# What is Fast API

A modern web framework for building WEB APIs using python, which is designed to be fast, easy to use, efficient and highly performant.

It is built on top of Python pydantic (a data validation and serialization library) and starlette (a high-performance asynchronous WEB framework).

Supports Swagger documentation, also it automatically includes created endpoints at <ip address>:<port>/docs ex: [127.0.0.1:8000/docs](http://127.0.0.1:8000/well_hello).

# What is routing in FAST API

IT is about creating and using FAST APIs with large scale networks.

# What is Alembic in FAST API

It is about changing and optimizing database for FAST API

# Why FAST API

It is one of the fastest performing WEB frameworks, It is really really very fast.

It is fast w.r.t performance and development. It includes data validation, serialization, and documentation as well.

With few lines of code, we can have an Restful API created

Has an active community

Let’s you develop APIs with fewer bugs and validation included.

# Where does FAST API fit in the architecture?

Fast API will be the backend server which will receive RESTFUL APIs from Front end and with proper authentication, will send data back to Front end users

A computer screen shot of a computer

Description automatically generated

# Installation of fast api

Fast api is available as a python package, but for fast api to run, it needs a web server called uvicorn

You can run the below commands to install

pip install fastapi

pip install “uvicorn[standard]”

# CRUD operations Translated to HTTP REQUEST METHODS in FASTAPI

Create -> POST API/ HTTP request method (To remember difference between POST and PUT, you need to create(POST) an object, then only you can update (PUT), an analogy for this in real time is you need to write a letter (POST) then only you can drop (PUT) into post box

Read -> GET API/ HTTP request method

Update -> PUT API/ HTTP request method

Delete -> DELETE API/ HTTP request method

# How is fast api implemented:

After you install fastapi and uvicorn packages, you can create a simple python file like below

from fastapi import FastAPI  
  
app = FastAPI() #to use FastAPI here  
  
  
BOOKS = [  
 {"title":"1", "author":'Author One', 'category':'fiction'},  
{"title":"2", "author":'Author Two', 'category':'fiction'},  
{"title":"3", "author":'Author Three', 'category':'science'},  
{"title":"4", "author":'Author Four', 'category':'science'},  
{"title":"5", "author":'Author Five', 'category':'Political'},  
{"title":"6", "author":'Author Six', 'category':'Political'},  
]  
  
  
@app.get("/books") #books here is the endpoint.  
async def read\_all\_books(): #by default all functions are async, as it is the framework starlette that fastapi uses.  
 return BOOKS

voila!, then run the python file using the command **uvicorn <name of the python file>:<the variable that you assign** FastAPI() **to>**, your backend application is running in your local.

Example:

uvicorn books:app --reload #here the reload is used so that when you change the code of the python file, It will automatically incorporate changes to backend server.

## Does order of declaration of endpoints matter?

Yes

If you have 2 endpoints,

1: /book/{dynamic parameter}

Suppose this API

on sending /book/pages will return page count of the book.

On sending /book/genre will return the genre of the book

2: /book/title\_of\_book

This API will simply return the titles of the available books

If you declare the /book/{dynamic parameter} api first and /book/title\_of\_book second, then what happens is when FE queries for /book/title\_of\_book will go to first API

## Types of parameters, we can include in API end point:

We can have path parameters or query parameters or both combined.A computer screen shot of a code

Description automatically generated

## POST API

A screen shot of a computer program

Description automatically generated

# IMP - (high level) Description of packages that are used to achieve major functionalities to develop an application

* Starlette – light weight ASGI framework used to build web applications (for all API handling stuff- routing, connecting APIs)
* Pydantic – used for data modelling, data handling, error validation.
* Sqlalchemy – to perform data operations with the database (from which the backend fetches data)
* python-multipart – to accept form data (data in form(similar to google forms)).
* from fastapi.security import OAuth2PasswordRequestForm #passed as a dependency injection to user login
* pip install "python-jose[cryptography]" 🡪 to implement JWT so that, we authorize API requests.
  + To generate signature you can use
    - openssl rand -hex 32
* passlib – to implement authorization

# What is pydantic

Pydantic is a python library that handles data modelling, data parsing and efficient error handling.

Commonly used for data validation, and also how to handle data that comes to FAST API application.

# Major differences between pydantic1 and pydantic2

.dict() function is now renamed to .model\_dump()

schema\_extra function within a Config class is now renamed to json\_schema\_extra

# What are http exceptions

HTTP exceptions are something which will be raised in order to cancel the functionality of method in which it is called and return the status code to the user.

Example: if a fb user is trying to find a profile with name “zhing Zhong zhung” if that user is not in FB, then we need to return 404 resource not found error.

Refer to implementation of this in book2.py

# Connect Database with application using Fast API

We need SQL Alchemy (a ORM) so that using python we can connect and operate on a database.

pip install sqlalchemy.

sqlalchemy offers

* ORM
* Helps to create an engine (it helps in giving python ability to connect to SQL database.) with many features such as
  + Check many threads (as each thread can communicate independently).
  + from sqlalchemy import create\_engine
  + engine = create\_engine(SQL\_ALCHEMY\_DATABASE\_URL, connect\_args= {'check\_same\_thread': False})
* Create session (a session represents a connection between application and database).
  + from sqlalchemy.orm import sessionmaker
  + SessionLocal = sessionmaker(autoflush=False, autocommit = False, bind=engine)
* Create a database object, so that we can interact with database later.
  + from sqlalchemy.ext.declarative import declarative\_base
  + Base = declarative\_base() #to interact with tables of database

# Routers in fastapi

As the application grows, it will be difficult to accommodate all APIs in single python file, therefore it is suggested to have routers and APIs of particular topic to be in particular python file and then included in the main fast API file with the help of routers.

# DB for FASTAPI

■SQLite3 strives to provide local data storage for

individual applications and devices.

■SQLite3 emphasizes economy, efficiency and simplicity.

■For most small / medium applications, SQLite3 will

works perfectly.

■SQLite3 focuses on different concepts than a

production Database Management System.

■MySQL & PostgreSQL focuses on a big difference

compared to SQLite3.

■These **production (PostgreSQL, MySQL) DBMS focuses on scalability,**

**concurrency and control**.

■SQLite3 runs in-memory, which allows development of a

SQLite3 data to be easy, as it is part of your application!

■Production DBMS run on their own server and port. Which

means you need to make sure the database is running, and

have authentication linking to the DBMS

■(SQLite3) For deployment you can deploy a SQLite3 database

along with the application

■(Prod DBMS) For deployment you will need to also deploy the

database separate from the application

## Postgres

### When to choose which type of db you want?

■If you application is going to have 10s of thousands of

users, it may be wise to switch to a production DBMS

■If you application is only you, and a few others, SQLite3

will work great

### Features of POSTGRESQL

* Production ready
* Open source RDBMS
* Needs server
* Highly scalable
* Secure

There are additional elements you will be installing while installing Postgresql - <https://sbp.enterprisedb.com/getfile.jsp?fileid=1258641>

* Pgadmin - GUI framework to interact with POSTGRESQL
* Stackbuilder – acts like an add-on installer to help us download additional installers, tools or applications to complement (achieve more using) POSTGRESQL

### To connect POSTGRESQL to fastapi application

pip install psycopg2-binary

assign SQL\_ALCHEMY\_DATABASE\_URL, the following values

SQL\_ALCHEMY\_DATABASE\_URL = (

"postgresql://postgres:<postgres password>@localhost/TodoApplicationDatabase"

)

engine = create\_engine(SQL\_ALCHEMY\_DATABASE\_URL)

### Don’ts while using mysql

Don’t use password with @, fastp api connection with postgres will get confused and @ before the local host in the SQL\_ALCHEMY\_DATABASE\_URL

## Mysql

### Features of POSTGRESQL

* Production ready
* Open source RDBMS
* Needs server
* scalable
* Secure