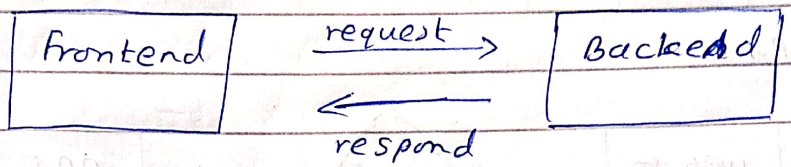
FastAPI

classmate
Date 19-6-25
Page

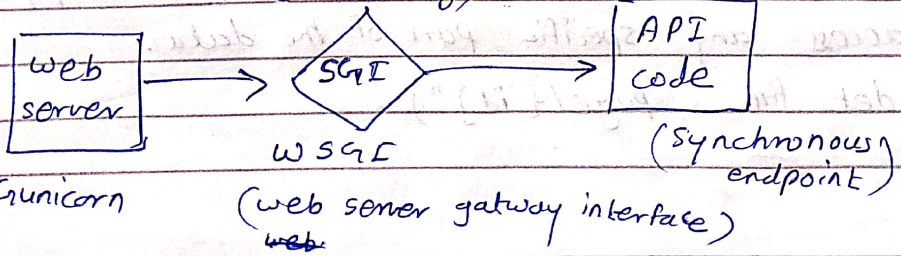
- What is API ?
  - It is set of rules & protocols that allows one software application to intract another.



- need of API ?
  - before API software in monolithic architecture like src → [frontend / backend] - frontend & backend code is tightly couple
  - Now, we will able to create frontend & backend seperatly.
    src — [frontend] → communicate by API.
    src — [backend]

- Fast API
  - It is modern & high performance web framework for building APIs with python.
  - fast API build by using two python libraries which is starlette & pydantic
    - starlette — It is manage how your API receive request & sends back responds by HTTP protocols.
    - pydantic — It used to check if data comming into your API is correct & in right data types / valid.

- flask (werkzeug)



disadvantage: - performance issue
- Include I/p wait time & high latency
- debugging challenging

FastAPI:

uvicorn — ASGI (starlette) (Asynchronous server gateway interface) — Asynchronous endpoint. (support to async & await API)

* HTTP method: (hypertext transfer protocol)
- It is standard action to used to communicate with servers over the HTTP protocol.
- There are four types to intract with dynamic software (CRUD).
  - C - create (POST
  - U - update (PUT)
  - R - retrive (GET)
  - D - delete (DELETE)

* Fast API setup.

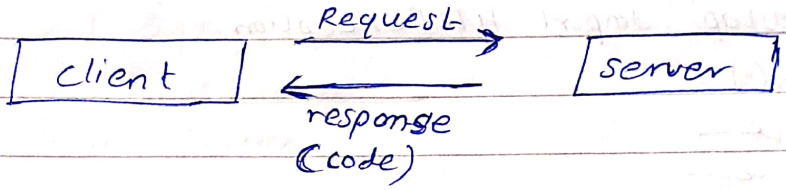
```
pip install fastapi
pip install pydantic
from fastapi import FastAPI
app = FastAPI()
```

* Params:
- It is use to pass the specific parameter/query to access any specific part of the data.

Syntax: `def fun_("page/{id}"):`

- Path function :
- The path() function in fastAPI is used to provide metadata, validation rules & documentation hints for path parameter. in your API endpoints.
- like → title, description etc.

- HTTP status code :
- this is three digit code returned by a web server to indicate the status of API.



- they help to client understand.
  - 2xx → successfully receive & processed
  - 3xx → redirection (further action needs to be taken)
  - 4xx → client error. (something wrong with the request from client).
  - 5xx → server error (wrong on server).

* Some example of code

1)
   - 200 : ok → get/post succeeded.
   - 201 : created → resource created after post
   - 204 : No content → success but no data ex (after delete request).

2)
   - 400 : bad request → invalid request (wrong data).
   - 401 : unauthorized → No / invalid authentication login required
   - 403 : forbidded → Authenticated but no permission (logged but not allow).

- 404 : Not found → data not present (DB).

3) - 500 : Internal server error. (broke on server)
- 502 : bad gateway → like fail to reach backend
- 503 : service unavailable → server is down or overloaded

- User define code
- to return this code use HTTPException function

4code // from fastapi import HTTPException
def fun():
    if ___
    ___
    raise HTTPException(status_code = x, detail = " ")

- Path & query parameter :
- path is use to provide the information related to the parameter

ex: def fun(id: str = Path(..., description = " ", example = " "
    title = " ")):
    ___ ___

- it is use to understand the client which type of data can fill & more about data which can pass.

query parameter :
- it is optional key-value pair appended to the end of URL to pass additional data for filtering, sorting, searching & pagination without altering the endpoints path itself.

Syntax : ~~def~~ /path ? obj = parameter & obj = parameter
↑ Syntax — use to add more query

- Use Query function present in fastapi

ex:
```
@app.get("/sort")
def sort_patient(sort_by: str = Query(..., description=" "),
                 order: str = Query(..., description=" ")):

    valid_field = ["height", "weight", "bmi"]
    if sort_by not in valid_field:
        raise HTTPException(status_code=400, details=
            f"Invalid field {valid_field}").
```

- Pydantic : data validation in Python

python is dynamic typing : ex: a=25, a="onkar"

- what is pydantic:
- It is powerfull data validation & setting management library based on python type hints.
- It is primary used in fastAPI & other modern python project.
- Data validation :- it is ensure that data is in valid form otherwise raise the error.
- Automatic type conversion : convert data type (ex- "37"=37) - make API more robust & user friendly
- it is fast & lightweight
- pydantic version-2 written in rust

```
from pydantic import BaseModel, Field, EmailStr, field_validator,
                     model_validator, computed_field
from typing import List, Dict, Annoted, Optional
```

- Some example:
- Create model

```
class Patient(BaseModel):
    name: str
    age: ~~field~~(
    age: int = Field(gt=0, lt=120)
    email: EmailStr
    linkedin: AnyUrl
```

* ~~Annoted~~ Annotated
- use to add description

ex: `name: Annotated[str, Field(max_length=30, title="", description="", examples=["onkar"])]`

- `field_validator("name")` : use to customized field for validation.

Syntax:

```
@field_validator("name")
@classmethod
def capital_name(cls, value):
    if value.islower():
        raise ValueError("name should be capital")
    return value
```

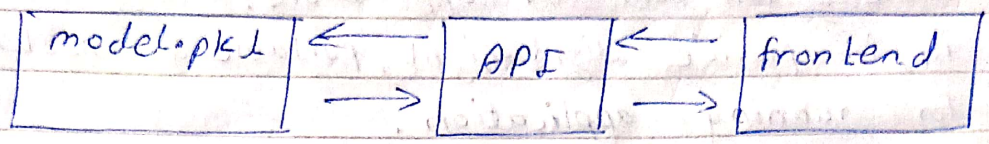
```
@model_validator(mode="after")
def custom_val(cls, model):
    if model.age > 60:
```

} customize all ~~data~~ model

```
@computed_field
@property
def bmi(self) -> float:
    bmi = round(self.weight / (self.height**2), 2)
    return bmi
```

→ use to add data/field by using user input data

* Serving ML model with FastAPI :



model building — FastAPI — Streamlit