# 2022 Cyber Group - 2

***2022BCY0002 - Dania Eram, 2022BCY0013 – Raghavendra, 2022BCY0024 – Gayatri, 2022BCY0035 – Suraj, 2022BCY0046 – Sanjay, 2022BCY0057 - Shresth***

# Software Design Document [SDD]

***Abstract:***

*The document proposes a Library Management System leveraging NoSQL's flexibility for diverse data, focusing on scalability and modularity. It employs microservices for compartmentalization and an API Gateway for centralized access. While query clarification isn't explicitly discussed, it can be seamlessly integrated through various approaches like query categorization, user-friendly interfaces, NLP-powered suggestions, feedback loops, and knowledge base integration. This highlights the system's adaptability and potential to cater to evolving user needs, including query clarification.*

## ***Table Of Contents:***

# *1. Introduction:*

## *1.1 Purpose:*

*This document aims to provide an exhaustive overview of the Library Management System design. It is intended for use by developers, testers, and project managers. The purpose of this document is to furnish a comprehensive insight into the Library Management System design and its intended audience comprises developers, testers, and project managers. The document is drafted to provide a clear understanding of the system's design and functioning, encompassing its architecture, modules, and data flow. The information provided in the document will enable the intended audience to assess the system's capabilities, identify areas of improvement, and make informed decisions regarding its implementation. Therefore, this document carries paramount importance in the development and management of the Library Management System. The purpose of this document is to describe the system design of a Library Management System.*

## *1.2 Scope:*

*The scope of this document is to provide a comprehensive overview of the Library Management System design. It is intended for use by developers, testers, and project managers. The purpose of this document is to furnish a comprehensive overview of the Library Management System design, and its intended audience comprises of developers, testers, and project managers. The document is drafted with the objective to provide a clear understanding of the system's design and functioning, which includes its architecture, modules, and data flow. The information provided in the document will enable the intended audience to evaluate the system's capabilities, identify areas of improvement, and make informed decisions regarding its implementation. Therefore, this document is of utmost importance in the development and management of the Library Management System.*

## *1.3 Definitions:*

| | |
|---|---|
| *SDD* | *Software Design Document* |
| *UI* | *User Interface* |
| *DFD* | *Date Flow Diagram* |
| *LMS* | *Library Management System* |

## *1.4 References:*

*This document has been prepared with the reference of IEEE std 1016: 2005.*

## *1.5 Overview:*

*The Library Management System is a sophisticated web-based application that provides users with the ability to borrow books from their library. Furthermore, it facilitates users to submit their queries, which are then answered by the professors or other users. This feature ensures that the users receive prompt and accurate feedback, thereby enhancing their overall experience of their queries of the users by lecturers or other users and showing the best solution using upvotes.*

## *2. System Overview:*

### *2.1 System Context:*

*The Library Management System is a sophisticated web-based application that provides users with the ability to borrow books from their library. Furthermore, it facilitates users to submit their queries, which are then answered by the professors or other users. This feature ensures that the users receive prompt and accurate feedback, thereby enhancing their overall experience of their queries of the users by lecturers or other users and showing the best solution using upvotes.*

### *2.2 System Functionality:*

*The Library Management System offers a range of features that facilitate seamless management of library functions. These include user enrollment and login functionality, the ability to search for available books, borrowing and return notifications, and the option to submit queries and feedback from other users.*

### *2.3 User Characteristics:*

- *Register and manage user accounts (students, faculty, staff)*
- *Track user borrowing history and fines*
- *Allow users to search for books and view availability*
- *Enable users to reserve books*
- *Implement user roles and permissions for access control*
- *Allow users to submit queries related to books, research, or library services*
- *Categorize queries for efficient routing*
- *Assign queries to relevant professors or experts for response*
- *Track query status and resolution*
- *Enable users to rate and provide feedback on responses*

### *2.4 Constraints:*

- *The system must be scalable to handle an increasing number of bikes and users.*
- *The system must be available 24/7 except for maintenance downtime*
- *The system must be able to handle many concurrent users.*

### *2.5 Assumptions and Dependencies:*

*The user's internet connection is stable and meets the minimum requirements for accessing the system.*

## *3. Architectural Design:*

### *3.1. Presentation Layer:*

- *Web Application: Developed using a front-end framework like React or Angular.*
- *Mobile App (optional): Built with a platform-specific framework like React Native or Flutter.*
- *API Gateway: Acts as a single-entry point for all API requests from the UI components and mobile app.*

### *3.2. Business Logic Layer:*

- *Microservices: Each core functionality (Book Management, User Management, Borrowing, Query Management) implemented as a separate microservice for scalability and maintainability.*
- *API Server: Handles communication between the API Gateway and microservices, routing requests and responses.*
- *Business Logic: Each microservice implements its business logic, interacting with the data layer and other microservices as needed.*

### *3.3. Data Access Layer:*

- *NoSQL Database: MongoDB Atlas for storing unstructured data like query responses, user feedback, etc.*
- *Data Access Layer: Handles interaction with the database(s), providing CRUD operations and complex data queries.*

### *3.4. Additional Components:*

- *Authentication and Authorization Service: Manages user authentication and authorization with JWT tokens or similar mechanisms.*
- *Messaging Queue: Facilitates asynchronous communication between microservices for tasks like sending notifications or processing queries.*
- *Caching Layer (optional): Improves performance by caching frequently accessed data.*
- *Logging and Monitoring: Logs system events and monitors performance metrics for troubleshooting and optimization.*

### *3.5. Communication and Integration:*

- *API Gateway: Receives requests from the UI/mobile app and routes them to the appropriate microservices.*
- *Microservices: Communicate with each other using RESTful APIs or message queues.*
- *Data Access Layer: Uses standard database protocols to interact with the database(s).*

### *3.6. Security Considerations:*

- *Implement secure authentication and authorization with role-based access control.*
- *Encrypt data at rest and in transit using industry-standard algorithms.*
- *Regularly perform security audits and penetration testing.*
- *Implement secure coding practices to prevent vulnerabilities.*

### *3.7. Scalability and Performance:*

- *Microservices architecture allows for horizontal scaling of individual functionalities.*
- *Caching layer can improve performance for frequently accessed data.*
- *Database configuration and optimization techniques can ensure efficient data retrieval.*

### *3.8. Deployment Options:*

- *Cloud deployment (e.g., AWS, Azure) offers flexibility and scalability.*
- *On-premises deployment provides more control but requires additional IT infrastructure.*

# *4. User Interface Design:*

## *4.1 Overview:*

*Target Users: Students, faculty, staff (consider specific user roles and needs)*

*Overall Design Goals: User-friendly, intuitive, accessible, responsive across devices*

*Technology Stack: Consider front-end frameworks like React or Angular*

## *4.2 Interface Identification and Description:*

*1. Home Page: Provides an overview of the system, with quick access to key functionalities (search books, manage account, submit query, etc.)*
*2. Book Search: Powerful search with filters (title, author, genre, availability), book details page with information and borrowing/reservation options*
*3. User Account: Manage profile, view borrowing history, fines, and reservations*
*4. Borrowing and Returns: Manage borrows, track due dates, renew books, initiate returns*
*5. Query Management: Submit queries, track status, receive and rate responses*

## *4.3 Interface Design:*

- *General Principles:*
  - *Clean and uncluttered layout with clear hierarchy*
  - *Consistent design elements across interfaces*
  - *Intuitive navigation and user flow*
  - *Responsive design for different screen sizes*
  - *Accessibility features for users with disabilities*
- *Visual Design:*
  - *Use of colors, fonts, and icons that align with your library's branding and user expectations*
  - *High-contrast text and background for readability*
  - *Appropriate use of visuals (book covers, icons) to enhance understanding*
- *Information Architecture:*
  - *Logical organization of information within each interface*
  - *Use of labels, tooltips, and clear instructions where needed*
  - *Search functionality with relevant filters and sorting options*

## *4.4 Interface Features:*

- *Search: Autocomplete suggestions, advanced search filters, search by genre/author/ISBN*
- *Book details: Cover image, description, availability status, user reviews, related books*
- *Borrowing/Reservation: Easy checkout process, due date reminders, renew option*
- *Query Management: Clear categorization of queries, progress tracking, rating/feedback system*
- *Personalization: User profile with preferences, reading lists, saved searches*
- *Notifications: Alerts for due dates, fines, query responses*

# *5. Data Design*

## *5.1 Overview:*

- ***Objectives:*** *Leverage NoSQL's flexibility and scalability for diverse data types and potential future expansion.*
- ***Database Selection:***
    - ***Document database (MongoDB, Couchbase):*** *Store structured and semi-structured data like books, users, borrows, queries, and responses.*

## *5.2 Data Description and Relationships:*

- ***Document Database:***
    - ***Collections:***
        - *Books: Title, author, ISBN, genre, edition, publication date, availability status, location, related books, etc.*
        - *Users: Name, ID, email, contact information, role, borrowing history, fines, reading list, preferences, etc.*
        - *Borrows: Book ID, user ID, borrow date, due date, return date, late fees, etc.*
        - *Queries: Subject, description, user ID, assigned to, response, rating, timestamp, etc.*
        - *Additional collections (optional): Publishers, reviews, keywords, categories, etc.*
    - ***Relationships:***
        - *Embedded references within documents (e.g., Book referencing User in borrow history).*
        - *Use separate collections for related entities (e.g., Publisher documents linked to Book documents).*
        - *Utilize denormalization for frequently accessed data to improve query performance.*

### *5.3 Data Dictionary:*

- *Each document collection defines its own schema using JSON-like structures.*
- *Define data types, constraints, and validation rules within each document to ensure data integrity.*
- *Consider using schema validation tools to enforce consistency and data quality.*

### *5.4 Data Storage and Integration:*

- *Store all data in the chosen document database(s).*
- *Utilize indexing strategies for efficient querying and retrieval based on frequently accessed fields.*
- *Consider sharding for horizontal scaling to handle large datasets.*
- *Explore options for data visualization and analytics directly within the document database, or export data to dedicated analytics platforms.*
- *Be prepared for potential challenges with complex queries or data analysis that might benefit from relational databases.*

## *6. Component Design:*

### *6.1 Overview:*

- ***Microservices Architecture:*** *Divide the system into independent, self-contained services for modularity, scalability, and maintainability.*
- ***NoSQL Databases:*** *Utilize the flexibility and scalability of NoSQL databases for various data types (document, key-value).*
- ***Component Communication:*** *Employ RESTful APIs or message queues (e.g., RabbitMQ, Kafka) for communication between services.*

### *6.2 Component Identification and Description:*

1. ***Book Management:***
    - *Manages book information (add, edit, delete, search).*
    - *Tracks book availability and location (borrowed, available).*
    - *Generates reports on book usage and popularity.*
2. ***User Management:***
    - *Registers and manages user accounts (students, faculty, staff).*
    - *Tracks user borrowing history and fines.*
    - *Allows users to search for books and view availability.*
    - *Implements user roles and permissions for access control.*
3. ***Borrowing and Returns:***
    - *Handles book borrowing and return processes.*
    - *Issues and tracks due dates.*
    - *Calculates and manages late fees.*
    - *Integrates with library self-service kiosks (optional).*
4. ***Query Management:***
    - *Allows users to submit queries related to books, research, or library services.*
    - *Categorizes queries for efficient routing.*
    - *Assigns queries to relevant professors or experts for response.*
    - *Tracks query status and resolution.*

5. ***Data Access Layer (DAL):***
    - *Interacts with the NoSQL databases (MongoDB, Redis) for data storage and retrieval.*
    - *Implements CRUD operations and complex data queries.*
    - *Manages data consistency and integrity across services.*
6. ***API Gateway:***
    - *Acts as a single-entry point for all API requests from the UI/mobile app.*
    - *Routes requests to the appropriate microservices.*
    - *Implements authentication and authorization mechanisms.*

### *6.3 Component Functionality:*

- *Each component has its own well-defined functionality independent of other components.*
- *Components communicate with each other through defined APIs or message queues.*
- *Data access is handled through the DAL, ensuring consistency and security.*

### *6.4 Interface Description:*

- *Each microservice exposes well-documented RESTful APIs for communication.*
- *API documentation includes request/response formats, parameters, and error codes.*
- *Consider using tools like Swagger or API Blueprint for API documentation.*
- *Internal communication between services can be used for message queues for asynchronous messaging.*

### *6.5 Component Implementation:*

- *Choose a programming language and framework based on your team's skills and preferences (e.g., Node.js with Express for microservices, Python with Django for API Gateway).*
- *Utilize libraries and frameworks for working with NoSQL databases*
- *Implement robust logging and monitoring for each component for troubleshooting and performance analysis.*
- *Consider containerization technologies like Docker for easy deployment and scaling of microservices.*

## *7. Appendices:*

### *7.1 Glossary:*

- ***NoSQL:*** *Non-relational database offering flexibility and scalability for diverse data types.*
- ***Microservices:*** *Independent, self-contained services collaborating to build an application.*
- ***API Gateway:*** *Single entry point for API requests, routing them to appropriate microservices.*
- ***Document Database:*** *Stores data in JSON-like documents with flexible schema.*
- ***Key-value Store:*** *Stores data in key-value pairs, ideal for frequently accessed small data.*
- ***RESTful API:*** *Web API adhering to REST principles for consistent communication.*
- ***CRUD:*** *Create, Read, Update, Delete operations on data.*
- ***DAL:*** *Data Access Layer responsible for interacting with databases.*
- ***Schema:*** *Defines the structure and organization of data.*
- ***Denormalization:*** *Duplicating data for faster access and performance trade-offs.*
- ***Sharding:*** *Dividing a database horizontally across multiple servers for scalability.*
- ***Caching:*** *Storing frequently accessed data in memory for faster retrieval.*

### *7.2 Analysis Models (Optional):*

### *7.3 Design Models:*

### *7.4 Data Models:*

### *7.5 Change Log:*

| *Date* | *Change* | *Description* |
|---|---|---|
| *2024-02-25* | *Initial document created* | *Basic design outline for Library Management System with NoSQL.* |