8:53:53

Environment Variables :-

- Environment variables are variables that are configured on and specific to system/computer (say).

- Env variables are used to store confidential information like secret key, passwords, or deployment specific info like database name, url. Thus, they do not reflect in source code, and thus, becomes secure.

- When an env variable is configured on a system, any program/app running on the system can access it.

On Windows :-

- On search, we can type Environment variable, and from there, we can find user and system specific env. variables.

Accessing Env. Variables in cmd :-

- An environment variable named, say, Path, can be accessed in cmd as follow :-

```
echo %Path%
```

→ will print its value

to access, use % symbol before and after the environment variable name

To access environment variable in python, we use getenv() method of os module :-

```
import os
print( os.getenv("Path") )
```

Takes the name of the environment variable as string.
Returns the value contained in the env variable

Environment File :-

Going to system settings and creating the environment variables manually is cumbersome, specially when we have many env. variables.

So, do it at once, we use Environment File.

9:04:05

For robust development, in our code, when using env variable, we must use checks first, that the environment variable is set [else, our application might crash]

Value of environment variable will always be a string initially. So, we must perform necessary type conversion.

⇒ These checks and required type conversions for the environment variables can be done using Pydantic Models [schemas]

⇒ For it, we need to create a Pydantic model class, that will extend the BaseSettings class, imported from pydantic.
This class will contain the list of all the environment variables, that we need to set, as static variables (with their types defined).

Ex:

```
from pydantic import BaseSettings

class Settings(BaseSettings):
    database_username: str = "postgres"
    database_password: str = "Ultimate99"
    secret_key: str = "qwerty12345"
```

(default values)

These are the environment variables to be used

⇒ This Pydantic model will perform the checks, and the conversion to type mentioned.

⇒ In this model, the variable names made are not case-sensitive, and all are converted to lower-case.

⇒ By providing default values, we can work in our development env, without the need to set those environment variable values.

⇒ To now access the environment variables, we need to create an instance of the pydantic model class, and then access the static member variables of the object :-

```
settings = Settings()
print( settings.database_password)
```

pydantic model class

environment variable accessed as static member

⇒ When object is created as Settings() call, all the environment variables are read from system and set to the variables, and then the object is returned.

⇒ If default value is not set, and also variable is not set in system, it will throw an error, of type Validation Error.

⇒ For better organisation, we keep the code of environment variables, in config.py file, and in main.py, we do :-

```
from .config import settings
```

the object

we can import this in any file, where we need environment variables.

So, for our project, the schema will be like this ⟶

```python
from pydantic import BaseSettings


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 = ".env"
```

This nested class will make the pydantic model automatically import environment variables from file

it means current working dir

path of the file containing the environment variables

Now, in our FastAPI folder (where our project is kept), we create a .env named file. (dot is must)
and in it, create all the environment variables.
This is a easy way, for development.
[This .env file will be outside the folder, which contains the main.py file]

In general, all env variables are in UpperCase.

In this .env file, we assign values to env variables, without quotes like this :- ⟶

```
DATABASE_HOSTNAME=localhost
DATABASE_PORT=5432
DATABASE_PASSWORD=password123
DATABASE_NAME=fastapi
DATABASE_USERNAME=postgres
SECRET_KEY=09d25e094faa6ca2556c818166b7a95
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=30
```

without quotes

We add Config nested class in the Settings schema, and in it, assign path of our .env file to env_file variable, such that the schema automat. imports the environment variables from file.

The .env file must always be in the .gitignore file, while working with git ves.