FINAL PROJECT

Requirements

1. **Introduction**:
    - *Entry*:
        - Create a .NET project with a clear purpose statement.
        - Include a README file explaining the project's goals and how to run it.
    - *Intermediate*:
        - Implement basic error handling (e.g., display friendly messages for invalid inputs).
    - *Advanced*:
        - Add logging to track application events (e.g., using Serilog or NLog).
2. **POO** (Object-Oriented Programming):
    - *Entry*:
        - Define at least 5 classes representing real-world entities (e.g., Library, Customer, Order).
        - Include properties, methods, and constructors.
    - *Intermediate*:
        - Implement inheritance or composition between classes.
        - Apply encapsulation by setting access modifiers for class members.
        - Use abstract classes or interfaces.
    - *Advanced*:
        - Implement a custom collection class (e.g., a stack or queue).
        - Explore design patterns related to object creation (e.g., Factory Method).

3. **SOLID Principles**:
    - *Entry*:
        - Ensure each class adheres to the Single Responsibility Principle (SRP).
    - *Intermediate*:
        - Apply the Open/Closed Principle (OCP) by allowing extension without modification.
        - Apply ISP (Interface Segregation Principle)
    - *Advanced*:
        - Implement the Dependency Inversion Principle (DIP) using an IoC container.
        - Apply the Liskov Substitution Principle (LSP) in class hierarchies
4. **LINQ** (Language Integrated Query):
    - *Entry*:
        - Utilize LINQ to query collections (e.g., filtering, sorting, grouping).
    - *Intermediate*:
        - Join data from multiple sources (e.g., combining customers and orders).
    - *Advanced*:
        - Optimize LINQ queries for performance (e.g., avoiding unnecessary materialization).
        - Implement custom LINQ operators (e.g., custom aggregations).
5. **Delegates** and **Lambda Expressions**:
    - *Entry*:
        - Implement event handling using delegates.
    - *Intermediate*:
        - Use lambda expressions for concise code (e.g., sorting, filtering).
    - *Advanced*:
        - Create a custom delegate-based event system (e.g., event bus).
        - Explore dynamic method invocation.

6. **Entity Framework**:
    - *Entry*:
        - Set up an Entity Framework project.
        - Set up a database structure on a drawing for a better understanding.
    - *Intermediate*:
        - Define entities (tables) representing relevant data (e.g., Books, Students).
        - Establish relationships (one-to-many, many-to-many) between entities.
    - *Advanced*:
        - Implement database migrations and seed data.
        - Optimize database queries for your situation. Use both lazy loading and eager loading, depending on the scenario.
7. **Code Quality**:
    - *Entry*:
        - Follow consistent naming conventions.
    - *Intermediate*:
        - Organize code into meaningful folders (e.g., Models, Controllers, Services).
        - Write unit tests for critical components (e.g., business logic).
        - Apply principles: DRY (Don't Repeat Yourself), KISS (Keep It Simple,Stupid), YAGNI (You Ain't Gonna Need It).
    - *Advanced*:
        - Integrate static code analysis tools.
        - Write the documentation of your WebAPI endpoints that is describing the usage of that scenario.

8. **Web API**:
    - *Entry*:
        - Create a RESTful API with endpoints for CRUD operations (e.g., managing books).
        - All the solution logic should be accessible with swagger.
    - *Intermediate*:
        - Implement authentication (e.g., JWT tokens) and authorization.
    - *Advanced*:
        - Add versioning and rate limiting to the API.
        - Implement caching for frequently accessed data.
9. **Blazor**:
    - *Entry*:
        - Develop a Blazor application.
        - Develop pages that comply with your project goal.
        - The pages should have a logic scope and a clean view.
    - *Intermediate*:
        - Use components for UI elements (e.g., forms, lists).
        - Use the endpoints from your WebApi to get the information.
    - *Advanced*:
        - Apply validation and data binding.
10. **Design Patterns**:
    - *Entry*:
        - Apply one Creational Design Pattern.
    - *Intermediate*:
        - Apply one Structural Design Pattern.
    - *Advanced*:
        - Apply one Behavioral Design Pattern.

11. **Architecture**:
    - *Intermediate*:
        - Respect the given architecture.
        - The layer's scope should remain the same.
        - The communication between layers should remain as agreed.

NOTE:

1. The project's structure should have common sense and behave like a unit. (e.g. If you decide to create an Online store, all the developed functionality should be in this area).
2. All principles and best practices should be applied in the whole solution.
3. The general implementations (e.g error handling, validation etc.) should be applied in all the needed scenarios.