Technical Requirements Document: Microservices Architecture for Library Management System

Introduction:

- Overview: This document outlines the technical requirements for the development and implementation of a Library Management System (LMS) using a microservices architecture.
- Purpose: To establish a comprehensive understanding of the system's architectural components, their interactions, underlying technologies, and analytical capabilities.
- Scope: The LMS aims to automate and streamline diverse library functions, including book cataloging, user management, borrowing, analytical insights, and reporting.

System Architecture:

- Microservices:
  - User Management Service:
    - Responsibility: Handles user-related operations, including user registration, authentication, and profile management.
    - APIs: `/register`, `/login`, `/profile`.
  - Catalog Service:
    - Responsibility: Manages the catalog of books, including details like title, author, genre, and availability.
    - APIs: `/books`, `/books/{id}`, `/genres`.
  - Loan Service:
    - Responsibility: Handles book loan transactions, including borrowing, returning, and tracking due dates.
    - APIs: `/borrow`, `/return`, `/due-dates`.
  - Transaction Service:
    - Responsibility: Manages transaction history, including user interactions with the system.
    - APIs: `/transactions`.
  - Notification Service:
    - Responsibility: Sends notifications to users regarding due dates, overdue books, and other relevant information.
    - APIs: `/send-notification`.
  - Analytics Service:
    - Responsibility: Provides analytical insights into user behavior, popular books, and other metrics.
    - APIs: `/user-behavior`, `/popular-books`.

        - Reporting Service:
            - Responsibility: Generates reports and dashboards for administrators and stakeholders.
            - APIs: `/generate-report`.
        - Authentication Service:
            - Responsibility: Centralized authentication service used by other microservices to validate user identities.
            - APIs: `/validate-token`.

Component Interaction:

- The User Management Service interacts with the Authentication Service for user authentication.
- The Catalog Service communicates with the Loan Service to update book availability upon borrowings and returns.
- The Loan Service communicates with the Transaction Service to log borrowing and return transactions.
- The Loan Service may send notifications through the Notification Service for due dates and overdue books.
- The Analytics Service and Reporting Service utilize data from various microservices to provide insights and generate reports.

Technology Stack:

- Backend:
    - Java will serve as the primary language for the application layer of each microservice.
    - Spring Boot will be employed to expedite development and facilitate seamless integration.
- Frontend:
    - The UI will be developed using a combination of HTML, CSS, and JavaScript.
    - A modern frontend framework, such as Angular, React, or Vue.js, will be utilized to create dynamic and responsive user interfaces.
- Database:
    - A robust relational database management system (RDBMS), such as MySQL or PostgreSQL, will be chosen for efficient data storage and retrieval.
- Analytical Tools:
    - Integration with popular business intelligence tools such as Tableau or Power BI for advanced analytics and reporting.

Database Schema:

- A well-defined database schema will include tables for books, users, transactions, and other relevant entities, maintaining a normalized structure to support data integrity.

Security:

- Robust authentication and authorization mechanisms will be implemented to secure access to the system.
- HTTPS will be enforced for secure data transmission.
- Regular security audits and updates will address vulnerabilities and ensure a secure environment.

Scalability:

- The system architecture will be designed to scale horizontally, accommodating an increasing number of users and data points seamlessly.

Performance:

- Database queries will be optimized for efficiency, ensuring swift response times.
- Caching mechanisms will be implemented to reduce latency and enhance performance.

Analytics:

- The system will integrate tools for tracking user behavior, identifying popular books, and generating other relevant metrics.
- Customizable reports and dashboards will be designed to support data-driven decision-making.

Testing:

- A comprehensive testing strategy will include unit tests, integration tests, and end-to-end tests.
- Automated testing tools will be employed to guarantee the reliability and functionality of the system.

Deployment:

- Continuous integration and continuous deployment (CI/CD) pipelines will be established for seamless development and deployment workflows.
- Cloud platforms such as AWS or Azure will be considered for hosting, scalability, and resource management.
- Multiple deployment environments, including staging and production, will be maintained to ensure thorough testing before releasing updates to the live system.

Production Support:

- A dedicated support team will be in place to address production issues and provide timely resolutions.

- Regular monitoring and logging mechanisms will be implemented to proactively identify and address potential problems.

Quality Analysis:

- Continuous quality analysis will be conducted throughout the development lifecycle.
- Code reviews, static code analysis, and automated quality checks will be integral parts of the development process.

Unit Testing:

- Comprehensive unit tests will be written for each module to verify individual functionalities.
- Test-driven development (TDD) practices will be encouraged to ensure code reliability.

Performance Testing:

- Rigorous performance testing will be conducted to assess the system's response time, throughput, and scalability under various loads.
- Load testing, stress testing, and scalability testing will be part of the performance testing strategy.