No.	Student Name	Roles
1	Ahmed Adel Mohamed	Notifications
2	Osman Saad Osman	User Registration, Login & Password Management
3	Asmaa Hamdy Mahmoud	UI, Navigation, Privacy Policy screen, Configuration changes and contact us feature
4	Farah Mohamed Ahmed	Library, Profile and edit profile screens and comment section in Details screen
5	Yasmine Ali Abdelrahman	Home, Details, Search and Search Results screens
6	Ahmed Adel Elshahat	User profile screen, comment section and presentation

BOOKED

BOOKED is a library app that integerates the website “dbooks” with an android app to facilitate the user’s reading experience, it encompasses features like favouriting a book, downloading a book, reading a book, registering an account, adding comments, searching for books, managing your account and more. BOOKED also has user-friendly features like dark mode, arabic mode and visually appealing effects.

Components:

- **Registration Screen**
- **Login Screen**
- **Forgot Password Screen**
- **Home Screen**
- **Library Screen**
  .1. Faourites Tab
  .2. Downloads Tab
- **Search Screen**
  .1. Search Results Screen
- **Profile Screen**
- **Drawer**

.1. Dark mode
.2. Change Language to Arabic
.3. Privacy Policy Screen
.4. Contact us feature
.5. Logout

User Registration & Login: Sign Up or Login with Username, Email, and Password

When a user first enters the app, they’re prompted to register a new account or login if they already have an account, these screens have important thoughts goint into them, and they include:

- **Form Validation with Regular Expressions**

Email and password fields are validated using regular expressions to ensure strong security and proper input formatting.

Email Validation: Ensures the email is in a valid format

Password Validation: Enforces minimum password requirements

Firebase Authentication: Secure registration and login using Firebase to store user credentials.

- **Login & Password Management**

Secure Login with Firebase: Users log in with email and password, securely authenticated through Firebase.

Password Reset via Email: Users can reset their password with Firebase's recovery system by receiving a reset link via email.

- **State Management with MVI and MVVM Patterns**

MVVM (Model-View-ViewModel): The app uses MVVM for state handling in the UI layer, where the ViewModel manages data and interacts with Firebase to provide smooth data flow and UI updates.

MVI (Model-View-Intent): MVI is used to manage user intents (actions like sign up, login, and password reset), ensuring a reactive and predictable flow between user interactions and UI changes. This helps with handling events like login errors, registration success, etc.

State Handling: Real-time state management to handle UI transitions like loading states, success, or failure messages for sign-up, login, and password reset.

Home Screen

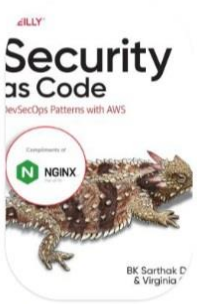
B O O K E D

Greetings!

Hop right back in

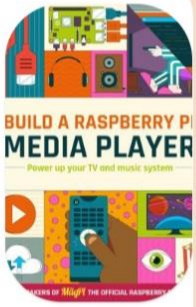
Learn Python the right way

Browse recently added books



Security as Code

Financing Investment in Times of Hig...



Build a Raspberry Pi Media Player

When a user first enters the app while logged in, the first screen to show is the home screen, it includes three main functions:

- Last Read Book

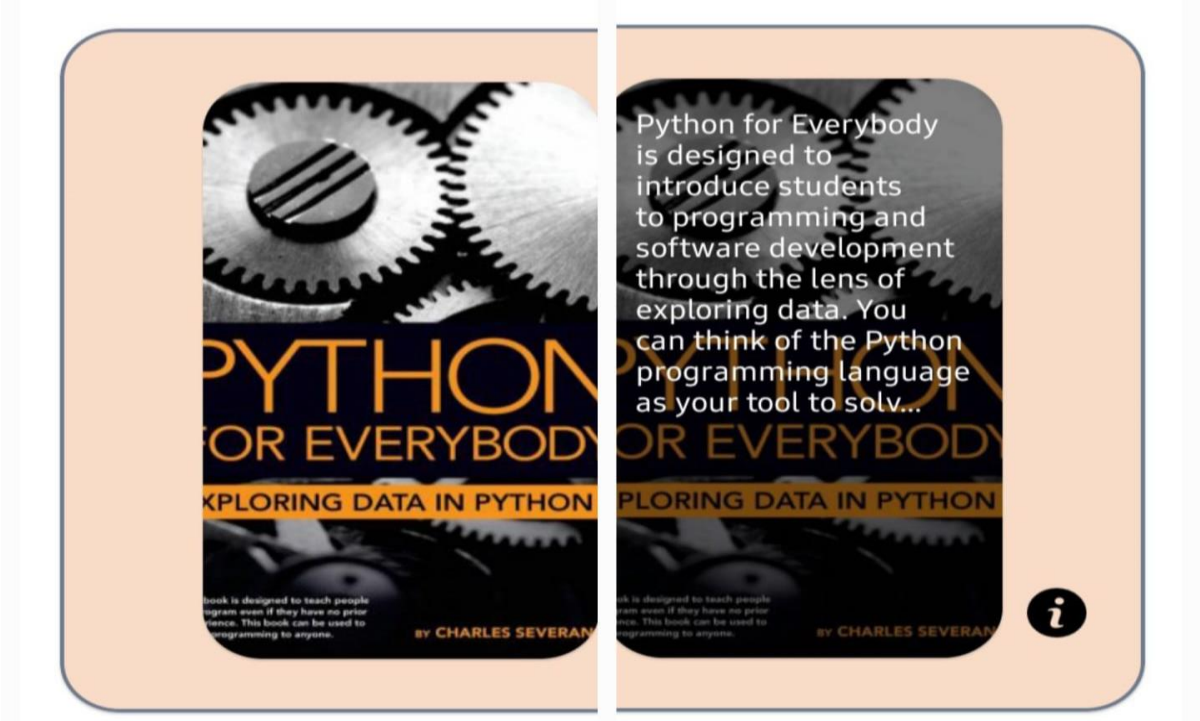
  It features that last book the user opened its pdf file from the app, it uses SharedPreferences to store the id of the book then call the book from the database, if not found, then call it from the API

- Recently added Books:

  It features a list of the last 20 added books in the website, the app lists them in a horizontal way

- Recommended Books Pager:

  The recommended books are featured in an auto-scroll list that scrolls horizontally by itself.The user can click on a book to stop it from autoscrolling and read the book's description then once finished they can click again so that the description is hidden and the list auto scrolls again. They can also click on the "view info" button to see the details of the desired book.



Details Screen

When any book is clicked in the app, this screen will be opened to show the details of the book clicked, these details include: Title, subtitle, authors, number of pages, year it was published in and its description. The details page also includes the comment section, this section allows the users to add real-time comments and view other's real-time comments, users can alos delete their own comments if they wish to.

The details activity's code makes use of a download helper class and a view model to effectively implement its features. The download helper implements download manager and uses a broadcast receiver to handle the download and its states well. It also saves its data in the presistent room databases to store the data.

The database related to details activity is as follows:

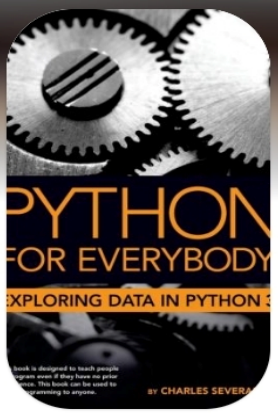
Local book: any book in the library is stored here (whether downloaded or favourited) and all of its data

Downloads: strores only the id of the book and the file path

Favourites: stores only the id of the book

And the three tables are connected together throuth the id as a primary and foreign key.

The comment section in the details page will be explained later in another section.



Python for Everybody

Exploring Data Using Python 3

Dr. Charles Severance

Read offline

pages: 247

year: 2016

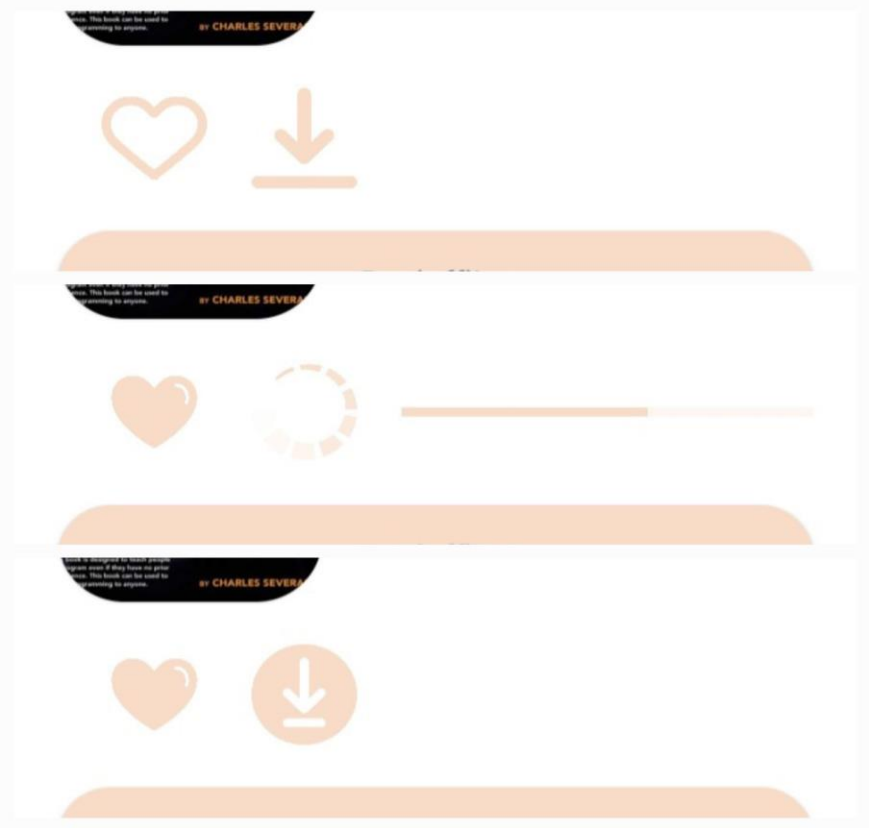
publisher: Self-publishing

Python for Everybody is designed to introduce students to programming and software development through the lens of exploring data. You can think of the Python programming language as your tool to solv...

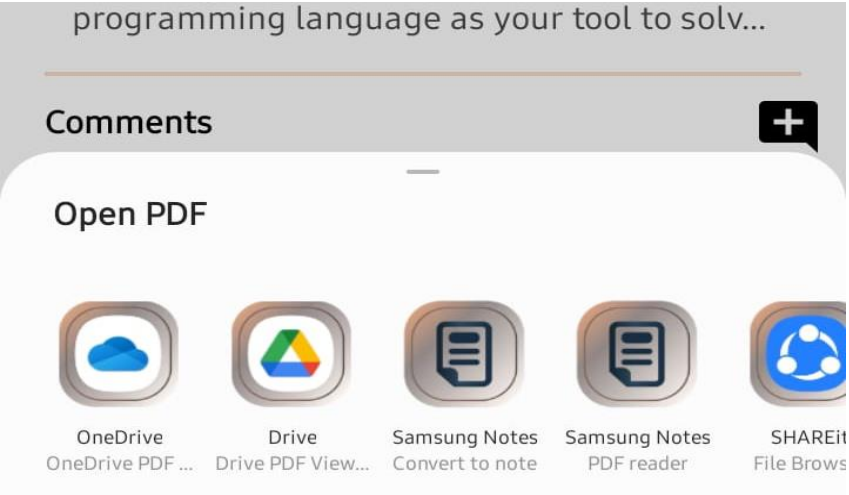
Comments

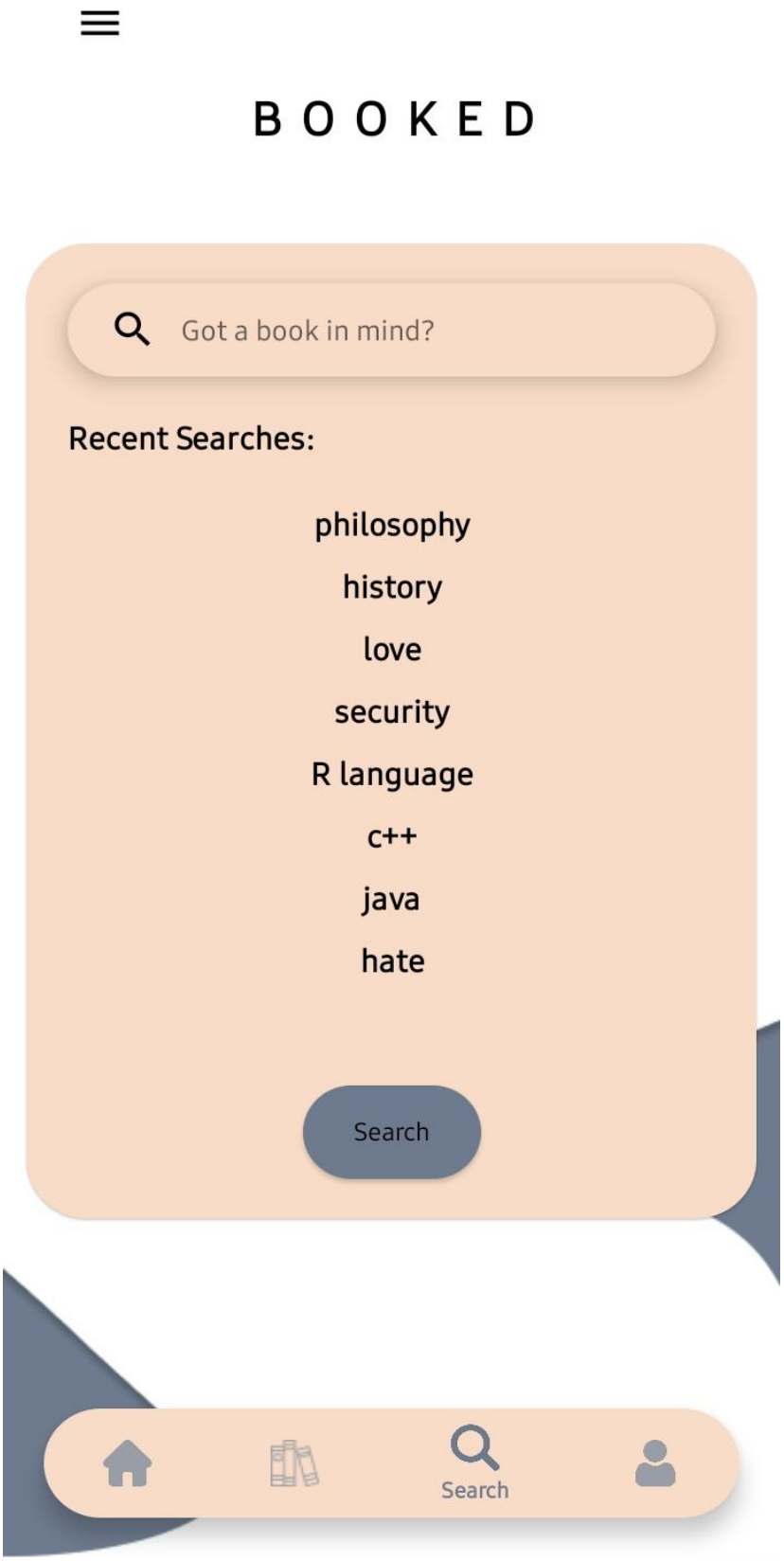
Yasmine 19/10/2024 23:51

This book is nice



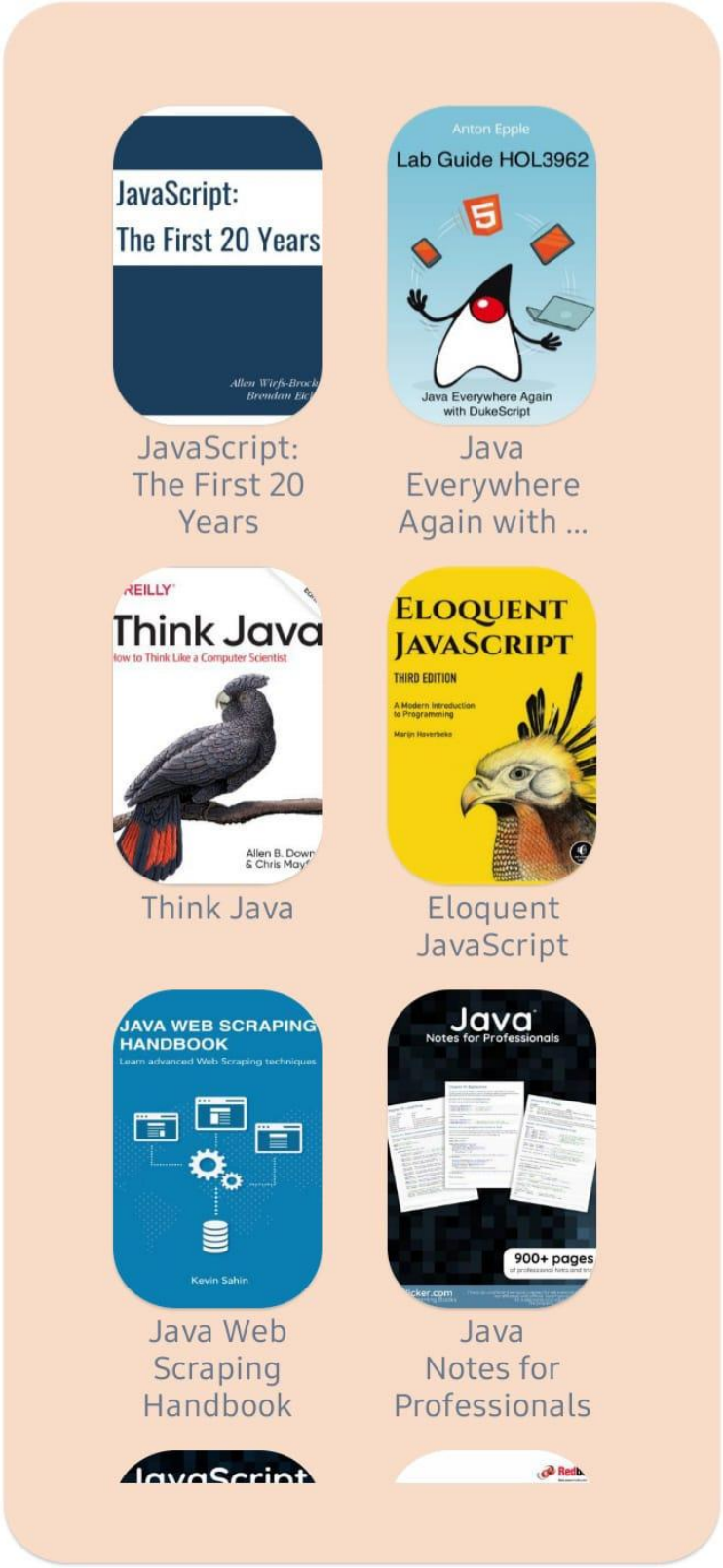
The details page feature multiple important functionalities and they include: add book to library, download the book. The page shows the book's downloading state whether not downloaded, downloading or download. It also lets the user read the book if it's download.





Search Screen

The Search screen enables the user to search for a book or a general term for which they wish to browse books related to, knowing that their search history's saved

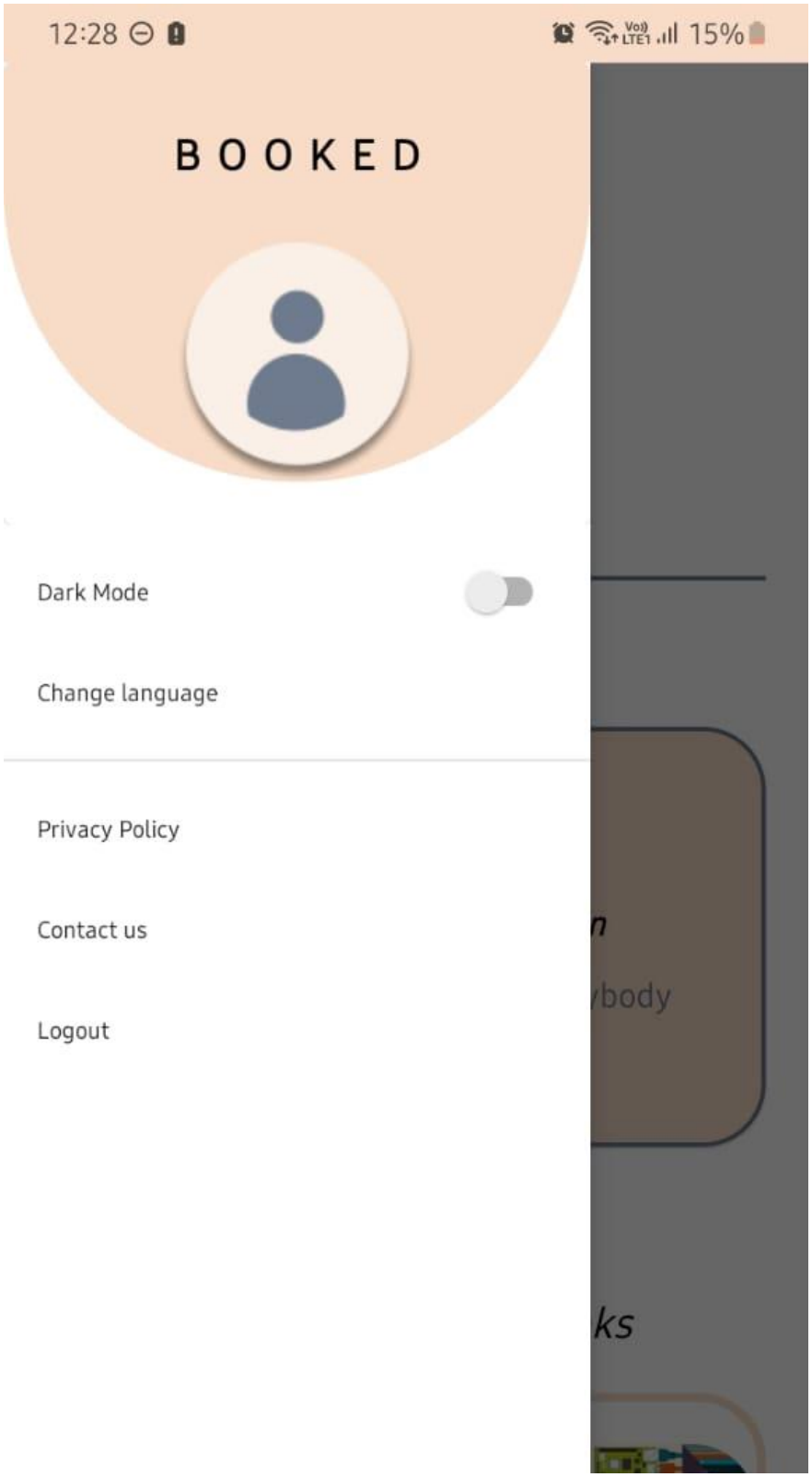


Search Results Screen

Reslts of the search, are shown in

this screen in a Grid view. It also a visually appealing icon to show if a search yielded no results.

Drawer


12:28
15%
BOOKED
Dark Mode
Change language
Privacy Policy
Contact us
Logout

Drawer Design includes a header using:

MaterialCardView: This serves as the overall container for the header.

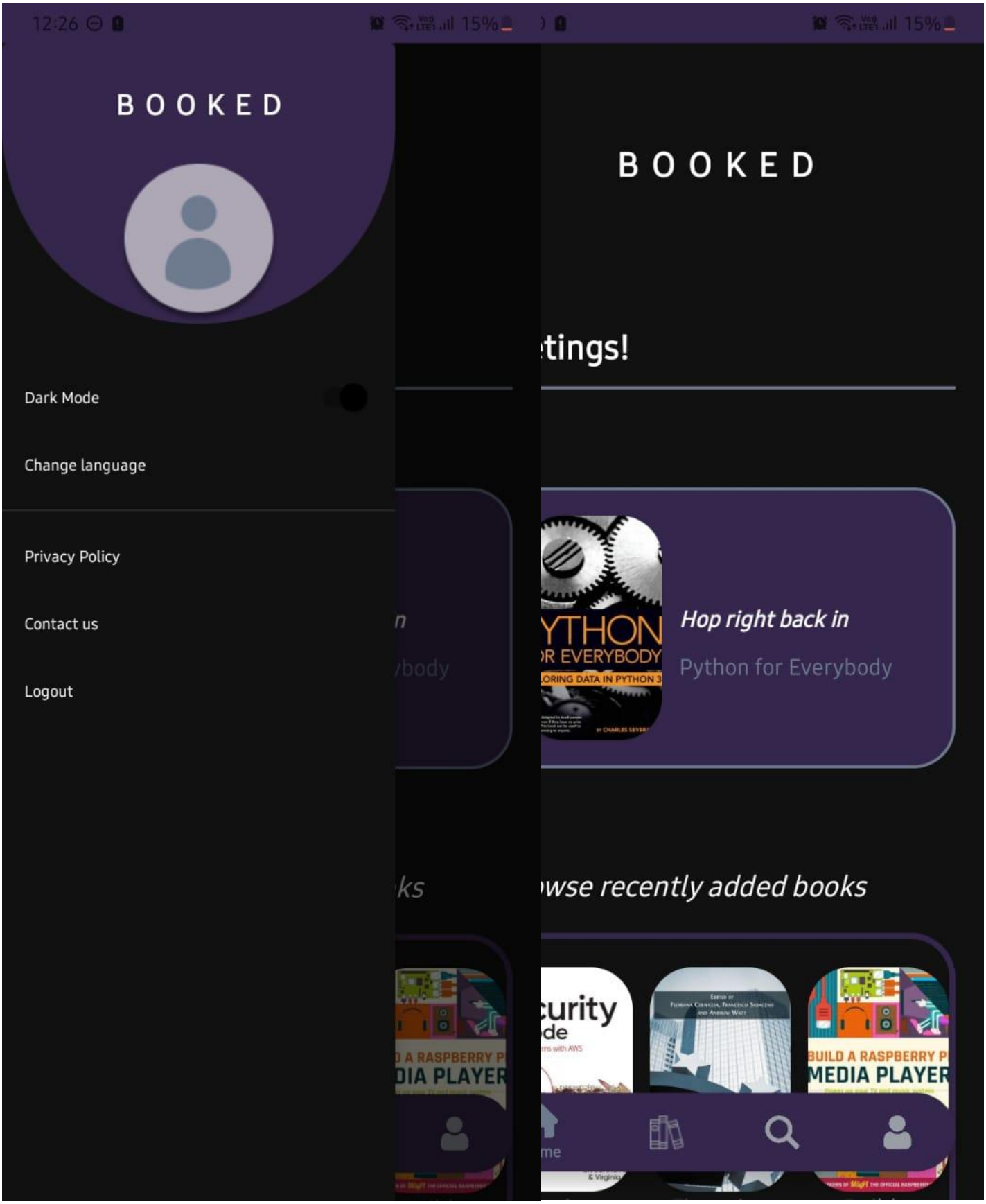
TextView : Displays the text Booked, representing the header content.

ImageView : Shows the profile picture (or "avatar") below the text.

The drawer has Five main functions and they are: dark mode, changing the language, privacy policy, contact us, and log out.

Dark mode

The first item in the menu is Dark Mode: used a UI that appears as a switch using ActionViewClass


12:26
15%
BOOKED
Dark Mode
Change language
Privacy Policy
Contact us
Logout
BOOKED
tings!
Hop right back in
Python for Everybody
wse recently added books

To enable the dark mode switch in the app, the coding process followed these steps:

Initialized Shared Preferences:

I created an instance of MySharedPreference to manage the app's settings, specifically for saving and retrieving the user's theme preference.

Set Up the Na vigation Drawer:

I retrieved the navigation drawer (NavigationView) using view binding to access its menu items.

Found the Dark Mode Switch:

I located the menu item for dark mode using its identifier (R.id.dark) and cast it to a SwitchCompat to allow toggle functionality.

Set Initial State:

I set the initial state of the switch based on the user's saved preference. This determines whether the switch should start in the checked (dark mode) or unchecked (light mode) position.

Handled Toggle Events:

I implemented a listener for the switch to handle user interactions. **When the switch is toggled:**

If checked, I changed the app's mode to dark using AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES).

If unchecked, I reverted to light mode with AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO).

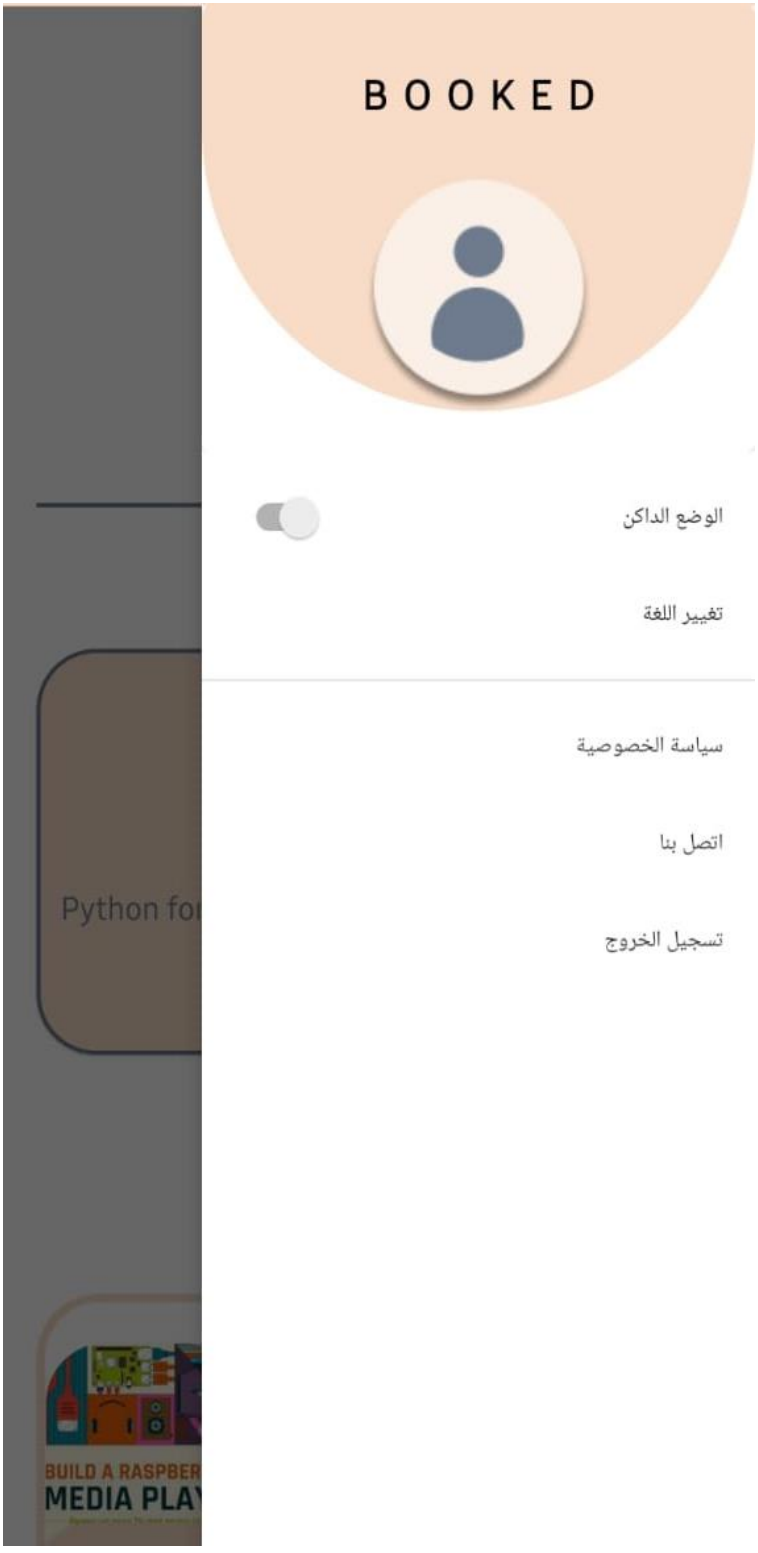
After each toggle, I saved the current state of the switch using a dedicated method to store the user's preference.

Saved Theme Preference:

I created a method to save the user's theme preference, ensuring that the selected mode persists even after the app is closed and reopened.

Language


BOOKED
مرحبًا!
عد للقراءة مجددًا
Python for Everybody
PYTHON FOR EVERYBODY
EXPLORING DATA IN PYTHON 3
استعرض الكتب المضافة حديثًا
Security as Code
BUILD A RASPBERRY PI MEDIA PLAYER
الصفحة الرئيسية


BOOKED
الوضع الداكن
تغيير اللغة
سياسة الخصوصية
اتصل بنا
تسجيل الخروج
Python for
BUILD A RASPBERRY PI MEDIA PLAYER

To implement the change language ,the coding process was as follows:

Handle Language Selection:

When the user selects the language option from the navigation drawer, we check the current language of the app.

If the current language is Arabic ("ar"), we call the setLocale("en") function to switch to English.

If the current language is English (or any other language), we call setLocale("ar") to switch back to Arabic.

Set Locale Function:

We created a function named setLocale that accepts a language code as an argument.

Inside this function:

We create a Locale object using the provided language code.

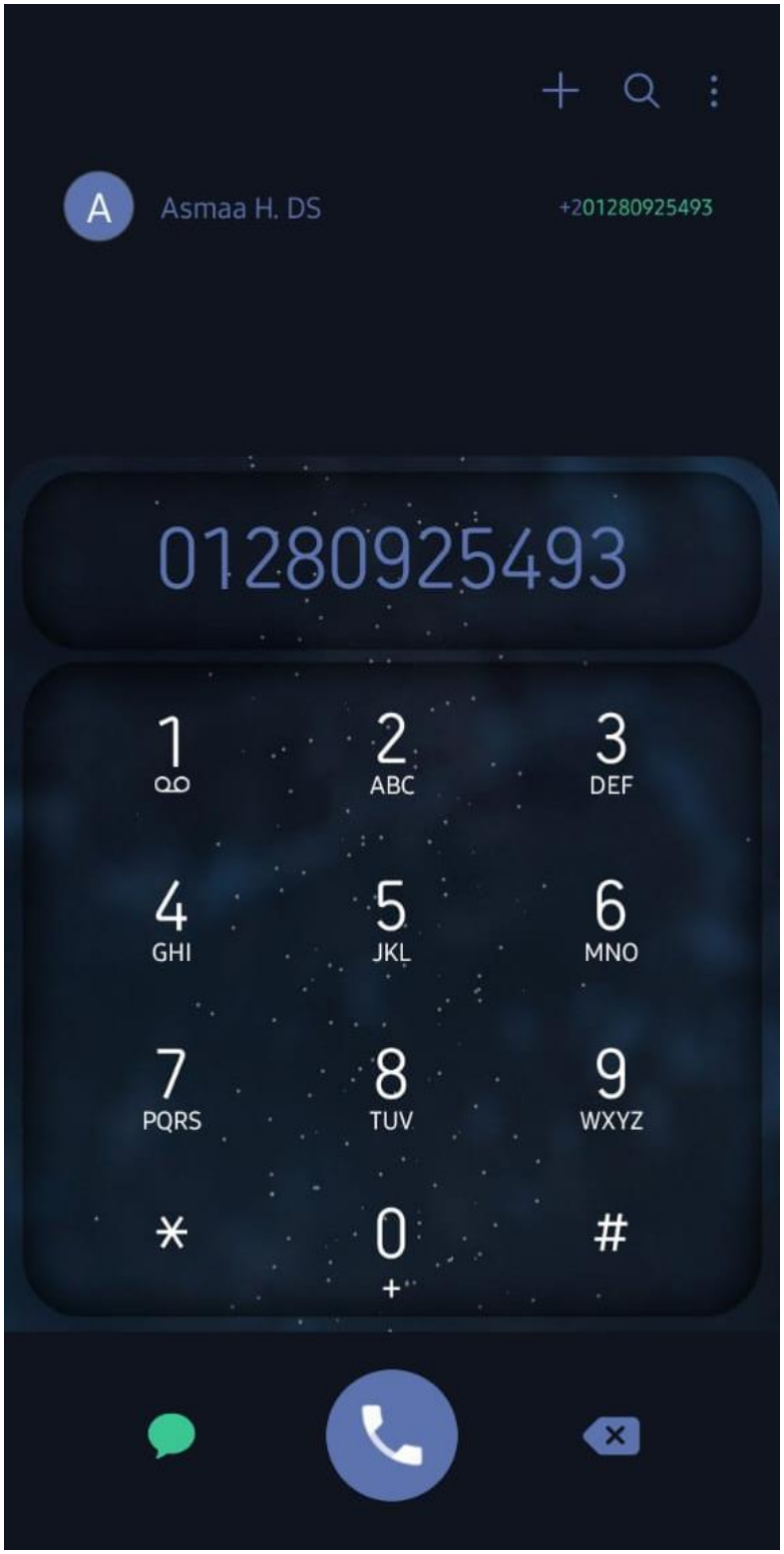
We set this locale as the default using Locale.setDefault(locale).

We update the app's configuration to apply the new locale.

We save the new locale in shared preferences so that it persists across app sessions.

Finally, we restart the main activity to apply the changes, ensuring that the user sees the app in the selected language immediately

Contact us


A
Asmaa H. DS
+201280925493
01280925493
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
*
0 +
#

To Implement the "Contact Us" Feature, We Did the Following:

Handle Contact Selection:

When the user selects the "Contact Us" option from the navigation drawer, we call the contactUs() function.

Define the contactUs Function:

Inside this function:

we define the phone number we want users to call, which is "01280925493".

We create an intent with the action Intent.ACTION_DIAL, which allows us to open the phone's dialer.

We set the data of the intent using Uri.parse("tel:$phoneNumber") to format the phone number correctly.

Finally, we start the activity with this intent, which launches the dialer, ready for the user to make the call.

To allow the app to dial phone numbers:

we declared the necessary permission in the AndroidManifest.xml file.

<uses-permission
android:name="android.permission.CALL_PHONE"/>

Bottom navigation

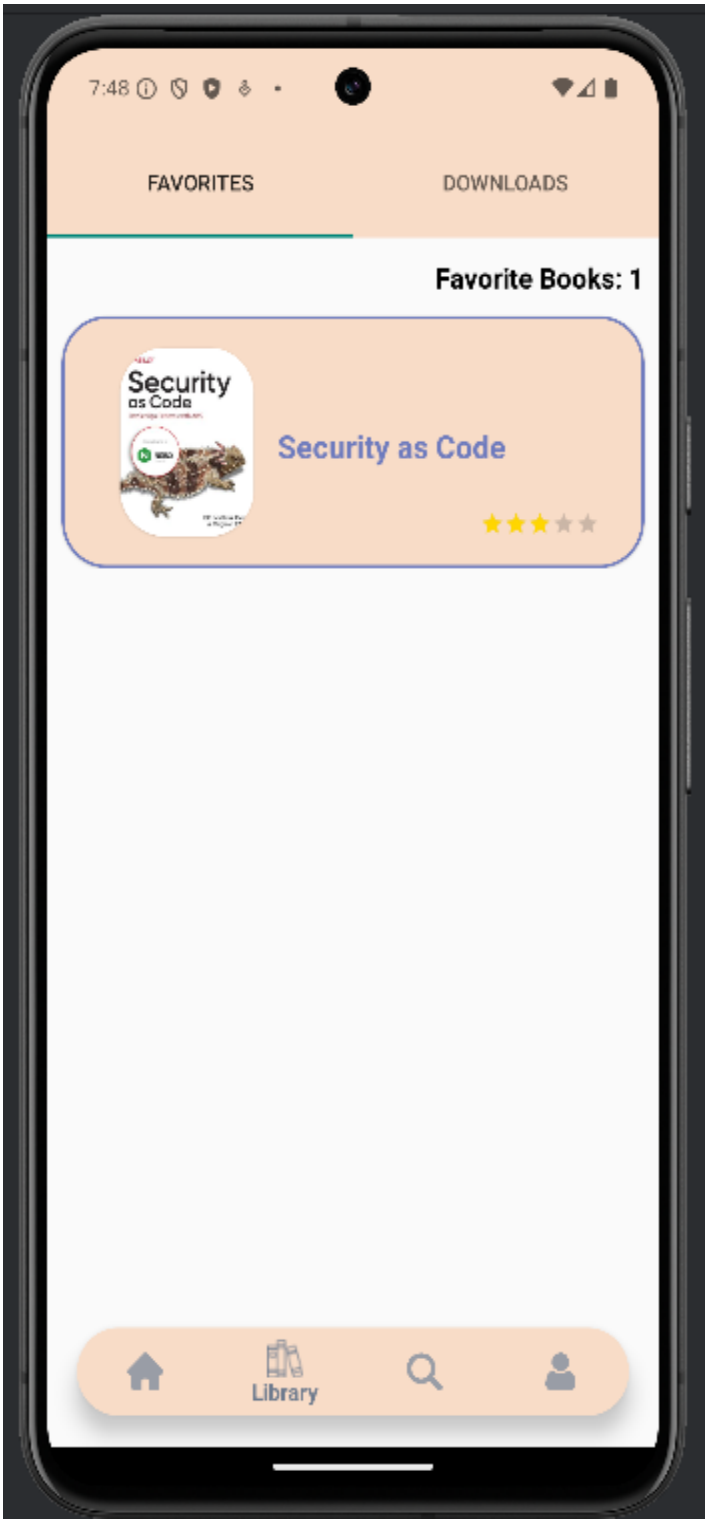
To implement a bottom navigation bar that transitions between different fragments:

An XML menu was created, with defined items and their attributes. Then, in the Kotlin code, we set up the navigation controller to manage these transitions and handled any intents that may specify a direct fragment to open.

Library Screen

-Favorites tab

- Downloads tab


7:48
FAVORITES
DOWNLOADS
Favorite Books: 1
Security as Code
Security as Code
Library

Library Fragment (Favorites & Downloads)

The **Library Fragment** allows users to manage their favorite and downloaded books in two distinct sections: **Favorites** and **Downloads**. Each section uses a **RecyclerView** to display the books saved by the user, leveraging a **Room database** for persistence. When a user marks a book as a favorite or downloads it, it is automatically saved locally and displayed under the corresponding tab. Clicking on any book in either section opens the **Details Activity**, which provides comprehensive information about the book.

This design ensures that users can access and view book details even without an active internet connection, thanks to the local storage of books in the Room database.

Key Components:

- **LibraryPagerAdapter**: Handles switching between the Favorites and Downloads tabs.
- **LibraryAdapter**: Binds the book data to the RecyclerView.
- **LibraryFragment**: Parent fragment that contains the Favorites and Downloads.
- **FavoriteFragment**: Displays the user's favorite books.
- **DownloadFragment**: Displays books that the user has downloaded.

More details:

Library Fragment (Favorites & Downloads)

Overview

The **Library Fragment** is a key component of the app that allows users to view and manage their favorite and downloaded books. It consists of two main tabs: **Favorites** and **Downloads**, each represented by individual fragments. These fragments display the user's books using a **RecyclerView** and retrieve the data from a local **Room database**. The data displayed in these lists is updated whenever the user marks a book as favorite or downloads a book.

When a book is clicked, the user is navigated to the **Details Activity**, which provides detailed information about the book. This ensures that even if the user is offline, they can still access the book details, as the data is stored locally.

Architecture and Components

1. LibraryFragment

- **Purpose**: Acts as the parent fragment that hosts the **FavoritesFragment** and **DownloadsFragment**.
- **View**: Uses a **ViewPager2** for tab-based navigation, allowing users to switch between their favorite books and downloaded books.
- **Data Handling**: The fragment initializes the **ViewModel**, which handles the retrieval and storage of book data from the Room database.

2. FavoritesFragment

- **Purpose**: Displays all books that the user has marked as favorites.
- **View**: Implements a **RecyclerView** to display favorite books in a list format.
- **Data Handling**: Books are retrieved from the **Room database** where favorite books are stored persistently.
- **Book Click Action**: When a user clicks on a book, it triggers the intent to open the **DetailsActivity**, passing the necessary data to display all the book's information.

3. DownloadsFragment

- **Purpose**: Displays all books that the user has downloaded.
- **View**: Similar to the **FavoritesFragment**, it uses a **RecyclerView** to display the downloaded books.
- **Data Handling**: Downloads are also saved in the **Room database** for offline access.
- **Book Click Action**: Like in the Favorites section, clicking a book navigates the user to the **DetailsActivity**.

Key Components:

a. LibraryPagerAdapter

- **Purpose**: Manages the different fragments (Favorites and Downloads) within the **ViewPager2** in the **LibraryFragment**.
- **Functionality**: The adapter returns the appropriate fragment (Favorites or Downloads) based on the selected tab.

b. LibraryAdapter

- **Purpose**: Acts as the adapter for the **RecyclerView**, responsible for binding book data to the view items.
- **Implementation**: It efficiently handles loading book covers, titles, and ratings, and provides a click listener for each book item to open the **DetailsActivity**.

c. Room Database Integration

- The app uses a **Room database** to store the user's favorite and downloaded books locally. This ensures that users have access to their collection even when offline.
- **Favorite books** are added or removed from the database when the user marks a book as a favorite.
- **Downloaded books** are saved when the user downloads a book, making them accessible through the **DownloadsFragment**.

d. DetailsActivity

- **Purpose**: This activity provides detailed information about a selected book, such as its description, rating, reviews, etc.
- **Offline Capability**: Since the app stores book information locally in the Room database, users can access this activity even when their device is offline.

Data Flow

1. **Adding a Favorite or Download**:
    - When the user marks a book as a favorite or downloads it, the app stores the book's data in the Room database.

2. **Displaying Favorites or Downloads**:
    - The **FavoritesFragment** and **DownloadsFragment** query the Room database to retrieve the relevant books and display them using the **LibraryAdapter**.
3. **Book Click Action**:
    - Clicking on any book item within the **RecyclerView** triggers an intent to open the **DetailsActivity**. The activity retrieves the book details, including title, author, description, and other relevant data, which can be displayed whether the user is online or offline.

UI and User Experience

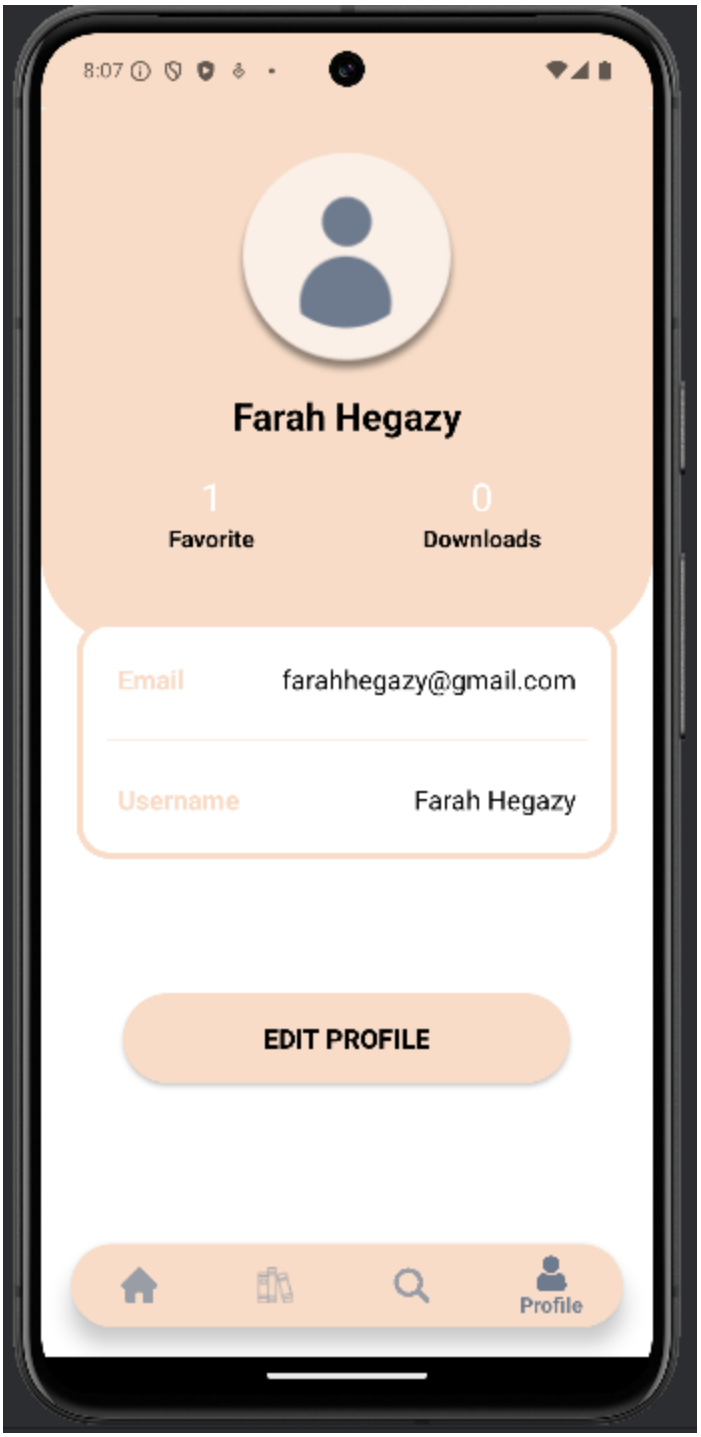
- **Empty States**: When the user has no favorite or downloaded books, an empty state image is shown with a message such as "Favorite Books are Empty" or "Downloaded Books are Empty".
- **Visual Elements**: The tabs and book lists are designed with a clean and minimalist UI, ensuring that users can easily navigate through their library.
- **Offline Access**: One of the main features of the Library Fragment is its offline access to book details. Users can mark books as favorite or download them, and later access these books even when they are not connected to the internet.

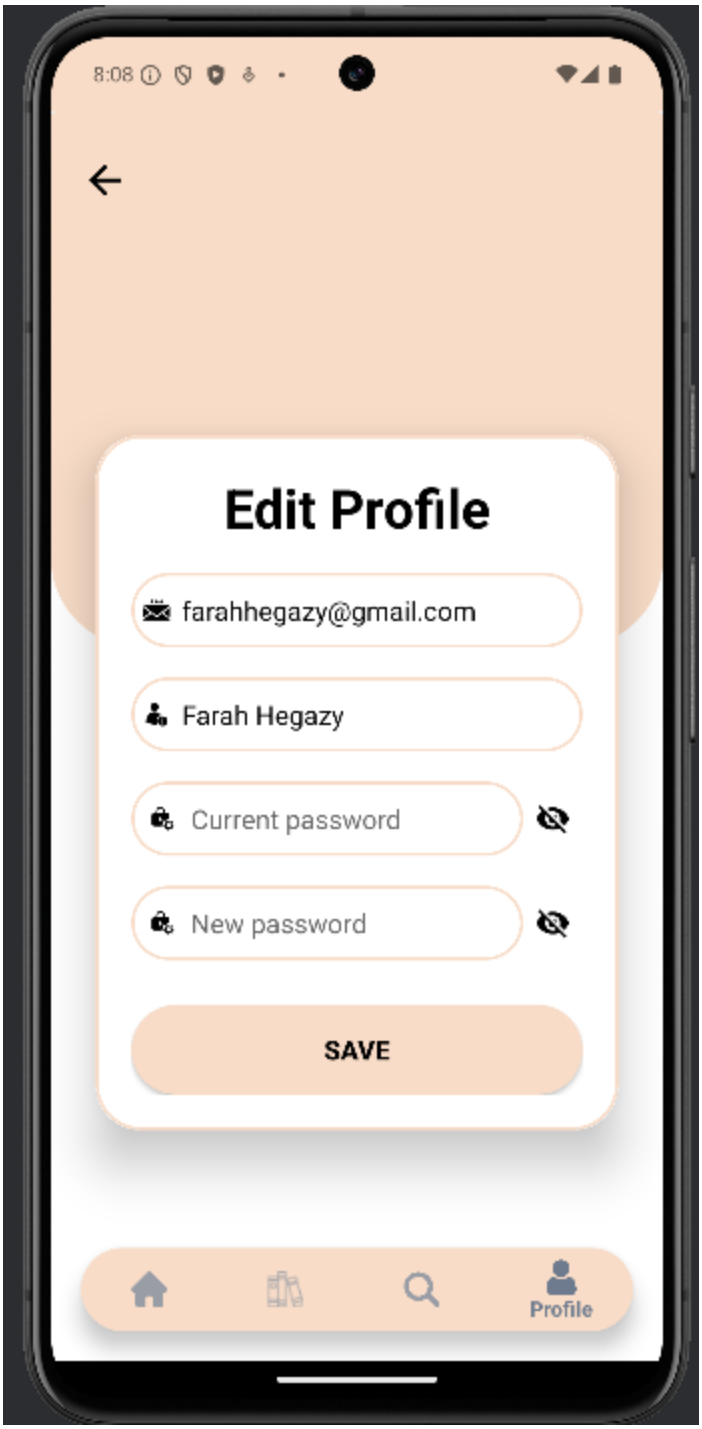
Technology Stack

- **Kotlin**: The entire fragment and associated components are developed using Kotlin.

- **Room Database**: Handles local data storage for favorites and downloads.
- **RecyclerView**: Efficiently displays lists of books in both the Favorites and Downloads sections.
- **ViewPager2**: Provides tabbed navigation between Favorites and Downloads.
- **MVVM Architecture**: Ensures separation of concerns and clean handling of data through ViewModels.

Profile Screen


8:07
Farah Hegazy
1
Favorite
0
Downloads
Email
farahhegazy@gmail.com
Username
Farah Hegazy
EDIT PROFILE
Profile


8:08
Edit Profile
farahhegazy@gmail.com
Farah Hegazy
Current password
New password
SAVE
Profile

User Profile Fragment and Account Management:

The user profile screen serves as a centralized hub for displaying and managing the key details associated with the app user's account. This includes:

- Username and Email Address.

- Favorites and Downloads:

The profile screen also surfaces the total number of books the user has marked as favorites, as well as the cumulative number of downloads. These metrics offer valuable insights into the user's engagement and content consumption within the app.

- Edit Profile Button:

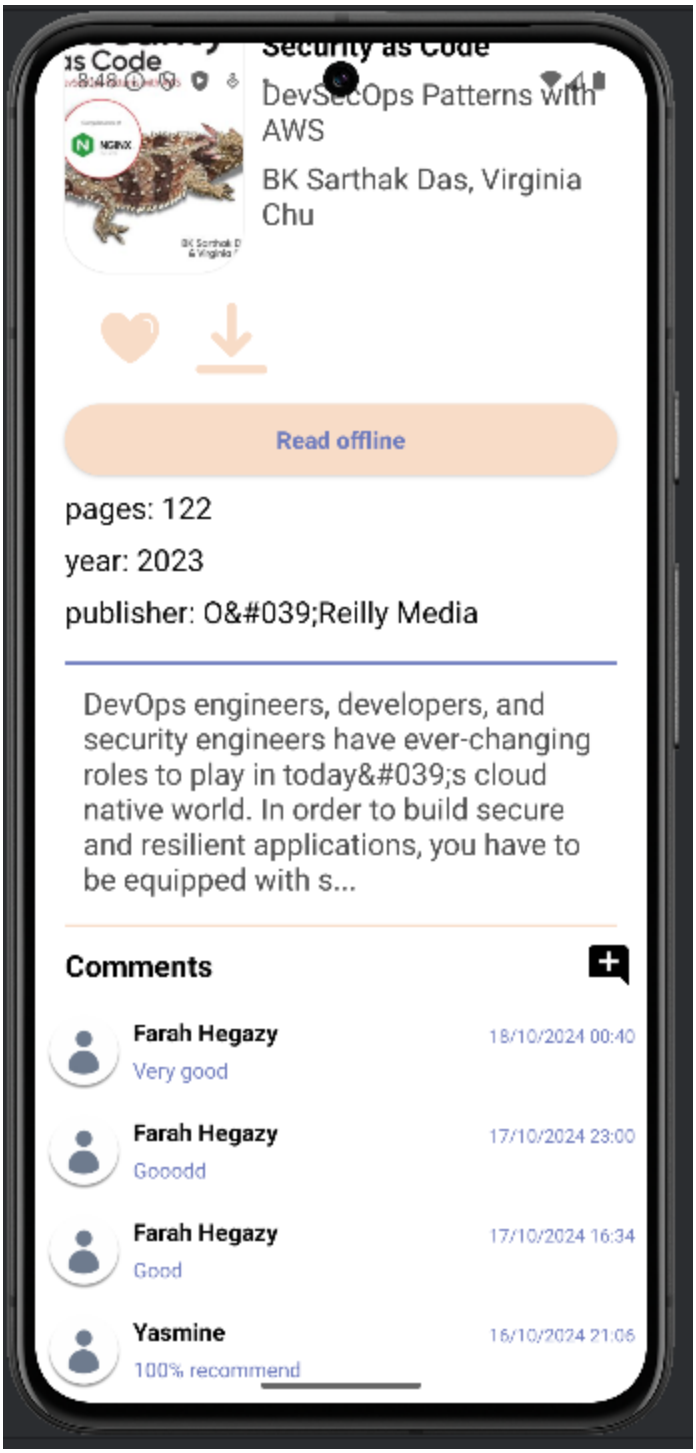
In the profile Fragment, there is an "Edit Profile" button, to provide a shortcut for users if they want to rapidly make changes to their account information. This smoothly moves the user to edit the profile Fragment.

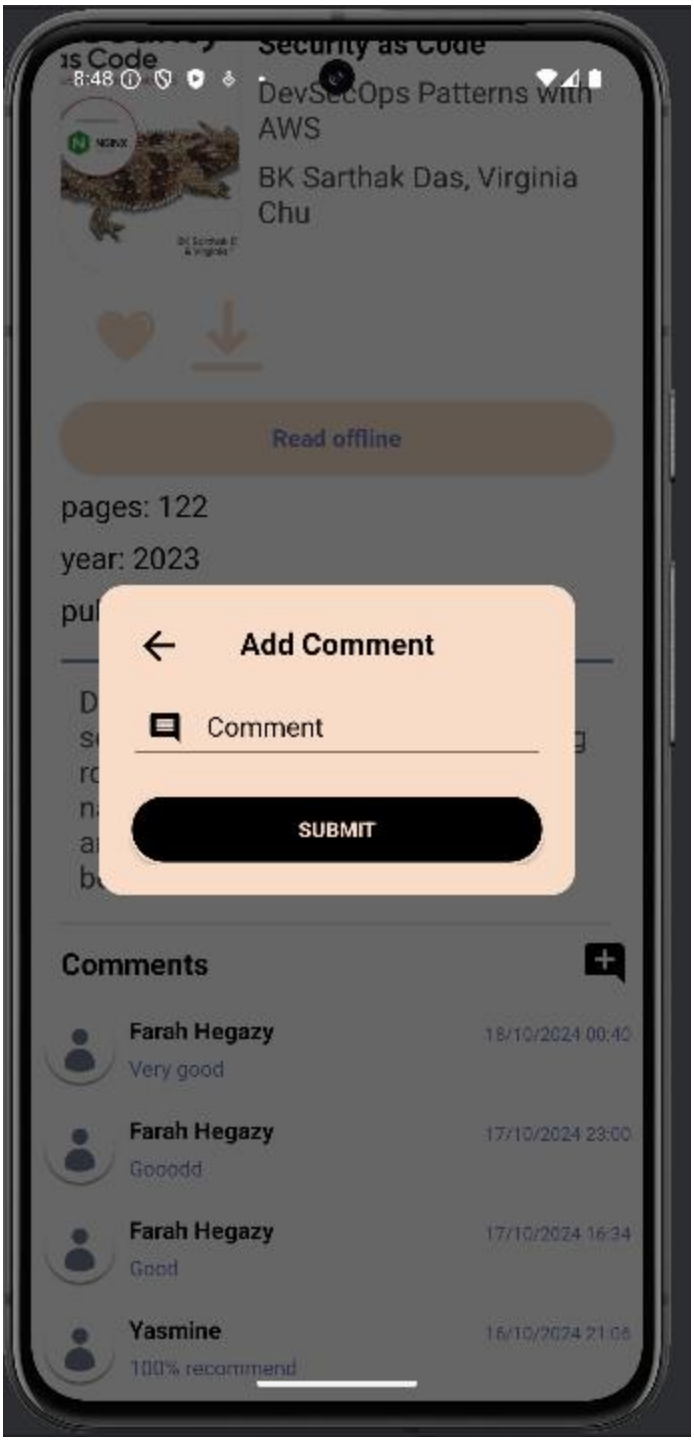
Edit Profile Fragment:

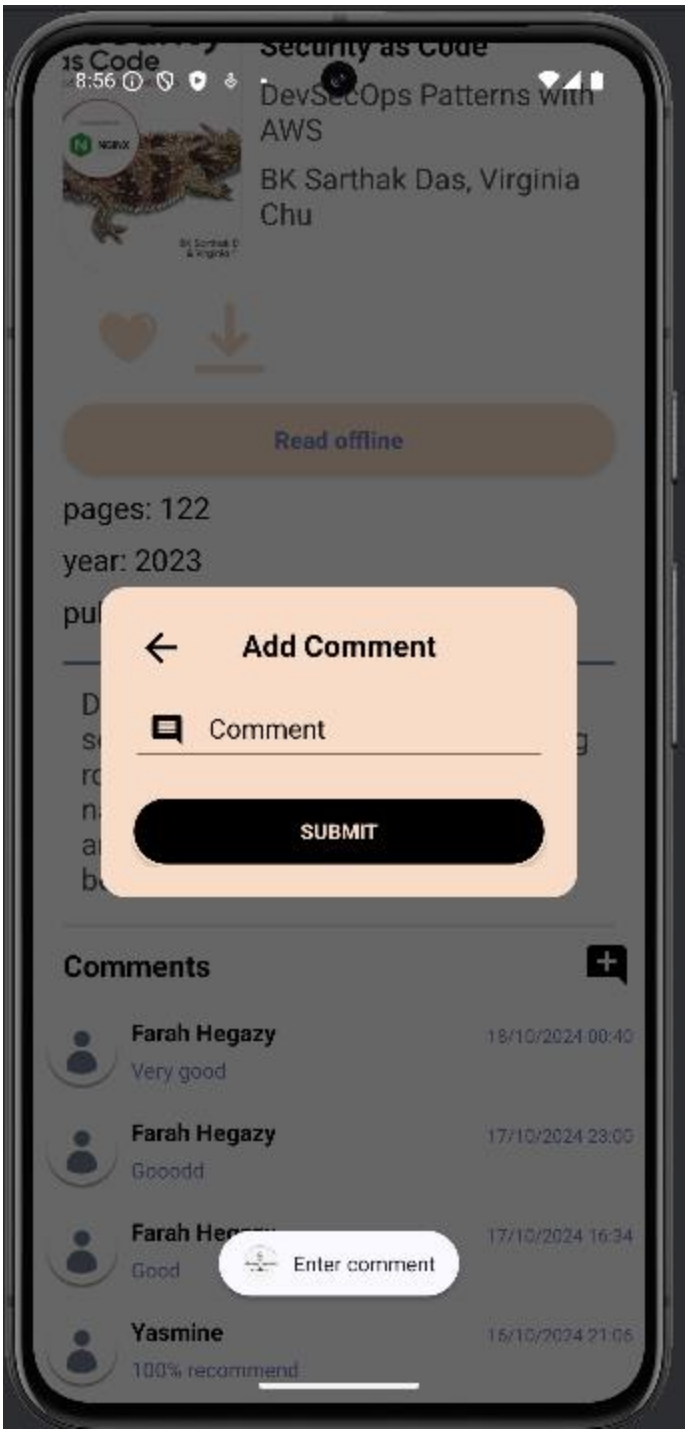
The edit profile screen presents fields for the user to update their **username, email address, and password**.

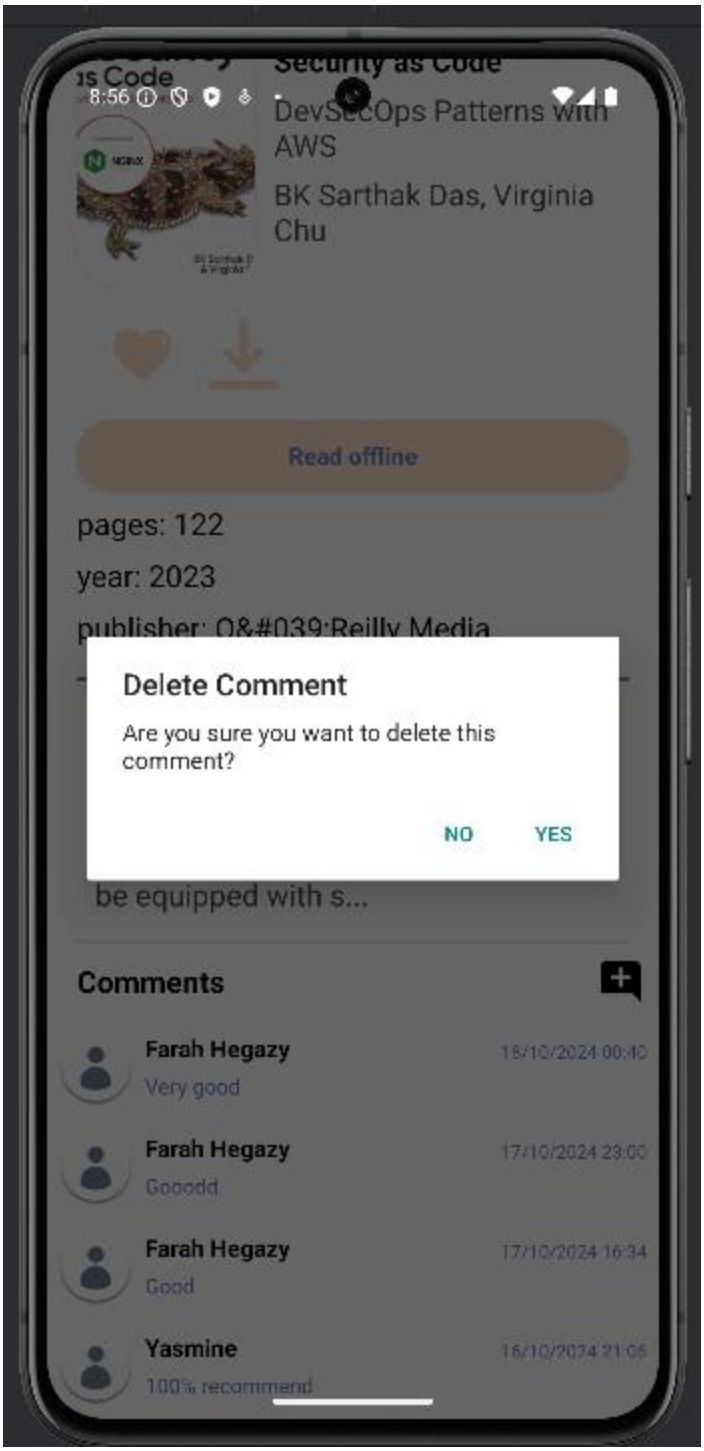
Once the desired changes have been made, the user can tap the "Save" button to commit the updates to their account information stored in Firebase.

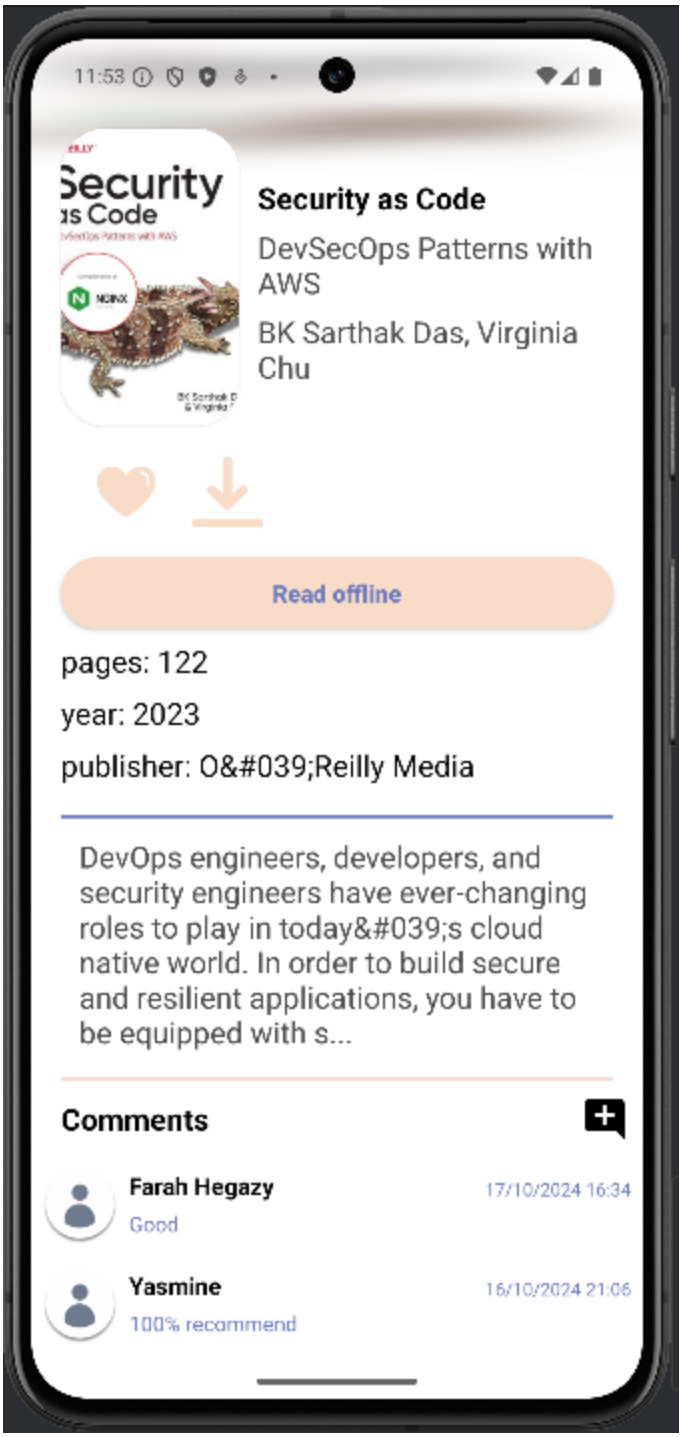
This edit profile functionality empowers users to manage their details directly within the app, enhancing the user experience and giving them control over their accounts.

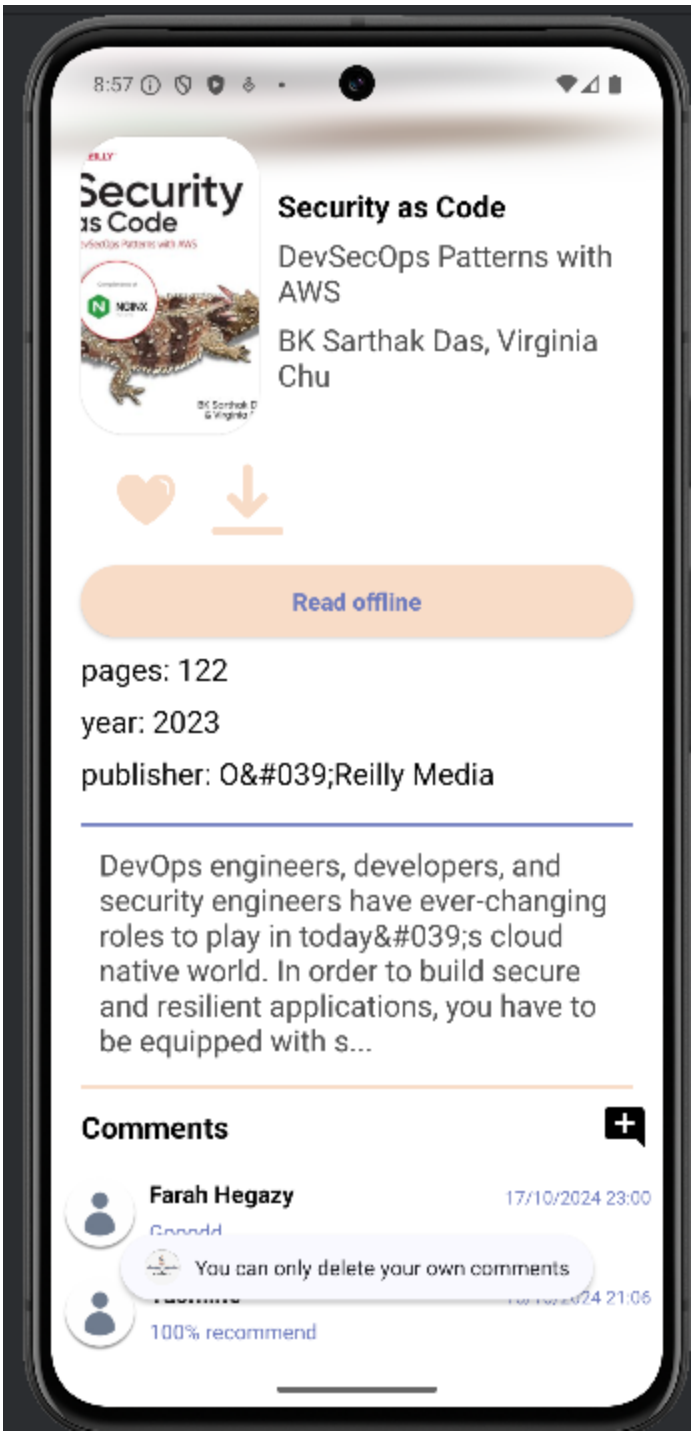

Security as Code
DevSecOps Patterns with AWS
BK Sarthak Das, Virginia Chu
Read offline
pages: 122
year: 2023
publisher: O'Reilly Media
DevOps engineers, developers, and security engineers have ever-changing roles to play in today's cloud native world. In order to build secure and resilient applications, you have to be equipped with s...
Comments
Farah Hegazy
18/10/2024 00:40
Very good
Farah Hegazy
17/10/2024 23:00
Gooodd
Farah Hegazy
17/10/2024 16:34
Good
Yasmine
16/10/2024 21:06
100% recommend


Security as Code
8:48
DevSecOps Patterns with AWS
BK Sarthak Das, Virginia Chu
Read offline
pages: 122
year: 2023
Add Comment
Comment
SUBMIT
Comments
Farah Hegazy
18/10/2024 00:40
Very good
Farah Hegazy
17/10/2024 23:00
Gooodd
Farah Hegazy
17/10/2024 16:34
Good
Yasmine
16/10/2024 21:06
100% recommend


Security as Code
DevSecOps Patterns with AWS
BK Sarthak Das, Virginia Chu
Read offline
pages: 122
year: 2023
Add Comment
Comment
SUBMIT
Comments
Farah Hegazy
18/10/2024 00:40
Very good
Farah Hegazy
17/10/2024 23:00
Gooodd
17/10/2024 16:34
Good
Enter comment
Yasmine
15/10/2024 21:06
100% recommend


Security as Code
8:56
DevSecOps Patterns with AWS
BK Sarthak Das, Virginia Chu
Read offline
pages: 122
year: 2023
publisher: O'Reilly Media
Delete Comment
Are you sure you want to delete this comment?
NO
YES
be equipped with s...
Comments
Farah Hegazy
18/10/2024 00:40
Very good
Farah Hegazy
17/10/2024 23:00
Gooodd
Farah Hegazy
17/10/2024 16:34
Good
Yasmine
16/10/2024 21:06
100% recommend


11:53
Security as Code
DevSecOps Patterns with AWS
BK Sarthak Das, Virginia Chu
Read offline
pages: 122
year: 2023
publisher: O'Reilly Media
DevOps engineers, developers, and security engineers have ever-changing roles to play in today's cloud native world. In order to build secure and resilient applications, you have to be equipped with s...
Comments
Farah Hegazy
17/10/2024 16:34
Good
Yasmine
16/10/2024 21:06
100% recommend


8:57
Security as Code
DevSecOps Patterns with AWS
BK Sarthak Das, Virginia Chu
Read offline
pages: 122
year: 2023
publisher: O'Reilly Media
DevOps engineers, developers, and security engineers have ever-changing roles to play in today's cloud native world. In order to build secure and resilient applications, you have to be equipped with s...
Comments
Farah Hegazy
17/10/2024 23:00
You can only delete your own comments
100% recommend

Comment Section - Book Details Activity

The comment section in the **Book Details Activity** allows users to interact with individual books by adding their thoughts, reviews, or feedback. This feature is designed to enhance user engagement and provide a platform for sharing opinions about each book.

Key Features

- **Add Comment:** Users can add comments by clicking on the "+" icon located in the comments section. Upon clicking, a dialog appears prompting the user to enter their comment. Once submitted, the comment is stored in Firebase at the following path:

collection("books").document(bookId).collection("comments").document(timestamp)

The data stored for each comment includes:

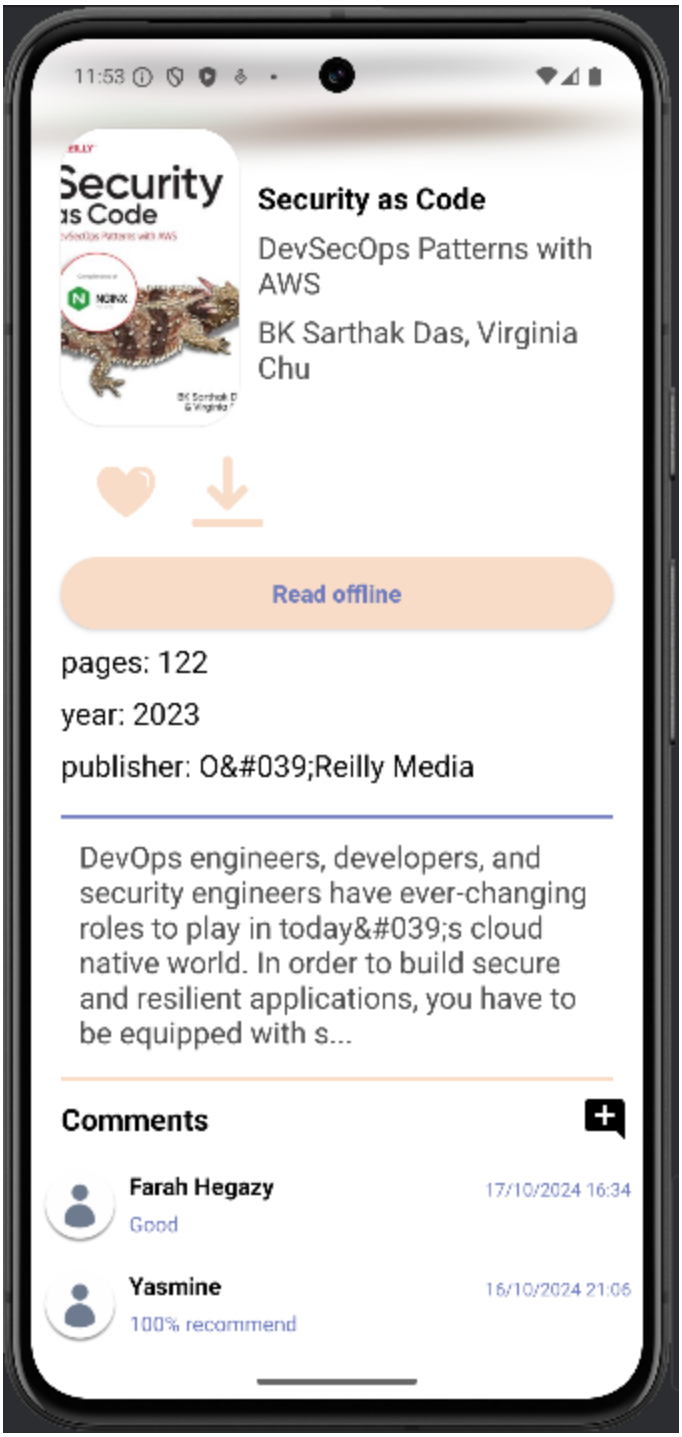
  - **id**: Unique identifier for the comment.
  - **timestamp**: Date and time the comment was created.
  - **uid**: User ID of the commenter.
  - **username**: Name of the user who wrote the comment.
  - **comment**: The actual text of the comment.
- **Delete Comment:** Users have the ability to delete comments they have written. If the user attempts to delete a comment, a confirmation dialog appears asking them to confirm or cancel the deletion. This ensures that only the comment author has control over their own comments.

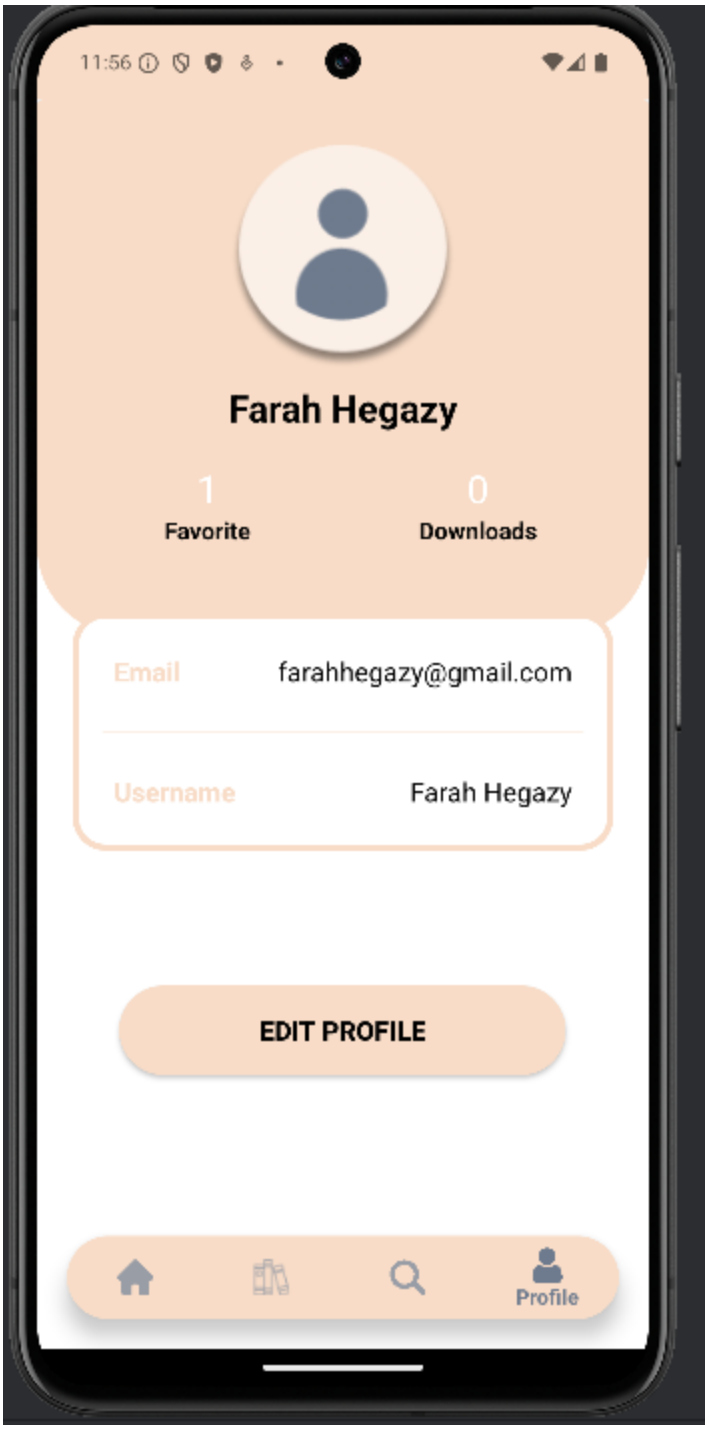
- **Display Comments:** The comments are displayed in a **RecyclerView**, sorted in descending order based on the timestamp. This arrangement ensures that the most recent comments are always shown at the top of the list.

User Interface

- The comment section is integrated into the book details page, providing a seamless user experience.
- Each comment item displays the username, comment text, and the date and time the comment was posted.
- The user-friendly design includes options to quickly add or delete comments with minimal navigation.

This implementation leverages Firebase's real-time capabilities, ensuring that comments are updated and synchronized efficiently across all users' devices.


11:53
Security as Code
DevSecOps Patterns with AWS
BK Sarthak Das, Virginia Chu
Read offline
pages: 122
year: 2023
publisher: O'Reilly Media
DevOps engineers, developers, and security engineers have ever-changing roles to play in today's cloud native world. In order to build secure and resilient applications, you have to be equipped with s...
Comments
Farah Hegazy
17/10/2024 16:34
Good
Yasmine
16/10/2024 21:06
100% recommend


11:56
Farah Hegazy
1
Favorite
0
Downloads
Email
farahhegazy@gmail.com
Username
Farah Hegazy
EDIT PROFILE
Profile


11:56
Yasmine
Email
tempacc@gmail.com
Username
Yasmine
Home

Comment Section Behavior in Details Activity

The **Comment Section** in the **Details Activity** of the app is designed to enhance the user experience by allowing users to engage with reviews and comments on specific books. One key feature of this section is the ability for users to interact with the profile pictures displayed next to the comments. The behavior of this interaction is dynamic and context-aware, adjusting based on whether the user viewing the comment is the same as the one who posted it. Below is a detailed explanation of this feature:

1. Profile Picture Interaction

- **When clicking on a profile picture**, the app first checks whether the profile belongs to the **current logged-in user** or a **different user**.
    - **If the profile picture belongs to the current user**, the app navigates the user to their own **ProfileFragment**, where they can view and manage their personal information, books, and activity.
    - **If the profile picture belongs to another user**, the app navigates to a separate **UserProfileFragment**, displaying public information about that user.

2. UserProfileFragment

- **User Information Displayed**: The **UserProfileFragment** is designed to provide an intuitive interface for viewing other users' profiles. This fragment shows the following key details:
    - **Username**: Displayed prominently to identify the user.

    - **Email**: The email of the user is shown, allowing others to contact them directly if necessary.
    - **Profile Picture**: The same profile picture clicked in the comment section is displayed, reinforcing the identity of the user being viewed.
- **Chat Icon**:
    - A **Chat Icon** is displayed at the top right corner of the fragment. In future updates, clicking this icon will initiate a chat with the selected user, leading to the opening of a **Chat Activity**.
    - This will enable users to communicate directly with one another, fostering a community around shared reading experiences.

3. Technical Implementation Details

- **Fragments Used**:
    - ProfileFragment: Displays the current user's profile with editable options.
    - UserProfileFragment: Displays public information of other users when their profile picture is clicked.
- **Navigation Logic**:
    - The app uses conditional navigation based on the **Firebase authentication** status of the user. When a user clicks a profile picture, a check is performed to determine if the clicked profile belongs to the current logged-in user.

    - **If yes**: The app navigates to ProfileFragment.
    - **If no**: The app navigates to UserProfileFragment, passing relevant user data to this fragment.
- **Future Features**:
    - In future releases, the **Chat Icon** will be connected to a real-time messaging system, allowing users to chat directly. This feature will leverage Firebase Realtime Database or Firestore for live messaging capabilities, ensuring seamless communication between app users.

4. Why This Feature Matters

- This feature adds a layer of social interaction to the app by enabling users to explore who is commenting on specific books.
- It encourages community engagement by offering a chat feature that will be integrated in the future, helping users discuss books and share recommendations directly within the app.
- By allowing users to see who else is engaging with content and giving them a direct line of communication, the app creates a more interactive and connected reading community.

This feature is part of a broader effort to create a rich, interactive user experience, making it easier for readers to connect, share, and discuss books on the platform.

Notification

Package: com.example.fcmpuchnotification

Purpose: This class extends FirebaseMessagingService and handles incoming Firebase Cloud Messaging (FCM) notifications. It parses the notification data and generates a custom notification on the Android device.

Explanation:

Constants:

channelId: Defines a unique identifier for the notification channel (introduced in Android O).

channelName: Specifies the name displayed for the notification channel in system settings.

onMessageReceived: This is the core method that receives incoming FCM messages.

It checks if the message contains a notification payload.

If a notification is present, it calls the generateNotification method to create and display the notification.

getRemoteView: This method creates a custom notification layout using a RemoteView object.

It inflates the layout defined in R.layout.notification.

It sets the text for the title and message views using the provided arguments.

It sets the image resource for the app logo using the R.drawable.logo resource.

generateNotification: This method builds and displays the notification.

It creates an Intent to launch the MainActivity when the notification is tapped.

It creates a PendingIntent to associate the Intent with the notification.

It builds a NotificationCompat.Builder object to configure the notification details:

Small icon (set using R.drawable.logo)

Auto-cancellation (clears the notification when tapped)

Vibration pattern

Sets the content view using the getRemoteView method with the notification title and message.

Sets the pending intent to launch the MainActivity.

It checks the Android version and creates a NotificationChannel object (required for Android O and above).

Finally, it uses the NotificationManager to display the notification with a unique ID.

Additional Notes:

This example uses a custom notification layout (R.layout.notification). Make sure this layout file exists in your project's resource directory.

The code retrieves resources (icons) using resource IDs (R.drawable.logo). Ensure these resources are properly defined in your project


2:09 PM | 0.4KB/s
Welcome to me
BOOKED
dola
where are u dude