

# ⚙️ Complete `create_user` Flow in FastAPI

## ✓ 1. `schemas.py` — Request & Response Models

```
from pydantic import BaseModel, EmailStr

class UserCreate(BaseModel):
    name: str
    email: EmailStr
    password: str

class UserResponse(BaseModel):
    id: int
    name: str
    email: EmailStr

class Config:
    orm_mode = True # important to work with ORM objects
```

---

## ✓ 2. `models.py` — SQLAlchemy User Model

```
from sqlalchemy import Column, Integer, String
from database import Base

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, nullable=False)
    email = Column(String, unique=True, index=True)
    password = Column(String, nullable=False) # store hashed password
```

---

## ✓ 3. `database.py` — DB Setup + Dependency

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

DATABASE_URL = "sqlite:///./test.db" # or PostgreSQL, MySQL etc.

engine = create_engine(DATABASE_URL, connect_args={"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

```
Base = declarative_base()

# Dependency to use in routes
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

---

#### ✓ 4. crud.py — Business Logic (Service Layer)

```
from sqlalchemy.orm import Session
from models import User
from schemas import UserCreate
from utils import hash_password # if using hashing

def create_user(db: Session, user: UserCreate):
    db_user = User(
        name=user.name,
        email=user.email,
        password=hash_password(user.password) # optional hashing
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user
```

---

#### ✓ 5. utils.py — (Optional) Password Hashing

```
from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def hash_password(password: str) -> str:
    return pwd_context.hash(password)
```

---

#### ✓ 6. main.py — FastAPI Route Using Dependency

```
from fastapi import FastAPI, Depends, HTTPException
from sqlalchemy.orm import Session
```

```
import models, schemas, crud
from database import engine, get_db

models.Base.metadata.create_all(bind=engine)

app = FastAPI()

@app.post("/users/", response_model=schemas.UserResponse)
def create_user(user: schemas.UserCreate, db: Session = Depends(get_db)):
    db_user = crud.create_user(db, user)
    return db_user
```

---

## Full Request-Response Flow

### 1. Request:

```
POST /users/
{
  "name": "Darshan",
  "email": "darshan@example.com",
  "password": "secret123"
}
```

### 2. FastAPI Route:

- Validates using `UserCreate`
- Injects `db` session via `Depends(get_db)`

### 3. Business Logic (CRUD Layer):

- Calls `create_user(db, user)`
- Hashes password (optional)
- Adds user to DB

### 4. SQLAlchemy Model:

- Maps to `users` table

### 5. Response: Returns `UserResponse` model (hides password!)

---

## Bonus: Folder Structure

```
project/
|
├─ main.py
```

- models.py
- schemas.py
- crud.py
- database.py
- utils.py
- requirements.txt

## ✓ Features Used

Concept	Usage
🧩 Pydantic Models	<code>UserCreate</code> , <code>UserResponse</code>
🗄️ SQLAlchemy ORM	<code>User</code> model in <code>models.py</code>
🧬 Dependency Injection	<code>Depends(get_db)</code> in route
🔗 DB Abstraction	<code>crud.create_user(db, user)</code>
🔒 Optional Security	<code>hash_password</code> via <code>passlib</code>

## 🔄 1. FastAPI `create_user` Flow – Arrow Diagram & Explanation

```
CLIENT (e.g., Postman, React) sends JSON payload
|
V
POST /users/          ♦ Request: { "name": "Darshan", "email": "...", "password":
"1234" }
|
V
@app.post("/users/")
def create_user(user: UserCreate, db: Session = Depends(get_db)):
    |
    ├── Validates body using → 🧩 Pydantic `UserCreate`
    └── Injects DB session → 🧬 Depends(get_db)
        |
        V
        Calls → crud.create_user(db, user)
                |
                ├── Creates new User ORM object
                ├── Hashes password (optional)
                └── db.add(), db.commit(), db.refresh()
                    |
                    V
                    Returns → UserResponse (without password)
```

```
|
V
Client receives JSON:
{
  "id": 1,
  "name": "Darshan",
  "email": "darshan@example.com"
}
```



## Data Flow Breakdown:

- **Pydantic Schema:** Input validated (`UserCreate`)
- **Dependency Injection:** `get_db` provides a DB session
- **Business Logic (CRUD):** `create_user` interacts with DB
- **ORM (SQLAlchemy):** Handles table & row management
- **Response Model:** Returns only `id`, `name`, `email`



## 2. JWT Authentication Flow in FastAPI

Let's now explore how **JWT Login works** using password validation, token generation, and protection of private routes.



### JWT Flow – Arrow Diagram

```
CLIENT sends login credentials:
POST /login
{ "username": "darshan", "password": "1234" }
|
V
@app.post("/login")
|
├── Verifies username in DB
├── Validates password using hash check
├── If valid:
|   |
|   V
|   Generate JWT using PyJWT
|   Return token → { "access_token": "<JWT>", "token_type": "bearer" }
```

Then, on protected routes:

```
CLIENT calls:
GET /profile with Header: Authorization: Bearer <JWT>
```

```
|
V
Depends(get_current_user)
|— Decodes JWT
|— Verifies signature
|— Gets user_id/email from payload
|— Loads user from DB
|
V
Returns secure data
```

---

## ✓ Code Breakdown

### 1 Generate Token – `auth.py`

```
from datetime import datetime, timedelta
from jose import JWTError, jwt

SECRET_KEY = "your-secret"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

def create_access_token(data: dict):
    to_encode = data.copy()
    expire = datetime.utcnow() + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

---

### 2 Login Route – `routes.py`

```
from fastapi import APIRouter, Depends, HTTPException
from fastapi.security import OAuth2PasswordRequestForm
from sqlalchemy.orm import Session
from auth import create_access_token
from utils import verify_password
from models import User
from database import get_db

router = APIRouter()

@router.post("/login")
def login(form_data: OAuth2PasswordRequestForm = Depends(), db: Session = Depends(get_db)):
    user = db.query(User).filter(User.email == form_data.username).first()
```

```

if not user or not verify_password(form_data.password, user.password):
    raise HTTPException(status_code=401, detail="Invalid credentials")

access_token = create_access_token(data={"sub": user.email})
return {"access_token": access_token, "token_type": "bearer"}

```

### 3 Protected Route

```

from fastapi import Depends
from fastapi.security import OAuth2PasswordBearer
from jose import JWTError, jwt
from sqlalchemy.orm import Session

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/login")

def get_current_user(token: str = Depends(oauth2_scheme), db: Session = Depends(get_db)):
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        email: str = payload.get("sub")
        if not email:
            raise HTTPException(status_code=401)
    except JWTError:
        raise HTTPException(status_code=401)

    user = db.query(User).filter(User.email == email).first()
    return user

```




Use in route:



```

@app.get("/me")
def read_profile(current_user: User = Depends(get_current_user)):
    return current_user








```

## ✓ Summary Table – JWT Auth Flow

Step	Action	Tool Used
 Login	POST /login with username/password	OAuth2PasswordRequestForm
 Validate	Check password hash, issue token	passlib, jose
 Token	JWT with sub (email) and exp	create_access_token()

Step	Action	Tool Used
 Protect	Decode token in protected routes	<code>get_current_user()</code>
 Response	Secured user data	Token-based access

## USER CREATION SYSTEM CHECKLIST

 Step	Description	File
 1	DB Engine + Session Setup	<code>database.py</code>
 2	SQLAlchemy Model for User	<code>models.py</code>
 3	Pydantic Request & Response Schemas	<code>schemas.py</code>
 4	User CRUD Logic	<code>db_user.py</code>
 5	API Route for <code>/users/</code>	<code>user.py</code>
 6	Main App Mounting Routes	<code>main.py</code>

## 1. `database.py` – Setup DB Connection & Session

```
# database.py
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

DATABASE_URL = "sqlite:///./users.db"

engine = create_engine(
    DATABASE_URL, connect_args={"check_same_thread": False} # SQLite specific
)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()

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





## 2. `models.py` – User Table Definition

---

```
# models.py
from sqlalchemy import Column, Integer, String
from database import Base

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, nullable=False)
    email = Column(String, unique=True, index=True)
    password = Column(String, nullable=False)
```



## 3. `schemas.py` – User Request & Response Schemas

---

```
# schemas.py
from pydantic import BaseModel, EmailStr

class UserCreate(BaseModel):
    name: str
    email: EmailStr
    password: str

class UserResponse(BaseModel):
    id: int
    name: str
    email: EmailStr

class Config:
    orm_mode = True
```



## 4. `db_user.py` – Business Logic Layer (CRUD)

---

```
# db_user.py
from sqlalchemy.orm import Session
from models import User
```

```

from schemas import UserCreate
from utils import hash_password

def create_user(db: Session, user: UserCreate):
    db_user = User(
        name=user.name,
        email=user.email,
        password=hash_password(user.password)
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

```

---

## (Optional) `utils.py` – Password Hashing

---

```

# utils.py
from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def hash_password(password: str) -> str:
    return pwd_context.hash(password)

```

---

## 5. `user.py` – Route for Creating User

---

```

# user.py
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session
from database import get_db
from schemas import UserCreate, UserResponse
from db_user import create_user

router = APIRouter()

@router.post("/users/", response_model=UserResponse)
def register_user(user: UserCreate, db: Session = Depends(get_db)):
    return create_user(db, user)

```

---



## 6. `main.py` – Mount Routers & Initialize DB

---

```
# main.py
from fastapi import FastAPI
import models
from database import engine
from user import router as user_router

models.Base.metadata.create_all(bind=engine)

app = FastAPI()

# Mount the user routes
app.include_router(user_router)
```



## Final Folder & File Structure

---

```
project/
├── main.py
├── models.py
├── schemas.py
├── db_user.py
├── user.py
├── database.py
├── utils.py
└── users.db (created on first run)
```



## Test the API

---

 **POST** `/users/`

```
{
  "name": "Darshan",
  "email": "darshan@example.com",
  "password": "secret123"
}
```

 **Response:**

```
{  
  "id": 1,  
  "name": "Darshan",  
  "email": "darshan@example.com"  
}
```

---