Library Management System Design

1. Functional Requirements

(What the system DOES)

User Authentication:

Users can register, log in, and reset passwords.

Librarians have admin access (add/update/delete books, manage users).

Book Search:

Search books by title, author, ISBN, or genre.

View real-time availability status.

Borrowing/Returning:

Borrow up to 5 books/user.

Automatic due date calculation (14-day loan period).

Return books with overdue fines (20 pesos/day).

Notifications:

Email/SMS reminders 3 days before due dates.

Alerts for overdue books.

Admin Features:

Add/update/remove books from the catalog.

Generate reports (popular books, overdue items).

2. Non-Functional Requirements

(How the system PERFORMS)

Performance:

Search results load in <1.5 seconds (even under peak load).

The system handles 300+ concurrent users with load balancing (NGINX) and caching.

Scalability:

Database supports 100,000+ books and 50,000+ users.

Designed for horizontal scaling (additional servers can be added via PostgreSQL sharding or read replicas).

Implements read/write database separation (e.g., primary-replica database setup) to improve query performance.

Uses caching (Redis/Memcached) for frequently accessed book data to reduce database load.

Security:

Passwords stored using **bcrypt** or **Argon2id** (salted, adaptive hashing).

HTTPS encryption for all data transfers.

Regular security audits.

Availability:

99% uptime (excluding scheduled maintenance).

Usability:

Intuitive UI for users with no technical background.

Reliability:

Daily automated backups.

Transaction rollback on errors (e.g., failed borrow/return).

3. Subsystems

(Modular Components)

Subsystem	**Responsibilities**	**Key Parameters**
User Management & Authentication	- Registration, login, profile updates. - Role-based access (user vs. librarian). - Session management.	- Max 5 books/user. - Password complexity rules. - JWT token expiration: 24 hours.
Book Inventory	- Add/update/remove books. - Track availability, ISBN, metadata.	- Unique ISBN per book. - Categorize by genre.
Borrowing System	- Process loans/returns. - Calculate due dates and fines.	- 14-day loan period. - 20 pesos/day overdue fine.
Notification System	- Send reminders via email/SMS. - Alert librarians about overdue books.	- 3-day pre-due reminder. - Daily overdue alerts.
Reporting Module	- Generate usage statistics. - Export lists of overdue books.	- Weekly/Monthly reports.

4. Key Parameters

Max Books/User: 5 (adjustable by librarians).

Database Scalability:

Initial capacity: 50,000 books, 10,000 users.

Scalable to 500,000 books and 100,000 users using sharding (PostgreSQL Citus) or cloud databases (AWS Aurora).

Performance Considerations:

Caching: Redis for frequent searches (e.g., popular genres).

Indexing: Optimize search with indexes on title, author, ISBN.

Security Parameters:

HTTPS with TLS 1.3.

Regular security audits.

Password policy: Minimum 8 characters with symbols/numbers.

Session timeout: 15 minutes of inactivity.

Database Model Comparison: Relational vs. NoSQL

Criteria	**Relational (SQL)**	**NoSQL**
Data Structure	Structured, table-based with fixed schema (rows and columns).	Flexible schema (document, key-value, graph, or wide-column stores).
Scalability	Horizontal (sharding, read replicas) + Vertical.	Horizontal scaling (adding more servers).
Consistency	ACID compliance (Atomicity, Consistency, Isolation, Durability).	BASE model (Basically Available, Soft state, eventually consistent).
Query Language	SQL (powerful for complex joins and transactions).	Varies by type (e.g., MongoDB uses JSON-like queries; limited joins).
Use Cases	Systems requiring complex transactions (e.g., banking, inventory).	High-speed, unstructured data (e.g., social media, IoT, real-time analytics).

Performance	Optimized for read-heavy operations and complex queries.	Faster for write-heavy operations and unstructured data.
Maintenance	Requires schema migrations; strict data integrity.	Schema-less design allows flexibility but risks inconsistent data without governance.
Examples	MySQL, PostgreSQL, SQL Server.	MongoDB (document), Cassandra (wide-column), Redis (key-value).

Trade-offs

Model	**Strengths**	**Weaknesses**
Relational	- ACID guarantees. - Complex queries with SQL. - Hybrid scaling.	- Schema changes require migrations.
NoSQL	- Horizontal scalability. - Flexible schema. - High write performance.	- No atomic transactions. - Manual joins for relationships

Justification for Choosing Relational Database

Scenario: Library management systems require transactional integrity (e.g., borrowing a book update both user loans and book availability) and structured data (fixed relationships between users, books, and loans).

Scalability

Why Relational (PostgreSQL)?

- Horizontal scaling: Achieved via read replicas (for search queries) and sharding (e.g., Citus extension).
- Cloud compatibility: AWS Aurora/Google Cloud SQL automate scaling for 100,000+ books.

- Vertical scaling: Upgrading server specs (CPU/RAM) handles initial growth (50k books).

Why Not NoSQL?

- Eventual consistency: Risk of stale data (e.g., a book appears available but is already borrowed).
- Transactional limits: NoSQL cannot atomically update User.borrowed_books_count and Book.availability_status.

Maintainability

Why Relational?

- **Structured schema:** Clear relationships (e.g., User ↔ Loan ↔ Book tables).
- **Data governance:** Constraints (e.g., foreign keys) enforce rules like "max 5 books/user".
- **Schema migrations**: Tools like Liquibase/Flyway automate schema changes (e.g., adding a new column to the Book table).
- **JSONB support:** Store semi-structured book metadata (e.g., genres, tags) while retaining ACID.

Why Not NoSQL?

- **Schema-less design:** Risk of inconsistent data formats (e.g., some books lack genre or author fields).
- **Manual joins:** Relationships (e.g., linking loans to users and books) require application-level code, increasing maintenance complexity.
- **No built-in constraints:** Rules like "max 5 books/user" must be enforced in code, increasing error risk.
- **Data cleanup:** Requires manual effort to fix inconsistencies (e.g., orphaned loan records).

Functional Fit

ACID Compliance: Critical for loan transactions (e.g., deducting a book's availability and updating a user's loan count in one atomic operation).

Complex Queries: SQL simplifies generating reports (e.g., “Top 10 most borrowed books”) with JOIN and GROUP BY.

Final Recommendation

Chosen Model: Relational Database (PostgreSQL)
Justification:

- Ensures transactional integrity for loans/returns.
- Structured schema aligns with library data relationships.
- Hybrid scaling (horizontal + vertical) meets 500,000-book scalability needs.
- JSONB allows flexibility for semi-structured metadata.

The **Library Management System** was initially designed as a **simple** solution for book borrowing, returning, and searching. However, to ensure **real-world usability and long-term efficiency**, the design was refined with **scalability, performance, and security enhancements**. Instead of just a basic system, it now supports **500,000 books and 100,000 users**, leveraging **PostgreSQL sharding and read replicas** for horizontal scaling. To maintain **fast search speeds (<1.5s)** and **support 300+ concurrent users**, **Redis caching, indexing, and NGINX load balancing** were integrated. Security was also strengthened with **bcrypt/Argon2id** password hashing, **HTTPS encryption, and session management**. With **99% uptime, automated backups, and ACID compliance**, the system is built not just for functionality but for **real-world reliability**, proving that even a "simple" system should be **designed for scalability and maintainability** to meet modern requirements.