**Library Management System API Layout**

**1. Entities & Relationships (One-to-Many)**

* **User** (Parent) → Can borrow multiple **Books** (Child)
* **Book** → Can have multiple **Loans** (track borrowed books)
* **Loan** → Links a User and a Book with a due date

**Database Tables:**

1. **Users** (id, name, email, createdAt)
2. **Books** (id, title, author, isbn, availableCopies)
3. **Loans** (id, userId, bookId, borrowDate, dueDate, returned)

**2. API Endpoints & Features**

**User Management**

* POST /users → Create a user
* GET /users/{id} → Retrieve user details
* PUT /users/{id} → Update user details
* DELETE /users/{id} → Delete user (Cascade: delete loans if required)
* GET /users/{id}/loans → Get all books borrowed by a user

**Book Management**

* POST /books → Add a new book
* GET /books → List all books (with Pagination)
* GET /books/{id} → Retrieve a book’s details
* PUT /books/{id} → Update book info (e.g., availableCopies)
* DELETE /books/{id} → Remove a book

**Loan Management (Borrow & Return)**

* POST /loans → Borrow a book (userId, bookId, dueDate)
* GET /loans → List all active loans (Filter by **date range**)
* PUT /loans/{id}/return → Mark a book as returned

**3. Implementing Assignment Features**

✅ **One-to-Many Relationship** → A user can borrow multiple books (User → Loans)  
✅ **Date Handling** → Loans have a borrowDate and dueDate, allowing date filtering  
✅ **DTOs** →

* UserDTO: {id, name, email, totalBooksBorrowed}
* BookDTO: {id, title, author, availableCopies}
* LoanDTO: {userId, bookId, borrowDate, dueDate}  
  ✅ **Error Handling** →
* If a book is unavailable → 400 Bad Request: No copies left
* If trying to borrow an invalid book/user → 404 Not Found  
  ✅ **HATEOAS** → Each API response includes links to navigate (e.g., a loan response links to the book & user details)  
  ✅ **Pagination** →
* GET /books?page=1&size=10 → List books in pages
* GET /loans?from=2024-01-01&to=2024-02-01 → Filter by date

**Next Steps**

* **Design your database schema (ERD diagram)**
* **Set up your API using a framework (Spring Boot, Express.js, Django, etc.)**
* **Implement controllers, services, and repositories**
* **Test with Postman**

This layout provides all the structure you need to develop the Library Management System API while ensuring you meet the assignment requirements.

**Library Management System API Layout**

**1. Entities & Relationships (One-to-Many)**

* **User** (Parent) → Can borrow multiple **Books** (Child)
* **Book** → Can have multiple **Loans** (track borrowed books)
* **Loan** → Links a User and a Book with a due date

**Database Tables:**

1. **Users** (id, name, email, createdAt)
2. **Books** (id, title, author, isbn, availableCopies)
3. **Loans** (id, userId, bookId, borrowDate, dueDate, returned)

**2. API Endpoints & Features**

**User Management**

* POST /users → Create a user
* GET /users/{id} → Retrieve user details
* PUT /users/{id} → Update user details
* DELETE /users/{id} → Delete user (Cascade: delete loans if required)
* GET /users/{id}/loans → Get all books borrowed by a user

**Book Management**

* POST /books → Add a new book
* GET /books → List all books (with Pagination)
* GET /books/{id} → Retrieve a book’s details
* PUT /books/{id} → Update book info (e.g., availableCopies)
* DELETE /books/{id} → Remove a book

**Loan Management (Borrow & Return)**

* POST /loans → Borrow a book (userId, bookId, dueDate)
* GET /loans → List all active loans (Filter by **date range**)
* PUT /loans/{id}/return → Mark a book as returned

**3. Implementing Assignment Features**

✅ **One-to-Many Relationship** → A user can borrow multiple books (User → Loans)  
✅ **Date Handling** → Loans have a borrowDate and dueDate, allowing date filtering  
✅ **DTOs** →

* UserDTO: {id, name, email, totalBooksBorrowed}
* BookDTO: {id, title, author, availableCopies}
* LoanDTO: {userId, bookId, borrowDate, dueDate}  
  ✅ **Error Handling** →
* If a book is unavailable → 400 Bad Request: No copies left
* If trying to borrow an invalid book/user → 404 Not Found  
  ✅ **HATEOAS** → Each API response includes links to navigate (e.g., a loan response links to the book & user details)  
  ✅ **Pagination** →
* GET /books?page=1&size=10 → List books in pages
* GET /loans?from=2024-01-01&to=2024-02-01 → Filter by date

**4. Ensuring Full Marks**

**Assignment Breakdown & Additional Considerations**

| **Criteria** | **Is It Covered?** | **Additional Notes** |
| --- | --- | --- |
| One-to-Many Relationship | ✅ Yes | Include ERD Diagram |
| Date Handling | ✅ Yes | Validate and format dates properly |
| DTOs | ✅ Yes | Ensure DTOs match assignment examples |
| Error Handling | ✅ Yes | Implement structured error messages |
| HATEOAS | ✅ Yes | Provide example responses in JSON |
| Pagination | ✅ Yes | Include metadata in paginated responses |
| Screencast | 🔸 Your task | Show API demo using Postman |
| Testing | 🔸 Your task | Cover different test cases (e.g., invalid inputs) |

**5. Final Thoughts**

If you **implement everything as planned**, your **technical API implementation** should be perfect. However, **marks also depend on:**

1. **Report Quality** → Clear explanations, well-structured, with diagrams.
2. **Code Readability** → Proper comments, modular code structure.
3. **Screencast** → Clearly demonstrating API functionality.

✅ **If you execute everything perfectly, you should aim for a high mark (70%+).**  
🔹 **Would you like a checklist for submission to make sure nothing is missed?** 🚀