University of Science and Technology of Southern Philippines

Alubijid |Balubal | Cagayan de Oro | Claveria | Jasaan | Oroquieta | Panaon |Villanueva

GROUP: 5
NAME: CLAVEYL C. DE LA CERNA

HAROLD VICADA

PRINCESS ONG

ZEILPHA GEANNE ALIMBOG

CHERYLANE PACLAR

1. Software Architectural Styles – Advantages & Disadvantages

Architecture Style	Advantages	Disadvantages
Layered	Clear separation of concerns, easy to maintain, scalable	Performance overhead, inflexible structure
Client-Server	Centralized management, better security, scalable	Server overload, dependency on network availability
Event-Driven	Highly scalable, loosely coupled components	Difficult to debug, increased complexity
Microservices	Independent services, easy to scale, fault-tolerant	Complex deployment, requires strong service coordination
Repository Style	Centralized data access, easy data sharing	Single point of failure, potential performance bottlenecks

2. Select an Existing Software System

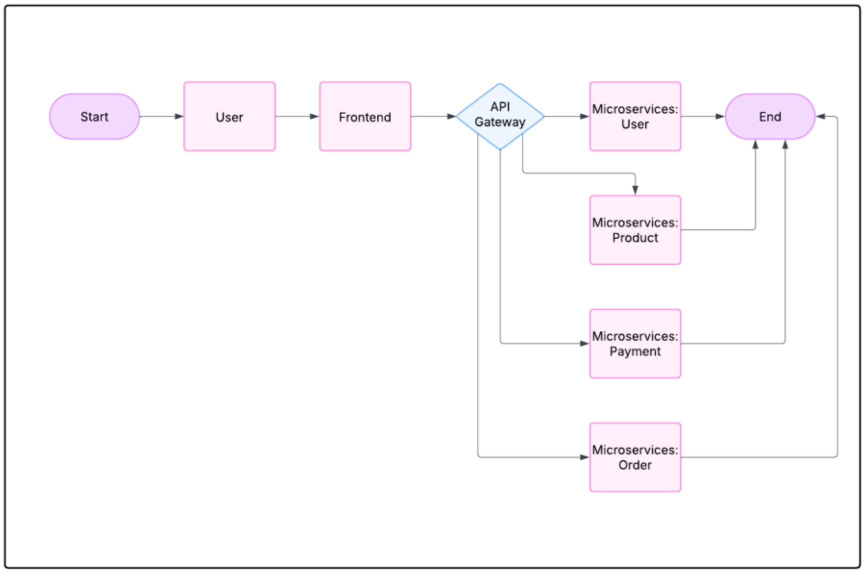
Chosen System:

- **E-commerce Platform**

3. Identify and describe the architecture style it follows.

- **Architecture Style: Microservices**
- **Description:**
  - Uses multiple independent services to handle different functionalities.
  - Services communicate via APIs, ensuring flexibility and scalability.

4. Basic Block Diagram:



5. Software application for a Daily Use Application

Chosen Application:

- **Spotify**

6. Identify its key components and their relationships.

Key Components:

- **User**
    a. Represents the end-user interacting with the system.
    b. Sends requests via the frontend.
- **Frontend**
    c. The UI or client application that the user interacts with.
    d. Sends API requests to the **API Gateway** for processing.
- **API Gateway**
    e. Acts as a single entry point for all client requests.
    f. Routes requests to the appropriate microservices.
    g. Can handle authentication, logging, and rate-limiting.
- **Microservices:**
    h. **User Microservice**
        i. Handles user-related operations like authentication, profile management, and authorization.
    i. **Product Microservice**
        i. Manages product details, inventory, and catalog information.
    j. **Payment Microservice**
        i. Handles transactions, billing, and payment processing.
    k. **Order Microservice**
        i. Manages order placement, tracking, and fulfillment.
- **End**
    l. Represents the completion of the user request.
    m. Can be the successful retrieval of data, order confirmation, or any other system response.

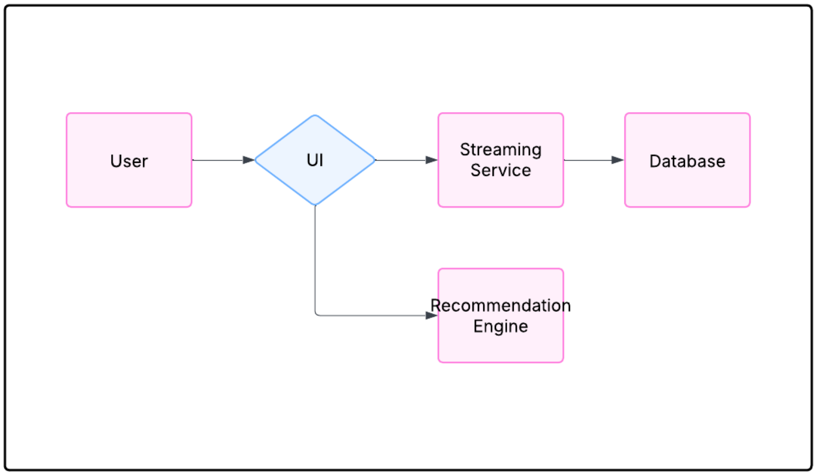
Relationships*:*

- The **User** interacts with the **Frontend**.
- The **Frontend** sends requests to the **API Gateway**.
- The **API Gateway** directs requests to different **Microservices**.

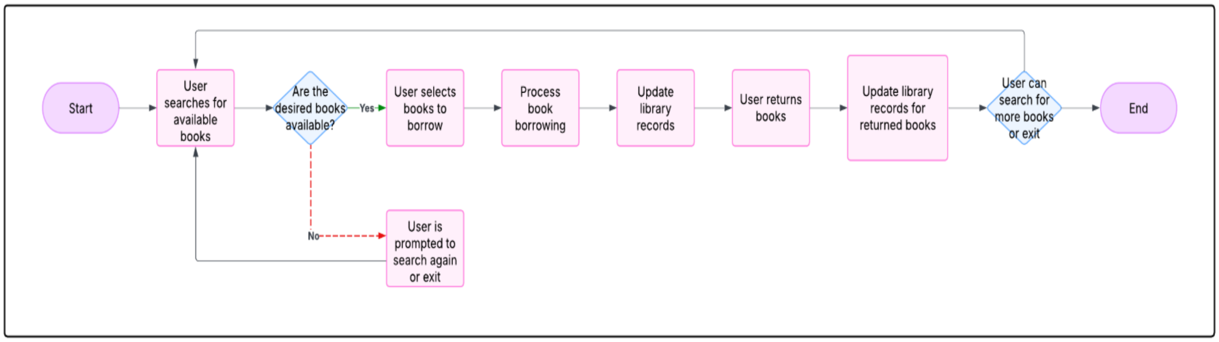
- The **Microservices** communicate with each other when necessary (e.g., User Microservice might call Product or Payment Microservices).
- The system eventually responds to the user, completing the process.

7. Create a UML component diagram showcasing its architecture

Components Diagram:



8. Design a simple library management system that allows users to borrow books, return books, and search for available books.


Start
User searches for available books
Are the desired books available?
Yes
User selects books to borrow
Process book borrowing
Update library records
User returns books
Update library records for returned books
User can search for more books or exit
End
No
User is prompted to search again or exit

9. Identify the functional and non-functional requirements of the system.

- **Functional Requirements:**
  - **Book Search** – Users must be able to search for available books in the library.
  - **Book Availability Check** – The system should determine whether the desired books are available.
  - **Book Borrowing** – Users must be able to select books to borrow.
  - **Borrowing Processing** – The system must process the borrowing transaction and update records.
  - **Library Record Update** – When books are borrowed, the system must update the inventory.
  - **Book Return** – Users must be able to return borrowed books.
  - **Return Processing** – The system must update records upon book returns.
  - **Re-Search or Exit Option** – After completing an action, users should have the option to search for more books or exit.

- **Non-Functional Requirements (System Quality Attributes):**
  - **Performance** – The system should handle multiple users searching, borrowing, and returning books simultaneously without delays.
  - **Scalability** – The system should support a growing number of books and users over time.
  - **Reliability** – The system should ensure accurate and consistent updates to book records.
  - **Usability** – The interface should be user-friendly, allowing users to interact with minimal training.
  - **Security** – User data and book records should be protected from unauthorized access.
  - **Availability** – The system should be accessible at all times with minimal downtime.
  - **Maintainability** – The system should allow easy updates and modifications without affecting existing functionality.

10. Break down the system into smaller subsystems (e.g., User Management, Book Inventory, Borrowing System).

- **User Management Subsystem**
  - Manages user registration, login, and authentication.
  - Maintains user details (e.g., name, contact information, borrowing history).
  - Enforces borrowing rules (e.g., max books per user).
  - Handles user roles (e.g., students, faculty, librarians).

- **Book Inventory Subsystem**
  - Stores and manages book details (title, author, category, availability status).
  - Tracks the number of copies available for each book.
  - Updates book status when borrowed or returned.
  - Allows librarians to add, remove, or update book records.

- **Borrowing System**
  - Handles book borrowing requests from users.
  - Checks book availability before allowing a user to borrow.
  - Updates the book inventory and user borrowing records.
  - Enforces borrowing limits (e.g., max books per user, due dates).
  - Notifies users of due dates and overdue books.

- **Returning System**
  - Processes book returns and updates records accordingly.
  - Checks for overdue books and applies penalties (if applicable).
  - Makes returned books available for other users to borrow.
  - Sends notifications for successful returns.

- **Security & Access Control System**

- Ensures user authentication (login/logout) and authorization (role-based access).
- Protects sensitive data (e.g., user information, borrowing records).
- Prevents unauthorized access or fraudulent transactions.

11. Define parameters such as max books borrowed per user, database scalability needs, and performance considerations.

Define Parameters:

- **Max Books Borrowed per User**:
    - A user can borrow up to **5 books at a time**.
    - Borrowing period: **14 days** with a **7-day renewal option**.
    - Overdue books will prevent further borrowing until returned.
- **Database Scalability Needs**:
    - Must handle **thousands of users and books** efficiently.
    - Support **real-time inventory updates** when books are borrowed/returned.
    - Allow **future expansion** (e.g., adding digital books, more libraries)
- **Performance Considerations**:
    - **Fast search queries** for books (indexing, caching strategies).
    - **Low-latency transactions** for book borrowing/returning.
    - **Optimized user authentication** for quick login and access.

12. Compare different database models (Relational vs NoSQL) and discuss trade-offs.

Compare Database Models (Relational vs NoSQL)

Feature	Relational Database (SQL)	NoSQL Database
Data Structure	Structured (tables, rows, columns)	Flexible (documents, key-value, graph)
Query Language	SQL (Structured Query Language)	NoSQL (JSON, key-value, column-family)
Scalability	Vertical (add more power to a single server)	Horizontal (add more servers easily)
Consistency	Strong ACID compliance (atomicity, consistency, isolation, durability)	Eventual consistency, but high availability
Performance	Optimized for complex queries and relationships	Faster for read-heavy applications
Use Case	Best for structured data (e.g., banking, HR systems)	Best for unstructured or rapidly growing data (e.g., real-time apps, big data)

13. Justify the chosen database model based on scalability and maintainability.

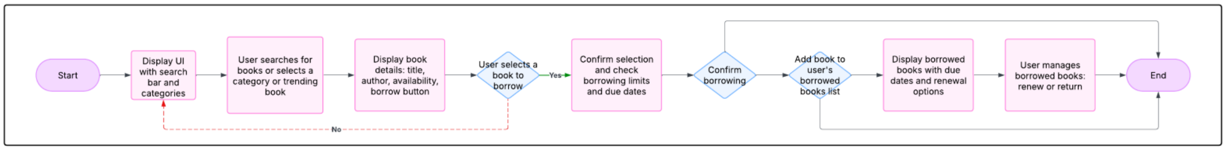
Justification for Chosen Database Model:

- **Relational Database (MySQL/PostgreSQL) is ideal** because:
  - It ensures **data consistency** (important for book availability tracking).
  - It supports **transactions** (book borrowing and returning must be atomic).
  - It allows **structured queries** for searching books efficiently.
  - It is easier to **maintain relationships** between books, users, and loans.

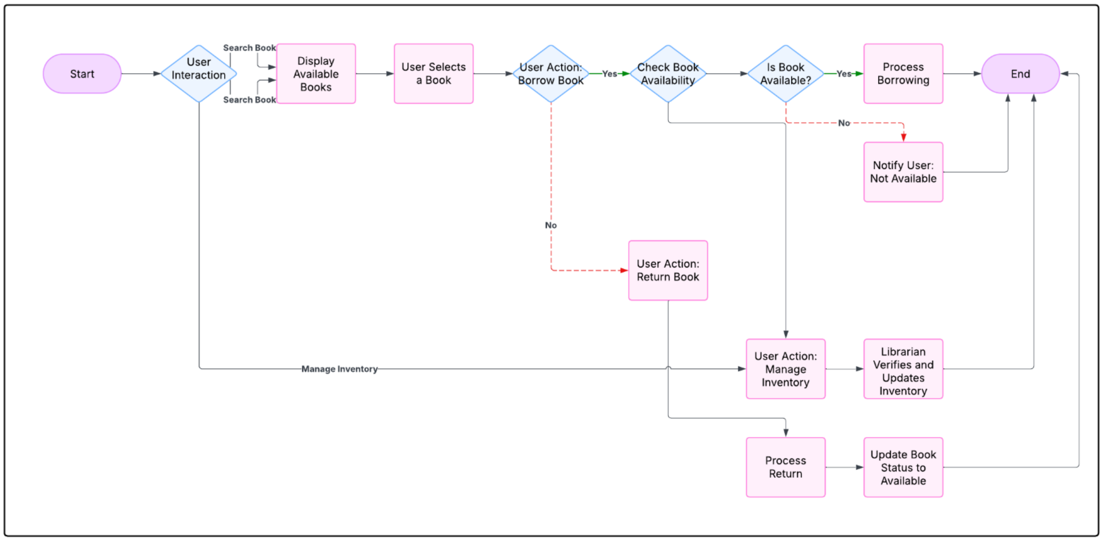
- **NoSQL (MongoDB) could be used** if the system required:

- **High-speed search and recommendations** (for AI-driven recommendations).
- **Scaling across multiple locations** with real-time synchronization.
- **Unstructured data** (e.g., storing user reviews, logs).

14. Sketch a high-level prototype UI for book search and borrowing.


Start
Display UI with search bar and categories
User searches for books or selects a category or trending book
Display book details: title, author, availability, borrow button
User selects a book to borrow
Yes
No
Confirm selection and check borrowing limits and due dates
Confirm borrowing
Add book to user's borrowed books list
Display borrowed books with due dates and renewal options
User manages borrowed books: renew or return
End

15. Create the following UML diagrams for the library management system:



16. Identify key actors and interactions.

- **The primary actors in the system are:**

- **User**
- **Librarian**

- **System (Library Management System)**

○ **Interactions:**

- User searches for a book.
- System displays available books.
- User selects a book to borrow.
- System checks book availability:
  - If available → Proceeds with borrowing.
  - If not available → Notifies the user.
- User returns a book.
- Librarian verifies and updates inventory.

17. Define classes such as User, Book, Loan, etc.

- **Define Classes (Class Diagram)**

The main classes in the system include:

- **User**
- **Book**
- **Loan**
- **Librarian LibrarySystem**

18. Illustrate a book borrowing process.

illustrate Book Borrowing Process (Sequence Diagram)

- User initiates book search.
- System retrieves book details and availability.
- User requests to borrow a book.
- System verifies book availability.

- If available, the system registers the loan and updates inventory.
- User returns the book, and the librarian updates the inventory.

19. Show system components and their interactions.

System Components and Their Interactions (Component Diagram)

The system is composed of:

- **User Interface (UI)** – Displays books, borrowing options, and notifications.
- **Book Inventory System** – Tracks book availability and records transactions.
- **Borrowing System** – Manages loan records and return processing.
- **Database** – Stores book, user, and loan information.

Conclusion:

This case study examined different software architectural styles, along with their benefits and drawbacks, and their practical applications via two software systems: an E-commerce platform (Microservices architecture) and Spotify (Microservices-based music streaming service).

We subsequently developed a Library Management System, dividing it into subsystems and clarifying its functional and non-functional requirements. The system was organized using components such as User Management, Book Inventory, Borrowing System, Returning System, and Security and Access Control to guarantee efficiency, reliability, and scalability.

In selecting a database, we assessed SQL and NoSQL options, concluding that relational databases (MySQL/PostgreSQL) are most appropriate for structured data and transaction management, while NoSQL (MongoDB) may improve real-time analytics and recommendation systems.

Ultimately, we produced UML diagrams, which included a component diagram, class diagram, sequence diagram, and system interaction diagram, to create a high-level design blueprint of the Library Management System.

Key Takeaways:

✓ Microservices architecture increases flexibility, scalability, and the management of independent services.
✓ Spotify and E-commerce platforms depend on API gateways to manage various microservices.
✓ Library Management Systems necessitate structured data, making relational databases a solid option.
✓ Functional and non-functional requirements direct system design, ensuring usability, performance, and security.
✓ UML diagrams offer visual clarity in system architecture and workflow.