Assignment 12: Service Layer and REST API Implementation

Objective:
For this task, you just need to create solution for a minimum of just three entities or models to lay the ground work for gropup work.

Build a **service layer** to encapsulate business logic and expose it via a **REST API**. Use the repository layer (Assignment 11) to persist data and ensure your API endpoints are fully documented and tested.

Scenario

Your system now has a domain model (Assignment 9), factories (Assignment 10), and a repository layer (Assignment 11). To make the application functional, you must:

1. **Implement service classes** to handle business operations (e.g., checkout a book, create a user).
2. **Expose these services** through RESTful endpoints (e.g., `POST /api/books`).
3. **Document the API** using OpenAPI/Swagger.
4. **Update GitHub Issues** to track services and API development tasks.

Tasks

1. Service Layer Implementation (40 Marks)

Task:

- Create **service classes** (e.g., `BookService`, `UserService`) that:
  - Use your **repositories** (Assignment 11) for persistence.
  - Implement business logic (e.g., validate inputs, enforce rules like *"users can't borrow more than 5 books"*).
- Example (Java):

```
public class BookService {
    private final BookRepository bookRepo;

    public BookService(BookRepository bookRepo) {
        this.bookRepo = bookRepo;
    }

    public Book checkoutBook(String bookId) {
        Book book = bookRepo.findById(bookId)
            .orElseThrow(() -> new BookNotFoundException(bookId));
        if (book.isCheckedOut()) {
            throw new BookAlreadyCheckedOutException(bookId);
        }
        book.checkOut();
        return bookRepo.save(book);
    }
}
```

Deliverables:

- A `/services` **directory** with service classes.
- **Unit tests** for business logic (e.g., test checkout limits).

---

2. REST API Development (40 Marks)

Task:

- Build RESTful endpoints for **CRUD operations** and **business workflows**:

HTTP Method	Endpoint	Description
`GET`	`/api/books`	Fetch all books
`POST`	`/api/books`	Create a new book
`PUT`	`/api/books/{id}`	Update a book
`POST`	`/api/books/{id}/checkout`	Check out a book

- Use a framework like:
  - **Java**: Spring Boot
  - **Python**: FastAPI/Flask
  - **C#**: ASP.NET Core

Example (Python with FastAPI):

```python
@app.post("/api/books/{book_id}/checkout")
def checkout_book(book_id: str):
    book = book_service.checkout_book(book_id)
    return {"message": f"Book {book_id} checked out", "book": book}
```

Deliverables:

- A `/api` **directory** with API controllers/routes.
- **Integration tests** (e.g., using Postman, pytest).

---

3. API Documentation (10 Marks)

Task:

- Document your API using **OpenAPI/Swagger**.
- Include:
    - Endpoint descriptions.
    - Request/response schemas.
    - Error responses (e.g., 404 if a book isn't found).

Example (FastAPI Auto-Docs):

```python
@app.get("/api/books", response_model=List[Book], tags=["Books"])
async def get_all_books():
    return book_service.get_all_books()
```

Deliverables:

- A `/docs` **directory** with OpenAPI YAML/JSON files (if not auto-generated).
- Screenshot of the **Swagger UI** (e.g., `http://localhost:8080/docs`).

---

4. GitHub Updates (10 Marks)

Task:

- **Close issues** related to service/API tasks.
- Create new issues for **bugs** (e.g., "Fix 500 error on checkout").
- Link commits to issues (e.g., `git commit -m "Close #21: Implement checkout endpoint"`).

Deliverables:

- Screenshot of your **GitHub Project Board** showing completed tasks.
- **CHANGELOG.md** summarizing API features and fixes.

Deliverables

1. **Service Layer**:
   - Service classes (`/services`).
   - Unit tests (`/tests/services`).
2. **REST API**:
   - API code (`/api`).
   - Integration tests (`/tests/api`).
3. **Documentation**:
   - OpenAPI/Swagger docs (`/docs`).
4. **GitHub Activity**:
   - Updated project board and issues.

Submission Guidelines

- **Format**: Push to your existing GitHub repository and test your url to make sure it works .
- **Grading**:
  - **40%**: Service layer correctness and test coverage.
  - **40%**: REST API functionality and documentation.
  - **10%**: API documentation completeness.
  - **10%**: GitHub activity.

Why This Matters

- **Architectural Integrity**: Separating concerns (repository → service → API) follows industry best practices.
- **Interoperability**: REST APIs allow integration with web/mobile apps.
- **Career Relevance**: Building APIs is a core skill for backend/full-stack roles.

Need Help?

- **Spring Boot**: Building a RESTful Web Service
- **FastAPI**: Tutorial
- **OpenAPI**: Swagger Documentation
