**Fastapi\_todo Documentation**

**Open command prompt**

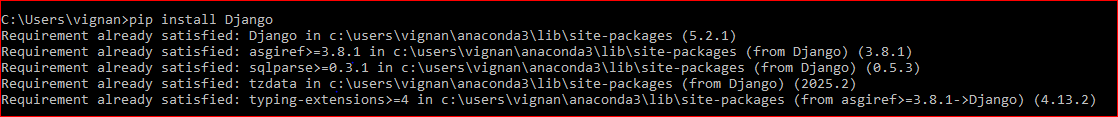
**Python --version**

It displays the **currently installed version of Python**.



**pip install Django**

It installs the **Django** web framework using Python's package manager, pip



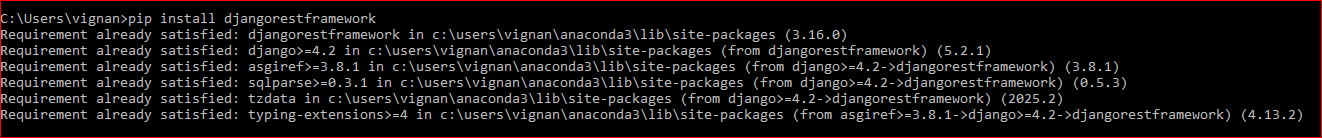
**django-admin –version**

This command checks the currently installed version of Django on your system



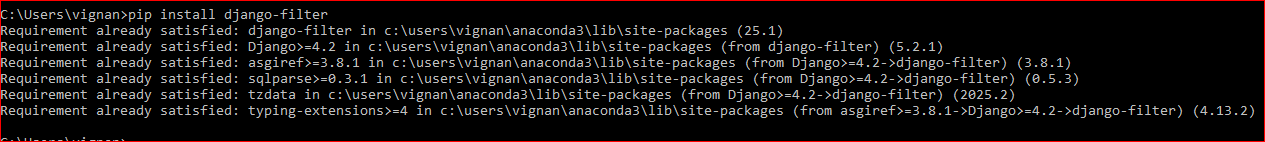
**pip install djangorestframework**

This installs **Django REST Framework (DRF)**, a powerful and flexible toolkit for building Web APIs in Django.



**pip install django-filter**

This installs **django-filter**, a powerful library used with **Django REST Framework** to easily add filtering capabilities to API views.



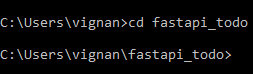
**django-admin startproject fastapi\_todo**

This command **creates a new Django project** named fastapi\_todo



**cd** **fastapi\_todo**

It changes the current directory to the newly created Django project folder named fastapi\_todo

.

**Again add the file fastapi\_todo in pycharm and continue doing project**

**Main.py**

from fastapi import FastAPI, HTTPException, Header, Request  
from fastapi.responses import JSONResponse  
from sqlmodel import Field, SQLModel, Session, create\_engine, select  
from cachetools import TTLCache  
import logging  
  
# --------------------- Logging Setup ---------------------  
logging.basicConfig(level=logging.INFO)  
  
# --------------------- App Init ---------------------  
app = FastAPI()  
  
# --------------------- Student Model ---------------------  
class Student(SQLModel, table=True):  
 id: int = Field(default=None, primary\_key=True)  
 name: str  
 age: int  
 grade: str  
  
# --------------------- Database Setup ---------------------  
sqlite\_file\_name = "students.db"  
engine = create\_engine(f"sqlite:///{sqlite\_file\_name}", echo=False)  
  
def create\_db\_and\_tables():  
 SQLModel.metadata.create\_all(engine)  
  
@app.on\_event("startup")  
def on\_startup():  
 create\_db\_and\_tables()  
  
# --------------------- Caching ---------------------  
cache = TTLCache(maxsize=100, ttl=30)  
  
# --------------------- HATEOAS + Caching + Async ---------------------  
@app.get("/students")  
async def get\_students(version: str = Header(default="v1")):  
 logging.info("Fetching students list")  
  
 if 'students' in cache:  
 logging.info("Serving from cache")  
 return cache['students']  
  
 with Session(engine) as session:  
 statement = select(Student)  
 results = session.exec(statement).all()  
  
 students\_with\_links = []  
 for student in results:  
 students\_with\_links.append({  
 "id": student.id,  
 "name": student.name,  
 "age": student.age,  
 "grade": student.grade,  
 "links": [  
 {"rel": "self", "href": f"/students/{student.id}"},  
 {"rel": "update", "href": f"/students/{student.id}"},  
 {"rel": "delete", "href": f"/students/{student.id}"}  
 ]  
 })  
 cache['students'] = students\_with\_links  
 return students\_with\_links  
  
# --------------------- Add Student (Idempotency) ---------------------  
@app.post("/students")  
def add\_student(student: Student):  
 with Session(engine) as session:  
 existing = session.exec(select(Student).where(Student.id == student.id)).first()  
 if existing:  
 raise HTTPException(status\_code=409, detail="Student with this ID already exists")  
  
 session.add(student)  
 session.commit()  
 session.refresh(student)  
 return student  
  
# --------------------- Webhook Receiver ---------------------  
@app.post("/webhook")  
def webhook\_receiver(data: dict):  
 logging.info(f"Webhook received: {data}")  
 return {"status": "received", "data": data}  
  
# --------------------- Versioning URI Based ---------------------  
@app.get("/v1/students")  
def v1\_students():  
 return {"version": "v1", "message": "Using v1 structure"}  
  
@app.get("/v2/students")  
def v2\_students():  
 return {"version": "v2", "message": "Using v2 structure with new features"}  
  
# --------------------- Deprecated Endpoint ---------------------  
@app.get("/students-deprecated")  
def deprecated\_students():  
 return JSONResponse(  
 content={"message": "This endpoint is deprecated. Please use /students"},  
 headers={"Deprecation": "true"}  
 )  
  
# --------------------- Get Single Student ---------------------  
@app.get("/students/{student\_id}")  
def get\_student(student\_id: int):  
 with Session(engine) as session:  
 student = session.get(Student, student\_id)  
 if not student:  
 raise HTTPException(status\_code=404, detail="Student not found")  
 return student  
  
# --------------------- Update Student ---------------------  
@app.put("/students/{student\_id}")  
def update\_student(student\_id: int, updated\_data: Student):  
 with Session(engine) as session:  
 student = session.get(Student, student\_id)  
 if not student:  
 raise HTTPException(status\_code=404, detail="Student not found")  
 student.name = updated\_data.name  
 student.age = updated\_data.age  
 student.grade = updated\_data.grade  
 session.add(student)  
 session.commit()  
 session.refresh(student)  
 return student  
  
# --------------------- Delete Student ---------------------  
@app.delete("/students/{student\_id}")  
def delete\_student(student\_id: int):  
 with Session(engine) as session:  
 student = session.get(Student, student\_id)  
 if not student:  
 raise HTTPException(status\_code=404, detail="Student not found")  
 session.delete(student)  
 session.commit()  
 return {"message": f"Student {student\_id} deleted"}

**database.py**

from sqlmodel import SQLModel, create\_engine  
  
DATABASE\_URL = "sqlite:///students.db"  
engine = create\_engine(DATABASE\_URL, echo=True)  
  
  
def create\_db\_and\_tables():  
 SQLModel.metadata.create\_all(engine)

**models.py**

from sqlmodel import SQLModel, Field  
from typing import Optional  
  
class Student(SQLModel, table=True):  
 id: Optional[int] = Field(default=None,primary\_key=True)  
 name: str  
 age: int  
 grade: str

**test.py**

from fastapi.testclient import TestClient  
from main import app  
  
client = TestClient(app)  
  
  
def test\_create\_student\_success():  
 response = client.post("/students", json={  
 "name": "Alice",  
 "age": 21,  
 "grade": "A"  
 })  
 assert response.status\_code == 201  
 assert response.json()["message"] == "Student added successfully"  
  
  
def test\_get\_all\_students():  
 response = client.get("/students")  
 print("Status Code:", response.status\_code)  
 print("Response JSON:", response.json())  
 assert response.status\_code == 200  
 assert "data" in response.json()  
  
  
  
def test\_partial\_update\_student\_success():  
 response = client.patch("/students/0",json={  
 "age": 22  
 })  
 assert response.status\_code == 200  
 assert response.json()["message"] == "Student updated successfully"  
 assert response.json()["data"]["age"] == 22  
 print("Status code:",response.status\_code)  
 print("Response JSON:", response.json())  
  
  
def test\_partial\_update\_invalid\_id():  
 response = client.patch("/students/999", json={"age":30})  
 assert response.status\_code == 404  
 assert "Student with ID" in response.json()["message"]  
 print("Status Code:", response.status\_code)  
 print("Response JSON:", response.json())  
  
  
def test\_create\_student\_invalid\_data():  
 response = client.post("/students", json={  
 "name": "Bob"  
 })  
 assert response.status\_code == 422  
 print("Status Code:", response.json())  
 print("Response JSON:", response.json())  
  
  
def test\_patch\_student\_invalid\_data():  
 response = client.patch("/students/0", json={  
 "age": "twenty"  
 })  
 assert response.status\_code == 422  
 print("Status Code:", response.status\_code)  
 print("Response JSON:", response.json())

**To run the code:**

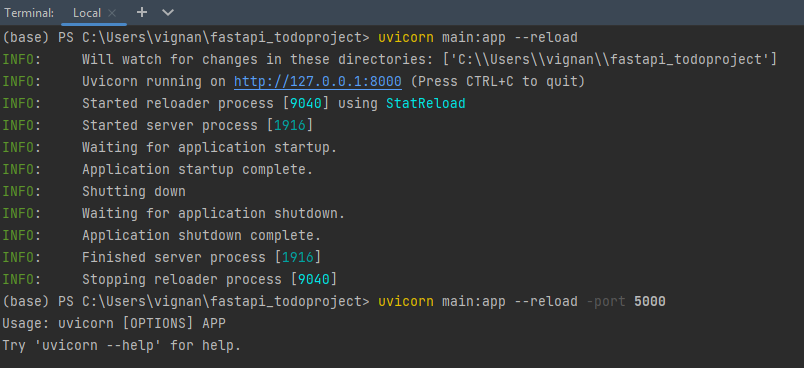
uvicorn main:app --reload

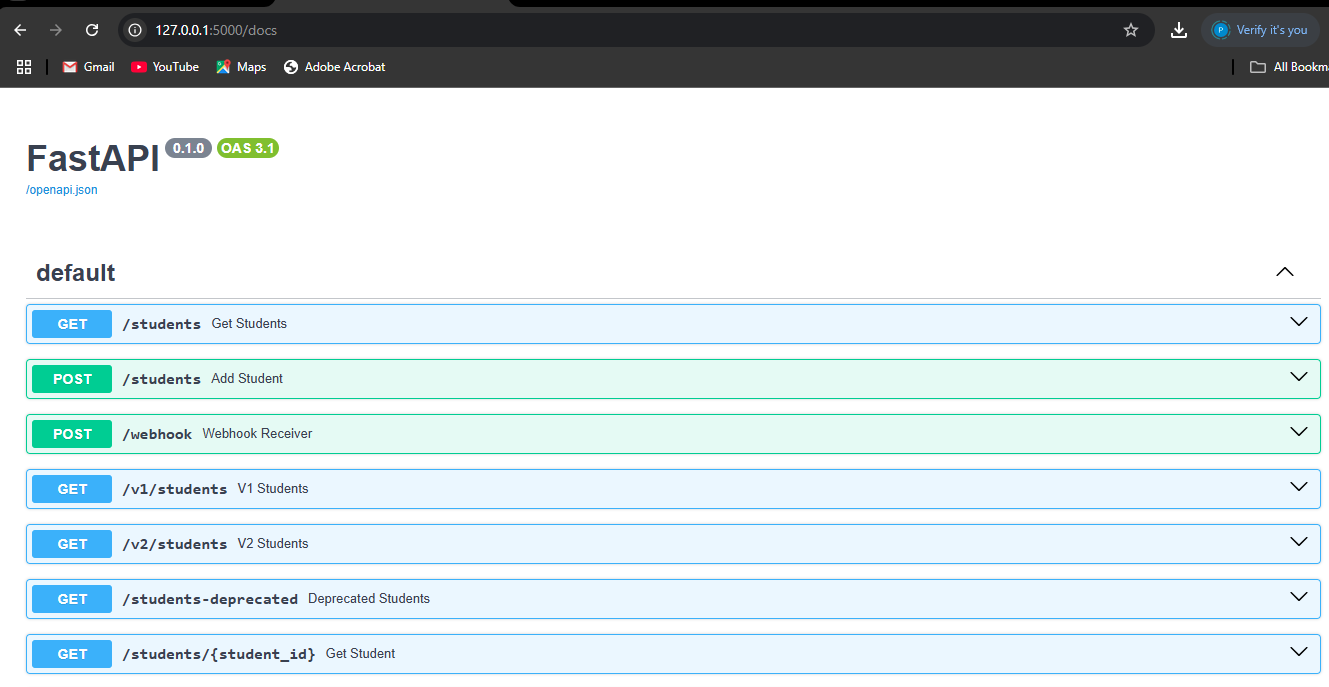
**Now link will be generated**

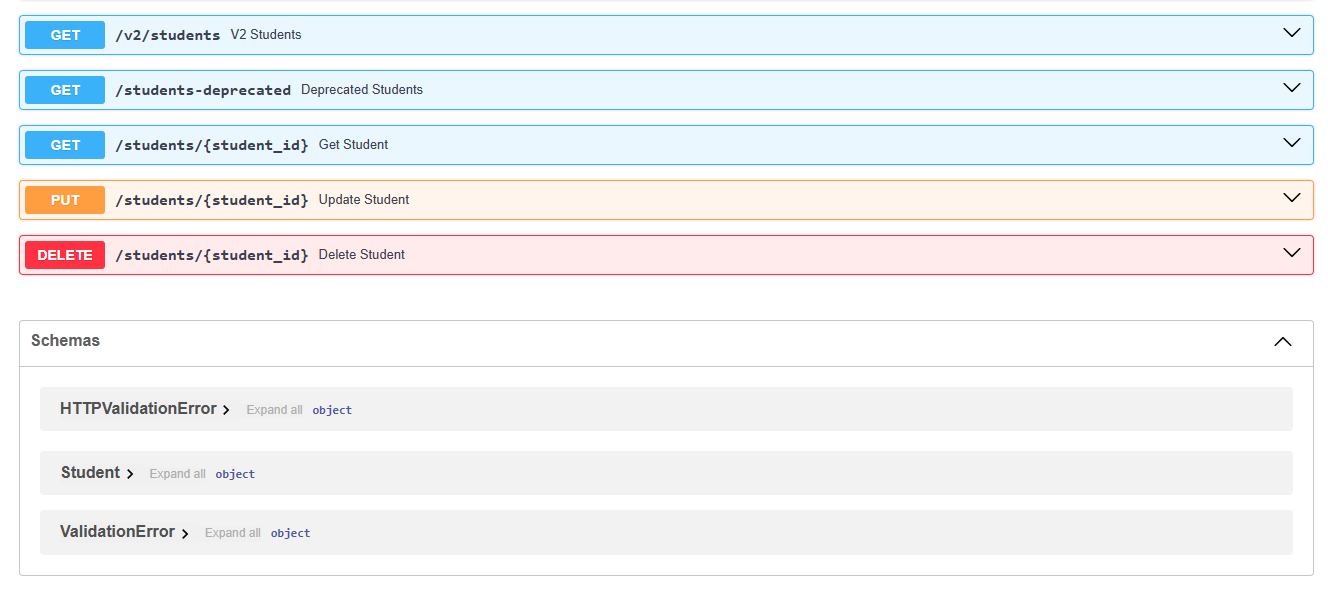
http://127.0.0.1:8000

**After opening link you will be able to see project**

**Output:-**







**Structure:**

