CRUD app using FastAPI + SQLite + SQLAlchemy.

---

1. Install dependencies

```
pip install fastapi uvicorn sqlalchemy
```

---

2. Project structure

```
main.py
```

---

3. Complete working example

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker, declarative_base

# Database setup (SQLite)
DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(DATABASE_URL, connect_args={"check_same_thread":
False})

SessionLocal = sessionmaker(bind=engine)
Base = declarative_base()

# Model
class UserDB(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String)
    email = Column(String)

# Create tables
Base.metadata.create_all(bind=engine)

# Pydantic schema
class User(BaseModel):
    name: str
    email: str

app = FastAPI()
```

```python
# Dependency
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

# CREATE
@app.post("/users/")
def create_user(user: User):
    db = next(get_db())
    db_user = UserDB(name=user.name, email=user.email)
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

# READ ALL
@app.get("/users/")
def get_users():
    db = next(get_db())
    return db.query(UserDB).all()

# READ ONE
@app.get("/users/{user_id}")
def get_user(user_id: int):
    db = next(get_db())
    user = db.query(UserDB).filter(UserDB.id == user_id).first()
    if not user:
        raise HTTPException(status_code=404, detail="User not found")
    return user

# UPDATE
@app.put("/users/{user_id}")
def update_user(user_id: int, updated_user: User):
    db = next(get_db())
    user = db.query(UserDB).filter(UserDB.id == user_id).first()
    if not user:
        raise HTTPException(status_code=404, detail="User not found")

    user.name = updated_user.name
    user.email = updated_user.email
    db.commit()
    return user

# DELETE
@app.delete("/users/{user_id}")
def delete_user(user_id: int):
    db = next(get_db())
    user = db.query(UserDB).filter(UserDB.id == user_id).first()
    if not user:
        raise HTTPException(status_code=404, detail="User not found")

    db.delete(user)
    db.commit()
    return {"message": "User deleted"}
```

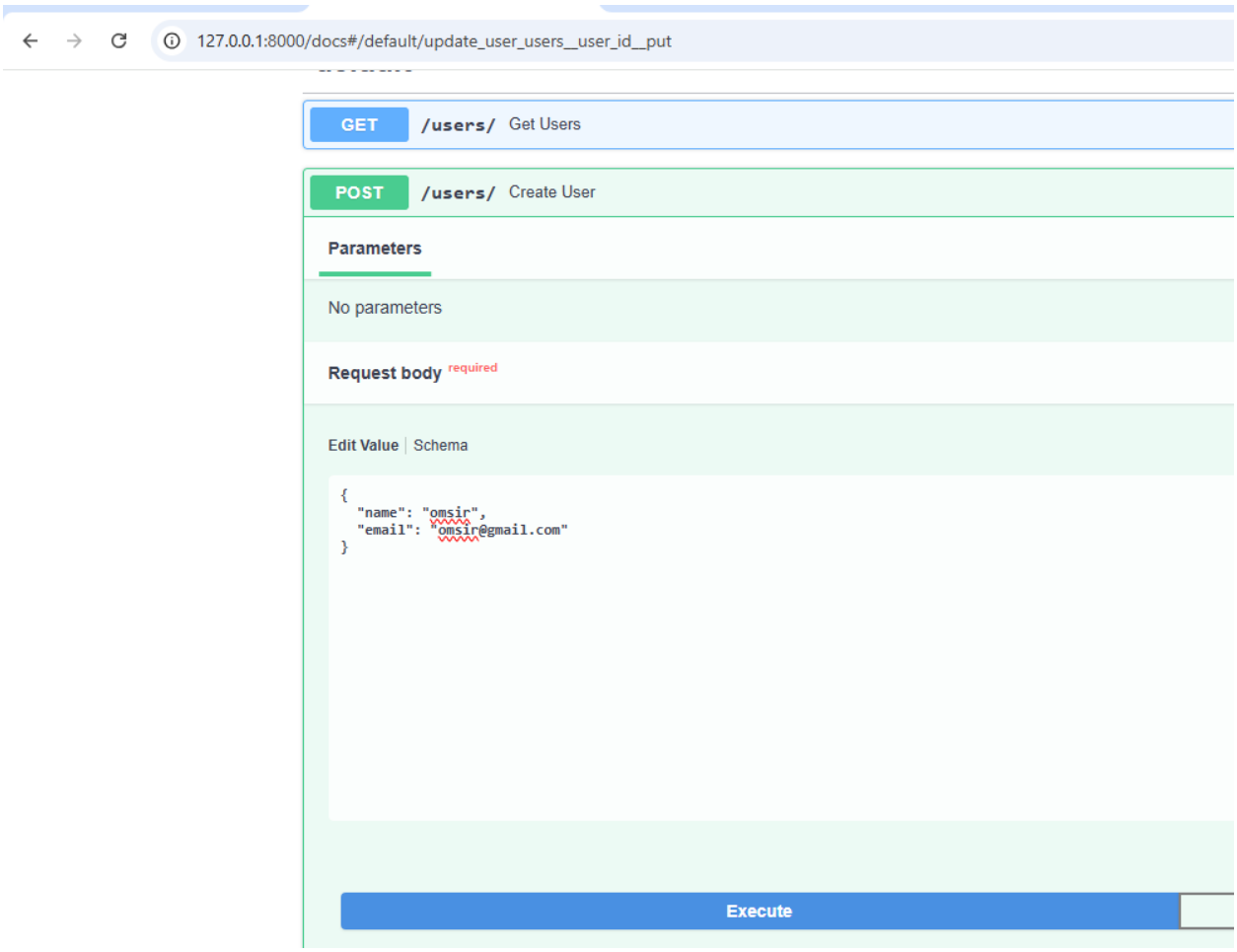
4. Run the app

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

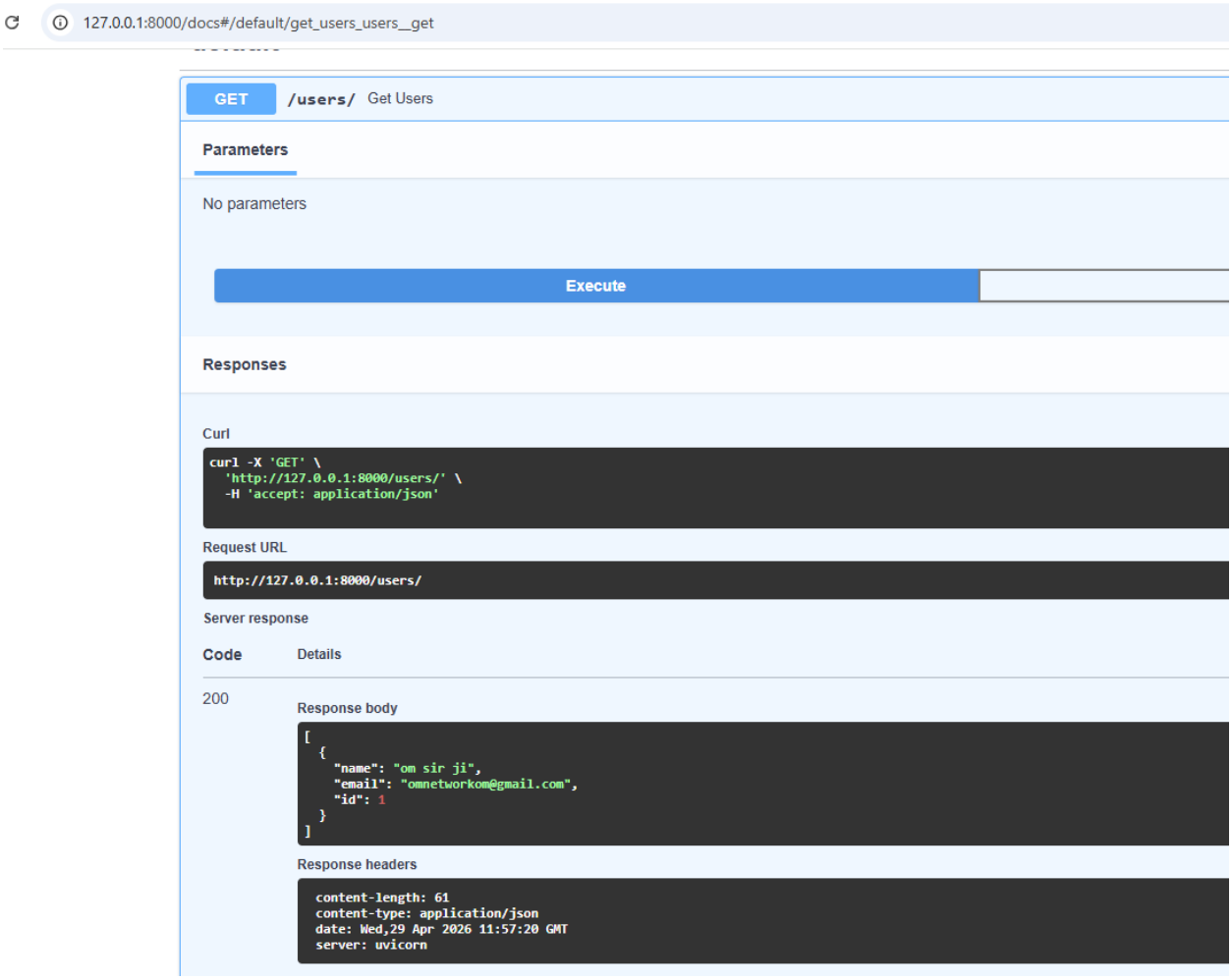
Open:-

 http://127.0.0.1:8000/docs

Post new users data:-



Get users details :-



Here's a **complete CRUD app using HTML + Jinja2 with FastAPI + SQLite + SQLAlchemy** (no separate frontend JS needed).

This is the "traditional server-rendered" style.

---

1. Project structure

```
project/
├── main.py
├── templates/
│      ├── index.html
│      ├── edit.html
├── test.db (auto-created)
```

---

2. Install dependencies

```
pip install fastapi uvicorn sqlalchemy jinja2 python-multipart
```

---

3. Backend (main.py)

```
from fastapi import FastAPI, Request, Form, HTTPException
from fastapi.responses import HTMLResponse, RedirectResponse
from fastapi.templating import Jinja2Templates
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker, declarative_base

# App
app = FastAPI()
templates = Jinja2Templates(directory="templates")

# DB
DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(DATABASE_URL, connect_args={"check_same_thread":
False})
SessionLocal = sessionmaker(bind=engine)
Base = declarative_base()

# Model
class UserDB(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String)
    email = Column(String)

Base.metadata.create_all(bind=engine)

def get_db():
    return SessionLocal()
```

```python
# READ (list users)
@app.get("/", response_class=HTMLResponse)
def read_users(request: Request):
    db = get_db()
    users = db.query(UserDB).all()
    return templates.TemplateResponse("index.html", {
        "request": request,
        "users": users
    })

# CREATE
@app.post("/add")
def add_user(name: str = Form(...), email: str = Form(...)):
    db = get_db()
    user = UserDB(name=name, email=email)
    db.add(user)
    db.commit()
    return RedirectResponse("/", status_code=303)

# DELETE
@app.get("/delete/{user_id}")
def delete_user(user_id: int):
    db = get_db()
    user = db.query(UserDB).filter(UserDB.id == user_id).first()
    if user:
        db.delete(user)
        db.commit()
    return RedirectResponse("/", status_code=303)

# EDIT PAGE
@app.get("/edit/{user_id}", response_class=HTMLResponse)
def edit_page(request: Request, user_id: int):
    db = get_db()
    user = db.query(UserDB).filter(UserDB.id == user_id).first()
    return templates.TemplateResponse("edit.html", {
        "request": request,
        "user": user
    })

# UPDATE
@app.post("/update/{user_id}")
def update_user(user_id: int, name: str = Form(...), email: str = Form(...)):
    db = get_db()
    user = db.query(UserDB).filter(UserDB.id == user_id).first()

    if not user:
        raise HTTPException(status_code=404)

    user.name = name
    user.email = email
    db.commit()

    return RedirectResponse("/", status_code=303)
```

4. Frontend templates

□ templates/index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>FastAPI Jinja CRUD</title>
</head>
<body>

<h2>Create User</h2>

<form action="/add" method="post">
    <input name="name" placeholder="Name" required>
    <input name="email" placeholder="Email" required>
    <button type="submit">Add</button>
</form>

<hr>

<h2>Users List</h2>

<ul>
    {% for user in users %}
        <li>
            {{ user.name }} - {{ user.email }}

            <a href="/edit/{{ user.id }}">Edit</a>
            <a href="/delete/{{ user.id }}">Delete</a>
        </li>
    {% endfor %}
</ul>

</body>
</html>
```

✏ templates/edit.html

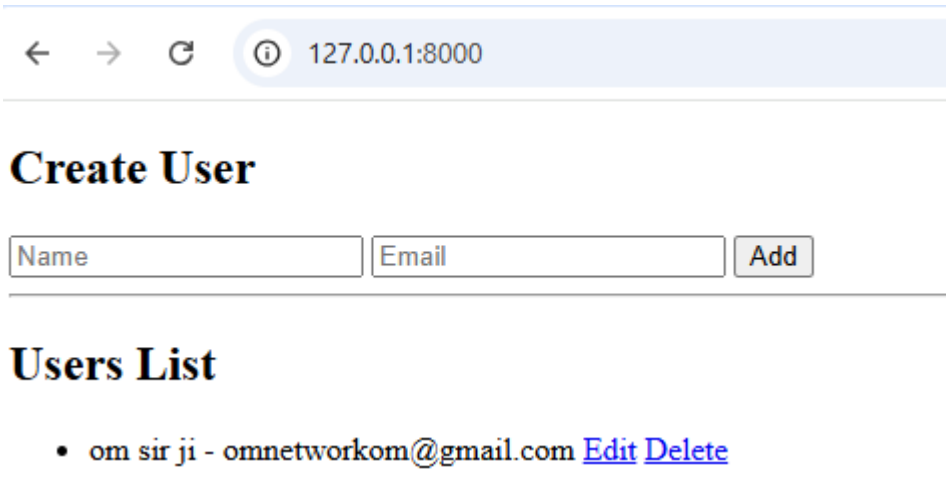
```
<!DOCTYPE html>
<html>
<head>
    <title>Edit User</title>
</head>
<body>

<h2>Edit User</h2>

<form action="/update/{{ user.id }}" method="post">
    <input name="name" value="{{ user.name }}" required>
    <input name="email" value="{{ user.email }}" required>
    <button type="submit">Update</button>
</form>

<br>
<a href="/">Back</a>

</body>
</html>
```

---

5. Run the app

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

Open:
 http://127.0.0.1:8000/



Here's a **complete working CRUD app** using:

- FastAPI (backend)
- SQLite (database)
- SQLAlchemy (ORM)
- HTML + JavaScript (`fetch`) frontend

---

1. Project structure

```
project/
├── main.py
├── test.db (auto-created)
├── index.html
```

□ 2. Install dependencies

```
pip install fastapi uvicorn sqlalchemy
```

---

3. Backend (main.py)

```
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker, declarative_base

# DB setup
DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(DATABASE_URL, connect_args={"check_same_thread":
False})
SessionLocal = sessionmaker(bind=engine)
Base = declarative_base()

# Model
class UserDB(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String)
    email = Column(String)

Base.metadata.create_all(bind=engine)

# Schema
class User(BaseModel):
    name: str
    email: str
```

```
app = FastAPI()

# allow frontend (CORS)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)

# helper DB
def get_db():
    db = SessionLocal()
    try:
        return db
    finally:
        pass

# CREATE
@app.post("/users")
def create_user(user: User):
    db = get_db()
    db_user = UserDB(name=user.name, email=user.email)
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

# READ ALL
@app.get("/users")
def get_users():
    db = get_db()
    return db.query(UserDB).all()

# UPDATE
@app.put("/users/{user_id}")
def update_user(user_id: int, user: User):
    db = get_db()
    db_user = db.query(UserDB).filter(UserDB.id == user_id).first()

    if not db_user:
        raise HTTPException(status_code=404, detail="User not found")

    db_user.name = user.name
    db_user.email = user.email
    db.commit()
    return db_user

# DELETE
@app.delete("/users/{user_id}")
def delete_user(user_id: int):
    db = get_db()
    user = db.query(UserDB).filter(UserDB.id == user_id).first()

    if not user:
        raise HTTPException(status_code=404, detail="User not found")
```

```
    db.delete(user)
    db.commit()
    return {"message": "deleted"}
```

---

4. Frontend (index.html)

```
<!DOCTYPE html>
<html>
<head>
    <title>FastAPI CRUD</title>
</head>
<body>

<h2>User CRUD</h2>

<!-- CREATE -->
<h3>Create User</h3>
<input id="name" placeholder="Name">
<input id="email" placeholder="Email">
<button onclick="createUser()">Create</button>

<hr>

<!-- LIST -->
<h3>Users</h3>
<button onclick="getUsers()">Load Users</button>
<ul id="userList"></ul>

<script>
const API = "http://127.0.0.1:8000/users";

// CREATE
async function createUser() {
    await fetch(API, {
        method: "POST",
        headers: {"Content-Type": "application/json"},
        body: JSON.stringify({
            name: document.getElementById("name").value,
            email: document.getElementById("email").value
        })
    });

    getUsers();
}

// READ
async function getUsers() {
    const res = await fetch(API);
    const data = await res.json();

    const list = document.getElementById("userList");
    list.innerHTML = "";

    data.forEach(user => {
        list.innerHTML += `
```

```
            <li>
                ${user.name} - ${user.email}
                <button onclick="deleteUser(${user.id})">Delete</button>
                <button onclick="updateUser(${user.id})">Update</button>
            </li>
        `;
    });
}

// UPDATE (simple prompt version)
async function updateUser(id) {
    const name = prompt("New name:");
    const email = prompt("New email:");

    await fetch(`${API}/${id}`, {
        method: "PUT",
        headers: {"Content-Type": "application/json"},
        body: JSON.stringify({name, email})
    });

    getUsers();
}

// DELETE
async function deleteUser(id) {
    await fetch(`${API}/${id}`, {
        method: "DELETE"
    });

    getUsers();
}
</script>

</body>
</html>
```

🚀 5. Run the app

Start backend:

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

Open frontend:

Just open `index.html` in browser

What you get

✔ Create user
✔ Read all users
✔ Update user (via prompt)
✔ Delete user
✔ Full REST API + frontend integration

Important notes

- CORS is enabled so HTML can talk to FastAPI
- SQLite database file is auto-created
- This is a **learning-level full-stack setup**