Digital Library Design Document

CMSC 4900 - Senior Project I

Group Members:

Camron Mellott

Josh Watson

Luke Joseph

Instructor Comments / Evaluation

Table of Contents

Abstract

The Digital Library application is an advanced library management system designed to assist users in organizing, discovering, and tracking their reading history. The System will allow users to create a personal account where they can manually upload books through ISBNs, mark books as either “Currently Reading” or “Completed”, log notes / reflections on each entry, and receive AI-generated book recommendations based on their past readings. The purpose of this document is to provide the team with a clear plan that constructs the project's requirements into an implementable design.

Description of the Document

Purpose and use

The purpose of this document is to explain the requirements defined by the specification document into a clear and applicable plan for the development. The Specification document outlined what the system must accomplish, whereas the design document outlines how the system will accomplish those requirements. This document describes the system architecture, modules involved, their responsibilities, and how they will interact with each other. This document will guide the team through the implementation phase, help maintain consistency across the project, and serve as a reference for any design decision made throughout the development process.

Ties to Specification Document

This document builds off the Specification document that defined the projects goals and required features. The Specification document listed these functionalities: allowing users to create profiles to save their progress, logging books as either "Currently Reading" or "Completed", scanning books using ISBNs, writing notes / reflections for each entry, and personal recommendations based on reading history. We will separate those requirements into modules, data structures, workflows, and design strategies. Every design decision made in this document is connected back to the acceptance criteria laid out in the specification document in order to ensure that the system design fully supports the expected end-user behavior and meets all requirements.

Intended Audience

The intended audience of this document includes the members of the development team and the client. The developers will use this document as a blueprint during the implementation phase. This document will ensure that everyone involved will have a shared understanding of how the Digital Library application will be constructed.

Project Block System Diagram

The Digital Library System will be composed of two main components, such as the client-side application and a cloud-based backend service. The client application will run on user devices such as smartphones, tablets, or computers. This will serve as the primary interface for uploading books through ISBNs, browsing/organizing users' personal collections, and viewing AI-generated recommendations.

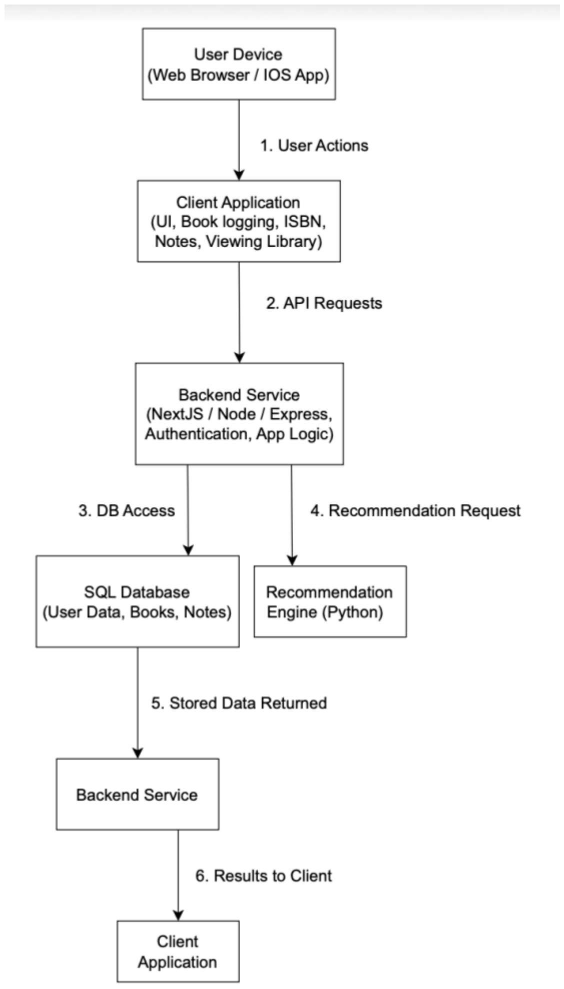
User Device (Web Browser / IOS App)
1. User Actions
Client Application (UI, Book logging, ISBN, Notes, Viewing Library)
2. API Requests
Backend Service (NextJS / Node / Express, Authentication, App Logic)
3. DB Access
4. Recommendation Request
SQL Database (User Data, Books, Notes)
Recommendation Engine (Python)
5. Stored Data Returned
Backend Service
6. Results to Client
Client Application

Figure 1

Design Details

System Modules and Responsibilities:

Architectural Diagram

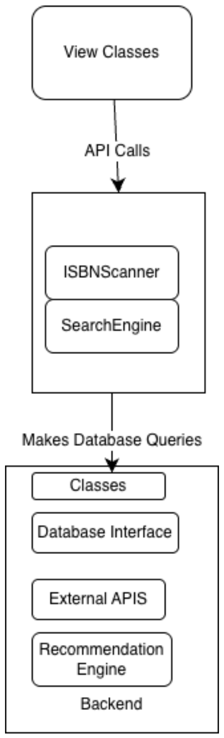

Figure 2: Architectural Diagram

Figure 2 description: The defined view classes will be used to view different interfaces within the application. Those different interfaces will make API calls to the search engine or ISBNScanner depending on user preference. Those results will be used in database queries to find book information, utilizing the appropriate classes. That information will be received from external APIs. In addition, the RecommendationEngine will also be queried using information from the databases.

User Management Module – Handles registration, login, authentication, and user profile management

Book Management Module – Manages adding books, categorizing books, and storing book data

Search Module – Processes user queries, interfaces with the database, returns search results

ISBN Scanner Module - Interfaces with device camera, decodes ISBN barcodes, fetches book data from external API.

Recommendation Engine Module – Analyzes user reading patterns, generates personalized recommendations using ML algorithms

Database Interface Module – Manages all operations with the SQL database

API Gateway Module – Handles communication between frontend and backend services

Module Cohesion:

Our project expresses module cohesion through well defined modules that each serve a specific purpose.

User Management Module demonstrates functional cohesion at the highest level. All functions within this module relate directly to user account operations such as registration, authentication, profile management, and session handling.

Book Management Module demonstrates functional cohesion as well. This module contains all features necessary to manage a user's personal library. User's can add books (manually or via ISBN), categorize them (Want to read, Currently Reading, Completed, and delete books. Having all book-related functions in one module will help keep the code readable and logical.

Search Module demonstrates functional cohesion by focusing exclusively on book discovery. All functions within contribute to the single task of helping users find books in the database.

All View Classes contain the code necessary to handle the user interface. They are independent of each other. These classes will include everything that the user sees within the interface, this includes: LoginView, LibraryView, SearchView, ScannerView, and RecommendationView.

Module Coupling:

For the mobile application, there are various types of View classes that will be utilized together to accomplish different screen views. The LibraryView, SearchView, ScannerView, and RecommendationView classes are all related to each other as part of the user interface layer, but each has its own specific purpose and can function independently.

The children of these views are the controller classes (SearchEngine, ISBNScanner, RecommendationEngine, BookManager). The controller classes respond to the actions that the user requests. The mobile application will be using the MVC (Model-View-Controller) system. This is a common system used in modern GUI applications, it allows for flexible structures where application functions and interface functions are independent of each other. This system ensures low coupling between the presentation and business logic.

The backend program will express sequential coupling with ISBNScanner, ExternalAPI and BookManager classes. When a user scans a book's ISBN barcode, the following classes will process:

1. ISBNScanner captures and decodes the barcode
2. ExternalAPI fetches book metadata using ISBN
3. BookManager stores the book data in the database and updates the user's library

Each class relies on the output of the previous class to successfully complete the workflow.

The RecommendationEngine demonstrates data coupling with other modules. It receives only the necessary data parameters from the Database Interface Module and returns a simple array of recommended Book objects. No complex data structures are utilized for the passing of information between modules.

Similarly, the Authentication Module exhibits data coupling with the User Management Module, only passing essential credentials such as username and password for validation without exposing internal details.

Design Analysis

Data Flow:

The dataflow of the Digital Library is handled through API endpoints that pass JSON payloads between the frontend and backend. The system follows a client-server architecture where data flows bidirectionally based on user actions.

Workflows:

Add Book via ISBN :

The user activated the ISBN scanner within the mobile application and captures a book's barcode. The image data is processed to extract the ISBN number, which is sent to the backend server via API request. The backend queries the External API Module to fetch the book data. This data is validated and stored in the SQL database, and associated with the user's account. The book object is then returned to the frontend, where it is displayed in the user's selected category.

Recommendation Workflow :

When the user requests recommendations, the frontend sends the userID to the backend. The Recommendation Engine queries the database to retrieve the user's reading history. This data is passed to the machine learning model, which analyzes reading patterns and generates a list of recommended books. The recommendations are returned to the frontend and displayed to the user.

Search Workflow:

The user enters a search query in the SearchView, which then sends the query string to the backend Search Engine. The engine constructs an SQL query and retrieves matching books from the database. Results are then returned to the frontend for display.

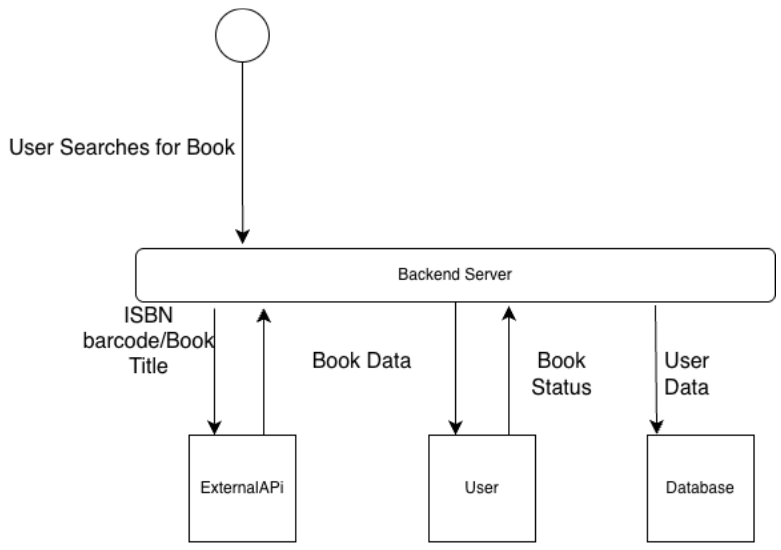

Figure 3: Dataflow Diagram

Figure 3: The dataflow diagram shows the interactions between the application and user. When a user searches for a book, the book title or ISBN contact the ExternalAPI and acquire the book data and returns it to the user. The user indicates the book status which is then packaged with the user's data and is stored in the database.

Design Organization

Detailed tabular description of Classes / Objects:

Class Name: User Class

Description: The User class represents an individual user account within the Digital Library System. It stores user credentials, profile information, and manages authentication state.

Data Members:

Member Name	**Type**	**Constraints**
userID	String	Primary key, not null
username	String	3-20 characters, alphanumeric, unique
passwordHash	String	Hashed, not null
email	String	Valid email format, unique
firstName	String	1-50 characters, alphabetic
lastName	String	1-50 characters, alphabetic
dateCreated	DateTime	Not null

Member Functions: validateCredentials(username,password), createUserProfile(userData), getLibrary()

Class Name: Book Class

Description: The book class represents book metadata retrieved from external APIs or manual user entry. It stores all relevant information about a book.

Data Members:

Member Name	Type	Constraints
bookID	String	Primary key, not null
isbn	String	10 or 13 digits, unique
title	String	1-500 characters, not null
author	String	1-200 characters, not null
genre	String	1-100 characters, nullable
description	String	0-2000 characters, nullable
coverImageURL	String	Valid URL format, nullable

Member Functions: SearchByTitle(title), fetchByISBN(isbn), getRecommendationFeatures()

Class Name: UserBook Class

Description: The UserBook class represents the association between a user and a book.

Data Members:

Member Name	Type	Constraints
userbookID	String	Primary key, not null
userID	String	Foreign to user, not null
bookId	String	Foreign to book, not null
status	Enum	Values: Want to Read, Currently Reading, Completed
dateAdded	DateTime	Not Null, auto generated

Member functions: addtoLibrary(userID,bookID,status), updateStatus(newStatus), deleteFromLibrary()

Class Name: ISBNScanner Class

Description: The ISBNScanner class interfaces with the device camera to capture and decode ISBN barcodes from physical books.

Member Name	**Type**	**Constraints**
cameraStream	Object	Camera API object, not null when active
scannerState	Enum	Values: Idle, Scanning, Processing, Error

Member functions: initializeCamera(), startScanning(), stopScanning(), decodeBarcode(imageData),validateISBN(isbn),fetchBookData(isbn)

Class Name: RecommendationEngine

Description: The RecommendationEngine Class analyzes user reading patterns and generates personalized book recommendations using machine learning algorithms.

Data Members:

Member Name	**Type**	**Constraints**
modelVersion	String	Version format
trainingData	Array	User reading history objects
recommendationCache	Object	UserId-keyed cache

Member functions: analyzeUserPatterns(userID), generateRecommendations(userID)

Class Name: ExternalAPI Class

Description: The ExternalAPI class manages communication with third party book data services such as Google Books API and Open Library API to fetch book metadata

Data Members:

Member Name	**Type**	**Constraints**
apiKey	String	Not null, stored securely
baseURL	String	Valid URL format, not null
requestCount	Integer	Non negative number, tracks API usage

Member functions: fetchBookByISBN(isbn), searchBooks(query)

Class Name: LibraryView Class

Description: The LibraryView class manages the user interface for displaying the user's personal book library.

Data Members:

Member Name	**Type**	**Constraints**
userID	String	Foreign key to User, not null
currentCategory	Enum	Values: Want to read, Currently Reading, Completed, All

Member functions: switchCategory(category)

Class Name: SearchView Class

Description: The SearchView class manages the user interface for searching and discovering books in the database

Data Members:

Member Name	**Type**	**Constraints**
searchQuery	String	0-200 characters
searchResults	Array	Array of book objects

Member functions: submitSearch(query), displayResults(results)

Class Name: ScannerView Class

Description: The ScannerView class manages the user interface for the ISBN barcode scanning feature

Data Members:

Member Name	**Type**	**Constraints**
cameraActive	Boolean	Indicates camera display status
scanningProgress	Boolean	Indicates active scan

Member functions: displayCamera(), showScanResult(bookData), showScanError(errorMessage)

Class Name: RecommendationView Class

Description: The RecommendationView Class mananges the user interface for displaying AI-generated book recommendations

Data Members:

Member Name	Type	Constraints
Recommendations	Array	Array of book objects
displayCount	Integer	Number of recommendations to show
dateAdded	DateTime	Not Null, auto generated

Member functions: displayRecommendations(recommendations)

Class Name: DatabaseInterface Class

Description: The DatabaseInterface class manages all database operations, providing a unified interface for SQL operations on User, Book, and UserBook tables.

Data Members:

Member Name	Type	Constraints
connectionPool	Object	Database connection pool, not null
queryTimeout	Integer	Milliseconds
transactionActive	Boolean	Indicated an active transaction

Member functions: connect(), disconnect(), executeQuery(query,parameters), beginTransaction(), commitTransaction(),rollbackTransaction()

Class Name: AuthenticationModule

Description: The AuthenticationModule class handles user authentication, session management, and security operations.

Data Members:

Member Name	Type	Constraints
sessionToken	String	JWT format, nullable
tokenExpiration	DateTime	Not null when session active
refreshToken	String	Nullable, used for token renewal

Member functions: login(username,password), validateSession(token), refreshSession(refreshToken)

Class Name: BookManager Class

Description: The BookManager class coordinates book related operations, serving as a controller between UI and data layers.

Data Members:

Member Name	Type	Constraints
activeUser	String	UserID of current user, not null
pendingOperations	Array	Queue of book operations

Member functions: addManually(bookData,status), addISBN(isbn,status), removeBook(userBookID),moveBook(userBookID, newStatus)

Functional Descriptions:

validateCredentials():

Input: The validateCredentials() function takes a username and password string as input.

Output: The function queries the database to retrieve the stored passwordHash for the given input, and compares the provided password with the hashed password.

Return Parameters: The function returns a Boolean indicating authentication success or failure, along with a session token if successful, or error message if authentication failure occurs.

Types: The data types used within are string, Boolean, and object

createUserProfile():

Input: Requires userData object containing username, password, email, firstName, and lastName

Output: creates a new User record in the database with a unique userID and hashed password

Return Parameters: returns a success Boolean and newly created userID, or throws exceptions for duplicate credentials.

Types: object, string and Boolean

getLibrary():

Input: None

Output: Retrieves all UserBook records associated with the authenticated user

Return Parameters: returns an array of UserBook objects with associated book data, or empty array if no books exist.

Types: array, object, string

fetchByISBN():

Input: 10 or 13 digit ISBN string

Output: Queries the external API database for a Book record matching given ISBN

Return Parameters: returns a Book object if found, null if not found, along with success boolean

Types: string, object, Boolean

searchByTitle():

Input: Receives a title string

Output: Queries database for Book records with titles matching search string

Return Parameters: returns an array of Book objects, or empty array if no matches are found

Types: string, array, object

getRecommendationFeatures():

Input: None

Output: Extracts features used by recommendation engine

Return Parameters: returns an object containing book info such as title, author, genre.

Types: Object, string, integer.

addToLibrary():

Input: The function requires a userID, bookID, and status

Output: The function will output a new UserBook record, linking the user to the book with the specified reading status

Return Parameters: returns an object containing: success Boolean, userBookID string, message string, or throws exception for invalid entries(string).

Types: string, enum, Boolean, object, DateTime.

updateStatus():

Input: The function requires userBookID, and newStatus

Output: Modifies the status field of the UserBook record and updates associated timestamps.

Return Parameters: returns an object containing success Boolean, updated UserBook object, and message, as well as error messages(strings).

Types: String ,enum, Boolean, object, DateTime.

deleteFromLibrary():

Input: userBookID

Output: Permanently removes the UserBook record from the database

Return Parameters: returns an object containing success Boolean, and message string, as well as error messages indicating deletion failure(string).

Types: String, Boolean, object.

initializeCamera():

Input: None

Output: The function requests camera permissions and initializes camera API

Return Parameters: returns an object containing success Boolean, and message string.

Types: Boolean, object, string

startScanning():

Input: None

Output: Mobile device camera is activated and displays a viewfinder for ISBN barcodes.

Return Parameters: returns an object containing success Boolean, scannerState enum and a message string, as well as error codes for denied camera permissions or camera failure (string).

Types: Boolean, enum, object, string

stopScanning ():

Input: None

Output: Deactivates the camera and closes viewfinder

Return Parameters: returns an object containing success Boolean, scannerState enum, and message string

Types: Boolean, enum, object, and string

decodeBarcode():

Input: receives imageData from camera stream

Output: Processes the image to detect and decode any ISBN barcodes

Return Parameters: returns an object containing success Boolean, isbn (string or null), and message string.

Types: string, Boolean, object, null

validateISBN():

Input: Receives isbn string

Output: Verifies the ISBN format using either the ISBN-10 or ISBN-13 format.

Return Parameters: Returns an object containing a valid Boolean, isbn string, format string, and a message string

Types: Object, string, integer.

fetchBookData():

Input: Receives isbn string

Output: Retrieves book metadata by making a call to the ExternalAPI and returning the result

Return Parameters: returns an object containing success Boolean, book object or null if book doesn't exist, and message string.

Types: string, object, Boolean, and null.

analyzeUserPatterns():

Input: Receives userID string

Output: Retrieves the user's reading history and extracts patterns based on preferred genres, and authors.

Return Parameters: returns a pattern object, within that object will include: genrePreference, and authorPreference arrays

Types: string, object, array

generateRecommendations():

Input: Requires a userID string

Output: analyzes user patterns and generates personalized book recommendations

Return Parameters: returns an object containing success Boolean, and recommendations in an array of book objects.

Types: string, array, object, boolean

fetchBookbyISBN():

Input: Receives isbn string

Output: The function sends a GET request to the external book API and extracts book metadata from the received JSON response

Return Parameters: returns an object containing success Boolean, book object, and message string or null if book is not found.

Types: Boolean, object, string, null

searchBooks():

Input: Receives query string

Output: The function sends a search request to the External API to find books matching the query string

Return Parameters: returns an object success boolean, results in the form of an array of book objects, a message string, and an empty array if no matches were found

Types: object, string, array, boolean

switchCategory():

Input: Receives category enum (Want to read, Currently Reading, Completed, All)

Output: The function filters the displayed books based on selected category

Return Parameters: The function returns an object containing a success Boolean, currentCategory enum and an array of UserBook objects called displayedBooks

Types: object, enum,array, Boolean

submitSearch():

Input: Receives query string

Output: The function sends the search query to the SearchEngine

Return Parameters: Returns an object containing a success Boolean, query string and requestID string

Types: object, string,boolean

displayResults():

Input: Receives an array of Book objects called results

Output: The function displays the search results in the UI.

Return Parameters: returns an object containing a success Boolean, and message string

Types: array, string, Boolean, object

displayCamera()

Input: None

Output: This function renders the camera viewfinder.

Return Parameters: returns an object containing a success Boolean, viewFinderActive Boolean, and message string

Types: Boolean, object, string

showScanResult():

Input: receives bookData book object

Output: displays the scanned book information

Return Parameters: returns an object containing a success boolean, and message string

Types: object, boolean, and string.

showScanError():

Input: receives errorMessage string

Output: displays an error message to the user with retry options.

Return Parameters: returns an object containing a success boolean

Types: string, boolean, and object.

displayRecommendations():

Input: receives array of Book objects called recommendations

Output: renders recommended books onto the screen

Return Parameters: returns an object containing a success Boolean, and string message

Types: string, boolean, array, and object

connect():

Input: none

Output: establishes connection to the SQL database using the connection pool

Return Parameters: returns an object containing success Boolean, connectionID string and message string

Types: string, boolean, and object.

disconnect():

Input: none

Output: closes the database connection

Return Parameters: returns an object containing a success Boolean and message string

Types: string, boolean, and object.

executeQuery():

Input: requires query string and parameters for the query in an array

Output: executes the SQL query to the database and returns results

Return Parameters: returns an object containing a success Boolean, and results array or null for no results

Types: string, array, boolean, null and object.

beginTransaction():

Input: none

Output: the function starts a database transaction

Return Parameters: returns an object containing a success Boolean, transactionID string and message string

Types: string, boolean, and object.

commitTransaction ():

Input: none

Output: The function commits the current transaction, making all changes permanent

Return Parameters: returns an object containing a success Boolean, transactionID string, and message string

Types: string, boolean, and object.

rollbackTransaction():

Input: none

Output: the function rolls back the current transaction, undoing all changes since beginTransaction()

Return Parameters: returns an object containing a success Boolean, transactionID string and message string

Types: string, boolean, and object.

login():

Input: requires a username string and password string

Output: validates credentials and creates a session if authentication is successful

Return Parameters: returns an object containing a success Boolean, sessionToken string, userID string, expiration DateTime, and message string

Types: string, boolean, and object.

validateSession():

Input: requires a JSON web token string called token

Output: verifies the session token's signature and expiration

Return Parameters: returns an object containing success Boolean, userID string, expires DateTime or null, and message string

Types: string, boolean, DateTime, null, and object.

refreshSession():

Input: requires refreshToken string

Output: generates a new session token using a valid refresh token

Return Parameters: returns an object containing success boolean, a JSON web token string called newSessionToken, expires DateTime, and message string

Types: string, boolean, DateTime, and object.

addManually():

Input: requires bookData object and status enum

Output: validates the book data, creates a Book record if it doesn't exist, and adds to user library

Return Parameters: returns an object containing a success Boolean, bookID string, userBookID string, and message string

Types: string, enum, boolean, and object.

addISBN():

Input: requires isbn string, status enum

Output: function fetches book data using the ISBN, creates a Book record if it doesn't exist, and adds to user library.

Return Parameters: returns an object containing success Boolean, bookID string, userBookID string, bookData Book object, and message string

Types: string, enum, boolean, and object.

removeBook():

Input: requires userBookID string

Output: Removes the book from the user's library by deleting the userBook record

Return Parameters: returns an object containing a success Boolean, and message string

Types: string, boolean, and object.

moveBook():

Input: requires userBookID string and newStatus enum

Output: updates the reading status of a book in the user's library

Return Parameters: returns an object containing success Boolean, updatedUserBook UserBook object, and message string

Types: string, boolean, and object

Files Accessed

The Digital Library application will only store a small amount of information on the user's device. The primary files that are stored locally will be the application package itself when the user installs the mobile version, or cached web data when accessing the web version through a browser. All long-term data such as user's profiles, logged books, reading completeness, notes, and recommendation history will be stored securely on the cloud-based backend server

Real-time Requirements

In order for the Digital Library application to maintain a smooth experience; tasks such as logging books, loading reading history, and generating recommendations should occur with minimal delay. This will ensure that users do not become frustrated and leave the application. A stable internet connection or mobile data is required for syncing data, retrieving book details, and accessing recommendations. If there is no connection available, the app will still be able to open, but core features will only function once there is a proper connection.

Messages

…

Narrative / PDL

Decision: Programming Language / Reuse / Portability

The main language we will be using to develop the application will be JavaScript. We will be utilizing a cross-platform native runtime called Capacitor to easily allow us to build for web and mobile use. The framework depends on Node.js and npm for package management and backend integration. This approach will allow us to do more and support more platforms while writing less code.

Implementation Timeline

Activity	Jan	Feb	Mar	Apr	May
Project Setup	✓				
Database Design	✓	✓			

Core Book Features		✓	✓		
ISBN Scanner			✓		
External API implementation			✓	✓	
Recommendation system				✓	✓
Testing	✓	✓	✓	✓	✓

Design Testing

Design testing will be done regularly throughout the development of the project. Each major feature will be tested independently of the rest of the application to ensure a comprehensive application. Features like the ISBN scanner, database functions, and the recommendation engine will need to be thoroughly tested prior to integration with the overall project. Once the application is complete, the team will begin debugging the application. We will try to give our application bad data and attempt to break it. This will ensure that our application was implemented effectively

The team intends to do thorough testing through personal use once a working version is available.

Sources

Flowchart Maker & Online Diagram Software. (n.d.). App.diagrams.net. https://app.diagrams.net/#

Appendix: Team Details

Camron Mellott:

- Abstract
- Description of Document
    - Purpose and Use
    - Ties to Specification
    - Intended Audience
- Project Block System Diagram
- Files Accessed
- Real-time requirements

Josh Watson:

- Design Details
    - System Modules and responsibilities

        - Architechtural Diagram
        - Module Cohesion
        - Module Coupling
    - Design Analysis
        - Data Flow Chart
    - Design Organization
    - Functional Description
        - Input/Output/return parameters/types
    - Implementation Timeline
    - Design Testing
    - Appendix: Team Details

Luke Joseph:

-

Appendix: Writing Center Report

Cal u Writing Center report

Staff: Jacob Kerfonta

Date: December 9th 2025, 4:30-5:00

Description / Notes: Looking over the groups project, I noticed no issues with the project that I can actually review. My expertise in programming is limited, so I could only look over spelling and sentence cohesion, which showed no issues.

Appendix: Workflow Authentication

I Camron Mellott, attest that I have performed the actions specified in this document

Signature: Camron Mellott Date: 12/09/25

I Josh Watson, attest that I have performed the actions specified in this document

Signature: [illegible] Date: 12/09/25

I Luke Joseph, attest that I have performed the actions specified in this document

Signature: Luke Joseph Date: 12/9/25