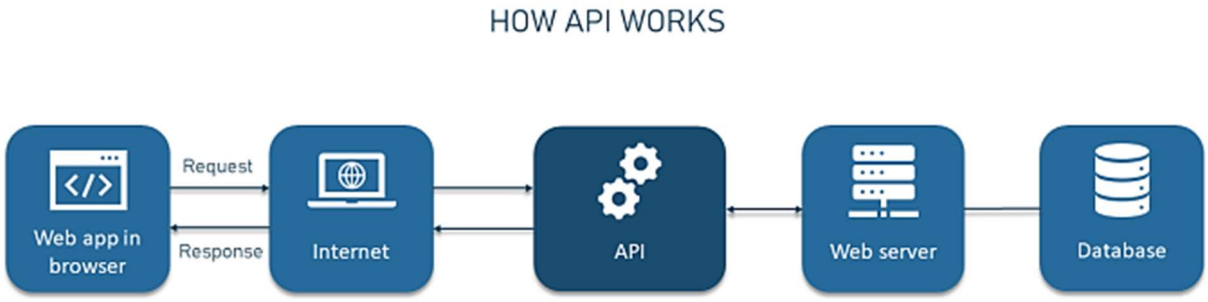
LEC # 05

API – Application Programming Interface – Two types – Fast API and Rest API

An application programming interface is a way for two or more computer programs or components to communicate with each other. It is a type of software interface, offering a service to other pieces of software.



Importance of API

Standard format input output

APIs are needed to bring applications together in order to perform a designed function built around sharing data and executing pre-defined processes.

FastAPI

FastAPI is a modern web framework for building APIs (AI powered APIs) with Python 3.7+ that is gaining widespread popularity. It is known for its high performance, auto-generated interactive documentation, and simplicity. FastAPI uses Python type hints for defining request and response models, which enables automatic validation and interactive documentation.

FastAPI vs Django REST Framework

FastAPI and Django Rest Framework are both powerful tools for building APIs in Python, but they cater to different use cases and preferences.

- If you need a high-performance, async-ready API with a shallow learning curve and interactive documentation, FastAPI is an excellent choice.
- If you are developing a full-stack Django application and need a comprehensive set of tools for API development, Django Rest Framework is a proven and reliable choice.

Sync vs Async

Synchronous means to be in a sequence, i.e. every statement of the code gets executed one by one. So, basically a statement has to wait for the earlier statement to get executed. It is blocking in nature.

Asynchronous is a non-blocking architecture, which means it doesn't block further execution while one or more operations are in progress.

Dependency Injection

"Dependency Injection" means, in programming, that there is a way for your code (in this case, your path operation functions) to declare things that it requires to work and use: "dependencies".

And then, that system (in this case FastAPI) will take care of doing whatever is needed to provide your code with those needed dependencies ("inject" the dependencies).

This is very useful when you need to:

- Have shared logic (the same code logic again and again).
- Share database connections.
- Enforce security, authentication, role requirements, etc.
- And many other things...

All these, while minimizing code repetition.

```python
from fastapi import FastAPI, Depends, Params

app = FastAPI()

# the dependency function:
def user_dep(name: str = Params, password: str = Params):
    return {"name": name, "valid": True}

# the path function / web endpoint:
@app.get("/user")
def get_user(user: dict = Depends(user_dep)) -> dict:
    return user
```

- The get_user() depends on user_dep() , it will run after user_dep() is executed.
- Depends(user_dep) means that user_dep() will be executed only when get_user() function is Used andwill NOT execute when get_user() function is Defined.

Work in VS code

- Create folder ned fast api using poetry
- Write poetry shell – to activate poetry
- To install packages: poetry add fastapi uvicorn httpx httpie request
- Create file main.py in sub folder

```python
users : list[dict[Any,Any]] = [
    {'name' : 'John', 'password' : '123'},
    {'name' : 'Jane', 'password' : '456'}
]

# the dependency function:
def user_dep(name: str = params, password: str = params):
    for d in users:
        if (d['name'] == name and d['password'] == password):
            return {"name": name,
```

```python
                    "valid": True,
                    "message": "Welcome " + name}
    else:
        return {"name": name, "valid": False}

# the path function / web endpoint:
@app.get("/user_dep")
def get_user(user: Annotated[dict, Depends(user_dep)]) -> dict:
    return user
```

Streamlit

- Create a Poetry Project
- Add Streamlit library
- Copy/Create your Streamlit Application in Poetry Folder.
- Create __init__.py file in your streamlit application folder
- Run streamlit app using following command
    - poetry run streamlit run ./00_helloworld/hello.py

Notes

- Session dictionary is created on Server.
- Cookies are created on Client.