**1. Beginner: Bank Account Simulator**

**Goal:** Practice defining classes, attributes, and methods.

**Key OOP Concepts:** Encapsulation, constructors, basic methods

**What to build:**

* A BankAccount class with attributes like account number, balance, and account holder.
* Methods: deposit(), withdraw(), get\_balance(), and maybe transfer() to another account.
* Optionally create a SavingsAccount subclass with interest calculation.

You’ll practice constructors (\_\_init\_\_), method organization, and basic inheritance.

**2. Intermediate: Library Management System**

**Goal:** Build a functional CLI app using multiple classes.

**Key OOP Concepts:** Inheritance, composition, collections

**What to build:**

* Classes: Book, User, Library
* Library manages lists of books and users, with methods like:
  + add\_book(), remove\_book()
  + register\_user()
  + lend\_book(user, book)
  + return\_book(user, book)
* Use composition: Library has Books and Users.
* Add search features (e.g. by title or author), and track who’s borrowed what.

**Drawing on existing repo examples like MostafaAhmed98’s “Library Management” repo . (chua xem)**

**3. 🔥 Advanced: Restaurant Ordering & Management Web App**

**Goal:** Use OOP in a real-world scenario with database and web framework.

**Key OOP Concepts:** Multiple classes, inheritance, abstraction, interaction with external components

**What to build:**

* Use **FastAPI**.
* Database: PostgreSQL
* Models / OOP classes: Restaurant, MenuItem, Order, Customer, Offer.
* Controllers/business logic: applying discounts, calculating totals, assigning delivery.
* Admin interface to manage menus, orders, and view reports.

This mirrors the **“Online Restaurant Food Ordering & Management System”** outlined in OOP project lists .

**Schema** trong FastAPI (dùng Pydantic) là **cấu trúc dữ liệu chuẩn** để:

| **🎯 Mục đích sử dụng** | **💡 Giải thích dễ hiểu** |
| --- | --- |
| ✅ Xác định dữ liệu nào sẽ được **gửi từ phía client** lên (POST/PUT) | Giúp kiểm tra đầu vào, ví dụ: tên phải là chuỗi, email phải đúng định dạng |
| ✅ Xác định dữ liệu nào sẽ được **trả về từ phía server** (GET) | Bạn không muốn trả hết thông tin (ví dụ: không trả password khi trả về user) |
| ✅ Tự động tạo **Swagger Docs** đẹp và rõ ràng | Mỗi field sẽ hiển thị loại, ví dụ: name: string, discount: float |
|  |  |

SQLAlchemy model là để giao tiếp với **database**, còn Pydantic schema là để giao tiếp với **người dùng/ứng dụng client**:

| **SQLAlchemy model** | **Pydantic schema** |
| --- | --- |
| Dùng để thao tác với database (thêm, sửa, xóa) | Dùng để **validate dữ liệu**, và **hiển thị ra ngoài API** |
| Có các kiểu như Column, ForeignKey... | Có kiểu như str, int, List... |
| Không có orm\_mode | Có orm\_mode = True để map từ SQLAlchemy sang |