COM4502

Library Management App

by

Mehmet Subaşı and Yusuf Özcan

PROJECT DEFINITION

A library management system is a system that ensures the orderly preservation, organization and management of the large number of books owned by libraries; in other words, it is a system that facilitates the daily operation of libraries. Using this system, books can be found, reserved, borrowed or returned instantly. It also provides information about how many books are available, how many are reserved, how many are borrowed, or how much the overdue penalty will be if the delivery of the book is delayed. In order to keep track of all these functions, all that is needed is for the library administrators to keep a database of the books in process with the delivery dates. This helps to provide better service for the user.

Our application provides all the necessary functionalities for a library management system. The simplicity and ease of usage are the priorities of the application. The application is designed for library administrators to keep track of the books. Since it is a mobile application, they are not obliged to computers to see available books, overdue books and such just by carrying the mobile phone they have access to all the data and can make needed changes.

We used Kotlin programming language for developing the application, Java for developing the backend and PostgreSQL for storing the data.

CONTENTS AND WORKING METHODS OF THE APPLICATION

1-) COMPONENTS

- <u>BOOK ITEM:</u> We used the Compose function to represent a book item (BookItem) whose function is to visually display the title, author, publisher, publishing house, year of publication and borrowing status of a particular book. We added styling features like shading and corner rounding by putting the book element inside a surface component. We used the row component to organize the book information. We also used an icon (Icon) to indicate whether the book has been borrowed or not. this icon will appear in red (borrowed) or green (available) depending on the 'isBorrowed' status.
- <u>USER ITEM:</u> We used the Compose function, which represents a user element (UserItem) whose function is to visually display a user's first name, last name and debt (if any). If the

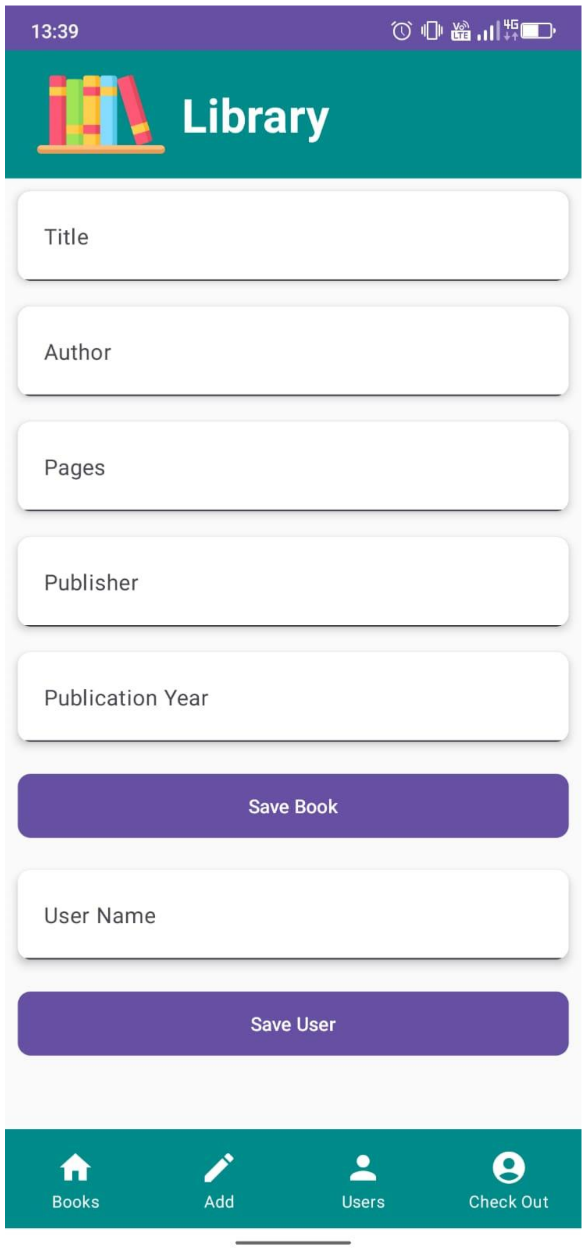
user has a debt, the icon will be colored red, if not, it will be colored green. We also tried to provide a visually pleasing user interface with design and style options.

- BOTTOM NAVIGATION BAR: We created a bottom navigation bar that allows users to easily navigate to different screens (book list, book add, user list and borrowed books). for each button we put four separate buttons (Books, Add, Users and Check Out) in a column layout.
- TOP NAVIGATION BAR: We have created a top navigation bar at the top of our app with a title (Library) and a logo.

2-) SCREEN

- ADDITION SCREEN: This screen presents a form to retrieve book information (title, author, number of pages, publisher and publication year) from the user. We achieved this by using TextField components. We also used Toast messages to provide feedback to the user while entering information. Finally, we added a button to save the book. We applied the same idea for adding users.

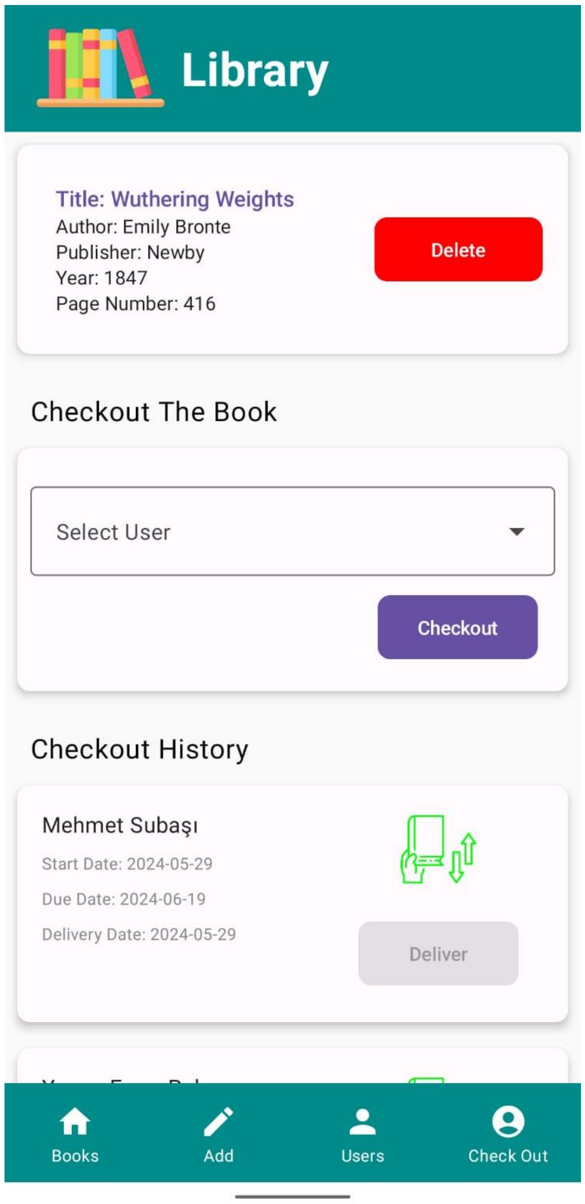
  We combined these two screens on one page. You can find the relevant image below. There is a section at the top of the page to add books. Information such as book name, author, page, publishing house and year of publication is requested. At the bottom there is a section to add users.



- LIBRARY NAVIGATION GRAPH We created the navigation graph of the application using the NavHost component. This determines the paths (routes) corresponding to the different screens (composable). It then calls the Composable function corresponding to each screen and passes the corresponding parameters.
- ROUTE ENUMERATION: We created an enum class called RouteEnum that determines the routes of the screens. This consists of elements containing the name (value) of different screens paired with a unique value. For example, the value of the BOOK_LIST_SCREEN element is defined as “BOOK_LIST_SCREEN”.

2.1-) BOOK SCREEN

- BOOK DETAILED SCREEN: First, we imported the necessary Jetpack Compose and navigation components. We defined a Composable function called BookDetailedScreen that takes a NavHostController parameter. This function creates the interface of the book detail screen. We used a surface for this and showed the screen title with the Text component. The visual for this section is as follows.
  As seen in the image below, old information about the book, movements in the book, which user purchased the book, and when they delivered it are shown in detail. If the book icon is green, it means that the user has read the book and returned it to the library. If the icon is red, it means that it has not been delivered yet.


Library

Title: Wuthering Weights
Author: Emily Bronte
Publisher: Newby
Year: 1847
Page Number: 416

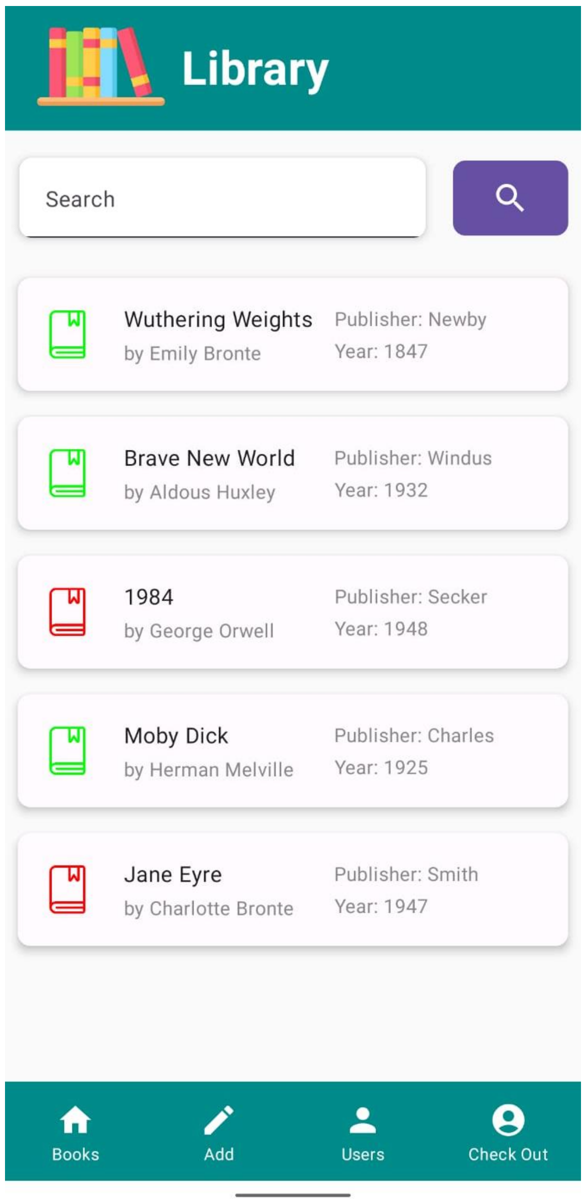
Checkout The Book

Checkout History

Mehmet Subaşı
Start Date: 2024-05-29
Due Date: 2024-06-19
Delivery Date: 2024-05-29

- <u>BOOK LIST SCREEN:</u> First, we imported the necessary Jetpack Compose and navigation components. Then we define a Composable function called BookListScreen. This function takes a NavHostController and optionally a MainActivity instance. This example is used to subscribe to the ViewModel and update the data. Also in the surface, there is the top bar (TopBar), refresh button and book list (LazyColumn).

  You can see the book list in the image below. General book information is included. If the book icon is green, it means that book is available in the library. But if the icon is red, it means that the book is not available in the library (it has not been checked in by the user yet).



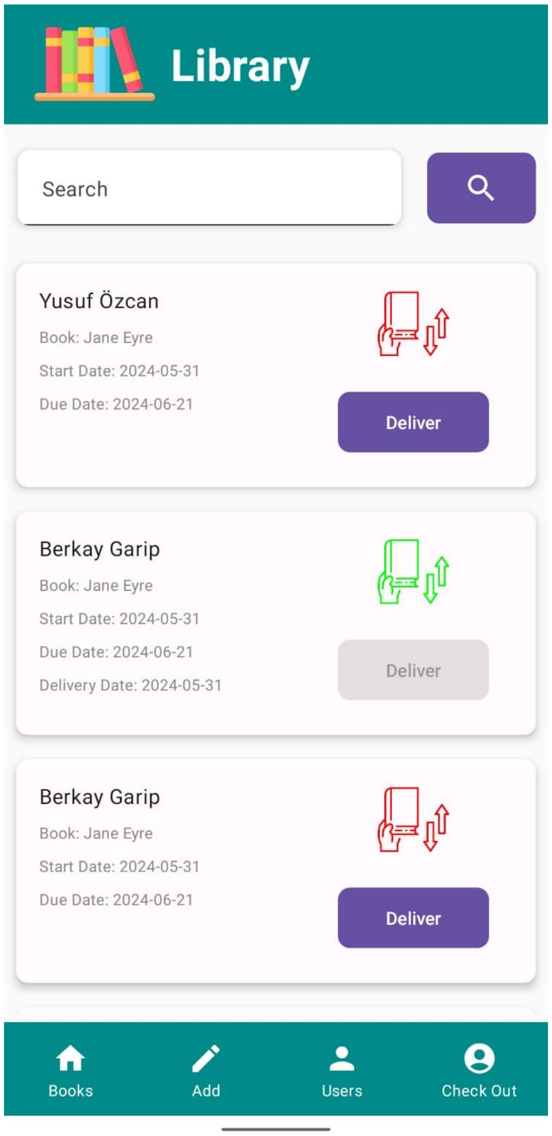
Library

Search

🟩	Wuthering Weights by Emily Bronte	Publisher: Newby Year: 1847
🟩	Brave New World by Aldous Huxley	Publisher: Windus Year: 1932
🟥	1984 by George Orwell	Publisher: Secker Year: 1948
🟩	Moby Dick by Herman Melville	Publisher: Charles Year: 1925
🟥	Jane Eyre by Charlotte Bronte	Publisher: Smith Year: 1947

2.2-) CHECK OUT SCREEN:

- <u>CHECK OUT LIST SCREEN i:</u> First, we imported the necessary Jetpack Compose and navigation components. We then defined a component function called CheckOutListScreen. This function takes a NavHostContoller parameter and interfaces to the borrow list screen using Jetpack Compose. For this we used a Surface and created a column in it. We added the top bar (TopBar), text, spacer and bottom bar (BottomNavBar) inside the column. We also added a function called CheckOutListScreenPreview to preview CheckOutListScreen's interface.
  Historical data is available on the screen, as shown in the image below. The dates of which user bought which book, whether they delivered it or not, are given and transferred to the interface.



Yusuf Özcan
Book: Jane Eyre
Start Date: 2024-05-31
Due Date: 2024-06-21

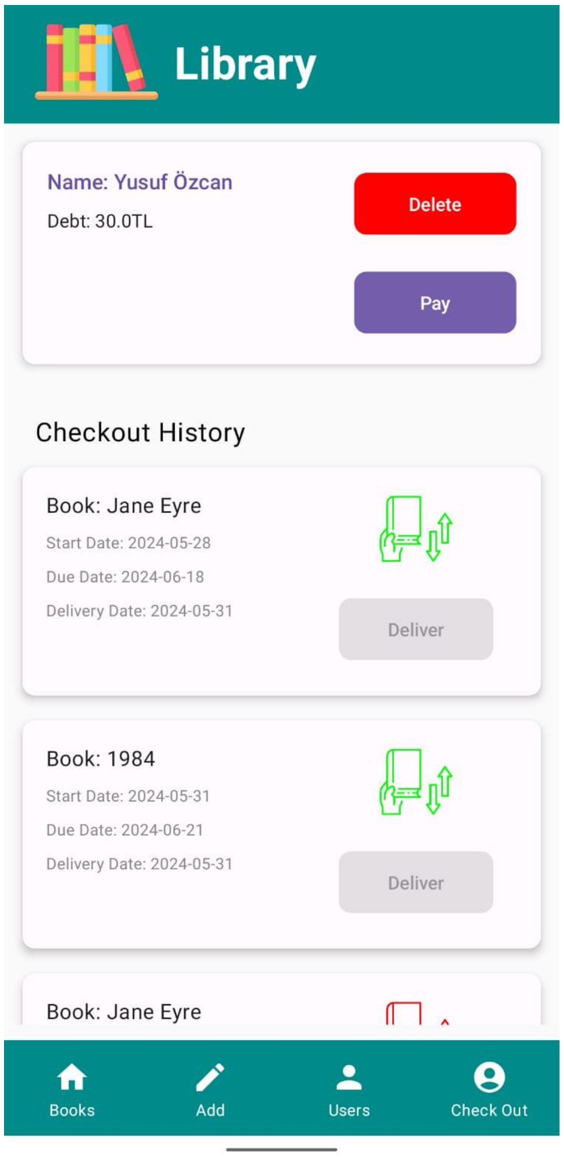
Berkay Garip
Book: Jane Eyre
Start Date: 2024-05-31
Due Date: 2024-06-21
Delivery Date: 2024-05-31

Berkay Garip
Book: Jane Eyre
Start Date: 2024-05-31
Due Date: 2024-06-21

2.3-) USER SCREEN:

- USER DETAILED SCREEN: We designed this screen to show a user's details.

  UserDetailedScreen is a Composable function that displays user details. We determined the background color and size with Surface components. We determined what information would appear on the screen with the text component. Finally, we adjusted the size and position of the components with Modifier. You can see how it looks on the screen in the image below. From the screen below, we can understand in detail which books each user has purchased, whether they have delivered them or not, and finally whether they have any debts. We can also pay the debt from this page and delete the user if we wish.



Name: Yusuf Özcan
Debt: 30.0TL

Checkout History

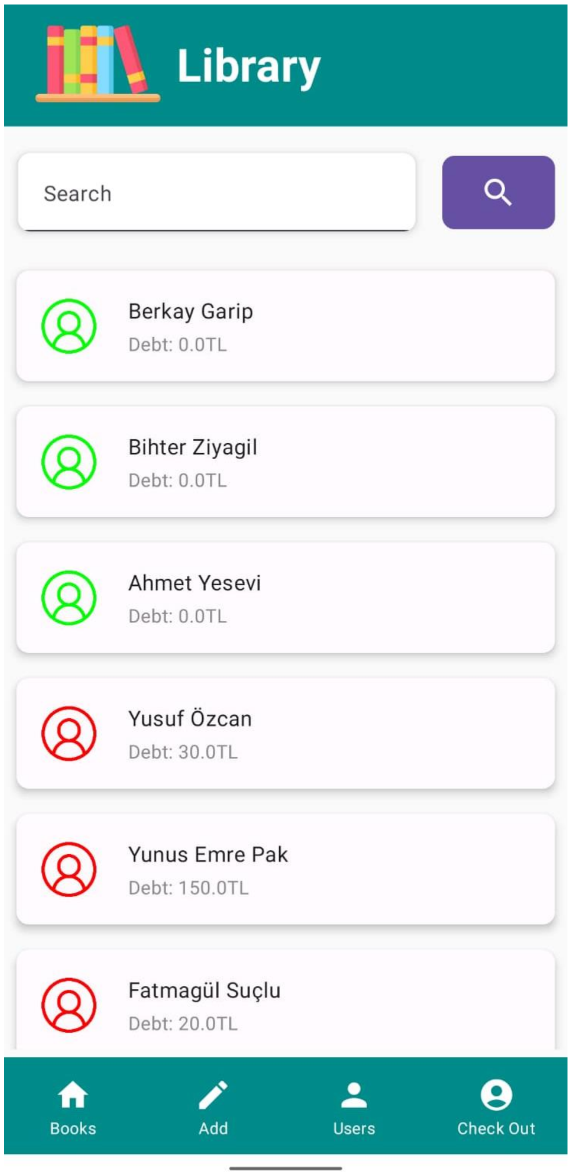
Book: Jane Eyre
Start Date: 2024-05-28
Due Date: 2024-06-18
Delivery Date: 2024-05-31

Book: 1984
Start Date: 2024-05-31
Due Date: 2024-06-21
Delivery Date: 2024-05-31

Book: Jane Eyre

- USER LIST SCREEN: This screen provides redirection to other screens to list users and show user details.

  UserListScreen is a Composable function that displays the list of users. We determined the background color and size with Surface components. We determined what information would appear on the screen with the text component. Finally, we adjusted the size and position of the components with Modifier. You can see how it looks on the screen in the image below. If the user icon is green, that user has no debt. If the icon is red, it means that user has a debt. There is also a filtering section at the top of the screen to search for users.



Berkay Garip
Debt: 0.0TL

Bihter Ziyagil
Debt: 0.0TL

Ahmet Yesevi
Debt: 0.0TL

Yusuf Özcan
Debt: 30.0TL

Yunus Emre Pak
Debt: 150.0TL

Fatmagül Suçlu
Debt: 20.0TL

3-) VIEW MODEL

- BOOK VIEW MODEL: We have created a class called BookViewModel which is used to manage book data in our application. This class acts as a bridge between the user interface and the data source. The getBookData() function is used to fetch the book list from the API. When the API request is initiated, the _isLoading value is set to true and the _isError value is set to false. If the request is successful, the retrieved data is posted to _bookData. If it fails or an error occurs, the onError function is called. The saveBook(book: Book) function is used to register a new book to the API. When the request is initiated, the _isLoading value is set to true and the _isError value is set to false. If the request is successful, the saved book is posted to _savedBook. If it fails or an error occurs, the onError function is called. The onError(inputMessage: String?) function is called when an error occurs and updates the error message. The _isError value is set to true and the _isLoading value is set to false. This class provides data management and error handling in accordance with the MVVM (Model-View-ViewModel) architecture, making the user interface cleaner and more manageable.

- USER VIEW MODEL and CHECKOUT VIEW MODEL are also functioning in the same way as written above.

CONCLUSION

Our app makes it easy for library managers to handle transactions with customers quickly and smoothly. This helps improve the user experience and cuts down on the work for library staff. It automates routine tasks and offers simple tools, letting managers spend more time picking good books.

For customers, fast transactions mean less waiting and more reading. The system makes checkouts, returns, and reservations easy, encouraging more visits to the library. Plus, it lets customers quickly see their borrowing history and current loans, so they can keep track of what they're reading.

Project Files: https://drive.google.com/file/d/10Kwpq3r7yqlyyW0-6IDwuezO1Z-iN7lr/view?usp=sharing