EECS 3311

SamePage Application Iteration 1 – Wiki

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 bookclubs, write book reviews, and create reading goals to track for your success!

Features (Iteration #1):

- View the latest releases of books
- Search for books by title
- Login or Sign up

Future Releases:

- **Iteration #2**:
  - Allowing Users to leave book reviews
  - Viewing the 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
- **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 Desgin documents found under Class Diagrams. All important packages, files, and code pertaining to the java application are located under **SamePage/src/main/java/com/eecs3311**.

Run the application:

- Go to **SamePage/src/main/java/com/eecs3311**
- There will be three 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
- **/App.java** - This is where you would start the application

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/forum

		s/index.php/t/1079233/

- 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 as 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 infrastrucure
- 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

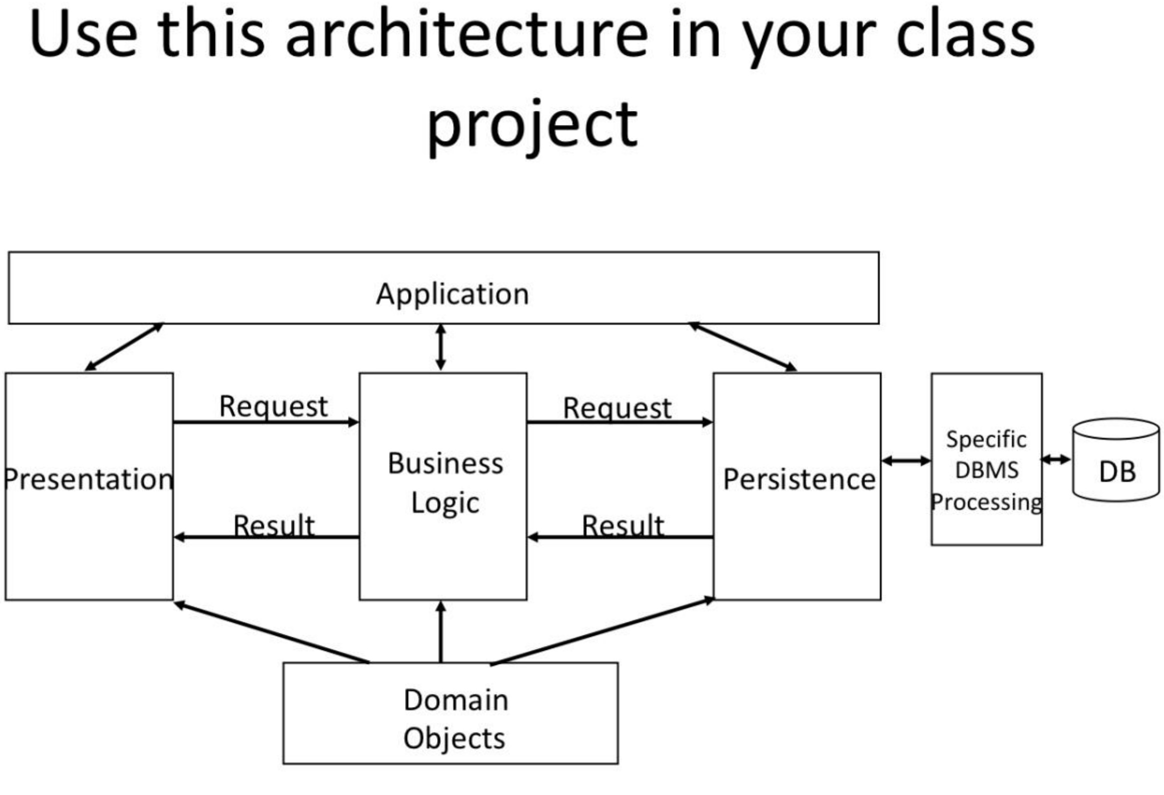
However, given time frame and feasibility, may be in best interest to not utilize backend framework.

- 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 whne implementing with Swing/FX

Architecture Choice and Justifications:

Relevant class material: Lecture week 3-1, mentions 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



Use this architecture in your class project

Design Patterns

High level comparison: **Model-View-Controller = MVC**

Model – responsible for business logic, state of application. Includes reading and writing data, persisting application state and tasks related to data management.
View – presenting data to 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 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, fragments

Presenter – gets input from View, processes the data with help of model and passes results back to the View after processing is done.

MVP in Swing examples:

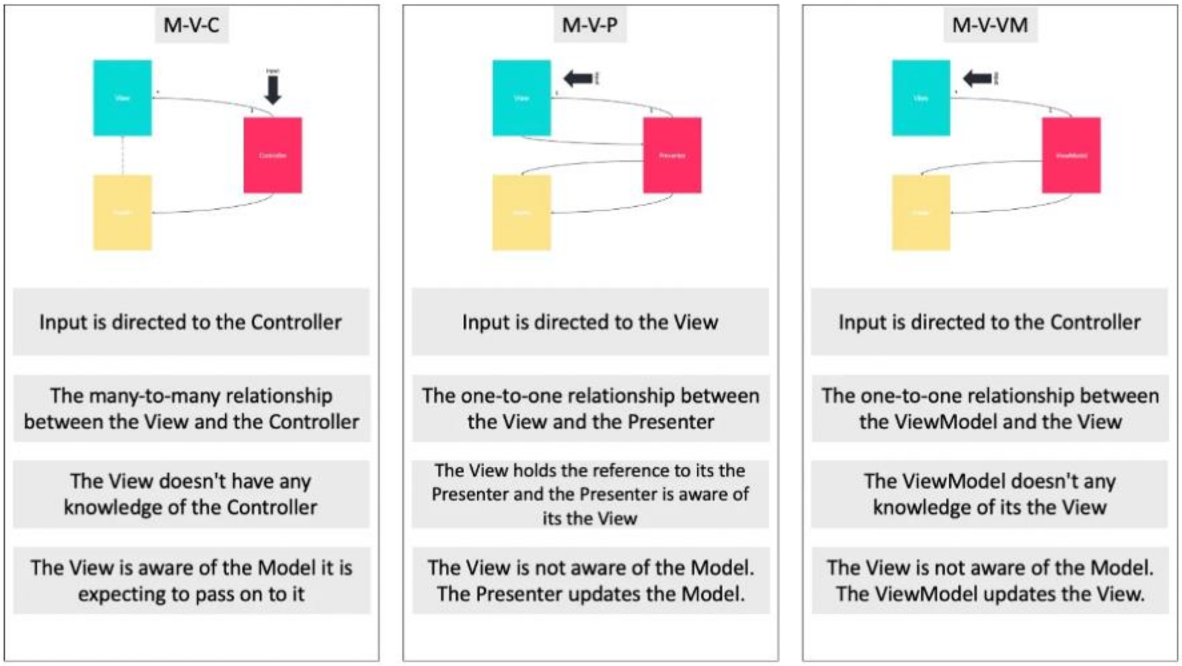
- https://sites.google.com/site/averagelosercom/Java/java-model-view-presenter-in-swing?pli=1
- https://riptutorial.com/swing/example/14137/simple-mvp-example

Model-View-ViewModel = MVVM

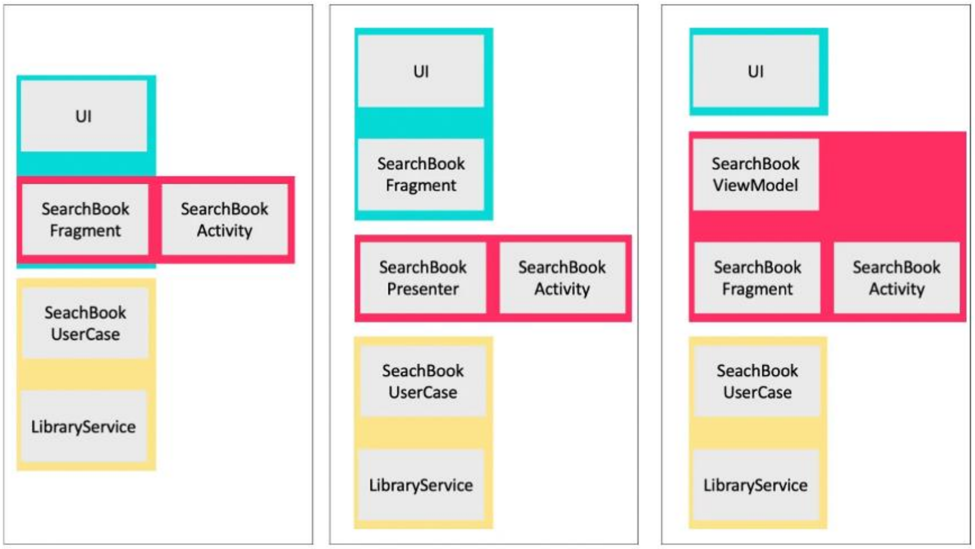
Model – similar to MVC, consisting of basic data required to run software

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

View-Model – one side abstraction of View and on 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



M-V-C	M-V-P	M-V-VM
Input is directed to the Controller	Input is directed to the View	Input is directed to the Controller
The many-to-many relationship between the View and the Controller	The one-to-one relationship between the View and the Presenter	The one-to-one relationship between the ViewModel and the View
The View doesn't have any knowledge of the Controller	The View holds the reference to its the Presenter and the Presenter is aware of its the View	The ViewModel doesn't any knowledge of its the View
The View is aware of the Model it is expecting to pass on to it	The View is not aware of the Model. The Presenter updates the Model.	The View is not aware of the Model. The ViewModel updates the View.



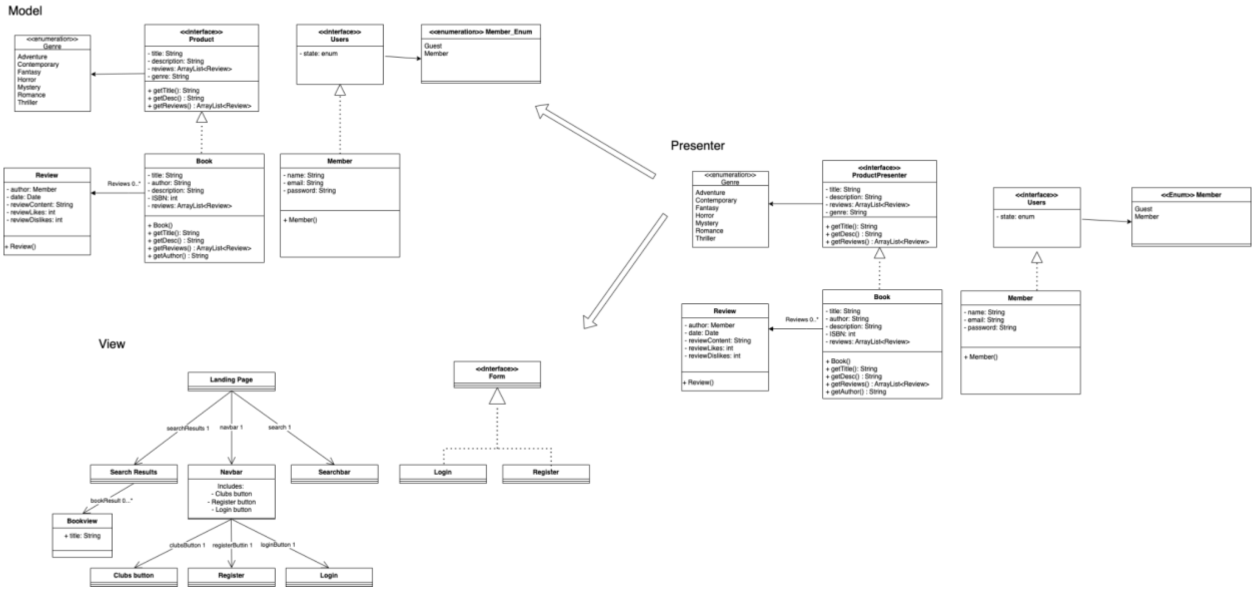
MVC vs MVP vs MVVM implementation for the BookSearch application

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)

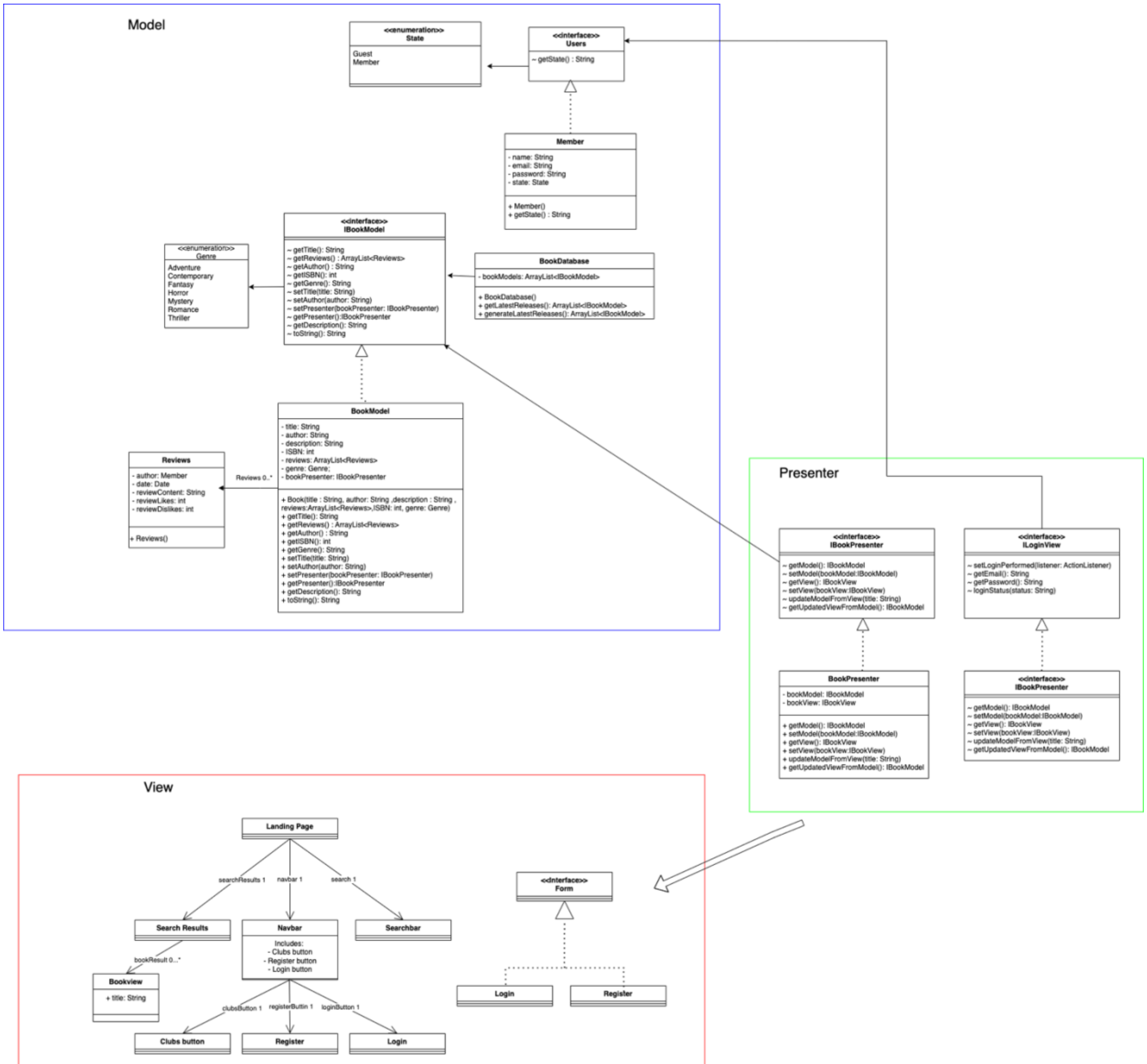
	Pros	Cons
MVP	- has a better separation of concerns - Presenter is not as **tightly coupled** as Controller, but has more responsibilities – updating model and updating view - Presenter and Model **follow Single Responsibility Principle** better since Model is not in charge of updating Presenter - Easier 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 in MVP, there is one presenter for evey model	- Less separation of concerns - Compared to how Presenter class in MVP, the same is 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

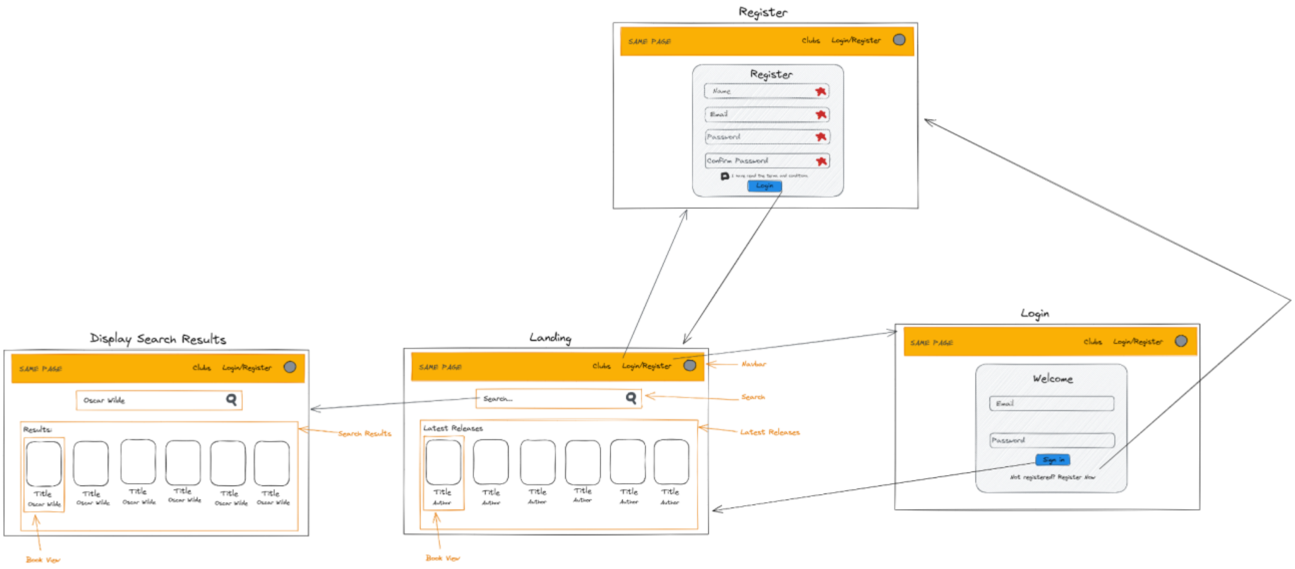
Initial Class Diagram/Sketch:



Updated Class Diagram: After understanding MVP pattern with SOLID design principles, diagrams were adjusted accordingly



Design Wireframe:



Iteration 1 - Big User Stories

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

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