JOMHACK
INSPIRE INNOVATION

PYTHON
BOOTCAMP

www.jomhack.com

FASTAPI (SQLITE)

FastAPI:

- Web framework for building APIs (endpoint)
- Type hints support
- Interactive API documentation (Swagger UI)
- pip install fastapi uvicorn

Pydantic:

- Data validation and parsing using Python type hints.
- IDE support with type hints
- JSON serialization/deserialization (Python to JSON to Python)
- pip install pydantic

FASTAPI (SQLITE)

JOMHACK
INSPIRE INNOVATION

Hypertext Transfer Protocol(HTTP) Methods:

- @app.get() - GET requests (Read)
- @app.post() - POST requests (Create)
- @app.put() - PUT requests (Update)
- @app.delete() - DELETE requests (Delete)
- @app.patch() - PATCH requests (Partial Update)

FASTAPI (SQLITE)

Imports & Initialization:

```python
from fastapi import FastAPI, HTTPException, status
from pydantic import BaseModel, EmailStr
from typing import List, Optional
from datetime import datetime
import sqlite3
from sql_database import DatabaseManager

app = FastAPI(title="SQLite Database API", version="1.0.0")
```

Basemodel:

```python
# Pydantic models for request/response
class UserCreate(BaseModel):
    name: str
    email: EmailStr
    age: int

class UserResponse(BaseModel):
    id: int
    name: str
    email: str
    age: int
    created_at: str

class PostCreate(BaseModel):
    user_id: int
    title: str
    content: str

class PostResponse(BaseModel):
    id: int
    user_id: int
    title: str
    content: str
    created_at: str

class PostResponseForUser(BaseModel):
    id: int
    title: str
    content: str
    created_at: str

# Initialize database
db = DatabaseManager()
```

FASTAPI (SQLITE)

post(CREATE):

```python
@app.get("/")
async def root():
    return {"message": "SQLite Database API", "version": "1.0.0"}

@app.post("/users/", response_model=dict, status_code=status.HTTP_201_CREATED)
async def create_user(user: UserCreate):
    """Create a new user"""
    try:
        user_id = db.create_user(user.name, user.email, user.age)
        if user_id:
            return {"message": "User created successfully", "user_id": user_id}
        else:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Failed to create user. Email might already exist."
            )
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

get(READ):

```python
@app.get("/users/", response_model=List[UserResponse])
async def get_all_users():
    """Get all users"""
    try:
        users = db.get_all_users()
        return [
            UserResponse(
                id=user[0],
                name=user[1],
                email=user[2],
                age=user[3],
                created_at=user[4]
            )
            for user in users
        ]
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

FASTAPI (SQLITE)

JOMHACK
INSPIRE INNOVATION

get(READ):

```python
@app.get("/users/{user_id}", response_model=UserResponse)
async def get_user(user_id: int):
    """Get a specific user by ID"""
    try:
        with sqlite3.connect(db.db_name) as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM users WHERE id = ?", (user_id,))
            user = cursor.fetchone()

        if not user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="User not found"
            )

        return UserResponse(
            id=user[0],
            name=user[1],
            email=user[2],
            age=user[3],
            created_at=user[4]
        )
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

post(CREATE):

```python
@app.post("/posts/", response_model=dict, status_code=status.HTTP_201_CREATED)
async def create_post(post: PostCreate):
    """Create a new post"""
    try:
        # Check if user exists
        with sqlite3.connect(db.db_name) as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT id FROM users WHERE id = ?", (post.user_id,))
            if not cursor.fetchone():
                raise HTTPException(
                    status_code=status.HTTP_404_NOT_FOUND,
                    detail="User not found"
                )

        post_id = db.create_post(post.user_id, post.title, post.content)
        if post_id:
            return {"message": "Post created successfully", "post_id": post_id}
        else:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Failed to create post"
            )
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

FASTAPI (SQLITE)

get(READ):

```python
@app.get("/users/{user_id}/posts", response_model=List[PostResponseForUser])
async def get_user_posts(user_id: int):
    """Get all posts by a specific user"""
    try:
        # Check if user exists
        with sqlite3.connect(db.db_name) as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT id FROM users WHERE id = ?", (user_id,))
            if not cursor.fetchone():
                raise HTTPException(
                    status_code=status.HTTP_404_NOT_FOUND,
                    detail="User not found"
                )

        posts = db.get_user_posts(user_id)
        return [
            PostResponseForUser(
                id=post[0],
                title=post[1],
                content=post[2],
                created_at=post[3]
            )
            for post in posts
        ]
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

get(READ):

```python
@app.get("/posts/", response_model=List[PostResponse])
async def get_all_posts():
    """Get all posts"""
    try:
        with sqlite3.connect(db.db_name) as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM posts ORDER BY created_at DESC")
            posts = cursor.fetchall()

        return [
            PostResponse(
                id=post[0],
                user_id=post[1],
                title=post[2],
                content=post[3],
                created_at=post[4]
            )
            for post in posts
        ]
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

FASTAPI (SQLITE)

JOMHACK
INSPIRE INNOVATION

delete(DELETE):

```python
@app.delete("/users/{user_id}", response_model=dict)
async def delete_user(user_id: int):
    """Delete a user and all their posts"""
    try:
        # Check if user exists
        with sqlite3.connect(db.db_name) as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT id FROM users WHERE id = ?", (user_id,))
            if not cursor.fetchone():
                raise HTTPException(
                    status_code=status.HTTP_404_NOT_FOUND,
                    detail="User not found"
                )

        success = db.delete_user(user_id)
        if success:
            return {"message": "User deleted successfully"}
        else:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Failed to delete user"
            )
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

delete(DELETE):

```python
@app.delete("/posts/{post_id}", response_model=dict)
async def delete_post(post_id: int):
    """Delete a specific post"""
    try:
        with sqlite3.connect(db.db_name) as conn:
            cursor = conn.cursor()
            cursor.execute("DELETE FROM posts WHERE id = ?", (post_id,))

            if cursor.rowcount == 0:
                raise HTTPException(
                    status_code=status.HTTP_404_NOT_FOUND,
                    detail="Post not found"
                )

        return {"message": "Post deleted successfully"}
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```