Fast API – Python

Introduction to FastAPI framework:

FastAPI framework, high performance, easy to learn, fast to code, ready for production.

FastAPI is a high-performing web framework for building APIs with Python 3.7+ based on standard Python type hints. It helps developers build applications quickly and efficiently.

Key features of FastAPI:

- **Performance**: On par with NodeJS and the Go language.
- **Speed**: Increase the development speed 2-3X.
- **Easy**: Great editor support. Completion everywhere. Easy to learn and use.
- **Robust**: Production-ready code with automatic interactive documentation.
- **OpenAPI based**: Fully compatible with OpenAPI and JSON Schema.

Installation:

Requirements**:**

FastAPI stands on the shoulders of giants:

- Starlette for the web parts.
- Pydantic for the data parts.

pip install fastapi

You will also need an ASGI server, for production such as Uvicorn or Hypercorn.

pip install “uvicorn[standard]”

Project Layout:

```
FAST-API-CRUD-TODO
        main.py
        schemas.py
        models.py
        database.py
        todo.db
        requirements.txt
```

Models:

A model is a Python class that inherits from the Model class. The model class defines a new Kind of data store entity and the properties the Kind is expected to take.

We have considered sqlite (The Virtual Database Engine of SQLite) for **FAST-API-CRUD-TODO Implementation.**

models.py:

```python
from sqlalchemy import Column, Integer, String

from database import Base

# Define To Do class inheriting from Base
class ToDo(Base):
    __tablename__ = 'todos'
    id = Column(Integer, primary_key=True)
    task = Column(String(256))
```

database.py:

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

# Create a sqlite engine instance
engine = create_engine("sqlite:///todo.db")

# Create a DeclarativeMeta instance
Base = declarative_base()

# Create SessionLocal class from sessionmaker factory
SessionLocal = sessionmaker(bind=engine, expire_on_commit=False)
```

schemas.py:

```python
from pydantic import BaseModel
# Create ToDo Schema (Pydantic Model)
class ToDoCreate(BaseModel):
    task: str

# Complete ToDo Schema (Pydantic Model)
```

```python
class ToDoAll(BaseModel):
    id: int
    task: str
    class Config:
        orm_mode = True
```

requirements.txt:

```
uvicorn
fastapi
sqlalchemy
```

Decorators:

A decorator in Python is a function that takes another function as an argument and extends its behavior without explicitly modifying it.

Syntactic Decorator

```
        Syntax:

        @decorator
        def func(arg1, arg2, ...):
            pass
```

Example: find out **the execution time of a function** using a decorator .

API Calls:

main.py

```python
app = FastAPI()# Initialize app

# Get API

@app.get("/")
def root():
    return "Welcome to Todo Application. Built with FastAPI and ❤️. "
```

```python
# Post API

@app.post("/todo", response_model=schemas.ToDoAll,
status_code=status.HTTP_201_CREATED)

def create_todo(todo: schemas.ToDoCreate, session: Session =
Depends(get_session)):
    # create an instance of Todo Model
    todo_obj = models.ToDo(task = todo.task)
    # Add the object into our database Table
    session.add(todo_obj)
    session.commit()
    session.refresh(todo_obj)
    # return the todo object
    return todo_obj
```

```python
# PUT API

@app.put("/todo/{id}", response_model=schemas.ToDoAll)
def update_todo(id: int, task: str, session: Session = Depends(get_session)):

    # Fetch todo record using id from the table
    todo_obj = session.query(models.ToDo).get(id)

    # If the record is present in our DB table then update
    if todo_obj:
        todo_obj.task = task
        session.commit()

    # if todo item with given id does not exists, raise exception and return 404
not found response
    if not todo_obj:
        raise HTTPException(status_code=404, detail=f"todo item with id {id} not
found")

    return todo_obj
```

```python
# DELETE API

@app.delete("/todo/{id}", response_model = str)
def delete_todo(id: int, session: Session = Depends(get_session)):

    # Fetch todo record using id from the table
    todo_obj = session.query(models.ToDo).get(id)

    # Check if Todo record is present in our Database,If not then raise 404 error
    if todo_obj:
        session.delete(todo_obj)
        session.commit()
    else:
        raise HTTPException(status_code=404, detail=f"todo item with id {id} not
found")

    return f"Todo task with id {id} successfully deleted"
```

Complete main.py:

```python
from typing import List
from fastapi import FastAPI, status, HTTPException, Depends
from database import Base, engine, SessionLocal
from sqlalchemy.orm import Session
import schemas
import models

# Create the database
Base.metadata.create_all(engine)

# Initialize app
app = FastAPI()

# provides database session to each request upon calling
def get_session():
    session = SessionLocal()
    try:
        yield session
    finally:
        session.close()

@app.get("/")
def root():
    return "Welcome to Todo Application. Built with FastAPI and ❤️. "
```

```
@app.post("/todo", response_model=schemas.ToDoAll,
status_code=status.HTTP_201_CREATED)
def create_todo(todo: schemas.ToDoCreate, session: Session =
Depends(get_session)):

    # create an instance of Todo Model
    todo_obj = models.ToDo(task = todo.task)

    # Add the object into our database Table
    session.add(todo_obj)
    session.commit()
    session.refresh(todo_obj)

    # return the todo object
    return todo_obj

@app.get("/todo/{id}", response_model=schemas.ToDoAll)
def read_todo(id: int, session: Session = Depends(get_session)):

    # Fetch todo record using id from the table
    todo_obj = session.query(models.ToDo).get(id)

    # Check if there is record with the provided id, if not then Raise 404
Exception
    if not todo_obj:
        raise HTTPException(status_code=404, detail=f"todo item with id {id} not
found")

    return todo_obj

@app.put("/todo/{id}", response_model=schemas.ToDoAll)
def update_todo(id: int, task: str, session: Session = Depends(get_session)):

    # Fetch todo record using id from the table
    todo_obj = session.query(models.ToDo).get(id)

    # If the record is present in our DB table then update
    if todo_obj:
        todo_obj.task = task
        session.commit()

    # if todo item with given id does not exists, raise exception and return 404
not found response
    if not todo_obj:
```

```python
        raise HTTPException(status_code=404, detail=f"todo item with id {id} not 
found")

    return todo_obj

@app.delete("/todo/{id}", response_model = str)
def delete_todo(id: int, session: Session = Depends(get_session)):

    # Fetch todo record using id from the table
    todo_obj = session.query(models.ToDo).get(id)


    # Check if Todo record is present in our Database,If not then raise 404 error
    if todo_obj:
        session.delete(todo_obj)
        session.commit()
    else:
        raise HTTPException(status_code=404, detail=f"todo item with id {id} not 
found")

    return f"Todo task with id {id} successfully deleted"

@app.get("/todo", response_model = List[schemas.ToDoAll])
def read_todo_list(session: Session = Depends(get_session)):

    # get all todo items
    todo_list = session.query(models.ToDo).all()

    return todo_list
```

To execute the application:

```
 pip install -r requirements.txt
```

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

<u>Best Practices:</u>

- **Project Structure**. Consistent & predictable
- use **Pydantic** for data validation
- Use dependencies for data validation
- **Chain dependencies** :
  - Dependencies can use other dependencies and avoid code repetition for similar logic.
- Follow the **REST**: Developing **RESTful API** makes it easier to reuse dependencies in routes
- **Docs**
- Use **Pydantic's** BaseSettings for configs
- **SQLAlchemy**: Set DB keys naming convention

Demo Application: TODO App

References:

- https://www.sqlite.org/index.html
- https://www.scaler.com/topics/python/python-decorators/
- https://www.toptal.com/python/build-high-performing-apps-with-the-python-fastapi-framework
- https://www.postman.com/downloads/
- https://www.starlette.io/