**Professional Project Idea: "Library Management System (LMS)"**

**This system manages books, members (customers), and librarians (employees), with a single admin. It perfectly fits the three user roles and all functional requirements.**

**1. The 6 Core Classes:**

1. **User (Abstract Base Class)**
   * **Attributes: user\_id, username, password, is\_active**
   * **Purpose: The base class for all users. Handles common login/logout functionality and the is\_active flag.**
2. **Admin (Inherits from User)**
   * **Attributes: (Inherits from User)**
   * **Purpose: Can manage all users (add, remove, activate/deactivate, view counts).**
3. **Librarian (Inherits from User)**
   * **Attributes: employee\_id (optional, can use user\_id)**
   * **Purpose: The "Employee" role. Can add, delete, update, and search for books.**
4. **Member (Inherits from User)**
   * **Attributes: member\_id, borrowed\_books (a list)**
   * **Purpose: The "Customer" role. Can borrow and return books.**
5. **Book**
   * **Attributes: book\_id, title, author, isbn, is\_borrowed**
   * **Purpose: Represents a book in the library's collection.**
6. **LibraryManager (The "System" Class)**
   * **Attributes: users\_list, books\_list**
   * **Purpose: This is the most important class. It orchestrates everything:**
     + **Manages the lists of all users and books.**
     + **Handles user registration (with unique username check).**
     + **Contains methods for all operations (borrow, return, add book, etc.).**
     + **Handles reading from and writing to files (users.txt, books.txt).**
     + **Contains input validation methods.**
     + **Uses try-except blocks for exception handling.**

**2. How It Meets All Your Requirements with 6 Classes:**

* **Users & Authentication (3 Types):**
  + **Admin, Librarian (Employee), Member (Customer). All inherit from User.**
  + **Unique Username: The LibraryManager.add\_user() method checks the users\_list before adding a new user.**
  + **Login/Logout: Handled by methods in the LibraryManager that search the users\_list.**
  + **Activation/Deactivation: The User class has an is\_active attribute. The LibraryManager checks this before allowing any operation.**
* **Permissions:**
  + **Admin: Can call methods in LibraryManager like view\_all\_users(), deactivate\_user(), get\_users\_count().**
  + **Librarian (Employee): Can call methods like add\_book(), delete\_book(), search\_book\_by\_id(title/id).**
  + **Member (Customer): Can call borrow\_book() and return\_book(). The borrow\_book() method is their "purchase" operation. It can also calculate fines, acting as the "invoice".**
* **OOP Principles:**
  + **Inheritance: Admin, Librarian, Member all inherit from User.**
  + **Encapsulation: Attributes are private (e.g., \_\_password). Methods like get\_username() are used.**
  + **Abstraction: The User class is abstract; you don't create a generic "User" object.**
  + **Polymorphism: While simple, you can have a display\_role() method that behaves differently for each user type.**
* **Data Structures & Functions:**
  + **Lists: The LibraryManager holds two main lists: list[User] users\_list and list[Book] books\_list.**
  + **Functions: All actions are methods within the LibraryManager or the classes themselves (e.g., book.is\_available()).**
* **Input Validation & Exception Handling:**
  + **Validation: Methods in LibraryManager will validate:**
    - **Phone Number (for member registration): Length 9, starts with 77/78/71.**
    - **Password: Length >7, contains letters, numbers, symbols.**
    - **Names: Not just numbers/symbols.**
  + **Exceptions: Try-Except blocks used in LibraryManager for file operations and when converting input (e.g., converting book\_id string to integer).**
* **File Handling:**
  + **The LibraryManager will have load\_data() and save\_data() methods.**
  + **These methods will read from/write to users.txt and books.txt to persist all data after the program closes.**