Perfect! Here's your **updated full project documentation** for the **Library Management System API**, with the new **`Payments`** table added and integrated into the model + business logic.

✅ Finalized Project: Library Management System API

📌 Summary of Key Features

Module	Details
📚 Book	Add/update books, total copies, available copies
👤 Customer	Register, track membership, borrow history
📦 Book Issue	Issue/return books with constraints
💳 Payments	Maintain a fee payment record (₹251/month)
🔁 Constraints	Max 3 active books per customer
⌛ Due Date + Fine	Fine if returned late (e.g., ₹10/day)
⭐ Review	One optional review per book per customer

🧠 Updated Entities (With Relationships)

1. `Customer`

```
java

Long id
String name
String email
String phone
LocalDate membershipDate
```

---

2. `Book`

```
java

Long id
String title
String author
String isbn
int totalCopies
int availableCopies
List<Review> reviews (OneToMany)
```

---

3. `IssuedBook`

```
java

Long id
Book book (ManyToOne)
```

```
Customer customer (ManyToOne)
LocalDate issueDate
LocalDate dueDate
LocalDate returnDate
Boolean isReturned
Double fineAmount
```

---

4. `Review`

```
java

Long id
Book book (ManyToOne)
Customer customer (ManyToOne)
String content
int rating (1–5)
```

---

5. `Payment`

```
java

Long id
Customer customer (ManyToOne)
LocalDate paymentDate
Double amount
Boolean isSuccessful
```

> 💡 **New Rule**: On issuing a book, check if the latest monthly payment exists (by `LocalDate.now().getMonth()` and year). Only allow issuing if successful payment of ₹251 exists for that month.

🔐 Business Rules

Rule	Logic
✅ Max books	A customer can have at most 3 active books issued
💳 Fee Payment	Customer must have paid ₹251 for the current month to issue books
🔁 Late Return Fine	Fine = ₹10/day after due date (configurable)
⭐ One Review	A customer can review a book only once
💸 Payment Record	Use `payments` table instead of `isFeePaid` flag for tracking

---

🧪 Key APIs Summary

👤 Customer

- `POST /customers` → Register new customer
- `GET /customers/{id}/books` → View current issued books
- `GET /customers/{id}/payments` → View payment history

📚 Book

- **`GET /books`** → List all books with availability
- **`POST /books`** → Add new book
- **`GET /books/{id}/reviews`** → Get all reviews for a book

🔄 Issue/Return

- **`POST /issues`** → Issue a book (check 3-book + current month payment)
- **`PUT /returns/{issueId}`** → Return a book, auto-calculate fine

💳 Payments

- **`POST /payments`** → Add payment entry for a customer
- **`GET /payments?month=07&year=2025&customerId=1`** → Validate fee paid for issuing logic

⭐ Review

- **`POST /books/{id}/review`** → Add review (check if already reviewed)

🧱 Additional Folder Suggestions

```bash
├── validator/               # For custom validation (limit, payment)
│   └── BookLimitValidator.java
├── scheduler/              # Optional cron/fine monitor
│   └── FineCalculatorJob.java
├── util/
│   └── FineUtils.java      # Fine calculation logic
├── service/payment/
│   └── PaymentValidationService.java
```

📊 Actuator Endpoints to Expose

- **`/actuator/health`**
- **`/actuator/metrics`**
- **`/actuator/beans`**
  *(Optional: enable or restrict via **`application.yml`**)*

🧪 Testing Strategy

Layer	Test Type
Model	Entity constraints (**`rating`**, **`email`**, **`date`**)

Layer	Test Type
Service	Business logic: issuing books, fine calc, payment check
Controller	Integration tests for `/issues`, `/payments`, `/returns`
Repository	JPA tests for payment history and book availability

---

🧾 DB Table Overview

Table	Fields/Relations
`customers`	`id`, `name`, `email`, `phone`, `membership_date`
`books`	`id`, `title`, `author`, `available_copies`, `total_copies`
`issued_books`	FK → `book_id`, `customer_id`
`reviews`	FK → `book_id`, `customer_id` *(with `unique` constraint)*
`payments`	FK → `customer_id`, `payment_date`, `amount`, `is_successful`

📅 Project Timeline (Still fits 3–4 days with focus)

Day	Task
Day 1	Project setup, MySQL schema, Models + Relationships
Day 2	Customer, Book, Issue APIs (including payment check)
Day 3	Review + Fine logic, Payment APIs, Actuator
Day 4	Tests, Swagger UI (optional), refactor/cleanup

🚀 Next Steps

Let me know if you'd like:

- Sample schema SQL for all 5 tables
- Full starter code structure
- Step-by-step model-to-API guide

Would you like me to now help scaffold models & repositories one-by-one, starting from `Customer`?

can you please create one pdf with all above data, i want to maintain same formatting which we're currently having