , value will be used and if not default value will be used

```
# Creating a Pydantic Class for Data Validation
class Post(BaseModel):
    title: str
    content: str
    published: bool = True

# if user provided a value for published it will be used else default option will be used
```

The screenshot shows the Postman interface. At the top, it says "FastAPI / CreatePosts". Below that, a POST request is made to "http://localhost:8000/createposts". The "Body" tab is selected, showing a JSON payload:

```

1
2   ...
3     "title": "14",
4     "content": "Check these Languages",
5     "published": false

```

Below the body, the "Body" tab is selected again, showing the response body:

```

1
2   {
3     "data": {
4       "title": "14",
5       "content": "Check these Languages",
6       "published": false
7     }

```

At the bottom right, the status is shown as "Status: 200 OK Time: 8 ms Size: 200 B".

If the user didn't provide any data for a field we are going to make it as None

```
# Creating a Pydantic Class for Data Validation
class Post(BaseModel):
    title: str
    content: str
    published: bool = True
    rating: Optional[int] = None
```

Each Pydantic Model has a Method Called Dict

```
print(new_post.dict())
```

we can use the Field() in our pydantic model as well

```
class Item(BaseModel):
    name: str
    description: Optional[str] = Field(
        None, title="The description of the item", max_length=300
    )
    price: float = Field(..., gt=0, description="The price must be greater than zero")
    tax: Optional[float] = None

# Here description is a Field and we define additional metadata and length condition
```

we can declare the response model from the endpoint

```
# we are returning list of Item Objects
@app.get("/items/", response_model=List[Item])
async def read_items():
    return items

# we can have union of two Object Models
@app.get("/items/{item_id}", response_model=Union[PlaneItem, CarItem])
async def read_item(item_id: str):
```

```

    return items[item_id]

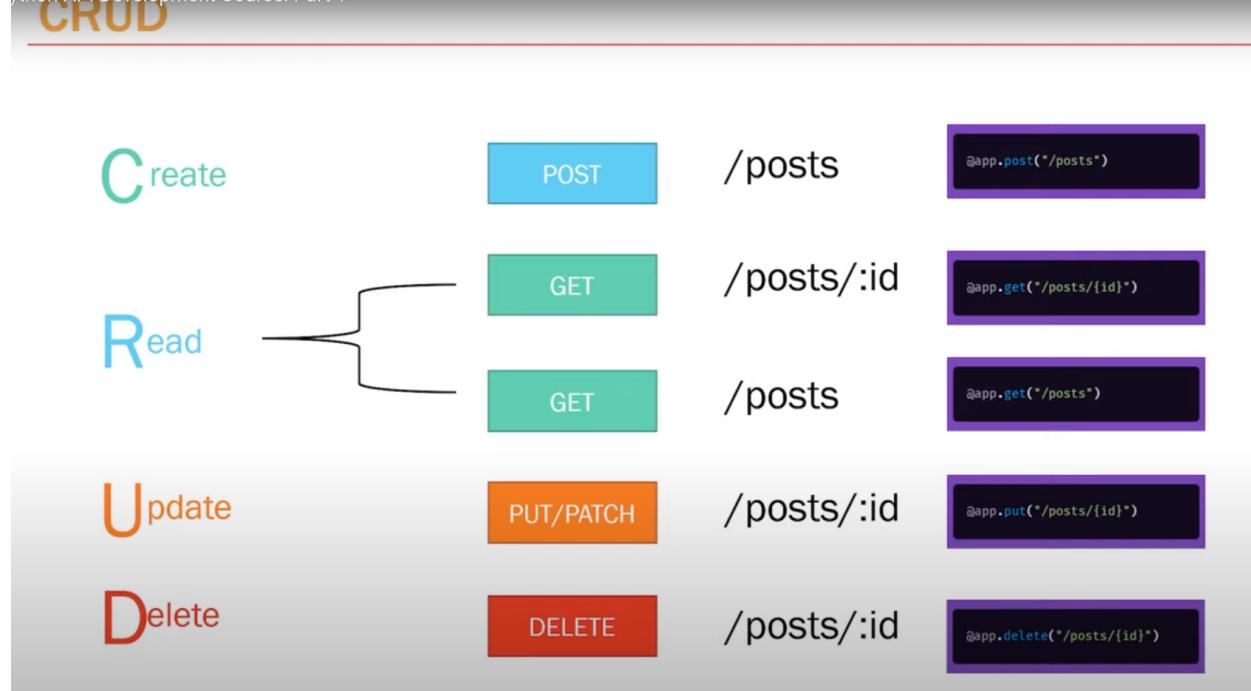
# it can Union or anyOf Union or anyOf

from fastapi import FastAPI, status
@app.post("/items/", status_code=status.HTTP_201_CREATED)
async def create_item(name: str):
    return {"name": name}

```

## Lets Perform CRUD Operation

Python API Development Course: Part 1



Always use Plurals that's the Standard Convention

Update - Put will change all the Fields

update - Patch will update only the specific Fields

We are storing all of our data in Memory Temporarily later we will be using the database

```

# Store Variables temporarily
my_posts = [
    {"title": "My Initial Post", "content": "My First Content", "id": 1}
]

@app.get("/posts")
def posts():
    return {"message": my_posts}

@app.post("/posts")
def create_post(post: Post):
    print(post)
    print(type(post))
    print(post.dict())
    post_dict = post.dict()
    post_dict['id'] = randrange(0, 100000)
    my_posts.append(post_dict)
    return {"data": post}

```

## lets Get a Single Post

```
def find_post(id):
    for post in my_posts:
        if post["id"] == id :
            return post

@app.get('/posts/{id}')
def get_post(id):
    print(id)
    post = find_post(int(id))
    return {"data": post}
```

instead of us converting to particular datatype we can give this to Fast Api

```
@app.get('/posts/{id}')
def get_post(id:int): # here we are telling FASTAPI we want ID as an integer , so fastapi will automatically convert the values when needed
    print(id)
    post = find_post(id)
    return {"data": post}
```

## Path Order Matters

```
@app.get('/posts/{id}')
@app.get('/posts/latest')

latest will never be executed and it always Throw the Error
So Careful on the Order
```

## Updating the Status Code

```
@app.get('/posts/{id}')
def get_post(id:int, response: Response):
    print(id)
    post = find_post(id)
    if not post:
        response.status_code = status.HTTP_404_NOT_FOUND
        return {"message": f"post with id {id} not Found"}
    return {"data": post}
```

Lets Update the Above Code using Fast API HTTP Methods

```
@app.get('/posts/{id}')
def get_post(id:int, response: Response):
    print(id)
    post = find_post(id)
    if not post:
        raise HTTPException(status_code=HTTP_404_NOT_FOUND, detail=f"post with id {id} is not Found")
    return {"data": post}
```

Here we are calling the FASTAPI HTTPException method , which makes the code more cleaner

Lets update the status code

```

@app.post("/posts", status_code=status.HTTP_201_CREATED)
def create_post(post:Post):
    print(post)
    print(type(post))
    print(post.dict())
    post_dict = post.dict()
    post_dict['id'] = randrange(0,100000)
    my_posts.append(post_dict)
    return {"data": post}

status_code=status.HTTP_201_CREATED ==> Using this we can update the status code to 201 which means we create a resource sucessfully

```

The screenshot shows a Postman interface with a POST request to `http://localhost:8000/posts ...`. The Body tab is selected, showing a JSON payload:

```

1
2   "title": 1421,
3   "content": "Check these Languages"
4
5

```

The response tab shows the result of the POST request:

Status: 201 Created Time: 7 ms

Body tab (Pretty view):

```

1
2   "data": {
3       "title": "1421",
4       "content": "Check these Languages",
5       "published": true,
6       "rating": null

```

## Delete

```

def find_post(id):
    for post in my_posts:
        if post["id"] == id :
            return post
def find_index_post(id):
    for index, post in enumerate(my_posts):
        if post['id'] == id:
            return index

@app.delete("/posts/{id}", status_code=status.HTTP_204_NO_CONTENT)
def delete_post(id: int):
    # find the index in the array that has the id
    index = find_index_post(id)
    # remove the Id from the array
    if index == None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail=f"id {id} doesnot exist")
    my_posts.pop(index)
    return Response(status_code=status.HTTP_204_NO_CONTENT)

```

## Update

```

@app.put("/posts/{id}")
def update_post(id:int, post:Post):
    print(post)
    index = find_index_post(id)
    if index == None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail=f"post with id {id} doesnot exists")

```

```

post_dict = post.dict()
post_dict['id'] = id
my_posts[index] = post_dict
return {'data': post_dict}

```

## Builtin Support For Documentation

The screenshot shows the FastAPI Swagger UI at [127.0.0.1:8000/docs](http://127.0.0.1:8000/docs). The main title is "FastAPI 0.1.0 OAS3". Below it, there's a link to "openapi.json". The "default" endpoint group is expanded, showing methods for /posts: GET (List Posts), POST (Create Post), GET (Get Post by ID), PUT (Update Post by ID), and DELETE (Delete Post by ID). Below the endpoints is a "Schemas" section containing definitions for HTTPValidationError, Post, and ValidationError.

The screenshot shows the detailed view for the "/posts" endpoint. It includes sections for "Parameters" (No parameters), "Responses", "Curl" command (curl -X 'GET' 'http://127.0.0.1:8000/posts' -H 'accept: application/json'), "Request URL" (http://127.0.0.1:8000/posts), and "Server response". The "Code" tab is selected, showing a 200 status code response body with JSON data representing two posts. The "Details" tab is also visible. At the bottom, there are "Response headers" with entries for content-length: 263 and content-type: application/json.

Lets Create a Folder app and move our main.py file over there

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

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
INFO: Shutting down
INFO: Waiting for application shutdown.
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ uvicorn app.main:app --reload
INFO: Will watch for changes in these directories: ['/home/esak/esak_2022/learnapi_beginners']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [10586] using watchdog
INFO: Started server process [10588]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

## Database

we are creating a sample table called products

The screenshot shows the pgAdmin interface for creating a new table. The 'Create - Table' dialog is open, and the 'Columns' tab is selected. The table structure is defined as follows:

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
<input checked="" type="checkbox"/>	product_id	bigint			<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> Yes
<input checked="" type="checkbox"/>	name	character varying			<input type="checkbox"/> No	<input type="checkbox"/> No
<input checked="" type="checkbox"/>	description	character varying			<input type="checkbox"/> No	<input type="checkbox"/> No
<input checked="" type="checkbox"/>	snapdate	date			<input type="checkbox"/> No	<input type="checkbox"/> No

At the bottom of the dialog, there are buttons for 'Cancel', 'Reset', and 'Save'. The 'Save' button is highlighted.

Equivalent Code

```
CREATE TABLE IF NOT EXISTS public.products
(
    id serial NOT NULL,      ----> For performing auto increment we have to use the serial
    name character varying,
    description character varying,
    snapdate date,
    price numeric,
    PRIMARY KEY (id)
);

ALTER TABLE public.products
OWNER to postgres;
```

lets add some data

	Id [PK] integer	name character varying	description character varying	snapdate date	price numeric
1	1	TV	Electronic Item	2021-11-18	200
2	3	Mobile	REdmi Mobile	2021-11-18	150
3	4	Mobile	Iphone	2021-11-18	200

Lets Add Another Column

Create - Column

General Definition Constraints Variables Security SQL

Default False

Not NULL? No

Type ✓ NONE IDENTITY GENERATED

⚠ Column type cannot be empty.

i ? Cancel Reset Save

	<b>Id</b> [PK] integer	<b>name</b> character varying	<b>description</b> character varying	<b>snapdate</b> date	<b>price</b> numeric	<b>is_sale</b> boolean
1	1	TV	Electronic Item	2021-11-18	200	false
2	3	Mobile	REdmi Mobile	2021-11-18	150	false
3	4	Mobile	Iphone	2021-11-18	200	false

Now the Constraint has been added sucessfully

we have added a timestamp field and we want postgress to fill it up . so we have added a default constraint called now()

Column	Type	Default
created_at	timestamp with time zone	now()

We can add the data using the SQL statement

```
INSERT INTO products(name, description, price, is_sale, inventory)
VALUES('Laptop acer', 'Acer Laptop', 234, 'true', 213)
```

we want our api to have the original Data , as of now postgress is returning 1 for sucess 0 for failure

```
INSERT INTO products(name, description, price, is_sale, inventory)
VALUES('Laptop Sony', 'Sony Laptop', 234, 'true', 3) returning * ;
```

Adding Multiple values

```
INSERT INTO products(name, description, price, is_sale, inventory)
VALUES ('Shoe', 'Reebok', 234, 'true', 30), ('shoe', 'Adidas', 300, 'true', 50), ('shoe', 'spark', 25, 'true', 200) returning * ;Update da
```

update

```
UPDATE products
SET description = 'spark - local', price = 30
WHERE id = 13 RETURNING *;
```

## Working With Database With Python

The screenshot shows a 'Create - Table' dialog box. The 'Columns' tab is selected, displaying five columns with the following properties:

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key
<input checked="" type="checkbox"/>	<input type="text" value="id"/>	serial			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	<input type="text" value="title"/>	character varying			<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	<input type="text" value="content"/>	character varying			<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	<input type="text" value="published"/>	boolean			<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	<input type="text" value="created_at"/>	timestamp with time zone			<input checked="" type="checkbox"/>	<input type="checkbox"/>

Below the columns, the 'Constraints' tab is active, showing:

- Default: now()
- Not NULL?: Yes

At the bottom right are buttons for Cancel, Reset, and Save.

Downloading and Installing Postgress driver

```
pip install psycopg2
```

To check the Connection

```

# Create Connection
while True:
    try:
        conn = psycopg2.connect(host ='localhost', database = 'learnfastapi',
                               user='postgres', password='esak@123', cursor_factory=RealDictCursor)
        cursor = conn.cursor()
        print("INFO: Database Connection was successfull")
        break
    except Exception as error:
        print(f"ERROR: {error}")
        print("ERROR: Connection FAILED")
        time.sleep(5)

```

Lets Retrieve all the data from the database

```

@app.get("/posts")
def posts():
    cursor.execute(""" SELECT * FROM posts""")
    posts = cursor.fetchall()
    return {"message": posts}

@app.post("/posts", status_code=status.HTTP_201_CREATED)
def create_post(post:Post):
    cursor.execute(""" INSERT INTO posts (title, content, published )
                      VALUES (%s, %s, %s ) RETURNING * """, (post.title, post.content, post.published))
    new_posts = cursor.fetchone()
    conn.commit()
    return {"data": new_posts}

@app.get('/posts/{id}')
def get_post(id:int):
    cursor.execute(""" SELECT * FROM POSTS WHERE id =  %s """, (str(id)))
    post = cursor.fetchone()
    if not post:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f"post with id {id} is not Found")
    return {"data": post}

@app.delete("/posts/{id}", status_code=status.HTTP_204_NO_CONTENT)
def delete_post(id: int):
    cursor.execute(""" DELETE FROM POSTS WHERE id = %s RETURNING * """, (str(id), ))
    post = cursor.fetchone()
    if post == None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail=f"id {id} doesnot exist")
    conn.commit()
    return Response(status_code=status.HTTP_204_NO_CONTENT)

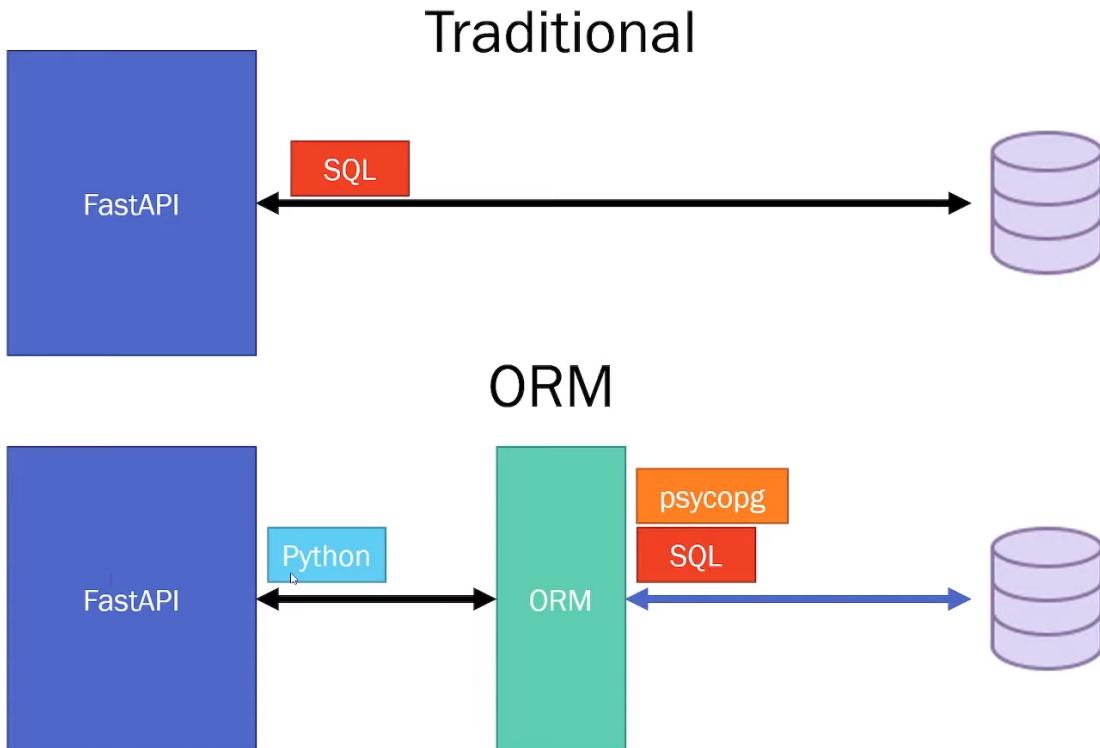
@app.put("/posts/{id}")
def update_post(id:int, post:Post):
    cursor.execute( """ UPDATE POSTS SET TITLE = %s , CONTENT=%s, PUBLISHED=%s
                      WHERE id = %s RETURNING * """,
                   (post.title, post.content, post.published, str(id)) )
    post = cursor.fetchone()

    if post == None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail=f"post with id {id} doesnot exists")
    conn.commit()
    return {'data': post}

```

## USING ORM

We perform all SQL operation using Python Code itself.



We can define Tables as python models

queries can be made exclusive through python code

sqlalchemy is one of the most popular ORM

## install SQLAlchemy

```
pip install sqlalchemy
```

```
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ pip install sqlalchemy
Collecting sqlalchemy
  Downloading SQLAlchemy-1.4.27-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.6 MB)
    |
    |██████████| 1.6 MB 5.2 MB/s
Collecting greenlet==0.4.17
  Downloading greenlet-1.1.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (153 kB)
    |
    |██████████| 153 kB 8.1 MB/s
Installing collected packages: greenlet, sqlalchemy
Successfully installed greenlet-1.1.2 sqlalchemy-1.4.27
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$
```

Lets Create a new File called database.py

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

SQLALCHEMY_DATABASE_URL = "postgresql://postgres:esak@123@localhost/learnfastapi"

engine = create_engine(SQLALCHEMY_DATABASE_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()
```

## Lets Define our models

```
from typing import ContextManager

from sqlalchemy.sql.expression import null
from sqlalchemy.sql.sqltypes import String
from .database import Base
from sqlalchemy import Column, Integer, String, Boolean
class Post(Base):
    __tablename__ = "posts"

    id = Column(Integer, primary_key=True, nullable=False)
    title = Column(String, nullable=False)
    content = Column(String, nullable=False)
    published = Column(Boolean, default=True)
```

`__tablename__ = "posts"` will be created in Postgress , this is the table name inside postgress  
all the models has to be extended from the BASE Class

Go to the `main.py` File and add the below line

```
models.Base.metadata.create_all(bind=engine)

this will create models in postgress automatically if the table is not exists
```

Lets Create a Dependency

Every Time we get the Request we are getting the Session Object and we issue the SQL Command and we close the Database Connection.

```
from . import models
from .database import SessionLocal, engine

# creating Dependency
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Lets Update the Path Parameters

```
@app.get('/sql')
def get_posts(db:Session = Depends(get_db)):
    return {'message': "Success"}
```

Lets Update Our published Column to Have default constraints as True . Add the Below Line to our Model

```
published = Column(Boolean, server_default='TRUE', nullable=False)
```

This wont Make any Changes to the Table in Postgress. Sqlalchemy is not intelligent enough to make changes to the database. So we have use another software called **alembic** . **For Doing Migrations we have to use the ALEMBIC**

Lets

Lets Lets Update the Path

```

@app.get('/api/v1/posts')
def get_posts(db:Session = Depends(get_db)):
    # lets Access the database Object
    posts = db.query(models.Post).all()
    return {'data': posts}

```

we are updating the Database using the ORM Models

```

@app.post('/api/v1/posts', status_code=status.HTTP_201_CREATED)
def create_posts(post:Post, db:Session = Depends(get_db)):
    # lets Access the database Object
    new_posts = models.Post(title= post.title, content=post.content, published=post.published)
    # lets add this Post to Database
    db.add(new_posts)
    # Lets Commit the changes
    db.commit()
    # Lets Return the Updated data
    db.refresh(new_posts)

    return {'data': new_posts}

```

If we have 50 Plus column in our database we cant go and edit one by one instead we can unpack the dict like below

```

new_posts = models.Post(**post.dict())

```

## CRUD Operation

```

@app.get('/api/v1/posts')
def get_posts(db:Session = Depends(get_db)):
    # lets Access the database Object
    posts = db.query(models.Post).all()
    return {'data': posts}

@app.post('/api/v1/posts', status_code=status.HTTP_201_CREATED)
def create_posts(post:Post , db:Session = Depends(get_db)):
    # lets Access the database Object
    #new_posts = models.Post(title= post.title, content=post.content, published=post.published)
    new_posts = models.Post(**post.dict())
    # lets add this Post to Database
    db.add(new_posts)
    # Lets Commit the changes
    db.commit()
    # Lets Return the Updated data
    db.refresh(new_posts)

    return {'data': new_posts}

@app.get("/api/v1/posts/{id}")
def get_post(id:int, db:Session = Depends(get_db)):
    post = db.query(models.Post).filter(models.Post.id == id).first()
    if post is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Post ID {id} is not created")
    return {"data": post}

@app.put("/api/v1/posts/{id}")
def update_post(id:int, post: Post, db:Session = Depends(get_db)):
    post_query = db.query(models.Post).filter(models.Post.id == id )
    dbpost = post_query.first()
    print(post)

    if dbpost is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail = "POST ID {id} is not Found")

    post_query.update(post.dict(), synchronize_session=False)
    #new_posts = models.Post(**post.dict())

    db.commit()

    return {"data": post_query.first()}

```

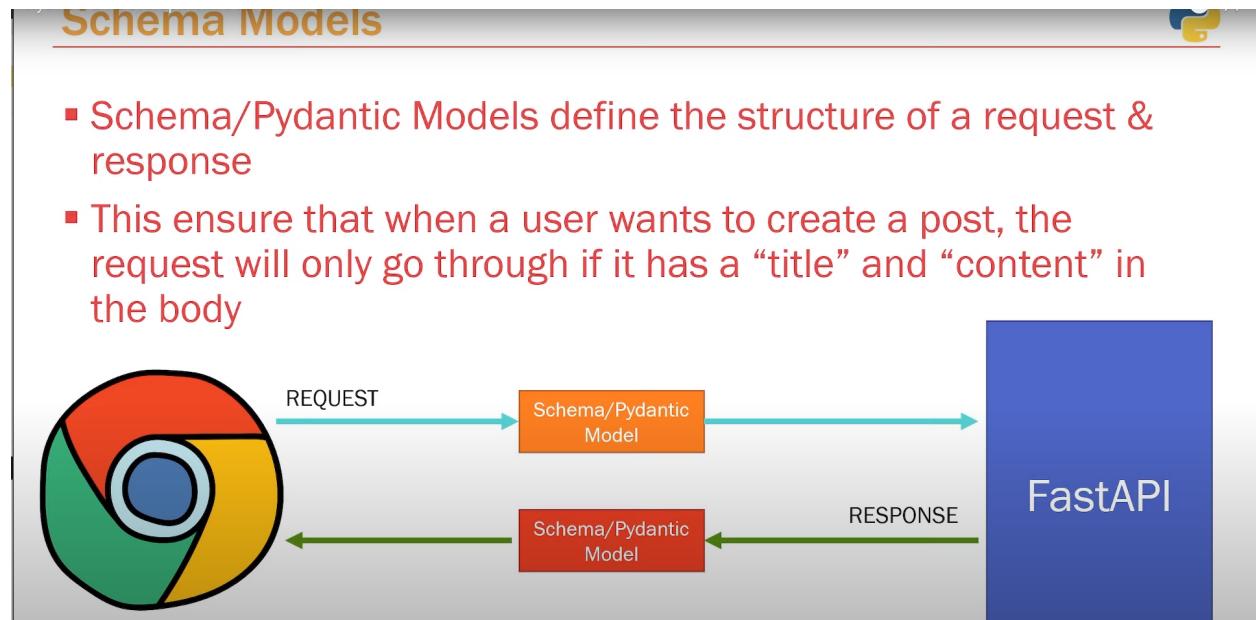
```

@app.delete("/api/v1/posts/{id}")
def delete_post(id:int, db:Session = Depends(get_db)):
    post = db.query(models.Post).filter( models.Post.id == id )
    if post.first() is None:
        raise HTTPException(status_code=HTTP_404_NOT_FOUND, detail= "POST ID {id} is not FOUND ")
    # delete the REcords
    post.delete(synchronize_session=False)
    db.commit()

    return {"data": "deleted"}

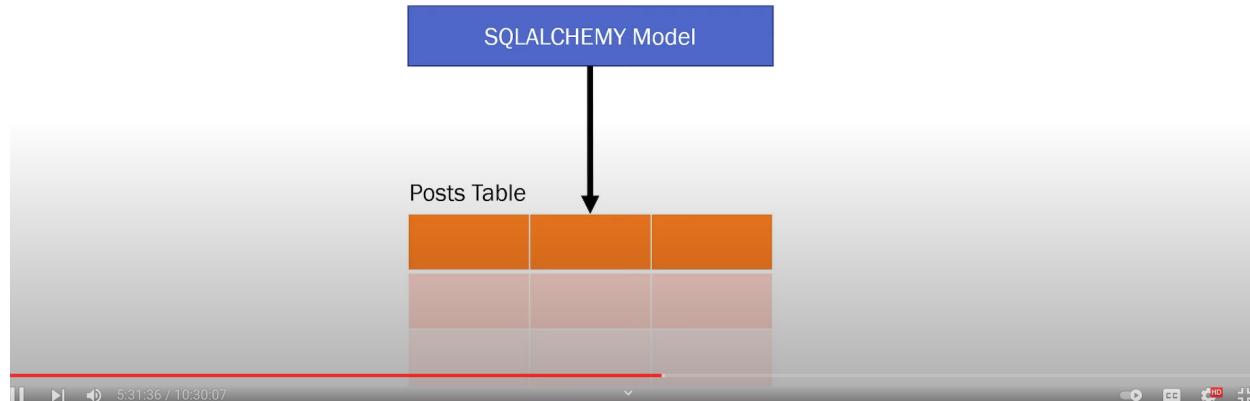
```

## ORM VS Schema Models





- Responsible for defining the columns of our “posts” table within postgres
- Is used to query, create, delete, and update entries within the database



## Pydantic Model Deep Dive

Lets Create a New File called schema.py.

```
class Post(BaseModel):
    title: str
    content: str
    published: bool = True
```

we can create Different Schema For Different Operation like update and create. Only the Fields satisfied the class property we can update it .

```
class PostBase(BaseModel):
    title: str
    content: str
    published: bool = True

class PostCreate(PostBase):
    pass
```

Lets Update the Schema for the response, we don't want to send all the data which returned from the table , we need to short it out.

### 1. Fastapi Convert the Dict into Json

so instead of return {"data":posts } we can rewrite its as return posts

```
@app.get("/api/v1/posts/{id}")
def get_post(id:int, db:Session = Depends(get_db)):
    post = db.query(models.Post).filter(models.Post.id == id).first()
    if post is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Post ID {id} is not created")
    return post
```

## 2.Handling the Response

The screenshot shows the Postman interface with a GET request to `http://localhost:8000/api/v1/posts...`. The Headers tab is selected, showing a table with one row labeled 'Key' and 'Value'. The Body tab is also selected, showing a JSON response with the following structure:

```
3   "published": true,
4   "title": "Esak Post 23",
5   "id": 3,
6   "created_at": "2021-11-22T13:01:43.522469+05:30",
7   "content": "Check these Languages"
8 }
```

I get all the details from the database .

let say we want to restrict it and we done need to display the ID and created\_at column

Create a new Schema Class for response

```
class PostResponse(BaseModel):
    title: str
    content:str
    published:bool

we will Get an Error saying like
pydantic.error_wrappers.ValidationError: 1 validation error for PostResponse
response
  value is not a valid dict (type=type_error.dict)

So we will update the PostResponse Class to have the below Code
Pydantic's orm_mode will tell the Pydantic model to read the data even if it is not a dict,
but an ORM model (or any other arbitrary object with attributes).
```

```
class PostResponse(BaseModel):
    title: str
    content:str
    published:bool

    class Config:
        orm_mode = True
```

Lets Update our Python Code to Handle the response Model

```
@app.post('/api/v1/posts', status_code=status.HTTP_201_CREATED, response_model=schemas.PostResponse)
def create_posts(post:schemas.PostCreate , db:Session = Depends(get_db)):
```

```

# lets Access the database Object
new_posts = models.Post(title= post.title, content=post.content, published=post.published)
new_posts = models.Post(**post.dict())
# lets add this Post to Database
db.add(new_posts)
# Lets Commit the changes
db.commit()
# Lets Return the Updated data
db.refresh(new_posts)

return new_posts

```

Now if we do a create response , our response will be restricted and we get what we have defined over the class

The screenshot shows the Postman interface for a POST request to `http://localhost:8000/api/v1/posts/`. The `Body` tab is active, displaying the following JSON payload:

```

1
2 ... "title": "My Journey Started with Python",
3 ... "content": "Check these Languages",
4 ... "published": true
5 ...
6

```

The response tab also displays the same JSON payload, indicating that the API only returns the fields specified in the schema.

We dont Get the Id and Created\_at Column as expected in our api response

Adding these Column Again and again is a painful task , so Lets Create a base class and we derive all the required or minimum field from it

```

All the required Fields
class PostBase(BaseModel):
    title: str
    content: str
    published: bool = True

Create the PostResponse Class
Here we add only the additional Fields

class PostResponse(PostBase):
    id: int
    created_at: datetime

    class Config:
        orm_mode = True

```

The screenshot shows a Postman interface with a POST request to `http://localhost:8000/api/v1/posts/`. The 'Body' tab is active, displaying a JSON payload:

```

1
2   ...
3     "title": "My Journey Started with Python3",
4     "content": "Check these Languages",
5     "published": true
6   ...

```

The response status is `201 Create`, and the response body is:

```

1
2   "title": "My Journey Started with Python3",
3   "content": "Check these Languages",
4   "published": true,
5   "id": 9,
6   "created_at": "2021-11-25T12:44:21.674721+05:30"

```

Lets add this response Model to the get path

```

@app.get('/api/v1/posts', response_model=schemas.PostResponse)
def get_posts(db:Session = Depends(get_db)):
    # lets Access the database Object
    posts = db.query(models.Post).all()
    return posts

we will be getting the Error Message
File "/home/esak/esak_2022/learnapi_beginners/venv/lib/python3.9/site-packages/fastapi/routing.py", line 137, in serialize_response
    raise ValidationError(errors, field.type_)
pydantic.error_wrappers.ValidationError: 4 validation errors for PostResponse
response -> title
  field required (type=value_error.missing)
response -> content
  field required (type=value_error.missing)
response -> id
  field required (type=value_error.missing)
response -> created_at
  field required (type=value_error.missing)
WARNING: WatchGodReload detected file change in ['/home/esak/esak_2022/learnapi_beginners/app/main.py']. Reloading...

```

If we understand the code we try to Pull a list of posts from the database . So we need to Mention this In our Response Model instead of saying it like a simple PostResponse , we have to define it as list of PostResponse schema

```

@app.get('/api/v1/posts', response_model=List[schemas.PostResponse])
def get_posts(db:Session = Depends(get_db)):
    # lets Access the database Object
    posts = db.query(models.Post).all()
    return posts

## response_model=List[schemas.PostResponse] => we are expecting the Response to be in this Format
## return posts ==> all the data is convert back to Json Automatically by the fast API

```

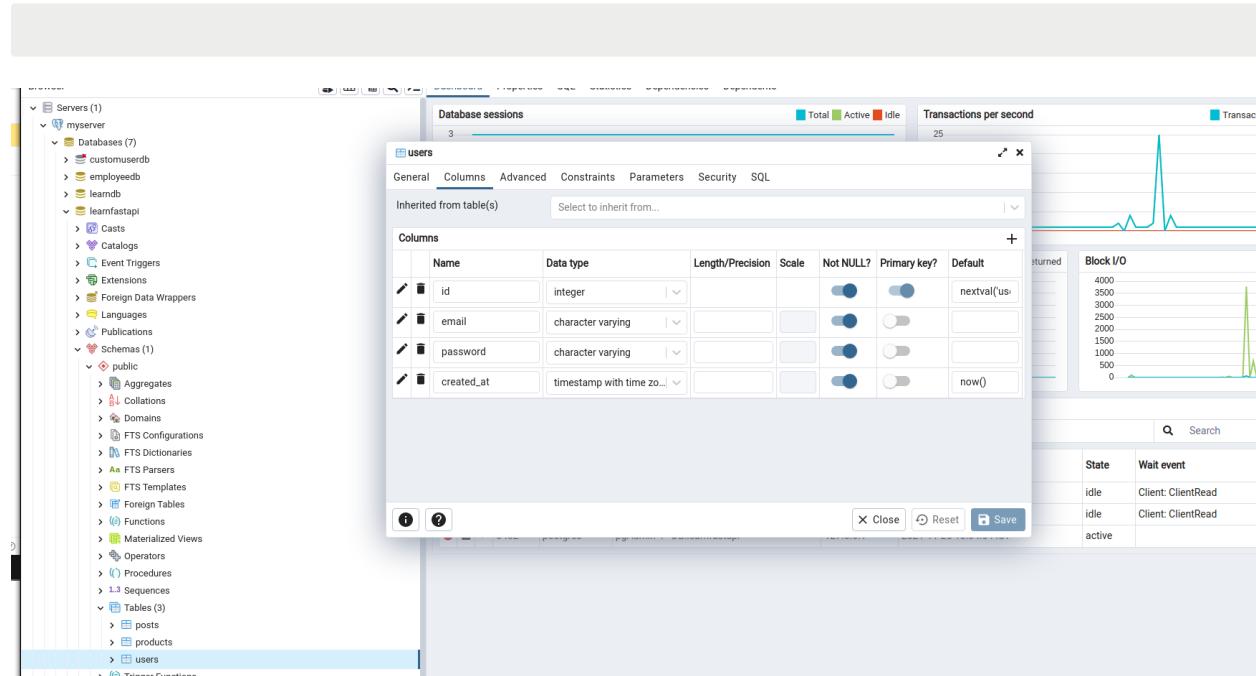
# Lets Add Users

Lets Create a new User database Model

```
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, nullable=False)
    email = Column(String, nullable=False, unique=True)
    password = Column(String, nullable=False)
    created_at = Column(TIMESTAMP(timezone=True), nullable=False, server_default=text('now()'))
```

Reload the Application and verify the Database



# Lets Create Path operation for user

Lets Create a Pydantic Schema

```
class UserCreate(BaseModel):
    email: EmailStr
    password: str

EmailStr --> will accept an Email , not any punch of Characters
```

Lets Create a new Response Model

```
class UserResponse(BaseModel):
    id: int
    email: EmailStr
    created_at: datetime

    class Config:
        orm_mode = True

@app.post("/api/v1/users", status_code=status.HTTP_201_CREATED, response_model=schemas.UserResponse)
def create_user(user: schemas.UserCreate, db: Session = Depends(get_db) ):
    new_user = models.User(**user.dict())
```

```

db.add(new_user)
db.commit()
db.refresh(new_user)
return new_user

```

POST http://localhost:8000/api/v1/users

Body (8) **Body** (selected)

JSON

```

1   "email": "santhosh123@gmail.com",
2   "password": "esak@123"
3
4

```

Body Cookies Headers (4) Test Results

Status: 201 Created

Pretty Raw Preview Visualize JSON

```

1   "id": 11,
2   "email": "santhosh123@gmail.com",
3   "created_at": "2021-11-25T13:18:24.489270+05:30"
4
5

```

## Hashing the Password

For hashing the Password we are going to use the Algorithm called Bcrypt and we are using the passlib library to that

```
pip install passlib[bcrypt]
```

```

INFO: Stopping reloader process [5207]
(venv) esak@esakpc:~/esak_2022/Learnapi_beginners$ pip install passlib[bcrypt]
Collecting passlib[bcrypt]
  Downloading passlib-1.7.4-py2.py3-none-any.whl (525 kB)
    |████████| 525 kB 5.9 MB/s
Collecting bcrypt>=3.1.0
  Downloading bcrypt-3.2.0-cp36-abi3-manylinux2010_x86_64.whl (63 kB)
    |████████| 63 kB 526 kB/s
Collecting cffi>=1.1
  Downloading cffi-1.15.0-cp39-cp39-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (444 kB)
    |████████| 444 kB 4.0 MB/s
Requirement already satisfied: six>=1.4.1 in ./venv/lib/python3.9/site-packages (from bcrypt>=3.1.0->passlib[bcrypt]) (1.16.0)
Collecting pycparser
  Downloading pycparser-2.21-py2.py3-none-any.whl (118 kB)
    |████████| 118 kB 10.2 MB/s
Installing collected packages: pycparser, cffi, passlib, bcrypt
Successfully installed bcrypt-3.2.0 cffi-1.15.0 passlib-1.7.4 pycparser-2.21
(venv) esak@esakpc:~/esak_2022/Learnapi_beginners$ 

```

Lets Create the User Password Hashed

```

from passlib.context import CryptContext
pwd_context = CryptContext(schemes=["bcrypt"], deprecated='auto')

@app.post("/api/v1/users", status_code=status.HTTP_201_CREATED, response_model=schemas.UserResponse)
def create_user(user: schemas.UserCreate, db: Session = Depends(get_db)):

    # Creating the Hashing Password
    hashed_password = pwd_context.hash(user.password)
    # Assign the Hashed Password to the User Object
    user.password = hashed_password
    new_user = models.User(**user.dict())

```

```
db.add(new_user)
db.commit()
db.refresh(new_user)
return new_user
```

Lets add a new user

	<b>id</b> [PK] integer	<b>email</b> character varying	<b>password</b> character varying	<b>created_at</b> timestamp with time zone
1	1	admin@gmail.com	admin@123	2021-11-25 13:06:08.250959+05:30
2	2	esak@gmail.com	esak@123	2021-11-25 13:06:29.616224+05:30
3	3	esak1@gmail.com	esak@123	2021-11-25 13:13:54.288403+05:30
4	5	esak121@gmail.com	esak@123	2021-11-25 13:15:02.795386+05:30
5	7	esak1221@gmail.com	esak@123	2021-11-25 13:16:08.445175+05:30
6	9	santhosh@gmail.com	esak@123	2021-11-25 13:17:50.123294+05:30
7	11	santhosh123@gmail.com	esak@123	2021-11-25 13:18:24.48927+05:30
8	12	santhosh123cre@gmail.com	\$2b\$12\$PaXyqj8A1...	2021-11-25 17:36:04.676784+05:30

Now the Password Is hashed

## Route For Fetching the User Information

```
@app.get("/api/v1/users/{id}", response_model=schemas.UserResponse)
def get_user(id:int, db:Session = Depends(get_db)):
    db_user = db.query(models.User).filter(models.User.id == id).first()
    print(db_user)
    if db_user is None:
        raise HTTPException(status_code=HTTP_404_NOT_FOUND, detail=f"User with Id - {id} Not Found")
    return db_user
```

## Routers

Lets Split our Path Operation and organize our Code little bit better

```
from fastapi import APIRouter

router = APIRouter()

Now we can call the path Parameters using the aouter Tag
@router.post("/api/v1/users", status_code=status.HTTP_201_CREATED, response_model=schemas.UserResponse)
def create_user(user: schemas.UserCreate, db:Session = Depends(get_db) ):

    # Creating the Hashing Password
    hashed_password = utils.hash(user.password)
    # Assign the Hashed Password to the User Object
    user.password = hashed_password
    new_user = models.User(**user.dict())
    db.add(new_user)
    db.commit()
    db.refresh(new_user)
    return new_user
```

Now add the Below Line to the main File

```
app.include_router(post.router)
app.include_router(user.router)

Once the Response Comes Here , first It will go to the POST File and look all the routes its has
```

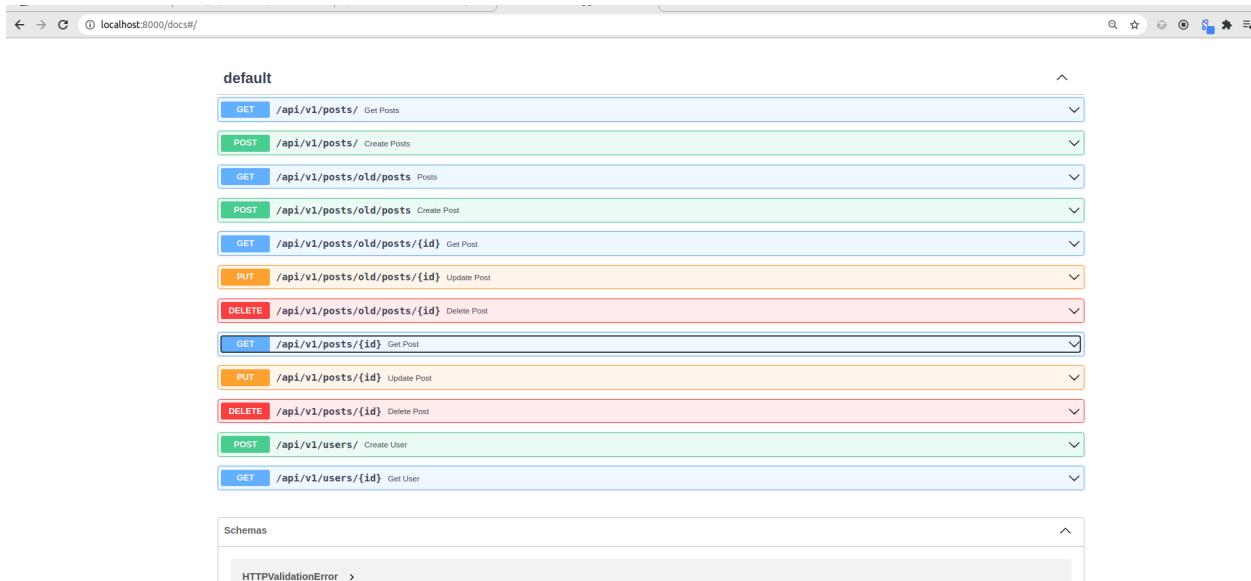
It will be really useful if we have lot of Routes to be add to our Folder

Lets Update the Router

```
"/api/v1/users"  ---> "/"
"/api/v1/users/{id}"  ---> "{id}"

For Doing this we have to Update the APIRouter() property
# Creating a Router
router = APIRouter(
    prefix='/api/v1/users'
)
```

we can check the Swagger Docs as below



we can Group routes based on the responsibility ,

we need to add the below Code

```
router = APIRouter(
    prefix='/api/v1/posts',
    tags= ['post']
)
```

FastAPI 0.1.0 OAS3

Openapi.json

post

- GET /api/v1/posts/ Get Posts
- POST /api/v1/posts/ Create Posts
- GET /api/v1/posts/old/posts Posts
- POST /api/v1/posts/old/posts Create Post
- GET /api/v1/posts/old/posts/{id} Get Post
- PUT /api/v1/posts/old/posts/{id} Update Post
- DELETE /api/v1/posts/old/posts/{id} Deletes Post
- GET /api/v1/posts/{id} Get Post
- PUT /api/v1/posts/{id} Update Post
- DELETE /api/v1/posts/{id} Delete Post

user

- POST /api/v1/users/ Create User
- GET /api/v1/users/{id} Get User

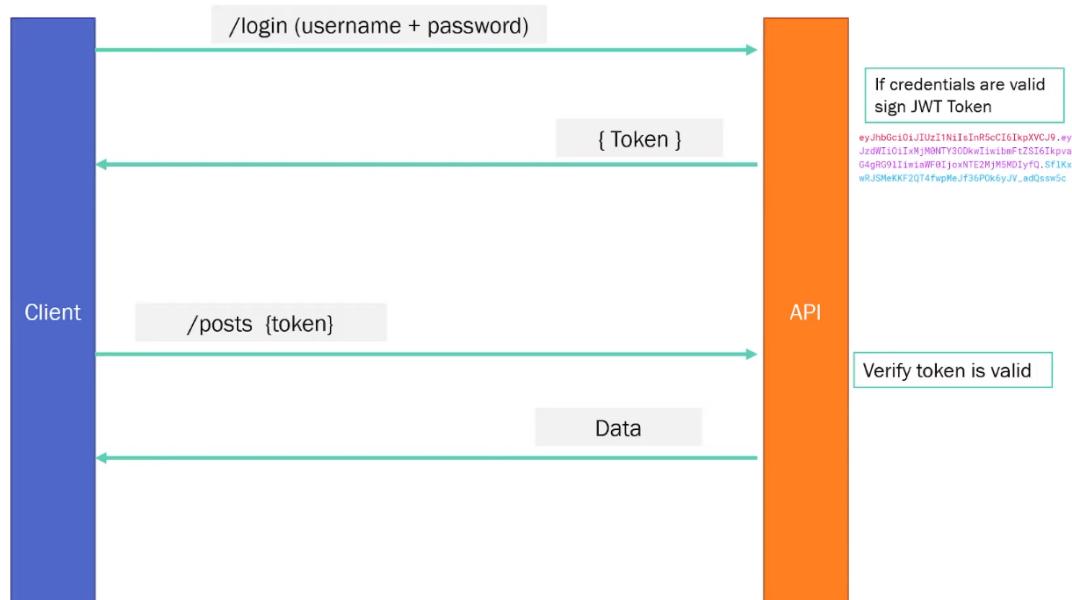
Schemas

## Authentication

we can Authentication in two ways

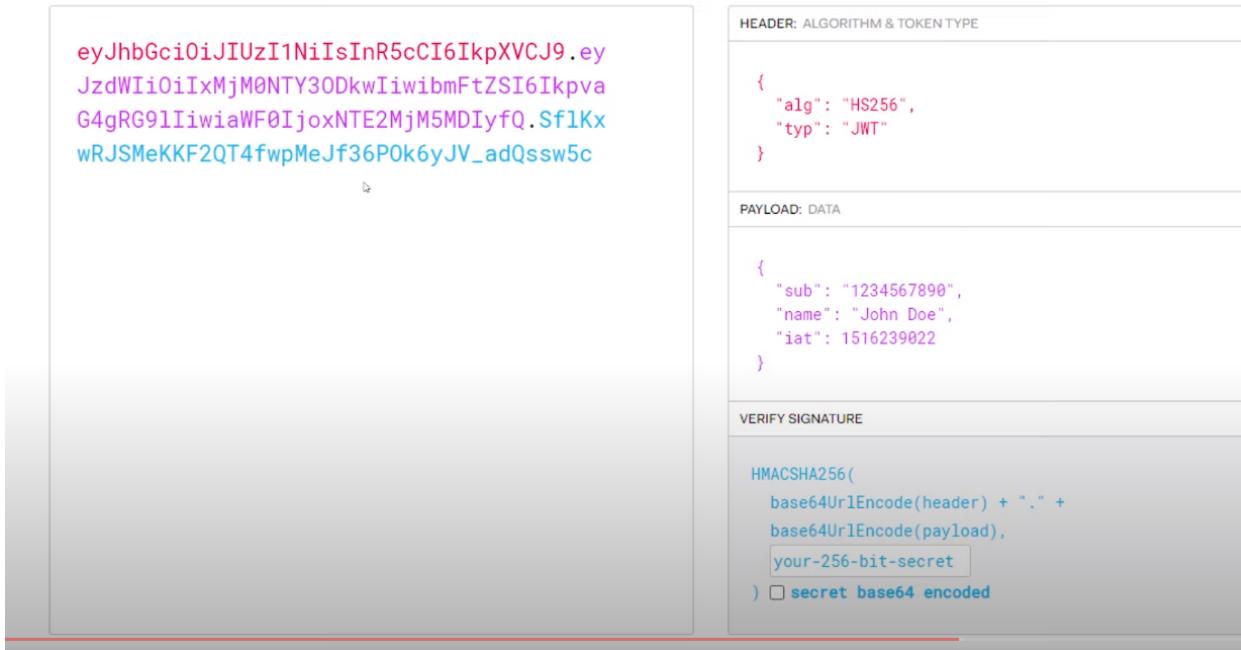
1. Session Based
2. Token based

### JWT Token Authentication



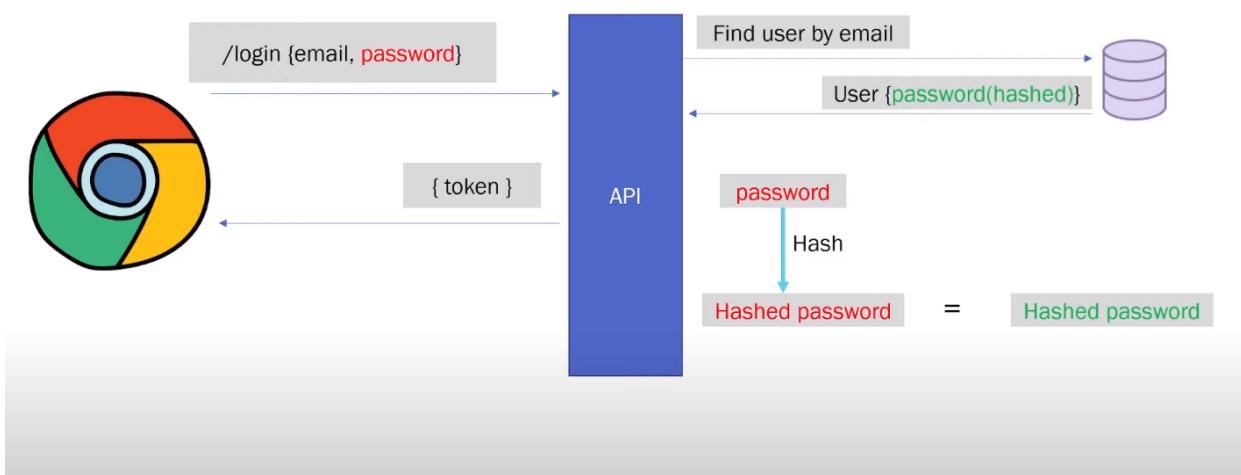
## JWT TOKEN

### JWT Deep Dive



## Logging Route

### Logging In User



### Lets Create a user Schema to Verify the request

```
# Defining the login Schema  
  
class UserLogin(BaseModel):  
    email: EmailStr  
    password: str
```

## Lets Create the login Routes

```
@router.post('/login')
def login(user_credentials: schemas.UserLogin , db:Session = Depends(get_db)):
    db_user = db.query(models.User).filter(models.User.email == user_credentials.email ).first()
    if db_user is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Invalid Credentials")

    # hasing the provided Password
    if not utils.verifyPassword(user_credentials.password, db_user.password):
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Invalid Credentials")

    # Create Token
    return {"Token": "Token"}
```

Lets Create the Token

## 1. Install python-jose

```
pip install python-jose[cryptography]
```

```
esak@esakpc:~/esak_2022/learnapi_beginners$ source /home/esak/esak_2022/learnapi_beginners/venv/bin/activate
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ pip install python-jose[cryptography]
Collecting python-jose[cryptography]
  Downloading python_jose-3.3.0-py3-none-any.whl (33 kB)
Collecting ecdsa!=0.15
  Downloading ecdsa-0.17.0-py2.py3-none-any.whl (19 kB)
Collecting rsa
  Downloading rsa-4.8-py3-none-any.whl (39 kB)
Collecting pyasn1
  Downloading pyasn1-0.4.8-py2.py3-none-any.whl (77 kB)
Collecting cryptography>=3.4.0
  Downloading cryptography-36.0.0-cp36-abi3-manylinux_2_24_x86_64.whl (3.6 MB)
Requirement already satisfied: cffi>=1.12 in ./venv/lib/python3.9/site-packages (from cryptography>=3.4.0->python-jose[cryptography]) (1.15.0)
Requirement already satisfied: pycparser in ./venv/lib/python3.9/site-packages (from cffi>=1.12->cryptography>=3.4.0->python-jose[cryptography]) (2.21)
Requirement already satisfied: six>=1.9.0 in ./venv/lib/python3.9/site-packages (from ecdsa!=0.15->python-jose[cryptography]) (1.16.0)
Installing collected packages: pyasn1, rsa, ecdsa, python-jose, cryptography
Successfully installed cryptography-36.0.0 ecdsa-0.17.0 pyasn1-0.4.8 python-jose-3.3.0 rsa-4.8
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$
```

Lets Create Function

```
from jose import JWTError, jwt
from datetime import datetime, timedelta

# SECRET KEY
# METADATA/ ALGORITHM
# EXPIRATION TIME

SECRET_KEY= "0258712wsdasdfesak12324374lopiutyyus"
ALGORITHM= "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES= 30

def create_access_token(data:dict):
    to_encode = data.copy()
    expire = datetime.now() + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})

    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

    return encoded_jwt
```

Lets Update the Login Route Path

```
@router.post('/login')
def login(user_credentials: schemas.UserLogin , db:Session = Depends(get_db)):
    db_user = db.query(models.User).filter(models.User.email == user_credentials.email ).first()
```

```

if db_user is None:
    raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Invalid Credentials")

# hasing the provided Password
if not utils.verifyPassword(user_credentials.password, db_user.password):
    raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Invalid Credentials")

# Create Token
access_token = oauth2.create_access_token(data = {"user_id":db_user.id})

return {"Token": access_token , "token_type": "bearer"}

```

Lets Check it

The screenshot shows a Postman request configuration for a POST method to the URL `http://localhost:8000/api/v1/auth/login`. The 'Body' tab is selected, showing a JSON payload with two fields: `"email": "esak@gmail.com"` and `"password": "esak@123"`. The response status is `200 OK`, with a response time of `538 ms` and a size of `281 B`. The response body is displayed in 'Pretty' format, showing the generated token and token type.

```

1
2   "Token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoyLCJleHAiOjE2Mzc4NzE0MzV9.yAEd65A1Lh_Fw0r9OnOSTpG5Uo5ZaLJb2-U2pq4MF00",
3   "token_type": "bearer"
4

```

Lets Verify the Token

The screenshot shows a JWT token being analyzed on jwt.io. The token is pasted into the 'Encoded' field:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoyLCJleHAiOjE2Mzc4NzE0MzV9.yAEd65A1Lh_Fw0r90n0STpG5Uo5ZaLJb2-U2pq4MF08
```

The token is decoded into three sections:

- HEADER:** ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- PAYOUT:** DATA

```
{
  "user_id": 2,
  "exp": 1637871435
}
```

- VERIFY SIGNATURE**

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) □ secret base64 encoded
```

A red error message '⊗ Invalid Signature' is displayed below the token fields.

## Lets Use FastApi Security Module

```
@router.post('/login')
def login(user_credentials: OAuth2PasswordRequestForm = Depends() , db:Session = Depends(get_db)):
    # By Default OAuth2PasswordRequestForm store the email in the Field called userName
    # if the user send email or username or anythin it stores in the username field
    # it Returns username, password
    db_user = db.query(models.User).filter(models.User.email == user_credentials.username ).first()
    if db_user is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Invalid Credentials")

    # hasing the provided Password
    if not utils.verifyPassword(user_credentials.password, db_user.password):
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Invalid Credentials")

    # Create Token
    access_token = oauth2.create_access_token(data = {"user_id":db_user.id})

    return {"Token": access_token , "token_type": "bearer"}
```

We dont send these Details using form-data

The screenshot shows the Postman interface with a POST request to `http://localhost:8000/api/v1/auth/login`. The 'Body' tab is selected, showing a form-data structure with two fields: 'username' (value: `esak@gmail.com`) and 'password' (value: `esak@123`). The status bar at the bottom indicates a `422 Unprocessable Entity` response.

To Create a New User we have to Login with Proper Credentials

```
@router.post("/", status_code=status.HTTP_201_CREATED, response_model=schemas.UserResponse)
def create_user(user: schemas.UserCreate, db: Session = Depends(get_db), get_current_user: int = Depends(oauth2.get_current_user)):
```

This Method depends on the `oauth2.get_current_user` method

```
token is coming from a request

def get_current_user(token: str = Depends(oauth2_scheme)):
    cred_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could Not validate Credentials",
        headers={"WWW-Authenticate": "bearer"}
    )

    return verify_access_token(token, cred_exception)
```

Lets Understand `verify_access_token`

```
def verify_access_token(token: str, cred_exception):
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        id: str = payload.get('user_id')
        if id is None:
            raise cred_exception

        token_data = schemas.TokenData(id=id)

    except JWTError as e:
        raise cred_exception

    return token_data
```

For Creating a Token

```
from fastapi import Depends, status
from fastapi.exceptions import HTTPException
from jose import JWTError, jwt
from datetime import datetime, timedelta
from . import schemas
from fastapi.security import OAuth2PasswordBearer

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="login")

# SECRET KEY
```

```

# METADATA/ ALGORITHM
# EXPIRATION TIME

SECRET_KEY= "0258712wsdasdfesak12324374lopiutyyus"
ALGORITHM= "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES= 30

def create_access_token(data:dict):
    to_encode = data.copy()
    expire = datetime.now() + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})

    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

    return encoded_jwt

```

POST <http://localhost:8000/api/v1/posts/>

Headers (9)

KEY	VALUE	DESCRIPTION	Bulk Edit
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9....		
Key	Value	Description	

Body Cookies Headers (5) Test Results

Status: 401 Unauthorized Time: 52 ms Size: 191 B Save

Pretty Raw Preview Visualize JSON

/code

```
Authorization Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.2323c4NTQ3Nzh9.zIPrYQH0p4coZc9o8fnaMjMmI4hoEddtQ9sdfsdfsdfsd
```

### Create Token schema

```

# Define the token

class Token(BaseModel):
    token: str
    token_type: str

class TokenData(BaseModel):
    id: Optional[str] = None

```

## Adding Env In Postman

Scratch Pad New Import

Overview POST Create POST Login POST Create DEL delete

DEV:FASTAPI

VARIABLE	INITIAL VALUE	CURRENT VALUE	Persist All	Reset All
<input checked="" type="checkbox"/> URL	http://127.0.0.1:8000	http://127.0.0.1:8000		
Add a new variable				

The screenshot shows the Postman interface with a yellow header bar. The main area displays a POST request to `((URL))/api/v1/posts/`. The request body is set to `JSON` and contains the following JSON:

```

1
2 "title": "My Journey Started with Python31.6",
3 "content": "Check these Languages Python",
4 "published": true,
5 "id": 14,
6 "created_at": "2021-11-25T21:20:04.619998+05:30"

```

The response status is `201 Created`. On the left sidebar, there's a tree view of collections, environments, and other API-related sections.

**When we create the Token using the Login Endpoint we want postman to automatically set the Environment Variable**

```
pm.environment.set("JWT", pm.response.json().Token);
```

The screenshot shows the Postman interface with a POST request to `((URL))/api/v1/auth/login`. The `Tests` tab contains the following JavaScript code:

```
1 pm.environment.set("JWT", pm.response.json().Token);
```

The response status is `200 OK`. The right sidebar shows snippets for environment variables and tokens. On the left sidebar, there's a tree view of collections, environments, and other API-related sections.

Setting this in path variables

The screenshot shows the Postman interface with the following details:

- Collection:** FastAPI
- Request:** POST /api/v1/posts/
- Authorization:** Bearer {{JWT}}
- Response Status:** 201 Created
- Response Body:**

```

1  {
2   "title": "My Journey Started with Python31.6",
3   "content": "Check these Languages Python",
4   "published": true,
5   "id": 15,
6   "created_at": "2021-11-25T21:26:38.395446+05:30"
    
```

## Setting up Relationships

id	email	password
101	kyle@gmail.com	\$2b\$12\$4Bpd
212	clay@gmail.com	d0C8oY1
378	mike@gmail.com	goWR2eK10

id	title	content	published
621	Top beaches	Random text	True
793	Sugar is bad	something	False
427	Favorite color	whatever	True

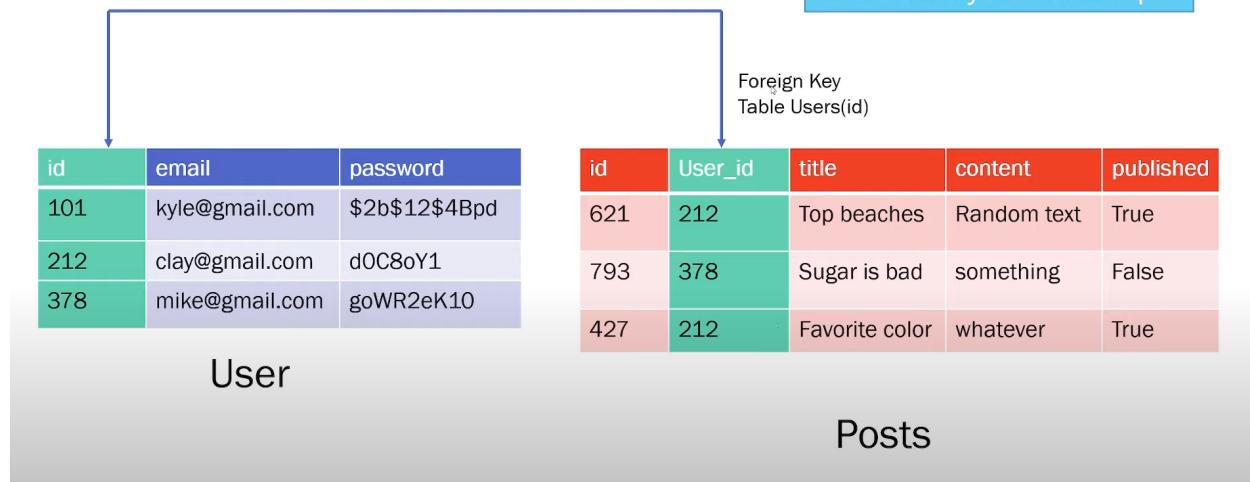
User

Posts

we are going to create a relationship between these 2 tables . we setup the Foreign Key



### One to Many Relationship



This is an Example of One To Many Relationship , one user can create many posts as they want

Lets Update the POST TABLE

The screenshot shows the "posts" table configuration screen. The "Constraints" tab is selected. Under the "Foreign Key" tab, a new constraint named "posts\_users\_fkey" is being defined. The "Columns" section shows a local column "user\_id" referencing a public.users table with a primary key "id". A red error message at the bottom states "Please specify columns for Foreign key." The "Action" section is visible below the columns.

we also need to Choose the action

CASCADE  $\Rightarrow$  So if a user is deleted all the respective posts also be deleted

Lets add these Column and Constraint using sqlalchemy code ,  
we have created a new Column named user\_id and we make it as a Foreign Key

```
class Post(Base):
    __tablename__ = "posts"

    id = Column(Integer, primary_key=True, nullable=False)
    user_id = Column(Integer, ForeignKey("users.id", ondelete="CASCADE" ), nullable=False )
    title = Column(String, nullable=False)
    content = Column(String, nullable=False)
    published = Column(Boolean, server_default='TRUE', nullable=False)
    created_at = Column( TIMESTAMP(timezone=True), nullable=False, server_default=text('now()'))
```

Lets update the schema

```
class PostResponse(PostBase):
    id: int
    created_at: datetime
    user_id: int

    class Config:
        orm_mode = True
```

Lets Update the Create Post

```
@router.post('/', status_code=status.HTTP_201_CREATED, response_model=schemas.PostResponse)
def create_posts(post:schemas.PostCreate , db:Session = Depends(get_db), current_user: int = Depends(oauth2.get_current_user)):
    print(current_user.email)
    print(current_user.id)
    # lets Access the database Object
    new_posts = models.Post(title= post.title, content=post.content, published=post.published)
    new_posts = models.Post(user_id =current_user.id , **post.dict())
    # lets add this Post to Database
    db.add(new_posts)
    # Lets Commit the changes
    db.commit()
    # Lets Return the Updated data
    db.refresh(new_posts)
```

```
    return new_posts
```

Lets Delete only the Post we have posted or he owns

```
@router.delete("/{id}")
def delete_post(id:int, db:Session = Depends(get_db), user:int = Depends(oauth2.get_current_user)):
    post_query = db.query(models.Post).filter( models.Post.id == id )
    # Above Db.query() will return a query Object
    # inorder to convert it to Post Object we have to use the first() method
    post = post_query.first()
    if post is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail= "POST ID {id} is not FOUND ")
    if post.user_id != user.id :
        raise HTTPException(status_code=HTTP_403_FORBIDDEN, detail="NOT AUTHORISED TO PERFORM DELETE THE POST")
    # delete the REcords
    post.delete(synchronize_session=False)
    db.commit()

    return {"data": "deleted"}
```

The screenshot shows the Postman interface with a DELETE request to `{{URL}}/api/v1/posts/5`. The Authorization tab is active, showing 'Type' set to 'Bearer Token'. A note says: 'Heads up! These parameters hold sensitive data. In your environment, we recommend using variables.' Below it, 'Token' is listed. The Body tab shows a JSON response with the following content:

```
1  {
2   |   "detail": "NOT AUTHORISED TO PERFORM DELETE THE POST"
3 }
```

Update the post

```
@router.put("/{id}", response_model=schemas.PostResponse)
def update_post(id:int, post: schemas.PostCreate, db:Session = Depends(get_db), user:int = Depends(oauth2.get_current_user)):
    post_query = db.query(models.Post).filter(models.Post.id == id )
    dbpost = post_query.first()
    print(post)

    if dbpost is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail = "POST ID {id} is not Found")

    if dbpost.user_id != user.id :
        raise HTTPException(status_code=HTTP_403_FORBIDDEN, detail="NOT AUTHORISED TO PERFORM DELETE THE POST")

    post_query.update(post.dict(), synchronize_session=False)
    #new_posts = models.Post(**post.dict())

    db.commit()

    return post_query.first()
```

## Return only the Posts Created by the user

```

@router.get('/myposts', response_model=List[schemas.PostResponse])
def get_posts(db:Session = Depends(get_db), current_user = Depends(oauth2.get_current_user)):
    # lets Access the database Object
    posts = db.query(models.Post).filter(models.Post.user_id == current_user.id ).all()
    return posts

```

We maintain the foreign key relationship with the user table , so we will get the id of the user but its not useful at the front end . we need to have the Username to be displayed in the Frontend. so we can achieve this using sql alchemy

```

class Post(Base):
    __tablename__ = "posts"

    id = Column(Integer, primary_key=True, nullable=False)
    user_id = Column(Integer, ForeignKey("users.id", ondelete="CASCADE" ), nullable=False )
    title = Column(String, nullable=False)
    content = Column(String, nullable=False)
    published = Column(Boolean, server_default='TRUE', nullable=False)
    created_at = Column( TIMESTAMP(timezone=True), nullable=False, server_default=text('now()'))

    owner = relationship("User") # it basically fetch the user based on the user_id

```

Lets Update the Pydantic Model

```

class PostResponse(PostBase):
    id: int
    created_at: datetime
    user_id: int
    owner: UserResponse

    class Config:
        orm_mode = True

```

The screenshot shows a Postman interface with the following details:

- Method:** GET
- URL:** {{URL}}/api/v1/posts/myposts
- Authorization:** Bearer Token (selected)
- Body:** (Pretty) JSON response (Status: 200 OK, Time: 13 ms)
- Response Data:**

```

4   "content": "Welcome to the application",
5   "published": true,
6   "id": 2,
7   "created_at": "2021-11-27T12:17:13.335504+05:30",
8   "user_id": 1,
9   "owner": {
10      "id": 1,
11      "email": "admin@gmail.com",
12      "created_at": "2021-11-25T19:33:18.851393+05:30"
13

```

## Query Parameters

Everything on the right of the Question mark is called the query parameters

we want our user to choose how many posts they want to see

```
@router.get('/', response_model=List[schemas.PostResponse])
def get_posts(db:Session = Depends(get_db), limit:int = 10 ):
    # Query Parameter
    print(limit)
    # lets Access the database Object
    posts = db.query(models.Post).limit(limit).all()
    return posts
```

For implementing Pagination in the Front end this skip parameter will be useful , First page we display 20 items and then in the second page we display next 20 items so we need to skip the first 20 when we displays the second page and so on

The screenshot shows a Postman interface. At the top, it says "GET {{URL}}/api/v1/posts?limit=1". Below that, the "Headers" tab is selected, showing "Content-Type: application/json". The "Body" tab is also visible. In the body section, there is a JSON object:

```
1   {
2     "title": "Welcom Post",
3     "content": "Welcome to the application",
4     "published": true,
5     "id": 2,
```

we have to provide the limit in the Function as an parameter

we want the user to search on some content , we have to search based on the title

```
{{URL}}/api/v1/posts?limit=5&search=day
if we want to search for words having spaces we have to use %20
{{URL}}/api/v1/posts?limit=5&search=day%20on

{{URL}}/api/v1/posts?search=aAdmin%20P0st
```

The screenshot shows a Postman interface. At the top, it says "GET {{URL}}/api/v1/posts?limit=5&search=day". Below that, the "Headers" tab is selected, showing "Content-Type: application/json". The "Body" tab is also visible. In the body section, there is a JSON object:

```
1   {
2     "title": "My Holiday Diary ",
3     "content": "First Tour",
4     "published": true,
5     "id": 5,
6     "created_at": "2021-11-27T12:32:49.635Z+05:30"
```

## Creating Environment Variables

Using pydantic we can have our env variables

```

from pydantic import BaseSettings

class Settings(BaseSettings):
    database_hostname: str = "localhost"
    database_port: str = "5432"
    database_password: str = "esak@123"
    database_name: str = "learnfastapi"
    database_username: str = "postgres"
    secret_key: str = "0258712wsdasdfesak12324374lopiutyus"
    algorithm: str = "HS256"
    access_token_expire_minutes: int = 30

    class Config:
        env_file = ".env"

settings = Settings()

```

we are setting the env variables in the class as default variables we can also provide it in the env file as below

```

class Settings(BaseSettings):
    database_hostname: str
    database_port: str
    database_password: str
    database_name: str
    database_username: str
    secret_key: str
    algorithm: str
    access_token_expire_minutes: int

    class Config:
        env_file = "app/.env"

settings = Settings()

```

and we need to add this env file to the calsss using the Config

```

DATABASE_HOST_NAME=localhost
DATABASE_PASSWORD=esak@123
DATABASE_PORT=5432
DATABASE_NAME=learnfastapi
DATABASE_USERNAME=postgres
SECRET_KEY=0258712wsdasdfesak12324374lopiutyus
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=30

```

## Build the Like/Voting System

1. User Should be able to like a post
2. should only be able to like a post once
3. retrieving posts should also fetch the number of likes for the post

## Vote Model



- Column referencing post id
- Column referencing id of user who liked the post
- A user should only be able to like a post once so this means we need to ensure every post\_id/voter\_id is a unique combination

Post_id	User_id
12	4
28	9
55	2
12	9
18	4
55	2

We cannot have a user like a post twice.  
Duplicate of row 3

## Composite Keys



- Primary Key that spans multiple columns
- Since Primary Keys must be unique, this will ensure no user can like a post twice

Primary Key

Post_id	User_id
12	4
28	9
55	2
12	9
18	4
55	2

Lets Create a new table votes

```
## Creating a new Table votes
class Vote(Base):
```

```

__tablename__ = "votes"

post_id = Column(Integer, ForeignKey('posts.id', ondelete='CASCADE'), primary_key=True) # refering the Posts Table Id Column
user_id = Column(Integer, ForeignKey('users.id', ondelete='CASCADE'), primary_key=True)

```

## Create Vote Route

lets create a new route

```

from fastapi import FastAPI, HTTPException, status, Response, Depends, APIRouter
from sqlalchemy.orm import Session
from .. import schemas, database, models, oauth2

router = APIRouter(
    prefix= "/api/v1/vote",
    tags= ['vote']
)

@router.post("/", status_code=status.HTTP_201_CREATED)
def vote(vote: schemas.Vote, db:Session = Depends(database.get_db), current_user: int = Depends(oauth2.get_current_user)):
    vote_query = db.query(models.Vote).filter(models.Vote.post_id == vote.post_id, models.Vote.user_id == current_user.id)
    found_vote = vote_query.first()
    if vote.direction == 1:
        print("upvoting")
        if found_vote:
            raise HTTPException(status_code=status.HTTP_409_CONFLICT, detail=f"User {current_user.id} has already voted on post {vote.post_id}")
        new_vote = models.Vote(post_id = vote.post_id, user_id = current_user.id)
        db.add(new_vote)
        db.commit()
        return {"message": "sucessfully added"}

    else:
        if not found_vote:
            raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail=f"Vote Doesnot Exists")
        vote_query.delete(synchronize_session=False)
        db.commit()
        return {"message": "sucessfully delete vote"}

```

The screenshot shows the Postman application interface. A POST request is being made to the URL `{{URL}}/api/v1/vote`. The 'Body' tab is active, displaying a JSON payload:

```

1
2   "post_id": 8,
3   "direction": 0
4

```

The response tab shows the following details:

- Status: 201 Created
- Body (Pretty): 

```

1
2   "message": "sucessfully delete vote"
3

```

**Now we want the aggregated Votes for a Post**

lets update the Code

```
@router.get('/', response_model=List[schemas.PostOut])
def get_posts(db:Session = Depends(get_db), limit:int = 10, skip:int = 0, search:Optional[str] = "" ):
    # Query Parameter
    print(limit)
    # lets Access the database Object
    posts = db.query(models.Post).filter(models.Post.title.contains(search)).limit(limit).offset(skip).all()
    # By default it will be left Inner Join
    #
    results = db.query(models.Post, func.count(models.Vote.post_id).label('votes')).join(models.Vote, models.Vote.post_id == models.Post.id)
    #print(results)
    return results
```

we have updated the get post Method

we have created a Left Outer Join

```
results = db.query
(models.Post.id, models.Post, func.count(models.Vote.post_id).label('votes'))
.join(models.Vote, models.Vote.post_id == models.Post.id, isouter=True).
group_by(models.Post.id).all()
```

For the Above Query Response will be like

```
{
    "id": 4,    --> we have select this Column in our query
    "Post": {   --> Name is Post because of Our Model Name is Post and all of its field are populated within the Double Quotes
        "title": "My Journey Started with Python31.6",
        "content": "Check these Languages Python",
        "published": true,
        "id": 4,
        "created_at": "2021-11-27T12:27:03.124094+05:30",
        "user_id": 2,
        "owner": {
            "id": 2,
            "email": "esak@gmail.com",
            "created_at": "2021-11-25T19:33:34.789176+05:30"
        }
    },
    "votes": 0   --> Another Column which we specified in the Query
},
```

Lets Update the Schema

```
# Schema class For Votes and post Join
class PostOut(BaseModel):
    Post: PostResponse
    votes: int

    class Config:
        orm_mode = True
```

Response of a Single Post

```
{
    "Post": {
        "title": "My Journey Started with Python31.6",
        "content": "Check these Languages Python",
        "published": true,
        "id": 4,
        "created_at": "2021-11-27T12:27:03.124094+05:30",
        "user_id": 2,
        "owner": {
            "id": 2,
            "email": "esak@gmail.com",
            "created_at": "2021-11-25T19:33:34.789176+05:30"
        }
    }
}
```

```
},
    "votes": 0
},
```

FastAPI / ListPosts

GET <{{URL}}/api/v1/posts?search=Admin%20Post>

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON ↻

```
1 [ ↴
2
3     "id": 4,
4     "Post": {
5         "title": "My Journey Started with Python31.6",
6         "content": "Check these Languages Python",
7         "published": true,
8         "id": 4,
9         "created_at": "2021-11-27T12:27:03.124094+05:30",
10        "user_id": 2,
11        "owner": {
12            "id": 2,
13            "email": "esak@gmail.com",
14            "created_at": "2021-11-25T19:33:34.789176+05:30"
15        }
16    },
17    "votes": 0
```

URL PATH WILL BE LIKE  
{{URL}}/api/v1/posts?search=Admin%20Post&limit=2&skip=4

## Database Migration

sqlalchemy has limitations to migrate the data , so we are going to use another tool called Alembic

- Developers can track changes to code and rollback code easily with GIT. Why can't we do the same for database models/tables
- Database migrations allow us to incrementally track changes to database schema and rollback changes to any point in time
- We will use a tool called Alembic to make changes to our database
- Alembic can also automatically pull database models from SQLAlchemy and generate the proper tables

```
pip install alembic
```

```
esak@esakpc:~/esak_2022/learnapi_beginners$ pip install alembic
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ pip install alembic
Collecting alembic
  Downloading alembic-1.7.5-py3-none-any.whl (209 kB)
    ━━━━━━━━━━ 209 kB 5.3 MB/s
Requirement already satisfied: SQLAlchemy>=1.3.0 in ./venv/lib/python3.9/site-packages (from alembic) (1.4.27)
Collecting Mako
  Downloading Mako-1.1.6-py2.py3-none-any.whl (75 kB)
    ━━━━━━━━━━ 75 kB 164 kB/s
Requirement already satisfied: greenlet<0.4.17 in ./venv/lib/python3.9/site-packages (from SQLAlchemy>=1.3.0->alembic) (1.1.2)
Requirement already satisfied: MarkupSafe>0.9.2 in ./venv/lib/python3.9/site-packages (from Mako>alembic) (2.0.1)
Installing collected packages: Mako, alembic
Successfully installed Mako-1.1.6 alembic-1.7.5
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$
```

alembic init alembic ==> Initiates a directory

it will create a alembic directory  
go to the env.py File and import the Base Class from our sqlalchemy models

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ alembic init alembic
Creating directory /home/esak/esak_2022/learnapi_beginners/alembic ... done
Creating directory /home/esak/esak_2022/learnapi_beginners/alembic/versions ... done
Generating /home/esak/esak_2022/learnapi_beginners/alembic/README ... done
Generating /home/esak/esak_2022/learnapi_beginners/alembic.ini ... done
Generating /home/esak/esak_2022/learnapi_beginners/alembic/env.py ... done
Generating /home/esak/esak_2022/learnapi_beginners/alembic/script.py.mako ... done
Please edit configuration/connection/logging settings in '/home/esak/esak_2022/learnapi_beginners/alembic.ini' before proceeding.
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$
```

Add the Following to the env file

```
from app.models import Base
target_metadata = Base.metadata
```

Go to Alembic.ini and update the postgres URI

```
sqlalchemy.url = postgresql+psycopg2://postgres:esak@123@localhost:5432//learnfastapi

Since it was hard coded we can override this env.py File

from app.models import Base
from app.config import settings
from urllib.parse import quote

# this is the Alembic Config object, which provides
# access to the values within the .ini file in use.

# Getting the Database URL
db_url = f'postgresql://{settings.database_username}:{settings.database_password}@{settings.database_hostname}/{settings.database_name}' % quote('esak@123')

config = context.config
config.set_main_option(
    "sqlalchemy.url" , db_url
)

)
```

Change the password

```
psql: FATAL:  role "esak" does not exist
Tesak@esakpc:~$ sudo -u postgres psql learnfastapi
[sudo] password for esak:
e could not change directory to "/home/esak": Permission denied
psql (14.1 (Ubuntu 14.1-1.pgdg21.04+1), server 13.5 (Ubuntu 13.5-1.pgdg21.04+1))
Type "help" for help.

learnfastapi=# ALTER ROLE postgres WITH esak@123 'esak123'
learnfastapi-#
```

## Using Models

we can create the Alembic Models manually

```
17
18
19 def upgrade():
20     op.create_table('posts',
21         sa.Column('id', sa.Integer(), nullable=False, primary_key=True), sa.Column('title', sa.String(), nullable=False),
22         sa.Column('content', sa.String(), nullable=False),
23         sa.Column('published', sa.Boolean(), nullable=False, server_default='TRUE'),
24         sa.Column('created_at', sa.TIMESTAMP(timezone=True), nullable=False, server_default=sa.text('NOW()'))
25     )
26
27
28
29
30 def downgrade():
31     op.drop_table('posts')
32
```

```
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ alembic revision -m "create post Table"
Generating /home/esak/esak_2022/learnapi_beginners/alembic/versions/c5b6cbc6060_create_post_table.py ... done
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$
```

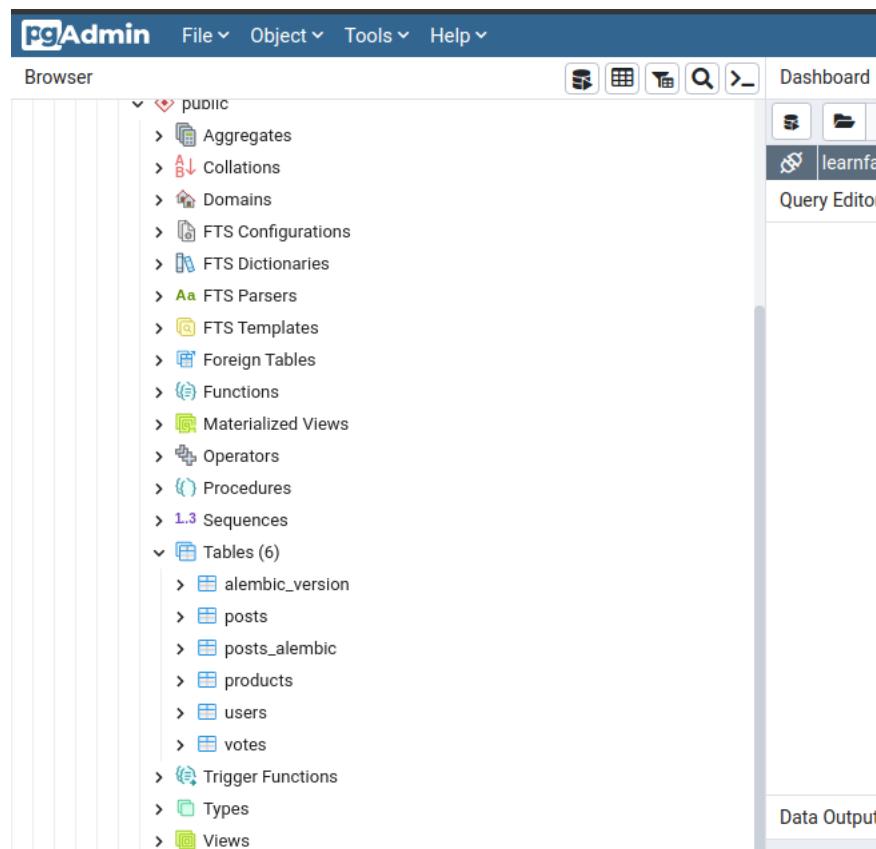
The screenshot shows the Visual Studio Code interface with several tabs open at the top: .env, config.py, post.py M, user.py, database.py, models.py, c5b6cbca6860\_create\_post\_table.py (active), alembic.ini U, and env.py U. The code editor displays Python migration code for Alembic:

```

File Edit Selection View Go Run Terminal Help
.env config.py post.py M user.py database.py models.py c5b6cbca6860_create_post_table.py alembic.ini U env.py U
alembic > versions > c5b6cbca6860_create_post_table.py downgrade
14     down_revision = None
15     branch_labels = None
16     depends_on = None
17
18 def upgrade():
19     op.create_table('posts_alembic',
20         sa.Column('id', sa.Integer(), nullable=False, primary_key=True), sa.Column('title', sa.String(), nullable=False),
21         sa.Column('content', sa.String(), nullable=False),
22         sa.Column('published', sa.Boolean(), nullable=False, server_default='TRUE'),
23         sa.Column('created_at', sa.TIMESTAMP(timezone=True), nullable=False, server_default=sa.text('NOW()'))
24     )
25
26
27
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
ValueError: invalid interpolation syntax in 'postgresql+psycopg2://postgres:esak340123@localhost:5432/learnfastapi' at position 35
(esak㉿esakpc:~/esak_2022/learnnapi_beginners) alembic current
esak%40123
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
(esenv) esak@esakpc:~/esak_2022/learnnapi_beginners$ alembic history
(esenv) esak@esakpc:~/esak_2022/learnnapi_beginners$ alembic upgrade head
esak%40123
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.runtime.migration] Running upgrade -> c5b6cbca6860, create post Table
(esenv) esak@esakpc:~/esak_2022/learnnapi_beginners$ 

```

The terminal window shows the command-line interaction for running migrations. The status bar at the bottom indicates the environment is 'main\*', Python version is '3.9.5 64-bit ('venv': venv)', and there are 0 errors.



it also support automatic Upgrades . alembic is intelligent enough to build the new additional added fields into the postgres table based on the models we have

```
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ alembic revision --autogenerate -m "add models to alembic"
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.ddl.postgresql] Detected sequence named 'products_id_seq' as owned by integer column 'products(id)', assuming SERIAL and omitting
INFO [alembic.ddl.postgresql] Detected removed table 'products'
INFO [alembic.ddl.postgresql] Detected sequence named 'posts_alembic_id_seq' as owned by integer column 'posts_alembic(id)', assuming SERIAL and omitting
INFO [alembic.ddl.postgresql] Detected removed table 'posts_alembic'
Generating /home/esak/esak_2022/learnapi_beginners/alembic/versions/542240f1f283_add_models_to_alembic.py ... done
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$
```

| IF we have any new Tables apart From Schema it will be deleted when we execute this Command

## Lets add a New Column to user Model

```
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, nullable=False)
    email = Column(String, nullable=False, unique=True)
    password = Column(String, nullable=False)
    phononenumber=Column(String)
    created_at = Column(TIMESTAMP(timezone=True), nullable=False, server_default=text('now()'))
```

I have added the New Column called phone Number

lets Run Alembic

```
alembic revision --autogenerate -m "update user Model"
alembic upgrade head
```

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
id	integer			On	On	nextval('us...')
email	character varying			On	Off	
password	character varying			On	Off	
created_at	timestamp with time zone			On	Off	now()
phononenumber	character varying			Off	Off	

So Far we used postman to send the request . request can come from anywhere including the Frontend application in JS from a webrowser

When we send the request from the Browser we get the below error

The screenshot shows the Chrome DevTools Console tab. The error message is:

```
> fetch('http://localhost:8000').then(res => res.json()).then(console.log)
< Promise {<pending>}
  × GET http://localhost:8000/ net::ERR_CONNECTION_REFUSED
  × Uncaught (in promise) TypeError: Failed to fetch
    at <anonymous>:1:1
    (anonymous) @ VM423:1
Promise.then (async)
(anonymous) @ VM423:1
```

Details: VM423:1 VM423:1

## python API Development Course: Part 2

### CORS



- Cross Origin Resource Sharing(CORS) allows you to make requests from a web browser on one domain to a server on a different domain
- By default our API will only allow web browsers running on the same domain as our server to make requests to it



#### Update the Middleware Settings

we have to update the domains inside the origin so that these domain can communicate with each other

```
from fastapi import FastAPI
from .database import engine
from . import models
from .routers import post, user, auth, vote
from .config import settings
# setting up Cors
from fastapi.middleware.cors import CORSMiddleware

print(settings.database_username)

# Creating the SQLALCHEMY Connection
# Since we have the Alembic Now we dont need the below line
#models.Base.metadata.create_all(bind=engine)

# Creating the FASTAPI APP
app = FastAPI()

# ADDING Middleware
```

```

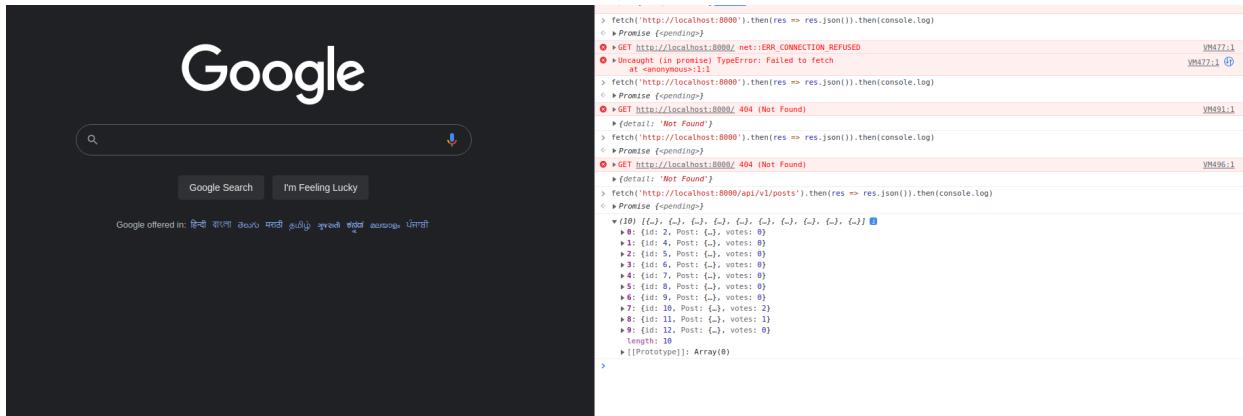
origins = [
    "http://localhost",
    "http://localhost:8080",
    "https://www.google.com"
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# ADDING ROUTER ROUTES
app.include_router(post.router)
app.include_router(user.router)
app.include_router(auth.router)
app.include_router(vote.router)

```

After that [google.com](https://www.google.com) domain can send request to our API



## Deployment

### 1. Setup Ubuntu machine

Create an Ec2 Machine in AWS (ubuntu) and execute the Following Commands

```

sudo apt update && sudo apt upgrade
sudo apt install python3-pip -y
sudo pip3 install virtualenv
sudo apt install postgresql postgresql-contrib -y
sudo apt-get install python3-virtualenv
apt install python3.8-venv

vi .env
set -o allexport; source /home/ubuntu/.env; set +o allexport;

update this to .profile File

```

```
ubuntu@ip-172-31-85-183:~$ psql -U postgres
psql: error: FATAL:  Peer authentication failed for user "postgres"
ubuntu@ip-172-31-85-183:~$
```

we need to login as postgress user and we can connect to it

```
sudo -i -u postgres

# logg into database
psql

# change Password
\password
esak123

# to Exit the postgress
\q
```

```
ubuntu@ip-172-31-85-183:/etc/postgresql/12/main$ ls
conf.d  environment pg_ctl.conf pg_hba.conf pg_ident.conf postgresql.conf start.conf
ubuntu@ip-172-31-85-183:/etc/postgresql/12/main$ sudo vi postgresql.conf
ubuntu@ip-172-31-85-183:/etc/postgresql/12/main$ sudo vi pg_hba.conf
ubuntu@ip-172-31-85-183:/etc/postgresql/12/main$ pwd
/etc/postgresql/12/main
ubuntu@ip-172-31-85-183:/etc/postgresql/12/main$
```

By Default postgres uses peer authentication , using that we cant connect with pgadmin so we have updated peer to md5

```
# DO NOT DISABLE!
# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local    all            postgres                                md5
#
# TYPE  DATABASE        USER        ADDRESS             METHOD
#
# "local" is for Unix domain socket connections only
local    all            all                                     md5
# IPv4 local connections:
host     all            all          0.0.0.0/0            md5
# IPv6 local connections:
host     all            all          ::/0                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication   all                                     peer
host   replication   all          127.0.0.1/32          md5
host   replication   all          ::1/128              md5
```

After changing the Configuration restart the Application

```
sudo systemctl restart postgresql  
# Now Log in  
psql -U postgres
```

Create a Non Root User

```
sudo adduser apiuser  
su - apiuser
```

```
apiuser: only root may add a user or group to the system.  
ubuntu@ip-172-31-85-183:~$ sudo adduser apiuser  
Adding user `apiuser' ...  
Adding new group `apiuser' (1001) ...  
Adding new user `apiuser' (1001) with group `apiuser' ...  
Creating home directory `/home/apiuser' ...  
Copying files from `/etc/skel' ...  
New password:  
Retype new password:  
passwd: password updated successfully  
Changing the user information for apiuser  
Enter the new value, or press ENTER for the default  
    Full Name []: apiuser  
    Room Number []: apiuser  
    Work Phone []: apiuser  
    Home Phone []: apiuser  
    Other []: apiuser  
Is the information correct? [Y/n] y  
ubuntu@ip-172-31-85-183:~$ su - apiuser  
Password:  
apiuser@ip-172-31-85-183:~$
```

Give sudo access

```
sudo usermod -aG sudo apiuser
```

Clone the Git Repo

```
git clone reponame .  
username esakkil2021  
password token
```

```
(venv) apiuser@apiuser@ip-172-31-85-183:~/app/src$ pwd
/home/apiuser/app/src
(venv) apiuser@apiuser@ip-172-31-85-183:~/app/src$ cd ..
(venv) apiuser@apiuser@ip-172-31-85-183:~/app$ 
(venv) apiuser@apiuser@ip-172-31-85-183:~/app$ ls -lrt
total 8
drwxrwxr-x 4 apiuser apiuser 4096 Dec  5 06:00 venv
drwxrwxr-x 5 apiuser apiuser 4096 Dec  5 06:06 src
(venv) apiuser@apiuser@ip-172-31-85-183:~/app$ cd src/
(venv) apiuser@apiuser@ip-172-31-85-183:~/app/src$ pip install -r requirements.txt
```

```
apiuser@apiuser@ip-172-31-85-183:~/app/src$ sudo apt install libpq-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  postgresql-doc-12
The following NEW packages will be installed:
  libpq-dev
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 136 kB of archives.
After this operation, 589 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 libpq-dev amd64 12.9-0ubuntu0.20.04.1 [136 kB]
Fetched 136 kB in 0s (6303 kB/s)
Selecting previously unselected package libpq-dev.
(Reading database ... 103264 files and directories currently installed.)
Preparing to unpack .../libpq-dev_12.9-0ubuntu0.20.04.1_amd64.deb ...
Unpacking libpq-dev (12.9-0ubuntu0.20.04.1) ...
Setting up libpq-dev (12.9-0ubuntu0.20.04.1) ...
Processing triggers for man-db (2.9.1-1) ...
apiuser@apiuser@ip-172-31-85-183:~/app/src$
```

## Start the application

```
uvicorn app.main:app
```

## Create RDS Instance

Update the Security group to include the 5432 port

Inbound rules (2)										
<input type="text" value="Filter security group rules"/> <span style="float: right;">Run Reachability Analyzer</span>										
	Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description	Manage tags	Edit inbound rules
<input type="checkbox"/>	-	sgr-0e70462bee7a341...	IPv4	PostgreSQL	TCP	5432	0.0.0.0/0	-		
<input type="checkbox"/>	-	sgr-07d5c5ce1a64a427f	-	All traffic	All	All	sg-9b7da4ac / default	-		

Verify that we can connect with pgadmin

## Run DB MIGRATION Tool

```
alembic upgrade head
```

Now Start the App

```
uvicorn app.main:app
but it throws the error

to Overcome this

uvicorn --host 0.0.0.0 app.main:app
```

```
(venv) apiuser@ip-172-31-85-183:~/app/src$ cd ../..
(venv) apiuser@ip-172-31-85-183:~/app/src$ uvicorn app.main:app
postgres
INFO:     Started server process [25662]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

## Run Files using Gunicorn

```
gunicorn -w 4 -k uvicorn.workers.UvicornWorker app.main:app --bind 0.0.0.0
```

**we want this Command to start automatically**

```
Lets Create our own service
cd /etc/systemd/system
vi apiserver.service
```

```
[Unit]
Description=demo fastapi application
After=network.target

[Service]
User=sanjeev
Group=sanjeev
WorkingDirectory=/home/sanjeev/app/src/
Environment="PATH=/home/sanjeev/app/venv/bin"
EnvironmentFile=/home/sanjeev/.env
ExecStart=/home/sanjeev/app/venv/bin/gunicorn -w 4 -k uvicorn.workers.UvicornWorker app.main:app --bind 0.0.0.0:8000

[Install]
WantedBy=multi-user.target
```

```
cd /etc/systemd/system/
sudo vi apiserver.service # copy the Content to it

systemctl start apiserver
systemctl status apiserver
systemctl restart apiserver
```

## Start the Service Automatically when it reboots

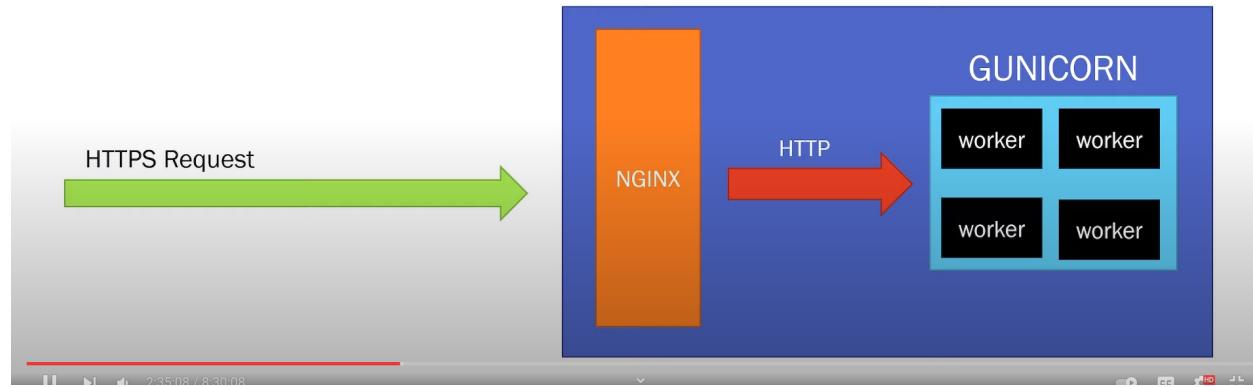
```
sudo systemctl enable api
```

## NGINX

nginx will act as a proxy Server



- High performance webserver that can act as a proxy
- Can handle SSL termination



```
sudo apt install nginx
systemctl start nginx

cd /etc/nginx/sites-available/
```

We need to Update the Configuration File

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    server_name _; # replace with specific domain name like sanjeev.com

    location / {
        proxy_pass http://localhost:8000;
        proxy_http_version 1.1;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $http_host;
        proxy_set_header X-NginX-Proxy true;
        proxy_redirect off;
    }
}
```

Now restart the service

```
systemctl restart nginx
```

## Enable the HTTPS

But a Domain Name from aws or any third party and setup in the Route53

Now we are going to enable the https

```
sudo apt install snapd
sudo snap install --classic certbot
sudo certbot --nginx
```

```

sanjeev@ubuntu-fastapi:/etc/nginx/sites-available$ certbot certonly --nginx -d sanjeev.xyz -d www.sanjeev.xyz
partner of the Let's Encrypt project and the non-profit organization that
develops Certbot? We'd like to send you email about our work encrypting the web,
EFF news, campaigns, and ways to support digital freedom.
-----
(Y)es/(N)o: N
Account registered.
Please enter the domain name(s) you would like on your certificate (comma and/or
space separated) (Enter 'c' to cancel): sanjeev.xyz www.sanjeev.xyz
Requesting a certificate for sanjeev.xyz and www.sanjeev.xyz

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/sanjeev.xyz/fullchain.pem
Key is saved at:      /etc/letsencrypt/live/sanjeev.xyz/privkey.pem
This certificate expires on 2021-12-01.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.

Deploying certificate
Successfully deployed certificate for sanjeev.xyz to /etc/nginx/sites-enabled/default
Successfully deployed certificate for www.sanjeev.xyz to /etc/nginx/sites-enabled/default
Congratulations! You have successfully enabled HTTPS on https://sanjeev.xyz and https://www.sanjeev.xyz

-----
If you like Certbot, please consider supporting our work by:
 * Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
 * Donating to EFF:                   https://eff.org/donate-le
-----
sanjeev@ubuntu-fastapi:/etc/nginx/sites-available$ |

```

If you're feeling more conservative and would like to make the changes to your Nginx configuration by hand, run this

## Setup a Firewall

we use security groups for handling Firewalls

```

sudo ufw allow http
sudo ufw allow https
sudo ufw allow ssh
sudo ufw allow 5432

## Enable the Firewall
sudo ufw enable

```

```

sanjeev@ubuntu-fastapi:/etc/nginx/sites-available$ sudo ufw status
Status: active

To                         Action      From
--                         --         --
80/tcp                      ALLOW      Anywhere
443/tcp                     ALLOW      Anywhere
22/tcp                      ALLOW      Anywhere
5432                         ALLOW      Anywhere
80/tcp (v6)                  ALLOW      Anywhere (v6)
443/tcp (v6)                 ALLOW      Anywhere (v6)
22/tcp (v6)                  ALLOW      Anywhere (v6)
5432 (v6)                   ALLOW      Anywhere (v6)

```

```

sanjeev@ubuntu-fastapi:/etc/nginx/sites-available$ |

```

# Dockerize the Application

Lets Create Docker Images

```
FROM python:3.9.7
WORKDIR /usr/src/app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Lets build this

```
docker build -t fastapi .
```

```
esak@esakpc:~/esak_2022/learnapibeginners$ docker build -t fastapi .
Sending build context to Docker daemon 124.7MB
Step 1/6 : FROM python:3.9.7
3.9.7: Pulling from library/python
bb7d5a84853b: Pull complete
f02b617c6a8c: Pull complete
d32e17419b7e: Pull complete
c9d2d81226a4: Pull complete
3c24ae8b6604: Pull complete
8a4322d1621d: Pull complete
0bde298e076a: Pull complete
e169b6c7c628: Pull complete
1b7366f8a3aa: Pull complete
Digest: sha256:8771691756bbf5beff80d64fca8f5b12e018352ddd9e30d8cdfe8cc3717b0e6
Status: Downloaded newer image for python:3.9.7
--> 208aa7e03e89
Step 2/6 : WORKDIR /usr/src/app
--> Running in 76fa91e89fda
Removing intermediate container 76fa91e89fda
--> 4a2f0569c6b7
Step 3/6 : COPY requirements.txt .
--> baed2ce8f533
Step 4/6 : RUN pip install --no-cache-dir -r requirements.txt
--> Running in 80c644336bad
Collecting alembic==1.7.5
  Downloading alembic-1.7.5-py3-none-any.whl (209 kB)
Collecting anyio==3.3.4
  Downloading anyio-3.3.4-py3-none-any.whl (78 kB)
Collecting asgiref==3.4.1
  Downloading asgiref-3.4.1-py3-none-any.whl (25 kB)
Collecting hcrnnt==3.2.0
```

## Write the Docker Compose File

```
version: '3'
services:
  api:
    build: .
    ports:
      - 8000:8000
```

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
esak@esakpc:~/esak_2022/learnapi_beginners$ docker-compose up -d
Creating network "learnapi_beginners_default" with the default driver
Building api
Sending build context to Docker daemon 124.7MB
Step 1/6 : FROM python:3.9.7
--> 208aa7e03e89
Step 2/6 : WORKDIR /usr/src/app
--> 4a2f0569c0b7
Step 3/6 : COPY requirements.txt .
--> Using cache
--> baed2ce9f533
Step 4/6 : RUN pip install --no-cache-dir -r requirements.txt
--> Using cache
--> 2a7a9dd0faff
Step 5/6 : COPY . .
--> 20391ab03985
Step 6/6 : CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
--> Running in 617159030fd4
Removing intermediate container 617159030fd4
--> b9b3c4d21cbf
Successfully built b9b3c4d21cbf
Successfully tagged learnapi_beginners_api:latest
WARNING: Image for service api was built because it did not already exist. To rebuild this image you must use 'docker-compose build' or 'docker-compose up --build'.
Creating learnapi_beginners_api_1 ... done
esak@esakpc:~/esak_2022/learnapi_beginners$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
c32d5eaa6fc learnapi_beginners_api "uvicorn app.main:ap..." 8 seconds ago Up 4 seconds 0.0.0.0:8000->8000/tcp, :::8000->8000/tcp learnapi_beginners_api_1
esak@esakpc:~/esak_2022/learnapi_beginners$ 

```

## Adding Postgress Service

we need to add the docker ip Address

```
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' 6805a3434dde
172.20.0.2
```

The screenshot shows the pgAdmin interface with a connection configuration dialog open. The title bar says '/postgres@Container'. The tabs at the top are 'Query' and 'Container', with 'Container' being the active tab. Below the tabs are several input fields:

- Host name/address:** 172.20.0.2
- Port:** 5432
- Maintenance database:** postgres
- Username:** postgres
- Kerberos authentication?**: A toggle switch is off.
- Role:** An empty text input field.
- Service:** An empty text input field.

At the bottom of the dialog are three buttons: 'Close', 'Reset', and 'Save'. To the right of the dialog, there is a table with one row containing the 'created\_at' timestamp: '2021-12-10 07:01:58.892116+00'.

Now we can access the POSTGRESS CONTAINER from our host PGADMIN

Docker Stores all the data in the FOLlowing Location

```
The contents of the /var/lib/docker directory vary depending on the driver Docker is using for storage.
```

## Docker File

```
FROM python:3.9.7

WORKDIR /usr/src/app

COPY requirements.txt ./

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

COPY . .

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

## Docker-Compose File

```
version: '3'
services:
  api:
    build: .
    depends_on:
      - postgres
    ports:
      - 8000:8000
    volumes: # example of Bind Mount
      - ./:/usr/src/app:ro
    command: unicorn app.main:app --host 0.0.0.0 --port 8000 --reload # weoverride the code

    # env_file:
    #   - app/.env
    environment:
      - ALGORITHM=HS256
      - DATABASE_HOSTNAME=postgres
      - DATABASE_PASSWORD=password123
      - DATABASE_PORT=5432
      - DATABASE_NAME=fastapi
      - DATABASE_USERNAME=postgres
      - SECRET_KEY=0258712wsdasdfesak12324374lopiutyyus
      - ACCESS_TOKEN_EXPIRE_MINUTES=30

  postgres:
    image: postgres
    environment:
      - POSTGRES_PASSWORD=password123
      - POSTGRES_DB=fastapi

    volumes:
      - postgres-db:/var/lib/postgresql/data

volumes:
  postgres-db:
```

We have these Images build and created in our Local

Lets Push these to DockerHub

The screenshot shows a Docker Hub repository page for the user 'esak2021' with the repository name 'fastapi'. The top navigation bar includes links for 'Explore', 'Repositories', 'Organizations', 'Help', 'Upgrade' (highlighted in yellow), and a user dropdown for 'esak2021'. The main content area displays the repository details, including a 'General' tab selected, a 'Tags' tab, a 'Builds' tab, a 'Collaborators' tab, a 'Webhooks' tab, and a 'Settings' tab. A prominent feature is the 'Advanced Image Management' section, which allows users to view all images and tags, clean up unused content, and recover untagged images. It is available with Pro, Team, and Business subscriptions. Below this, the repository name 'esak2021 / fastapi' is shown, along with a single tag 'fastapi' and a note that it was last pushed 'never'. A 'Docker commands' section provides a command to push a new tag: 'docker push esak2021/fastapi:tagname'. There is also a 'Public View' button. The 'Tags and Scans' section indicates that the repository is empty and vulnerability scanning is disabled. A 'Readme' section is present but empty. Automated build instructions are provided, suggesting manual pushing to GitHub or Bitbucket to trigger automated builds. A 'Login to DockerHub' section at the bottom contains terminal commands for creating a tag and pushing the image to Docker Hub.

```
Create tag for the Local Image s
By default latest will be used
docker image tag learnapi_beginners_api esak2021/fastapi

## Push the image to docker hub
docker push esak2021/fastapi
```

```
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ docker image ls
REPOSITORY          TAG      IMAGE ID   CREATED        SIZE
esak2021/fastapi    latest   f6d082773122 9 minutes ago  1.15GB
learnapi_beginners_api    latest   f6d082773122 9 minutes ago  1.15GB
<none>              <none>   8984e13db7e3  32 minutes ago  1.15GB
<none>              <none>   ab2e3fed57ac  39 minutes ago  1.15GB
postgres            latest   e94a3bb61224  7 days ago   374MB
python               3.9.7    208aa7e03e89  6 weeks ago   912MB
fourthproject_frontend latest   e3eb25470bfb  3 months ago   303MB
fourthproject_backend latest   26ea5cf8daed  3 months ago   94.9MB
nginx               latest   dd34e67e3371  3 months ago   133MB
python               3.9-alpine d4d6be1b90ec  3 months ago   45.1MB
gcr.io/k8s-minikube/kicbase v0.0.25  8768edd4356  5 months ago   1.1GB
node                15.13-alpine 3d9a0ea05233  8 months ago   112MB
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ docker push esak2021/fastapi
Using default tag: latest
The push refers to repository [docker.io/esak2021/fastapi]
91a9d6c99c25: Pushing [=>] 3.752MB/121.6MB
17fc7fdc7da9: Pushing [=====>] 13.51MB/121.8MB
087d02f7f2bd: Pushing [=====>] 4.096kB
f074ddf286c4: Pushing 3.072kB
f77bec5d5162: Mounted from library/python
7b656b8058c4: Waiting
02a38a00d553: Waiting
7fcfd2600f5ad: Waiting
8f56c3340629: Waiting
ba6e5ff31f23: Waiting
9f9f651e9303: Waiting
0b3c02b5d746: Waiting
62a747bf1719: Waiting
```

The screenshot shows the Docker Hub interface for the repository `esak2021/fastapi`. At the top, there's a search bar and navigation links for Explore, Repositories, Organizations, Help, Upgrade, and a user dropdown. Below the header, it says "Using 1 of 1 private repositories. [Get more](#)". The main content area has tabs for General, Tags, Builds, Collaborators, Webhooks, and Settings. Under the General tab, there's a section for "Advanced Image Management" with a link to "View preview". The repository details show "fastapi" as the tag and "a few seconds ago" as the last push time. To the right, there's a "Docker commands" section with a button to "Push a new tag to this repository" and a command line input field containing `docker push esak2021/fastapi:tagname`. Another section for "Tags and Scans" shows "VULNERABILITY SCANNING - DISABLED" with an "Enable" link. The "Automated Builds" section is available with Pro, Team, and Business subscriptions.

Lets Update our docker compose file for production Setup

```
version: '3'
services:
  api:
    image: esak2021/fastapi
```

```

depends_on:
  - postgres
ports:
  - 80:8000
# volumes: # example of Bind Mount
#   - ./usr/src/app:ro
#command: uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload  # weoverride the code

# env_file:
#   - app/.env
environment:
  - ALGORITHM=HS256
  - DATABASE_HOSTNAME=${DATABASE_HOSTNAME}
  - DATABASE_PASSWORD=${DATABASE_PASSWORD}
  - DATABASE_PORT=${DATABASE_PORT}
  - DATABASE_NAME=${DATABASE_NAME}
  - DATABASE_USERNAME=${DATABASE_USERNAME}
  - SECRET_KEY=${SECRET_KEY}
  - ACCESS_TOKEN_EXPIRE_MINUTES=${ACCESS_TOKEN_EXPIRE_MINUTES}

postgres:
  image: postgres
  environment:
    - POSTGRES_PASSWORD=${DATABASE_PASSWORD}
    - POSTGRES_DB=${DATABASE_NAME}

  volumes:
    - postgres-db:/var/lib/postgresql/data

volumes:
  postgres-db:

```

## Adding Test Cases

**we are using a library called 'pytest'**

```

(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ pytest
===== test session starts =====
platform linux -- Python 3.9.5, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /home/esak/esak_2022/learnapi_beginners
plugins: asyncio-3.3.4
collected 0 items

===== no tests ran in 0.04s =====
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ []

```

Lets Create out Test in the app Folder and create a File called `test_calculation.py` . Always the File should Start with `test_*` and add our test case .

```

from app.calculation import add
def test_add():
    assert 5 == add(1, 4)

test_add()

To Run this we have to use
pytest
pytest -v for more verbose
pytest -v -s To include the print statement as well

```

```

(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ pytest
===== test session starts =====
platform linux -- Python 3.9.5, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /home/esak/esak_2022/learnapi_beginners
plugins: asyncio-3.3.4
collected 1 item
tests/test_calculation.py . [100%]
===== 1 passed in 0.05s =====
(venv) esak@esakpc:~/esak_2022/learnapi_beginners$ []

```

Write the Test Cases as below

```
from app.calculation import add , sub, mul, div

def test_add():
    assert 5 == add(1, 4)

def test_sub():
    assert sub(9,5) == 4

def test_mul():
    assert mul(1,2) == 2

def test_div():
    assert div(4,2) == 2
```

We want to add more testcases then we have to do parameterization

```
@pytest.mark.parametrize(
    "num1, num2, result",
    [
        (3,3,6),
        (4,2,6),
        (4,5,9),
        (1,3,4)
    ]
)
def test_add(num1, num2,result ):
    assert add(num1, num2) == result
```

```
1  import pytest
2
3  from app.calculation import add , sub, mul, div
4
5  @pytest.mark.parametrize(
6      "num1, num2, result",
7      [
8          (3,3,6),
9          (4,2,6),
10         (4,5,9),
11         (1,3,4)
12     ]
13 )
14 def test_add(num1, num2,result ):
15     assert add(num1, num2) == result
16
17 def test_sub():
18     assert sub(9,5) == 4
19
20 def test_mul():
21     assert mul(1,2) == 2
22
23 def test_div():
24     assert div(4,2) == 2
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

cachedir: .pytest\_cache  
rootdir: /home/esak/esak\_2022/learnapi\_beginners  
plugins: anyio-3.3.4  
collected 7 items

tests/test\_calculation.py::test\_add[3-3-6] PASSED  
tests/test\_calculation.py::test\_add[4-2-6] PASSED  
tests/test\_calculation.py::test\_add[4-5-9] PASSED  
tests/test\_calculation.py::test\_sub PASSED  
tests/test\_calculation.py::test\_mul PASSED  
tests/test\_calculation.py::test\_div PASSED

===== 7 passed in 0.07s =====

(venv) esak@esakpc:~/esak\_2022/learnapi\_beginners\$

We are going to write test cases for the belwo code

```
class BankAccount():
    def __init__(self, starting_balance=0) -> None:
        self.balance = starting_balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        self.balance -= amount

    def collect_interest(self):
        self.balance *= 1.1
```

```
def test_bank_set_init_amount():
    bank = BankAccount(500)
    assert bank.balance == 500

def test_bank_default_amount():
```

```

bank = BankAccount()
assert bank.balance == 0

def test_bank_withdraw_amount():
    bank = BankAccount(500)
    bank.withdraw(100)
    assert bank.balance == 400

def test_bank_deposit_amount():
    bank = BankAccount(50)
    bank.deposit(10)
    assert bank.balance == 60

def test_bank_interest_amount():
    bank = BankAccount(50)
    bank.collect_interest()
    assert round(bank.balance, 2) == 55.00

```

in all of our test cases we are repeating the creation of bank object to overcome this , we have to use **fixtures . Fixtures will be called if we define them in the method parameters**

```

## Creating a Fixture for our Bank account class

@pytest.fixture
def zero_bank_account():
    return BankAccount()

@pytest.fixture
def bank_account():
    return BankAccount(50)

def test_bank_set_init_amount(bank_account):
    assert bank_account.balance == 50

def test_bank_default_amount(zero_bank_account):
    assert zero_bank_account.balance == 0

def test_bank_withdraw_amount(bank_account):
    bank_account.withdraw(100)
    assert bank_account.balance == 400

def test_bank_deposit_amount(bank_account):
    bank_account.deposit(10)
    assert bank_account.balance == 60

def test_bank_interest_amount(bank_account):
    bank_account.collect_interest()
    assert round(bank_account.balance, 2) == 55.00

```

Lets Parameterize the Test

```

@pytest.mark.parametrize(
    "deposited, withdraw, result",
    [
        (500, 300, 200),
        (400, 200, 600),
        (400, 500, 1100),
        (1000, 300, 700)
    ]
)
def test_bank_transaction(zero_bank_account, deposited, withdraw, result):
    zero_bank_account.deposit(deposited)
    zero_bank_account.withdraw(withdraw)
    assert zero_bank_account.balance == result

```

The screenshot shows a code editor with two panes. The left pane contains Python test code using pytest. The right pane shows the terminal output of the pytest run.

```

53     @pytest.mark.parametrize(
54         "deposited, withdraw, result",
55         [ (500,300,200),
56           (400,200,600),
57           (400,500,1100),
58           (1000,300,700) ]
59     )
60
61     def test_bank_transaction(zero_bank_account, deposited, withdraw,result):
62         zero_bank_account.deposit(deposited)
63         zero_bank_account.withdraw(withdraw)
64         assert zero_bank_account.balance == result
65
66
67

```

```

13     return x1 / x2
14
15
16     BankAccount():
17         if __init__(self, starting_balance=0) -> None:
18             self.balance = starting_balance
19
20         if deposit(self, amount):
21             self.balance += amount
22
23         if withdraw(self, amount):
24             if amount > self.balance:
25                 raise Exception ("ERROR: Insufficient Funds")
26             self.balance -= amount
27
28         if collect_interest(self):
29             self.balance *= 1.1

```

TERMINAL OUTPUT:

```

self = <app.calulation.BankAccount object at 0x7f7598d4850>, amount = 500
    def withdraw(self, amount):
        if amount > self.balance:
>           raise Exception ("ERROR: Insufficient Funds")
E           Exception: ERROR: Insufficient Funds
app/calulation.py:25: Exception
=====
FAILED tests/test_calculations.py::test_bank_withdraw_amount - Exception: ERROR: Insufficient Funds
FAILED tests/test_calculations.py::test_bank_transaction[400-200-600] - assert 200 == 600
FAILED tests/test_calculations.py::test_bank_transaction[400-500-1100] - Exception: ERROR: Insufficient Funds
=====
===== short test summary info =====
3 failed, 13 passed in 0.16s =====

```

if we except an Exception from the Code , then we can handle the scenario as below

```

def test_insufficient_funds(bank_account):
    with pytest.raises(Exception):
        bank_account.withdraw(200)

```

we want to stop the test when there is an Failure

```
pytest -v -x
```

## Test Our Application

we want to create our own test database

```

from fastapi.testclient import TestClient
from app.database import get_db, Base
from app.main import app
from app import schemas
from app.config import settings
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

import pytest

TEST_HOST_URL=f'postgresql://{{settings.database_username}}:{{settings.database_password}}@172.21.0.2:5432/testfastapi'
# create Engine
engine = create_engine(TEST_HOST_URL)

print(f"INFO:: USERNAME ::{settings.database_username}")
print(f"INFO:: PASSWORD ::{settings.database_password}")
print(f"INFO:: HOSTNAME ::{settings.database_hostname}")

TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)


```

```

@pytest.fixture(scope="module")
def session():
    Base.metadata.drop_all(bind=engine)
    Base.metadata.create_all(bind=engine)

```

```

db = TestingSessionLocal()
try:
    yield db
finally:
    db.close()

@pytest.fixture(scope="module")
def client(session):
    def override_get_db():

        try:
            yield session
        finally:
            session.close()
    app.dependency_overrides[get_db] = override_get_db
    yield TestClient(app)

def test_root(client):
    res = client.get("/")
    print(res.json())
    print(res.json().get('message'))
    assert res.json().get('message') == 'Welcome to api'
    assert res.status_code == 200

def test_create_user(client):
    res = client.post(
        "/api/v1/users/", json={"email": "hello123@gmail.com", "password": "password123"})
    print(res)
    print(res.status_code)
    print(res.json())
    new_user = schemas.UserResponse(**res.json())
    assert new_user.email == "hello123@gmail.com"
    assert res.status_code == 201

# here the login test is dependednt on the above test case
# Each and every test should be independent
def test_user_login(client):
    res = client.post(
        "/api/v1/auth/login", data={"username": "hello123@gmail.com", "password": "password123"})
    print(res)
    print(res.status_code)
    print(res.json())

    assert res.status_code == 200

```

Lets Create a new File called conftest.py.

in here we are going to create all of our fixtures inside of it. pytest will call these fixtures when ever needed

```

from fastapi.testclient import TestClient
from app.database import get_db, Base
from app.main import app
from app.config import settings
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base
from jose import JWTError, jwt
import pytest

TEST_HOST_URL='postgresql://{}:{}@172.21.0.2:5432/testfastapi'
# create Engine
engine = create_engine(TEST_HOST_URL)

print(f"INFO:: USERNAME ::{settings.database_username}")
print(f"INFO:: PASSWORD ::{settings.database_password}")
print(f"INFO:: HOSTNAME ::{settings.database_hostname}")

TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

@pytest.fixture(scope="function")
def session():
    Base.metadata.drop_all(bind=engine)
    Base.metadata.create_all(bind=engine)
    db = TestingSessionLocal()

```

```

try:
    yield db
finally:
    db.close()

@pytest.fixture(scope="function")
def client(session):
    def override_get_db():

        try:
            yield session
        finally:
            session.close()
    app.dependency_overrides[get_db] = override_get_db
    yield TestClient(app)

# Creating a Fixture
# this will create a user in the database
@pytest.fixture()
def create_test_user(client):

    user_data = {
        "email": "hello456@gmail.com",
        "password": "password123"
    }
    res = client.post("/api/v1/users/", json=user_data)
    assert res.status_code == 201
    print(res.json())
    new_user = res.json()
    new_user['password'] = user_data['password']
    return new_user

```

## Implement a CICD pipeline

```

name: Build and Deploy api
# Trigger the Branch when we push the Code to Following Branches
on:
  push:
    branches:
      - "main"
      - "anotherbranch"
      - "dev"

# Trigger the Branch when we create PR to Following Branches
  pull_request:
    branches:
      - "main"
      - "test"

```

push the code to the Github

Go to Actions Tab, a Workflow is triggered automatically

The screenshot shows the GitHub Actions workflow runs page for the repository 'esak21/learnapi\_beginners'. The 'Actions' tab is selected. Under the 'Workflows' section, there is a card for 'Build and Deploy api'. A search bar is present above the workflow runs table. The table has one row labeled 'added workflow' with a green checkmark, which triggered the run. The run details show it was triggered by a commit pushed by esak21, branch main, and occurred 20 seconds ago. There are 199 actions in the run.

To Store the Environment Variables in Github

The screenshot shows the GitHub Settings page for the repository 'esak21/learnapi\_beginners'. The 'Settings' tab is selected. On the left, a sidebar lists options: Options, Manage access, Security & analysis, Branches, Webhooks, Notifications, and Integrations. The main area is titled 'Actions secrets' and contains a table with one row for 'DATABASE\_HOSTNAME'. It shows the secret value, an update timestamp of 'Updated 2 minutes ago', and 'Update' and 'Remove' buttons.

To access this in Code

```
jobs:  
  job1:  
    env:  
      DATABASE_HOSTNAME: ${{ secret.DATABASE_HOSTNAME }}
```

We can setup secrets in using Environments

The screenshot shows the 'Environment secrets' section of a GitHub repository's settings. On the left is a sidebar with 'Actions', 'Environments', 'Secrets', 'Pages', and 'Moderation settings'. The main area has a heading 'Deployment branches' with a note about limiting what branches can deploy. A dropdown menu shows 'All branches'. Below is a table of secrets:

Secret	Last updated	Actions
ACCESS_TOKEN_EXPIRE_MINUTES	Updated now	<button>Update</button> <button>Remove</button>
ALGORITHM	Updated 22 seconds ago	<button>Update</button> <button>Remove</button>
DATABASE_HOSTNAME	Updated 4 minutes ago	<button>Update</button> <button>Remove</button>
DATABASE_NAME	Updated 1 minute ago	<button>Update</button> <button>Remove</button>
DATABASE_PASSWORD	Updated 2 minutes ago	<button>Update</button> <button>Remove</button>
DATABASE_PORT	Updated 3 minutes ago	<button>Update</button> <button>Remove</button>
SECRET_KEY	Updated 43 seconds ago	<button>Update</button> <button>Remove</button>
+ Add Secret		

## Entire Code for CI

```

name: Build and Deploy api

on: [push , pull_request ]

jobs:
  job1:
    environment:
      name: QA
    env:
      DATABASE_HOSTNAME: ${{secrets.DATABASE_HOSTNAME}}
      DATABASE_PORT: ${{secrets.DATABASE_PORT}}
      DATABASE_PASSWORD: ${{secrets.DATABASE_PASSWORD}}
      DATABASE_NAME: ${{secrets.DATABASE_NAME}}
      DATABASE_USERNAME: ${{secrets.DATABASE_USERNAME}}
      SECRET_KEY: ${{secrets.SECRET_KEY}}
      ALGORITHM: ${{secrets.ALGORITHM}}
      ACCESS_TOKEN_EXPIRE_MINUTES: ${{secrets.ACCESS_TOKEN_EXPIRE_MINUTES}}
    services:
      postgres:
        image: postgres
        env:
          POSTGRES_PASSWORD: ${{secrets.DATABASE_PASSWORD}}
          POSTGRES_DB: ${{secrets.DATABASE_NAME}}
        ports:
          - 5432:5432
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
    runs-on: ubuntu-latest
    steps:
      - name: Pull Git Repo
        uses: actions/checkout@v2

      - name: Setup Python Version 3.9
        uses: actions/setup-python@v2
        with:
          python-version: '3.9'
      - name: Upgrade Pip
        run: python -m pip install --upgrade pip
      - name: Install dependencies
        run: pip install -r requirements.txt
      - name: Run the Test cases
        run: |
          pip install pytest
          pytest -v -s

```

## Lets Setup the Continous Delivery

The screenshot shows the Docker Hub interface for the repository `esak2021/fastapi`. The repository has one tag, `latest`, which was pushed a minute ago. The Docker commands section shows the command `docker push esak2021/fastapi:tagname`. The Tags and Scans section indicates that vulnerability scanning is disabled. The Automated Builds section shows that manual pushing is available. Below the repository page is a GitHub Actions YAML configuration file:

```
name: Build and Deploy api

on: [push , pull_request ]

jobs:
  Build:
    environment:
      name: QA
    env:
      DATABASE_HOSTNAME: ${secrets.DATABASE_HOSTNAME}
      DATABASE_PORT: ${secrets.DATABASE_PORT}
      DATABASE_PASSWORD: ${secrets.DATABASE_PASSWORD}
      DATABASE_NAME: ${secrets.DATABASE_NAME}
      DATABASE_USERNAME: ${secrets.DATABASE_USERNAME}
      SECRET_KEY: ${secrets.SECRET_KEY}
      ALGORITHM: ${secrets.ALGORITHM}
      ACCESS_TOKEN_EXPIRE_MINUTES: ${secrets.ACCESS_TOKEN_EXPIRE_MINUTES}
    services:
      postgres:
        image: postgres
        env:
          POSTGRES_PASSWORD: ${secrets.DATABASE_PASSWORD}
          POSTGRES_DB: ${secrets.DATABASE_NAME}
        ports:
          - 5432:5432
        options: >
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
    runs-on: ubuntu-latest
    steps:
      - name: Pull Git Repo
        uses: actions/checkout@v2

      - name: Setup Python Version 3.9
        uses: actions/setup-python@v2
        with:
          python-version: '3.9'
      - name: Upgrade Pip
        run: python -m pip install --upgrade pip
```

```

- name: Install dependencies
  run: pip install -r requirements.txt
- name: Run the Test cases
  run: |
    pip install pytest
    pytest -v -s
- name: Login To Docker Hub
  uses: docker/login-action@v1
  with:
    username: ${{ secrets.DOCKER_HUB_USERNAME }}
    password: ${{ secrets.DOCKER_HUB_ACCESS_TOKEN }}

- name: setup Docker build
  id: builder
  uses: docker/setup-buildx-action@v1

- name: build and push
  id: docker_build
  uses: docker/build-push-action@v2
  with:
    context: ../
    file: ./Dockerfile
    builder: ${steps.buildx.outputs.name}
    push: true
    tags: ${secrets.DOCKER_HUB_USERNAME}/fastapi:latest
    cache-from: type=local,src=/tmp/.buildx-cache
    cache-to: type=local,dest=/tmp/.buildx-cache
- name: Image diges
  run: echo ${steps.docker_build.outputs.digest}

Deploy:
runs-on: ubuntu-latest
needs: [Build] # Job run in sequential Order
steps:
- name: push changes to production
  run: echo "Deploying to Production Steps"

```