**INDY-1 Blue — Book Dating App**

**Software Design Document**

**CS 4850 - Section 02 – Fall 2024**

**Professor Perry – Aug 30, 2024**

|  |  |
| --- | --- |
| A person with long hair smiling  Description automatically generated  Chelsea Mandel  Documentation | A person wearing sunglasses and a life jacket  Description automatically generated  Sebastian Arechaga  Developer |
| A person taking a selfie  Description automatically generated  Obie Ezeobi  Developer | A person smiling at the camera  Description automatically generated  Keaton Moneymaker  Developer |

**Team Members:**

|  |  |  |
| --- | --- | --- |
| Name | Role | Cell Phone / Alt Email |
| Obie Ezeobi | Developer | [678.655.0628](tel:6786650628)  [oezeobi@students.kennesaw.edu](mailto:oezeobi@students.kennesaw.edu) |
| Chelsea Mandel | Documentation | [423.285.4514](tel:4232854514)  [cmandel1@students.kennesaw.edu](mailto:cmandel1@students.kennesaw.edu) |
| Sebastian Arechaga | Developer | [678.213.6014](tel:6782136014)  [sarechag@students.kennesaw.edu](mailto:sarechag@students.kennesaw.edu) |
| Keaton Moneymaker | Developer | [706.260.5079](tel:7062605079)  [kmoneym1@students.kennesaw.edu](mailto:kmoneym1@students.kennesaw.edu) |
| Sharon Perry | Project Owner or Advisor | [770.329.3895](tel:7703293895)  [Sperry46@kennesaw.edu](mailto:Sperry46@kennesaw.edu) |

# Table of Contents

[Table of Contents 2](#_Toc1191354940)

[1. Introduction and Overview 3](#_Toc981967455)

[2. Design Considerations 3](#_Toc1775929159)

[2.1. Assumptions and Dependencies 3](#_Toc905752029)

[2.2. General Constraints 3](#_Toc1329235783)

[2.3. Development Methods 4](#_Toc421592599)

[3. Architectural Strategies 4](#_Toc433931375)

[4. System Architecture 6](#_Toc399413108)

[5. Detailed System Design 6](#_Toc1253936698)

[5.1. Classification 6](#_Toc480239238)

[5.2. Definition 6](#_Toc452473990)

[5.3. Constraints 6](#_Toc1087125928)

[5.4. Resources 7](#_Toc1926523489)

[5.5. Interface/Exports 7](#_Toc834418617)

[6. Glossary 7](#_Toc581513254)

[7. Bibliography 7](#_Toc1850953208)

# **1.** **Introduction and Overview**

This document outlines the design of the software application, detailing how the requirements specified in the SRS will be met. The document is intended to provide a high-level overview of the system architecture and provide details on the individual components that make up the system. This document will provide guidance to the development team regarding the structure and architecture of the system being developed.

# **2.** **Design Considerations**

## ***2.1.*** ***Assumptions and Dependencies***

The following are assumptions or dependencies regarding the software and its use. Use of the application is dependent on a stable internet connection. Account creation is dependent on the user having access to an email address or an accepted third-party authentication service. It is assumed that end users will interact with the product in a predictable manner. Familiarity with online profile management, as in the case of selecting a username and password and interacting with profile settings, is assumed. Familiarity with the concept of online dating website norms will, to an extent, be assumed on the part of the user. Such norms include an understanding of the binary accept or reject system and the user-preference guided suggestions as these will not be explicitly explained.

## ***2.2.*** ***General Constraints***

Describe any global limitations or constraints that have a significant impact on the design of the system's software (and describe the associated impact). Such constraints may be imposed by any of the following (the list is not exhaustive):

* Hardware or software environment
* End-user environment
* Availability or volatility of resources
* Standards compliance
* Interoperability requirements
* Interface/protocol requirements
* Data repository and distribution requirements
* Security requirements (or other such regulations)
* Memory and other capacity limitations
* Performance requirements
* Network communications
* Verification and validation requirements (testing)
* Other means of addressing quality goals
* Other requirements described in the requirements specification

## ***2.3.*** ***Development Methods***

The Agile Software Development methodology will be the primary approach for this software design. Focus will be placed on maintaining a steady work pace, delivering working software consistently, keeping open and frequently used communication channels, prioritizing simplicity of design over intricate design, and adapting to change. Some portions of the methodology that will not be applicable to this project are the maintenance process phase and the distinction of multiple teams.

# **3.** **Architectural Strategies**

HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) will be used for the structure and layout of the pages. React JS will be used for constructing the UI (user interface). React JS was chosen in consideration of the Agile Software Development methodology for its simple approach to UI construction.

The source code will utilize the JavaScript programming language. MongoDB will be used for document storage. Apache Web Server will host the web server. The Google Books database of books will be accessed for book candidates.

The application may be extended to a mobile application that functions on IOS and Android. Additional monetization may be added such as premium accounts with exclusive features.

# **4.** **System Architecture**

The subsystems of this system will include the book candidate search and recommendation subsystem and the user profile system. The book candidate search and recommendation subsystem will include components performing book database retrieval and book filtering. The user profile system will include components performing user personal information retrieval and update and saved books list management.

# **5.** **Detailed System Design**

Most components described in the System Architecture section will require a more detailed discussion. Other lower-level components and subcomponents may need to be described as well. Each subsection of this section will refer to or contain a detailed description of a system software component. The discussion provided should cover the following software component attributes:

## ***5.1.*** ***Classification***

## *5.1.1* *Database Retrieval Component*

Database retrieval will be handled by a class performing the necessary operations by accessing the database of books.

## *5.1.2* *Filtering Component*

Filtering will be performed through a function.

## *5.1.3 User Personal Information Customization*

Retrieval and updating of the user profile information will be handled by a class accessing the database of user information.

# *5.1.4 Saved Books Component*

The list of saved books will be managed by a class.

## ***5.2.*** ***Definition***

#### 5.2.1 Database Retrieval Component

As mentioned in 3.2.1 of the SRS, there will be a selection of books which can be suggested to the user. The database retrieval component uses a live database like Google Books or Open Library to supply book information.

#### 5.2.2 Filtering Component

In this case, filtering refers to skewing the book search to a user-defined filter chosen based on their book interests. Referring to 3.2.3 of the SRS, the filtering component will aid in narrowing down the results of a user’s book candidates.

#### 5.2.3 Profile Customization

Addressing the requirement 3.3.1 in the SRS, profile customization displays and allows editing of saved user information including login credentials and book preferences.

#### 5.2.4 Saved Books

Addressing the requirement 3.3.2 in the SRS, the saved books component will allow viewing, updating, and sorting of the saved books list.

## ***5.3.*** ***Constraints***

#### 5.3.1 Database Retrieval Component

As the database will fuel our supply of books for the program, there needs to be an adequate response time to maintain the user’s reservoir of goodwill. There will need to be a limit on the storage used by the database as the cache of books may overflow a user’s device if is not managed.

#### 5.3.2 Filtering Component

A precondition constraint for the filtering component would be the scope of categories in the filter. There will be categories to select from on the filtering component such as genre or read length, but we will have to have enough options to satisfy the use of a filtering feature in the first place.

#### 5.3.3 Profile Customization

There are a handful of constraints to consider for profile customization. A few would include appropriate profile update periods, limiting profile biographies and how to store them, and a profile’s state being saved and updated each session.

#### 5.3.4 Saved Books

A constraint for the book saving component would be the ensured storage and access to each individual user’s saved books list and the ability to update it in real time. The component should maintain the saved state of each user’s list synced between the server and the user’s latest changes.

## ***5.4.*** ***Resources***

#### 5.4.1 Database Retrieval Component

The Google Books database is needed to provide the collection of books that the application draws from which will be stored using MongoDB. A potential race condition would be many users attempting to fetch the same data about a book at once and bog down the system. The solution we could apply would be a client-based cache to be used instead of making iterative external calls to the database.

#### 5.4.2 Filtering Component

Using MongoDB, we will create our own filtering component which can surf through the queries of profile and books alike. There is a possibility of simultaneous searches slowing down overall performance, but this would be resolved by a stateless filter component.

#### 5.4.3 Profile Customization

A MongoDB database is needed to store user information. Also, a race condition can be found when a user attempts to update their profile while logged into two separate sessions, on different devices for example, and they attempt to update their profile. Optimistic locking would be a solution to this race condition, the record of the profile is versioned, so each update checks the database and refers to the server and session copy before making an edit.

#### 5.4.4 Saved Books

A connected MongoDB database will link a user’s profile with their saved list of books. Similar to the profile customization, the saved books component would have a race condition if updating their list on multiple devices, which could also be solved by optimistic locking.

## ***5.5.*** ***Interface/Exports***

The set of services (resources, data, types, constants, subroutines, and exceptions) that are provided by this component. The precise definition or declaration of each such element should be present, along with comments or annotations describing the meanings of values, parameters, etc. .... For each service element described, include (or provide a reference) in its discussion a description of its important software component attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

#### 5.5.1 Database Retrieval Component

##### *Classification:*

A database retrieval class. (BookDBFetcher)

##### *Definition:*

‘BookDBFetcher’ will interface with whichever external database source is chosen and fetch it for the user. (Google Books, Open Library, etc.)

##### *Responsibilities:*

* + Retrieve book data from external sources.
  + Cache the data and format it to suit the interface.

##### *Constraints:*

* + Speedy response time for positive user experience.
  + Limited cache size to prevent heavy storage use.

##### *Composition:*

* + Functions: queryBook(String input), fetchDetails(String bookName), cacheData(List<String> bookData)
    - ‘queryBook’ will search for a particular book (input) and display the results
    - ‘fetchDetails’ will be a function that runs once someone inspects a book closer and it will return the data necessary to display its metadata
    - ‘cacheData’ will run in the background as someone is loading or reading about a book and will return data which is cached for later use

##### *Uses:*

* + The Filtering component will use ‘BookDBFetcher’ to retrieve data for the search to apply its filter to.

##### *Resources:*

This component depends on Google Books or OpenLibrary to retrieve data about multitudes of books which is then stored by a MongoDB database.

##### *Processing:*

The class is called to search for many books, then the results are formatted for use, and cached for later.

##### *Interface:*

* + ‘queryBook(String input) returns a List<String>
  + The returned data is a list of many books.

#### 5.5.2 Filtering Component

##### *Classification:*

The filtering will be done through a function (searchFor).

##### *Definition:*

This function uses the user-defined filter to search for a narrowed selection of books.

##### *Responsibilities:*

* + Find books based on their genre, read time, time written, etc.
  + Narrow the user’s choices and start searching through a smaller sample.

##### *Constraints:*

* + The limiting constraint of a filter component is the pre-defined categories necessary to facilitate a proper layer of filters which aids the user experience.

##### *Composition:*

* + searchFor(bookFilter filter1) returns a list of queries matching the filter

##### *Uses:*

* + The saved books component will likely use this to quickly sort through certain types of books a user has previously saved.

##### *Resources:*

The filter will use a MongoDB database, but it will be developed by us. Another resource used by the filter would be system memory used to cache the results.

##### *Processing:*

A user chooses on the website UI what preferences they want to look for in a book and the ‘searchFor’ component will fetch books using the ‘BookDBFetcher’ and apply a filter to the fetched results and then convert them to a viewable format for the user.

##### *Interface:*

* + ‘searchFor(bookFilter filter1)’ returns the population of books that fit the filtered search.
  + ‘filter1’ is the filter object created after the user has selected their choices.
  + bookFilter is an object type which will consist of the many attributes chosen by the user to filter the search.

#### 5.5.3 Profile Customization

##### *Classification:*

A class which stores a profile object in an independent database (ProfileManager).

##### *Definition:*

‘ProfileManager’ is a class that notes all the changes a user wants to save and updates the database with their decision.

##### *Responsibilities:*

* + Fetching user data to display.
  + Taking in user input and updating database.

##### *Constraints:*

* + Limited biography lengths.
  + Updating and saving each profile state.

##### *Composition:*

* + Functions: fetchProfile(String username), updateProfile(String username, Profile newProfile)

##### *Uses:*

* + Used by the front-end to fetch data for a user’s page.
  + Used by the back-end to update data for user changes to their profile.

##### *Resources:*

An external database will store all of the profiles, we will use MongoDB to make one.

##### *Processing:*

A user will view their page and the ‘fetchProfile()’ method will retrieve their details. If they want to update them they can press and button and the ‘updateProfile()’ will send the updated ‘Profile’ object to the database to update.

##### *Interface:*

* + fetchProfile(String username)
    - Returns a List<String> which stores all the data for someone’s profile
    - Username is the id to target a certain user’s profile.
  + updateProfile(string username, Profile newProfile)
    - Profile is an object type which stores all the attributes that make up a profile and newProfile is the updated version of someone’s profile after they finish changing it.

#### 5.5.4 Saved Books

##### *Classification:*

A class which stores the saved books for a user (SavedBookList).

##### *Definition:*

‘SavedBookList’ deals with fetching a certain user’s list of saved books and handles editing and making changes to the list.

##### *Responsibilities:*

* + Store and revise user-saved books.
  + Synchronize identical lists across devices.

##### *Constraints:*

* + The class must have an adequate response time as the many changes will bog down the experience. There must also be ensured access to the database.

##### *Composition:*

* + Functions: fetchSavedBooks(String username), addBook(String username, String bookName), removeBook(String username, String bookName)

##### *Uses:*

* + This class fetches the data displayed by the front-end while also updating the data stored by the back-end similar to the “Profile Customization” component.

##### *Resources:*

Once again, ‘SavedBookList’ will have its own MongoDB database created and utilized solely by this class.

##### *Processing:*

When a user goes to their saved books page the class uses ‘fetchSavedBooks’ to retrieve the saved data for that user’s account. Then use either ‘addBook’ or ‘removeBook’ to make changes and send updates to the database.

##### *Interface:*

* + fetchSavedBooks(String username)
    - Returns a List<String> of saved books tied to the user’s profile (username).
  + addBook(String username, String bookName) and removeBook(String username, String bookName)
    - A void statement.
    - Updates the data tied to the user’s account and updates the user’s UI to show their changes, of either removing or adding books.

# **6.** **Glossary**

**Agile Software Development:** A methodology for software development emphasizing adaptability and communication.

**Candidate:** A book in the user’s queue of suggestions pending acceptance or rejection.

**CSS:** Cascading Style Sheets, a language for styling HTML elements.

**HTML:** HyperText Markup Language, a language for constructing the basic elements of web pages.

# **7.** **Bibliography**

A list of referenced and/or related publications.

Optimistic Locking with Version Number - Amazon Dynamodb. Amazon Web Service. (n.d.). https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBMapper.OptimisticLocking.html

*Don’t Make Me Think, Revisited* | Krug, Steve

<https://www.geeksforgeeks.org/stateful-vs-stateless-architecture/>