EECS 3311

SamePage Application - Wiki

Table of Contents

Application Details

💡 Description

SamePage is a desktop application where Users can find new and interesting books by browsing through the SamePage catalogue. Find reading buddies through book clubs, write book reviews, and create reading goals to track your success!

⚙️ Features

- **Iteration 1**
    - View the latest releases of books
    - Search for books by title
    - Log in or Sign up
- **Iteration 2 (Current Release)**
    - Allowing Users to leave book reviews
    - Viewing all the book information on the latest release page and search result page
    - Favourites section - Allowing Users to favourite books they like or want to read in the future
    - **Client Requested Feature**: User Request to Add New Book to App
        - Users can now request to add a missing book to the app if not found

📐 Future Releases

- **Iteration 3**:
    - Allowing Users to set and keep track of reading goals
    - Finding and joining Book Clubs
    - Adding friends to create BookClubs

✏️ Design and Development

Design Architecture documents found under Preliminary Research. UML Design documents are found under Class Diagrams. All important packages, files, and code pertaining to the java application are located under **SamePage/src/main/java/com/eecs3311**.

💻 Running the application

- Go to **SamePage/src/main/java/com/eecs3311**
- There will be 4 Packages and one Java File called **App.java**
- Click on the **App.java** file and run the application from here

📦 Important Packages

Under **SamePage/src/main/java/com/eecs3311**

- **/model** – contains all the model classes and dummy data for the models
- **/presenter** – contains the presenter class that link the needed model and view classes

- **/view** – contains the classes needed to develop the views for the model class and the other GUI components not related to the model class
- **/persistence** – contains the Database interface and classes to access the “real” and “stub” databases
- **/App.java** - This is where you would start the application

Iteration 1 Notes

🔬 Preliminary Research

UI Design Choice and Justifications

Frameworks/Tools:

Variables to Consider

- DB service to be used
- Desktop application
- GUI design
- Testing???
- API calling
- Implementation of SOLID principles
- IDE compatibility
- Learning curve

UI Development
Some good frameworks are:

- Java Swing
- JavaFX
- JavaAWT
    - Swing is an extension of JavaAWT, hence all AWT features are in Swing, yet it is more lightweight and has an improved performance

By far the two best choices for desktop application GUI development are Swing and JavaFX.

Java Swing vs. Java FX∫

	Java Swing	JavaFX
MySQL Connection https://www.javatpoint.com/example-to-connect-to-the-mysql-database	Can connect to DB service using core java functionality.	Can connect to DB service using core java functionality
API Calling	Can use web client like HTTPClient to call API in Java Swing http://www.mkyong.com/webservices/jax-rs/restful-java-client-with-apache-httpclient/	Can use spring web client to call REST service from JavaFX https://edencoding.com/connect-javafx-to-a-rest-api/
Testing https://github.com/SICKAG/gui-check	JUnit testing is supported with Swing, unsure about eclipse implementation, might be more difficult.	Supports JUnit testing in various IDE, possible in Eclipse https://www.eclipse.org/forums/index.php/t/1079233/

IDE Compatability	Swing is independent from Java so any IDE can be used, however, it is recommended to use NetBeans. Intellij, Eclipse, DrJava. Swing allows for better rapid deployment of an application due to its more mature IDE support.	Intellij is the most recommended for JavaFX, but it also works with Eclipse and NetBeans.
Learning Curve	Both frameworks have a similar learning curve. There are more resources to learn swing because it's been around for longer, may be easier to learn because of this.	JavaFX uses CSS while swing doesn't.

- JavaFX is younger and being developed at a faster rate, meaning in the future it will almost certainly surpass Swing in terms of usability and industry relevance
    - Meaning it might be more valuable for us to learn FX
- Swing has a wider library of UI components, FX is being updated and adding more at a faster rate
- FX may support mvc architecture better than Swing in some IDEs, unsure about eclipse
    - https://link-intersystems.com/blog/2013/07/20/the-mvc-pattern-implemented-with-java-swing/
    - https://edencoding.com/mvc-in-javafx/
- If rapid efficient development of an app with easier access to GUI elements and libraries is required, use SWING
- If developers want a modern touch and implementation of animations and effects is wanted, use FX
- If developers want better applications for desktop and mobile app, use FX

Backend Frameworks

- Spring Boot
    - Dependency Injection (DI) (Inversion of Control) – In this principle, rather than the application taking control of the flow sequentially, it gives the control to an external controller who drives the flow. The external controller is the events. When some event happens, the application flow continues. This gives flexibility to the application. In Spring, IoC is done by DI which are of three types – setter injection, method injection and constructor injection.
    - In Spring, objects are called beans and there is a BeanFactory that manages and configures these beans. You can think of the beanfactory as a container that instantiates, configures and manages the beans.
    - Focuses on business logic and takes care of infrastructure

- Hibernate
    - Object-Relational Mapping database for java applications
    - Directly maps java classes to corresponding database tables
    - Implements Java Persistence API
- Struts
    - Used for web application development

Best frameworks to consider are Spring Boot and Hibernate

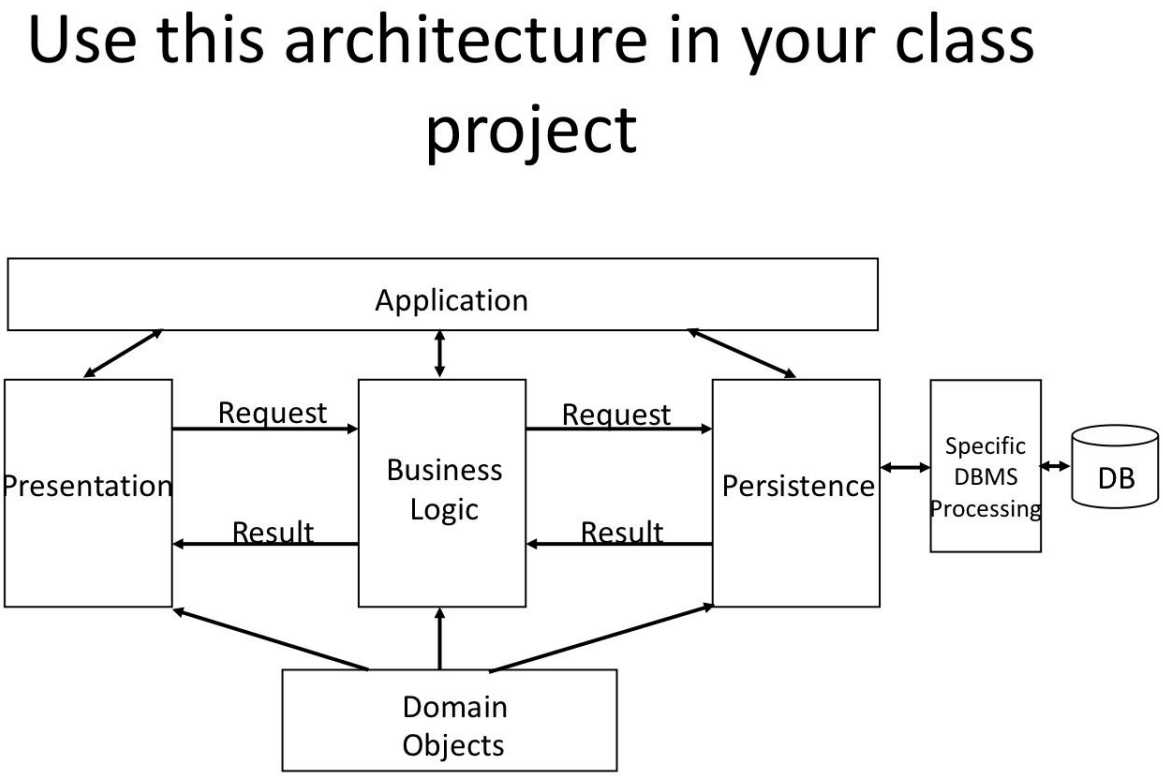
However, given time frame and feasibility, maybe in the best interest to not utilize backend framework.

- Can increase learning curve time, which we do not have a lot of
- Can increase unknowns
- May be unnecessary due to the size of the application and the features it must contain
- May cause complications when implementing with Swing/FX

Architecture Choice and Justifications

Relevant class material: Lecture week 3-1, mentions the benefit of architecture(presentation layer) to keep GUI separate from business processing resulting:

- Easier to substitute different GUI without having to modify any code outside of GUI
- Easy to apply patterns like MVC



Design Patterns

High-level comparison: **Model-View-Controller = MVC**
Model – responsible for business logic, state of the application. Includes reading and writing data, persisting application state and tasks related to data management.
View – presenting data to the user and handling user interaction
Controller – the view layer and model layer are glued together by one or more controllers

Model-View-Presenter = MVP
Model – represents a set of classes that describe business logic and data. Also defines business rules for data and how it can be manipulated.
View – used for making interactions with users like XML, activity, and fragments
Presenter – gets input from View, processes the data with help of the model and passes results back to the View after processing is done.

MVP in Swing examples:

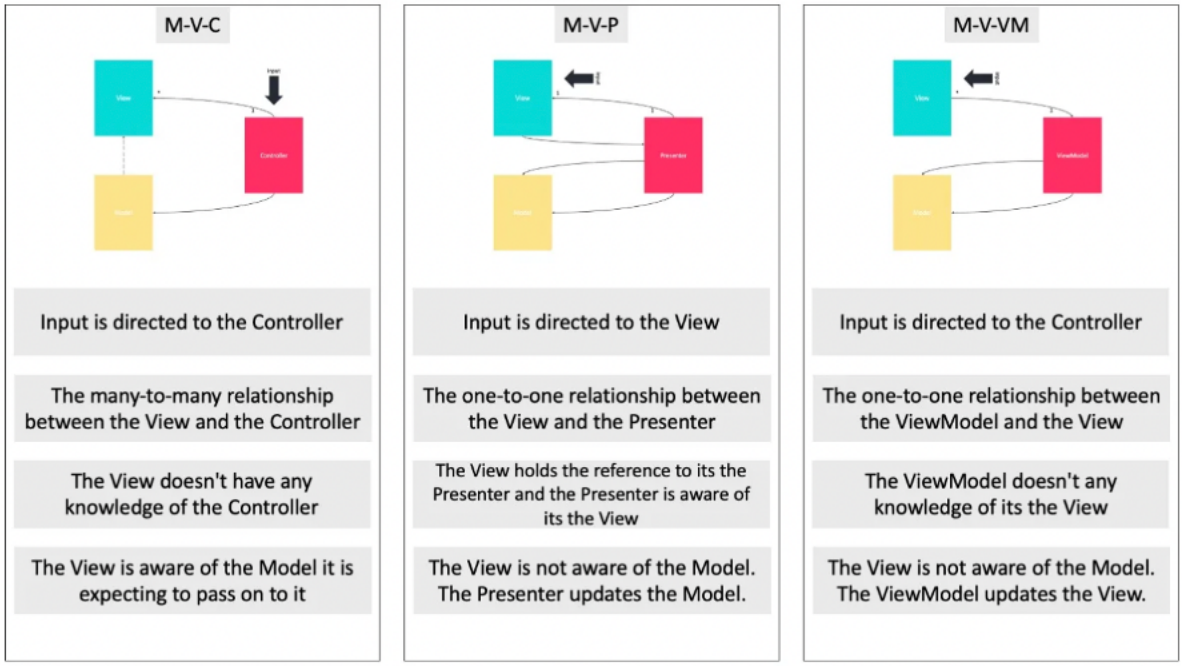
- https://sites.google.com/site/averagelosercom/Java/java-model-view-presenter-in-swing?pli=1
- https://riptutorial.com/swing/example/14137/simple-mvp-example

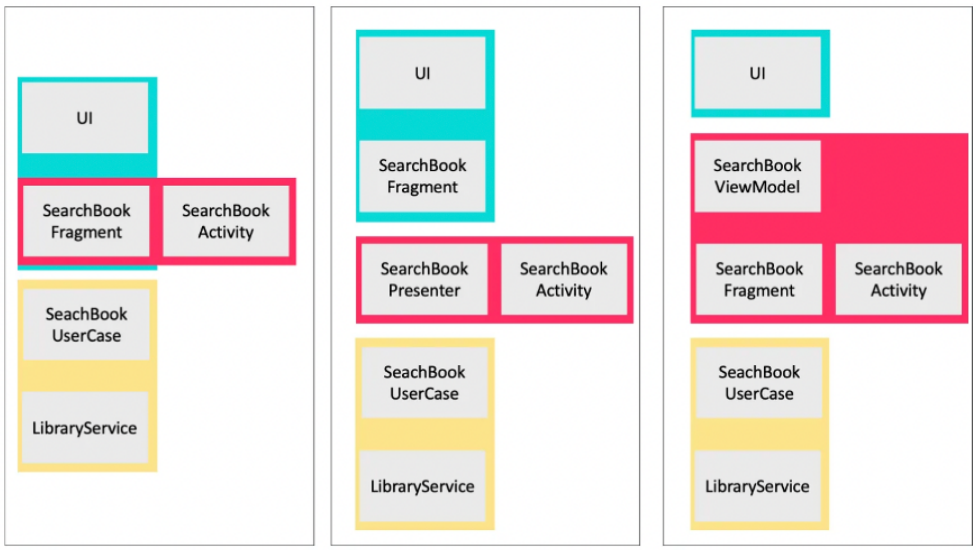
Model-View-ViewModel = MVVM

Model – similar to MVC, consisting of basic data required to run software

View – is a graphical interface between user and design pattern, similar to the one in MVC. Displays output of data processed

View-Model – one side abstraction of View and on the other side provides wrapper of Model data to be linked. Model converted to View and also contains commands that the View can use to influence Mode





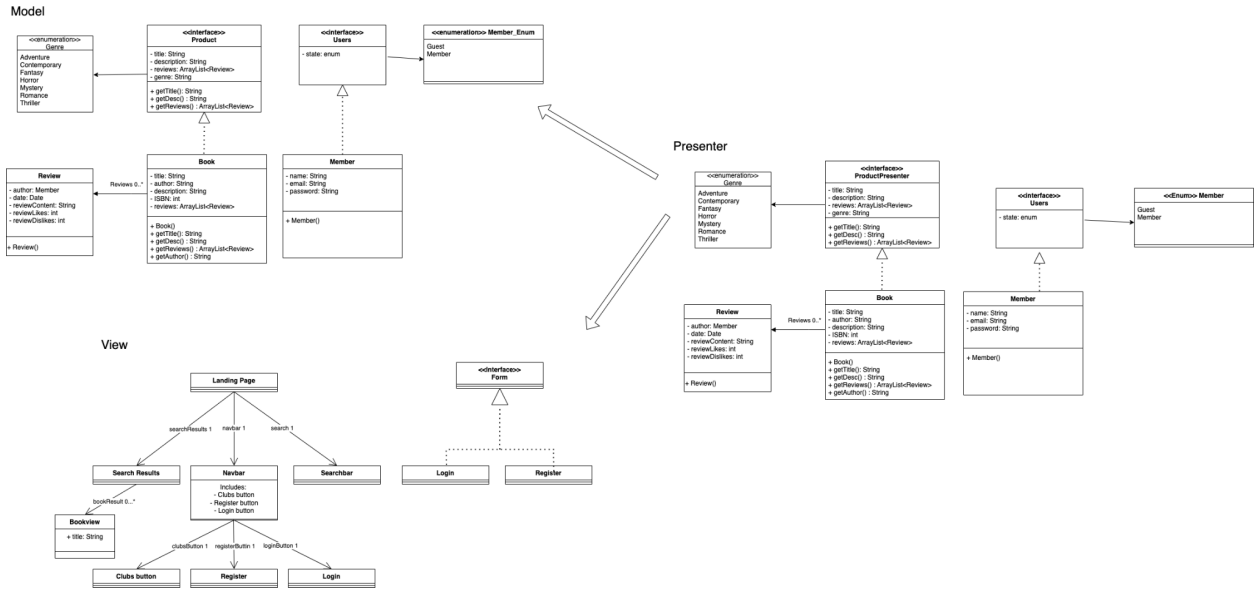
MVC vs MVP vs MVVM implementation for the BookSearch application

Performance Evaluation – UI performance, **MVP** is highly reliable. Data binding in MVVM creates additional overload.
Modifiability – less changes in MVP and MVVM (with MVVM contributing a lot towards maintainability)

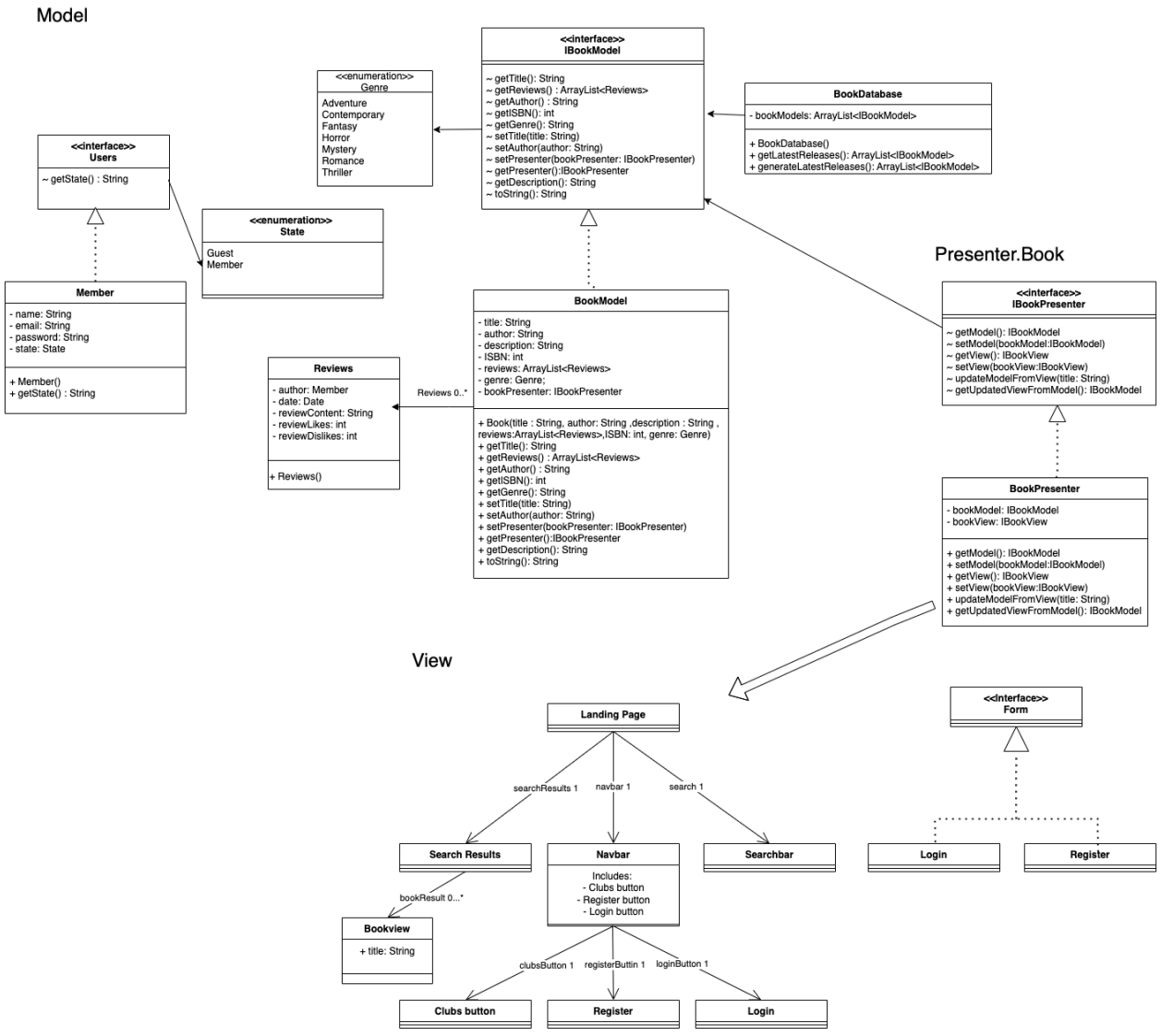
	Pros	Cons
MVP	- has a better separation of concerns - Presenter is not as **tightly coupled** as Controller, but has more responsibilities – updating model and updating view - Presenter and Model **follow Single Responsibility Principle** better since Model is not in charge of updating Presenter - Easier to unit test b/c of low coupling	- One presenter class manages one view at a time - Can lead to double the amount of classes for views
MVC	- One controller can select different views based upon required operations where as	- Less separation of concerns - Compared to how Presenter class in MVP, the same is

	in MVP, there is one presenter for evey model	done in the model resulting in lower cohesion - Controller and View are in related fragments - higher coupling - Model disrupts the Single Responsibility Principle in MVC as it is also in charge of updating controller - Inputs handled by controller that instructs mode for **further** operations
MVVM	- Loose coupling with the View - UI can be changed without modifying the ViewModel as long as the contract does not change - Better Testability: Since the ViewModel does not “see” the presentation layer or controller layer, ViewModel can be unit tested without UI elements	- Debugging can be difficult for complex data binding - Hard to design ViewModel for larger applications - Can be overkill for simple UIs - Generalizing the viewModel upfront can be difficult and large-scale data binding can lead to low performance

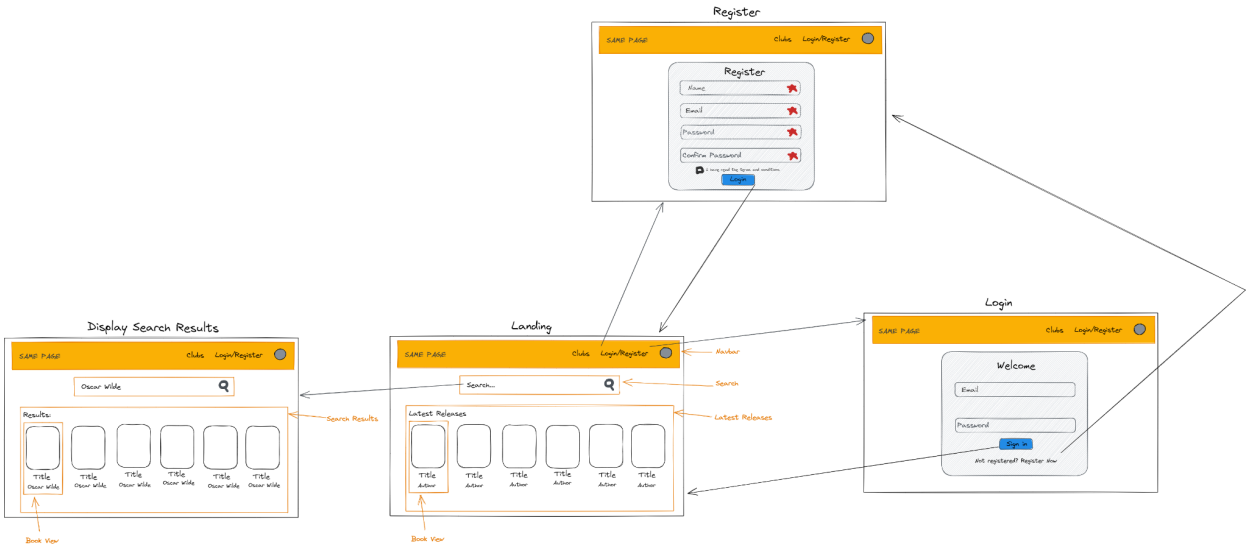
Initial Class Diagram/Sketch



Updated Class Diagram



Design Wireframe



Iteration 1 - Big User Stories

Goal

Create MVP of project, which includes landing page, search functionality and registration/login system

Big User Stories

Landing Page

As a guest or member, I want to be able to open Same Page and view a number of books on the homepage.

Priority: High **Cost**: 6 days

Search for Book

As a guest or member, I want to be able to search for a specific book.

Priority: High **Cost**: 6 days

Register/Login

As a guest or member, I want to have the ability to register a new account with SamePage or login to an existing account.

Priority: Medium **Cost**: 5 days

📙 Iteration 1 Detailed Planned Stories

Display Login/Register Buttons

Display the login and register buttons and their corresponding pop-up input boxes on the landing page.

Priority: High **Big User Story**: Landing **Cost**: 3 days

Display Latest Releases

Display the most recent book releases of any genre on the landing page.

Question: What external source can provide this information?

Priority: Medium **Big User Story**: Landing **Cost**: 3 days

Search Bar

Display a search bar where text can be entered or filters can be applied to search for a book.

Priority: High **Big User Story**: Search **Cost**: 4 days

Display Search Results

Display the results after searching for a book in the search bar.

Priority: High **Big User Story**: Search **Cost**: 2 days

Register New User

Add functionality to register an account with an email, username, and password.

Priority: Medium **Big User Story**: Login/Register **Cost**: 2 days

Login Existing User

Add functionality to securely log into Same Page with an existing email and password.

Priority: Medium **Big User Story**: Login/Register **Cost**: 3 days

Login Page

Create a separate login page with email and password input fields.

Priority: High **Big User Story**: Landing **Cost**: 1 days

Register Page

Create a separate register page with name, email, password, and confirm password input fields.

Priority: High **Big User Story**: Landing **Cost**: 1 days

Display Navbar

Create navbar component with login, register, clubs, and a profile icon.

Priority: High **Big User Story**: Landing **Cost**: 1 days

Iteration 2 Release Notes

🧰 Bug Fixes

- Login and Register crash on Exit
    - The app would crash when the exit button is closed on the Login or Register page and the Login and Register popups could be opened up multiple times. The app has implemented designated pages for Login and Register. There are no more pop-ups. The app can exit normally

🚀 Iteration 2 - What's New?

- Book Information Display
    - Users can click a book from the Search Results or Latest Releases page and see the relevant book information like the author, genre, preface, and user reviews
- Book Reviews
    - Users can submit multiple reviews on any book
- Favourite Books
    - Users can favourite books they like or want to read in the future
    - Dedicated Favourites page in the Profile section to view all your favourite books
- Client Feature – Add New Books
    - Users can request to add a new book to the application if they cannot find it through the search

🚧 Changes/Updates to Iteration 1 Stories

- Book Search Functionality
    - Previously, Users could only type the book name in the search bar but the result(s) would not appear
    - Now, Users can use the search bar appropriately to find books in the current database
- Login and Register
    - Previously, Users could not register to the app and could only use one login credential to use the app and its features
    - Now, Users can register and log in to their SamePage accounts. All actions done by the User are saved into the database

🛠️ Major Changes from Iteration 1

Code Structure

- Reorganized Structure
    - The View Layer in Iteration 1 was hard to follow. There was a general idea of how the components would be structured and interact with each other but were not planned out thoroughly. Three majors problems that had to be solved were:
        - **Problem**: Components that could be reused weren't being reused or were hard-coded into the application violating
        - **Solution**: Added a components package to store reusable components and added a layout package to keep pages that the User would visit. Enforced Single Responsibility Principle to decrease coupling and increase cohesion between components and pages
        - **Problem**: Some pages were JFrame objects while others were JPanel objects. This would allow users to spam pop-ups and increase memory usage for the computer. Also, closing the pop-ups would shut down the entire app
        - **Solution**: Converted all view-related items to JPanels to keep consistency between components. Used interfaces to enforce common functions onto components. Implemented a Main class that extended the JFrame class which would hold the logic to display the current page the User is on. The Main class implemented a CardLayout to allow navigation between pages. All views stayed as JPanels
        - **Problem**: Pages used all kinds of functions – no structure between pages
        - **Solutions**: Added a common interface for all JPanel views to implement. This would allow consistency between components and pages
    - Config Files
        - **src/main/resources/config.properties**: This is a configuration file to create a connection to the database easily. Developers only update their passwords to connect. Used for development only and does not affect the SamePage application
- New Design Implementations
    - Mediator Pattern
        - **view/components/ResultsMediator.java**: Used the mediator pattern to communicate with the *view/components/SearchBar.java* and *view/components/ResultsPanel.java* components. Achieved a reduction in communication complexity between the components, higher cohesion, and lowered coupling, leading to an increase in Single Responsibility
    - Singleton Pattern:
        - **model/User.java**: The User class has a single instance and its credentials are updated by the *model/Login/LoginModel.java* class. The User class provides global access to all other classes
        - **persistence/Database.java**: The Database uses Singleton and can be globally accessed through all the classes. It is an extension of the *persistence/AbstractDatabase.java* class that initializes the connections for the different database accesses.

        - **view/Book/DisplayBookInformation.java**: It is an extension of the JFrame classes. The DisplayBookInformation class appears when a User clicks on a book to view its information. The DisplayBookInformation is a popup and uses the singleton pattern to only appear once to avoid spamming pop-ups. It can be accessed globally but is currently only used in *view/Book/BookView.java*
    - MVP (Model, View, Presenter)
        - Classes that use the MVP pattern can be found under view, presenter, and model packages and have their own packages to contain their classes. The MVP pattern was implemented for one class (Book) in iteration 1 and now the pattern has expanded to Login, Register, and Review. The reasoning for using this pattern was explained in iteration 1 and can be found under Iteration 1 Notes / Preliminary Research

New App Behaviour

- Functional Register, Login, and Search
    - Users can now register, log in, and search through the library catalogue
- Profile page
    - Users have a dedicated profile page to view their favourite books
- Add New Book – Client Feature
    - Users can request to add a new book to the app if the book is not visible in the app
- View Book Information
    - Users can see the average ratings of the book on the mini-display and can view more information about the book when expanding the mini-view
        - Total number of ratings and the average rating
        - View user reviews and ratings of the book
        - Book author and genre
    - The mini-display has a “Favourite” button that allows users to add the book to their favourites list
- Submit Reviews and Ratings
    - Users can submit a review and rating for multiple books and can also see the rating get updated dynamically with their input
    - Can view other user reviews and ratings and the day it was posted
- Favourite Books
    - Users can add books they are interested in to their favourites by clicking the “Favourite” button on the mini-display
    - The app dynamically updates the UI so that when a User has added the book to their favourites, the book’s button is then replaced with a “Remove from Favourites” button to allow the User to remove the book from their list
    - The User profile page is dynamically updated to include their current favourite books. The User has the option to remove the book from their favourite on the profile page which gets updated dynamically

- Redirect to the Landing Page on successful login
    - On successful login, Users are redirected to the Landing Page to indicate that the login was a success. Previously, a message would appear telling the user they had successfully logged in. That message is still present

Updated Class Diagram for Iteration 2

The updated class diagram can be found in the *Iteration 2* package called *Class Diagram*. It was too big to fit in this document.

📚 Iteration 2 User Stories

Goal

The goal of this iteration is to create more “interactivity” between the books themselves and the guests/members, for example, allow both guests and members to know more about the book, be able to favourite books and leave reviews

Big User Stories

User Reviews

As a member, I want to upload/leave a review of the book I have read.

Priority: High **Cost**: 5 days

Book Information

As a guest or member, when I click on the book, I should be able to view the main information of the book.

Priority: High **Cost**: 5 days

Favorites

As a member, I want to favorite books that I like or want to read later in the future.

Priority: High **Cost**: 5 days

Client Feature - Add New Book to App

As a user I can request to add the book to the App if I do not see it on SamePage.

Priority: High **Cost**: 5 days

Iteration 2 Detailed User Stories:

View Reviews

Display the book reviews upon clicking on a specific book.

Priority: High **Big User Story**: User Review **Cost**: 3 days

Add Reviews

Users logged in, and add reviews to the selected book upon clicking on a specific book.

Priority: High **Big User Story**: User Review **Cost**: 3 days

Display Favorite Books

Display the favourited user books in user information.

Priority: High **Big User Story**: Favorites **Cost**: 3 days

Add Favorite Books

Add a book to the user's favourites upon the user selecting a specific book from the search.

Priority: High **Big User Story**: Favorites **Cost**: 3 days

Remove Favorite Books

Remove a book from user favourites upon the user selecting a specific book from their user information list.

Priority: High **Big User Story**: Favorites **Cost**: 3 days

Display Book Information

Display book information such as title, author, publication date, summary, reviews, and ISBN number.

Priority: High **Big User Story**: Book Information **Cost**: 3 days

Client Feature – Request to Add New Book

If a book is not listed, the user can submit a request/suggest the app to add the new book to a queue.

Priority: High **Big User Story**: Add New Book **Cost**: 3 days