Al-Kitab Journal for Pure Sciences (2024); 8(1):40-62.
https://doi.org/10.32441/kjps.08.01.p5

Al-Kitab Journal for Pure Sciences
ISSN: 2617-1260 (print), 2617-8141(online)
https://isnra.net/index.php/kjps

BANK OF PASSWORDS: a secure Android password manager implemented based on specific requirements

Hussein Abdulkhaleq Saleh*

Directorate General of Education in Dhi Qar., Iraq

*Corresponding Author: hussein.abd.alkhaliq@gmail.com

ORCID: 0009-0003-3426-7168

Citation: Saleh HA. BANK OF PASSWORDS: a secure Android password manager implemented based on specific requirements. Al-Kitab J. Pure Sci. [Internet]. 2024 Mar 12 [cited 2024 Mar 12];8(1):40-62. Available from: https://doi.org/10.32441/kjps.08.01.p5.

Keywords: Secure warehouse for passwords, Passwords vault, Passwords keeper app, Android application.

Article History

Received	16	Jan.	2024
Accepted	08	Mar.	2024
Available online	12	Mar.	2024

Abstract:

Passwords serve as a vital means to safeguard our digital accounts. Many individuals resort to conventional methods like writing down passwords on paper or storing them on cloud services, often overlooking security risks, forgetting, and divulging is the most notable, which leads to loss of access to accounts, or potential breaches. In this paper, we propose the development of an Android application named "BANK OF PASSWORD" to address this issue. Our work focuses on creating a lightweight app equipped with essential functionalities desired by users, including password addition, updating, copying, searching, and deletion. To ensure the security of stored passwords, our approach incorporates various protective measures, such as access restriction through a login process and the utilization of SHA256 hashing and AES256 encryption for password encryption, where stored passwords are securely encrypted and stored as ciphertexts within an SQLite database. A fingerprint authentication was implemented as a second login method. Extensive testing of the application demonstrates the successful functioning of all proposed features and requirements on devices running API level 26 or above.

Keywords: Secure warehouse for passwords, Passwords vault, Passwords keeper app, Android application.

بنك كلمات المرور: مدير كلمات مرور آمن لنظام *Android* تم تنفيذه بناءً على متطلبات محددة

حسين عبدالخالق صالح*

المديرية العامة للتربية في ذي قار، ذي قار، العراق

hussein.abd.alkhaliq@gmail.com

الخلاصة:

تعتبر كلمات المرور وسيلة أساسية لحماية حساباتنا الرقمية. يلجأ العديد من الأفراد إلى الطرق التقليدية مثل كتابة كلمات المرور على الورق أو تخزينها على الخدمات السحابية، وغالبًا ما يغفلون عن مخاطر الأمان، او النسيان، أو الكشف، وهو الأمر الأكثر بروزًا، والذي يؤدي إلى فقدان الوصول إلى الحسابات، أو خروقات محتملة. في هذه الورقة، نقترح تطوير تطبيق *Android* يسمى "بنك كلمات المرور" لمعالجة هذه المشكلة. يركز عملنا على إنشاء تطبيق خفيف يوفر الوظائف الأساسية التي يرغب بها المستخدمون، بما في ذلك إضافة كلمة المرور وتحديثها ونسخها والبحث عنها وحذفها. لضمان أمان كلمات المرور المخزنة، يتضمن نهجنا إجراءات حماية متنوعة، مثل تقييد الوصول من خلال عملية تسجيل الدخول، واستخدام تجزئة *SHA256* وتشفير *AES256* لتشفير كلمة المرور، حيث يتم تشفير كلمات المرور المخزنة بشكل آمن وتخزينها كنصوص مشفرة داخل قاعدة بيانات *SQLite*. كذلك، تم تنفيذ مصادقة بصمة الأصبع كوسيلة اخرى لتسجيل الدخول. يوضح الاختبار المكثف للتطبيق التشغيل الناجح لجميع الميزات والمتطلبات المقترحة على الأجهزة التي تعمل بنظام *API level* *26* أو إصدار أحدث من نظام Android.

الكلمات المفتاحية: مستودع آمن لكلمات المرور، خزانة كلمات المرور، تطبيق حفظ كلمات المرور، تطبيق Android.

1. INTRODUCTION:

In today's digital landscape, individuals possess multiple private accounts, including significant ones such as banking, social networks, websites, emails, and even Wi-Fi networks. These accounts are typically protected by private passwords, known only to the account holder, serving as a barrier against unauthorized access. However, the management of these passwords has become a significant challenge, leading to a pressing need for a secure and efficient solution. This paper is motivated by the desire to address this issue and enhance the security and convenience of password management for Android device users. It is worth noting that many users often tend to reuse the same password across multiple accounts for the sake of convenience. This practice poses a significant security risk, as a compromise of one password can lead to unauthorized access to several accounts **[1]**. Therefore, it is highly recommended to utilize unique and complex passwords for each user account **[2]**.

Managing multiple passwords can become a challenging task, potentially resulting in users forgetting their passwords and losing access to their accounts. Some individuals resort to writing down details of their passwords on paper, or storing them in a text file in cloud services

like Google Drive, or OneDrive, attempting to keep them secure. However, these traditional methods are still susceptible to security threats, including loss, damage, or unauthorized access to sensitive information.

Motivated by the need to provide a secure and user-friendly solution to these challenges, we propose developing a mobile application that provides a secure storage environment for user passwords. Given that most users employ devices powered by Android OS, which holds a significant market share of 67.72% [3], the proposed application will be tailored specifically for Android OS devices. The 'Bank of Passwords' app aims to provide a secure solution for Android users to manage their passwords, where, focuses on simplicity, usability, offline, free-cost, open-source, and available for most Android devices while still implementing robust security features. To ensure a satisfactory user experience, the application should offer convenient password management functionalities while prioritizing security. This necessitates a thorough determination of the requirements before the implementation process .

This paper aims to outline the application's design and implementation processes, demonstrating how it addresses the challenges of password management while enhancing user security. The paper is organized as follows: Section 2 covers user requirements and technical requirements for the app, Section 3 provides a brief implementation plan, Section 4 addresses the UI design processes, Section 5 covers the implementation of background processes, Section 6 is dedicated to test the application, and Section 7 concludes the paper.

2. CORE REQUIREMENTS:

For developing useful and usable applications, we have to understand user needs; and the techniques for implementing them, which means there are user requirements and technical requirements [4].

2.1 User Requirements: User requirements can be written from the perspective of what the users need to satisfy them [4]. In our approach, we will adopt the following requirements:

- The application offers users two login methods, either by entering a password or by using their fingerprint if supported.
- The user can assign a login password when the application launches for the first time for security purposes.
- The application forces the user to login every time he runs the application.
- For every new password to be added, the app must store a name and value.
- The application forces the user to use a unique name for every new password to be stored.
- The application can display all stored passwords as a scrollable list.

- The user should be able to copy any password value stored inside the application and paste it where needed.
- The application provides the user with the ability to edit any stored password easily (name or value or both).
- The application allows the user to delete any password stored inside the application.
- The user can search for any password stored inside the app.
- The application allows the user to change the login password.
- The user can erase (reset) all stored data inside the application with one click.
- The user can exit the application.
- The login-password length should be a minimum of eight characters.
- There is no need to get any access permissions from the user.
- The language of this application is English.
- The application Works offline.

2.2 Technical Requirements: To meet users' requirements in Sec. 2.1, and make it feasible, we need to define some technical requirements for describing how the application will be implemented practically **[4]**, plus illustrate how the background processes would be performed in this application. For such an application, our approach adopts the following technical requirements:

- For development, a reliable IDE should be used.
- A recommended Android API level must be used for development.
- The programming language that is used for development is preferred to be a popular and easy option for beginners and hobbyists.
- For storing the data of the user passwords, the application should use an efficient storage technic.
- The application must check and confirm that the new stored password has a unique name.
- The stored data (especially password values) must be secured by encrypting the plaintext of all stored passwords.
- The application must have the ability to decrypt all encrypted passwords to their plaintext again when needed to display them in the GUI.
- To assign a password for login, such an app must have the ability to store this kind of password and retrieve it when compared with the entered value in the login process.
- When a login password is assigned, the application must force the user to write this password twice in two different fields for more confirmation.

- For security level increase, the login password should be encrypted and stored as a ciphertext.
- The ciphertext of the login password should be used as a key for encrypting all stored passwords.

3. IMPLEMENTATION PLAN:

Before proceeding with designing and programming, it's essential to prepare an appropriate IDE, where we intend to use Android Studio because it's the official IDE for developing Android applications [5]. According to a Google announcement on 19 November 2020, Google Play Console will require all new apps to target API level 30 (Android 11) or above as reported by Android Developers [6].

For that reason, our app will be developed with API level 32 (Android 11) as a target, and API level 26 as minimum. For programming, there are many languages for developing Android apps such as Java, Kotlin, and C++. We plan to develop this app with Java, to produce a small apk file size [7], and as being the better option from the beginner's point of view [8]. Based on the common architectural principles mentioned by Android developers for best application development practices, each application should have at least two layers:

1. a UI layer.
2. a data layer.

The UI layer is accountable for presenting the application data on the screen, while the data layer encompasses the business logic, consisting of rules that govern the creation, storage, and manipulation of data within the application [8]. These processes occur independently of the user in the backend. Drawing from the aforementioned information, we can divide the implementation approach into two essential components:

1. UI Design.
2. Back-end Processes Implementation.

As we transition from the theoretical framework and requirements, it becomes pivotal to visualize the operational flow of the application to grasp its comprehensive functionality and user interaction dynamics, as described in **Figure 1**. This diagram will serve as a preliminary roadmap in the next phases, guiding the development process and ensuring that all user requirements are met effectively and efficiently.

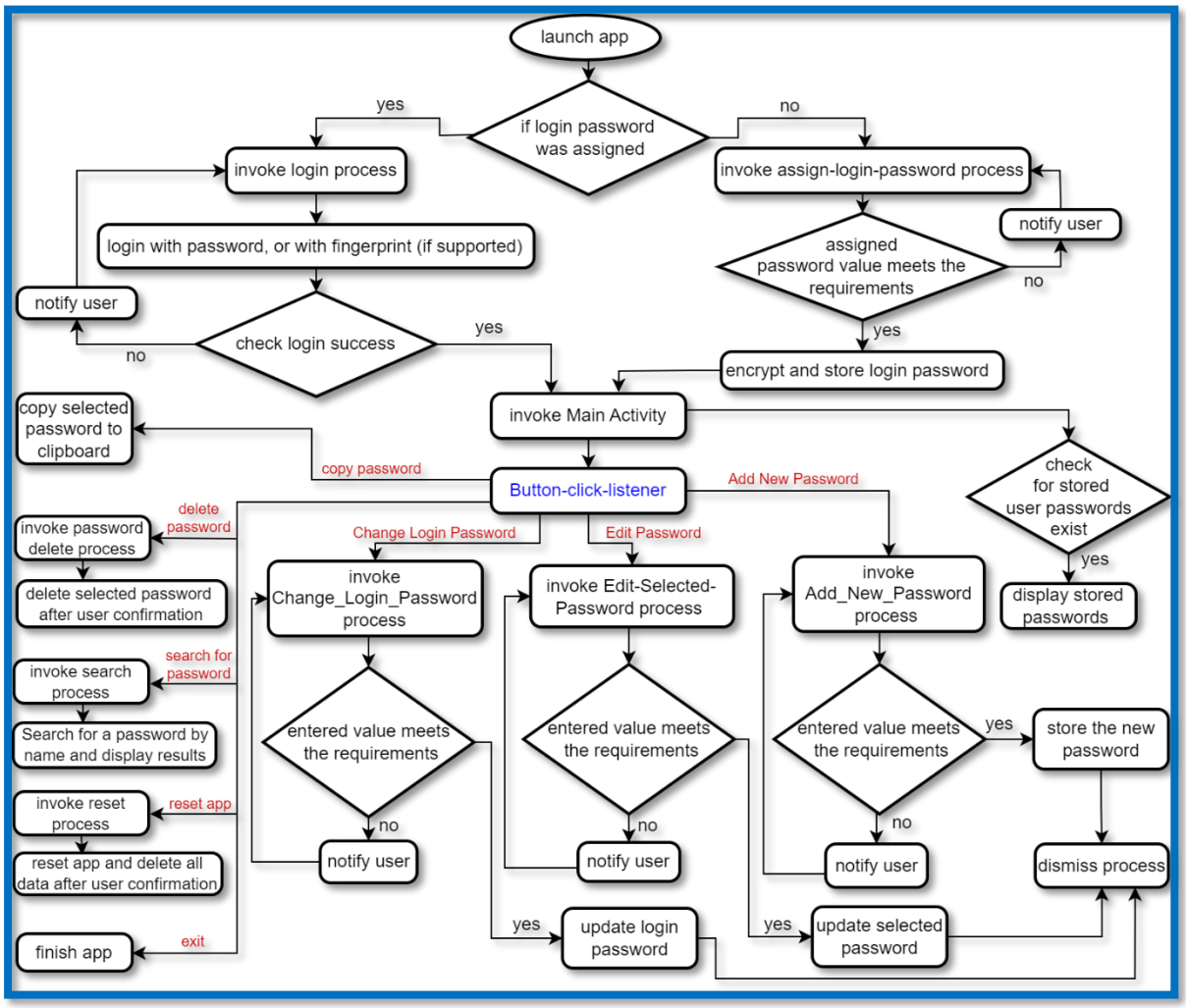

FIGURE 1: A Flowchart depicts the core functions and their processes in the 'Bank of Password' app.

4. UI DESIGN:

In this context, it is essential to gain a graphical design perspective to better comprehend the user requirements outlined in Section 2.1. This understanding allows us to visualize and create a feasible graphical user interface (GUI). Before embarking on the design phase, it is crucial to identify the primary tasks that the application will be carrying out based on the user requirements. Additionally, establishing the execution workflow, as depicted in **Figure 2**, helps to delineate the sequence of actions and interactions within the application.

4.1 Application's Main Tasks:

1. Assigning Login-password for the first time.
2. Login process.
3. Add a new password.
4. Display the stored passwords.
5. Delete password.

6. Copy password.
7. Update stored passwords.
8. Search for a password.
9. Change the login password.
10. Reset to default.
11. Exit.

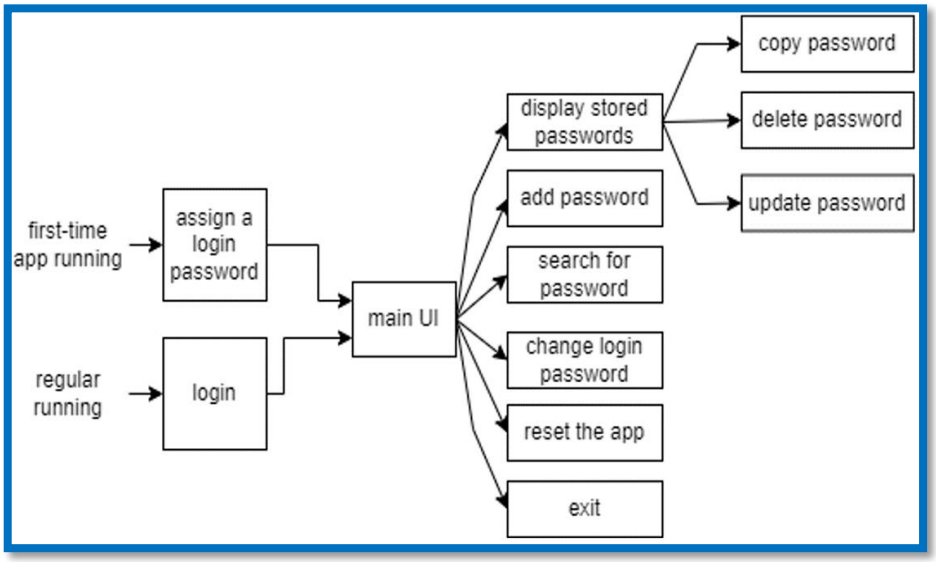

FIGURE 2: Diagram illustrating the workflow of UI functions execution.

Determining the main tasks and UI execution workflow provides a clear understanding and serves as a foundation for GUI building. These insights play a crucial role in shaping the background processes and programming logic required to implement all the application tasks. From a design perspective, each task can be visually represented on a single screen, and the most effective approach in Android development is to leverage the concept of activities. An Android activity represents a single screen that users see and interact with on their device **[9]**, serving as a window for the application to present its user interface.

Considering the nature of the tasks at hand, it becomes evident that certain tasks in our application necessitate dedicated activities, such as the login task and changing the login password. In contrast, others can be accommodated within the same activity, such as displaying the password list and performing a search. To address this situation, adopting a multiple-activity approach is recommended to ensure the appropriate graphical interface is built to cater to all tasks effectively.

4.2 Login-Password-Assigning Activity Design: Although this activity is not designated as the launch activity, it serves as the initial screen displayed to users after installing the application. The reason behind this design choice stems from the user requirements outlined in Section 2.1. Our application necessitates a login process to gain access, which, in turn, requires

a login password. Therefore, it is crucial to prompt the user to assign a login password before proceeding with any other actions within the application. The suggested layout components for this activity are depicted in **Figure 3**.

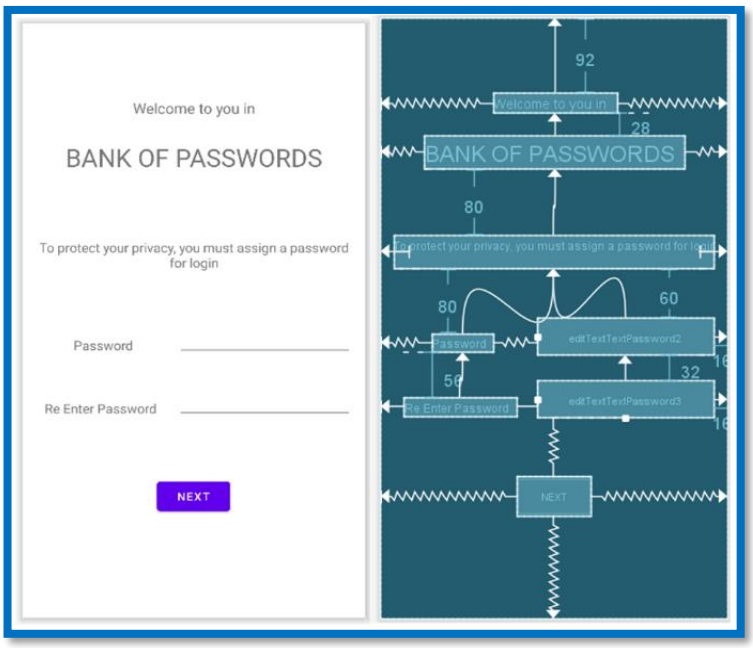

FIGURE 3: Layout design of the login-password-assigning activity.

(This layout consists of five TextViews for labeling, two EditText for getting login-password values, and a Button for executing.)

4.3 Login Activity Design: Once the user has been assigned a login password during the initial launch of the application, our app ensures user access verification in subsequent runs by initiating a login process. The user is granted entry only when the login password matches the correct value, as specified in the user requirements outlined in Section 2.1. Therefore, in regular app launches, it is imperative to display this activity as the first screen. The layout components for this activity are visually depicted in **Figure 4**.

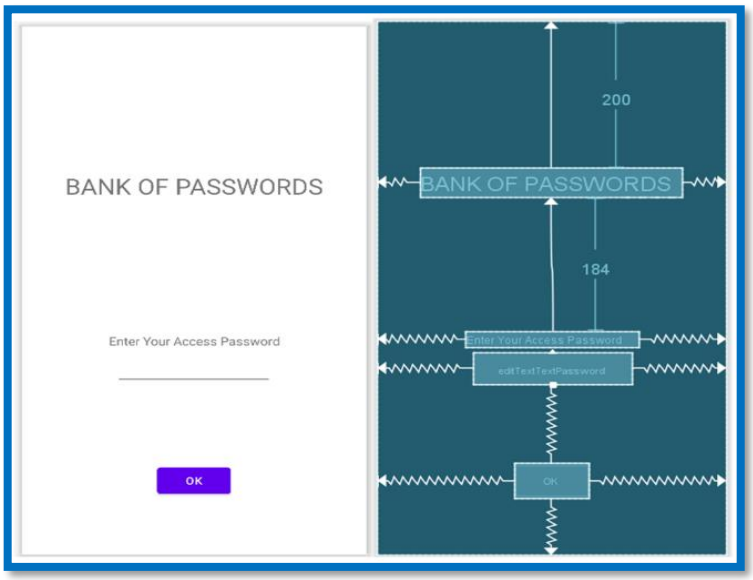

FIGURE 4: Layout design of the Login Activity

(This layout consists of five TextViews for labeling, two EditText for getting the values, on a Button for executing.)

4.4 Main Activity Design: The main activity is the central hub where the primary operations of the app take place, aligning with the execution workflow illustrated in **Figure 2**. The final UI layout design for this activity can be observed in **Figure 7**. The layout of this activity incorporates the following components:

- A constraint layout is utilized as the container for organizing all the elements.
- A TextView is positioned in the top left corner, dedicated to displaying the application name.
- An ImageView, clickable and located in the top right corner, features a settings icon. When clicked, it triggers the emergence of a Popup Menu, as demonstrated in **Figure 5**.
- A RecyclerView is situated in the middle of the screen, serving as a list to showcase the stored passwords. The proper functioning of this element requires the creation of a ViewHolder and an Adapter. The ViewHolder allows for customization of the appearance and behavior of each list element (list rows), while the Adapter is responsible for associating the data with the ViewHolder views. In our context, we recommend designing a custom ViewHolder layout that presents the information of each stored password (including the name and value), along with the relevant operations (update, copy, delete) for each list element, as depicted in **Figure 6**.
- A FloatingButton is included to facilitate the addition of new passwords.
- At the bottom of the screen, a SearchView is provided to enable password searching functionality.

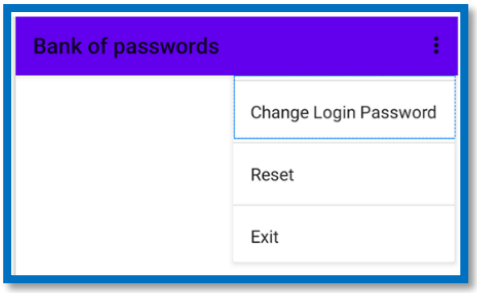

FIGURE 5: Layout design of the Setting Menu and its items.

(This layout consists of three items: change the login password, reset the application, and exit.)

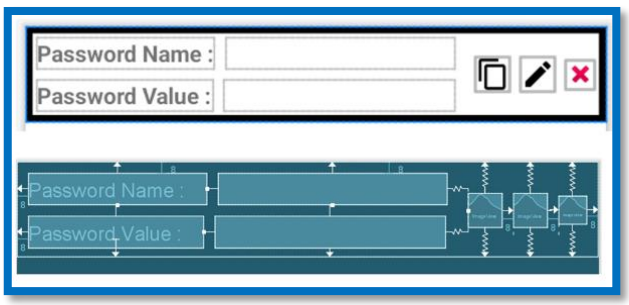

FIGURE 6: Layout design of the ViewHolder

(This layout contains: a constraint layout to hold all components; two TextViews for labeling; two TextViews for displaying password info; three clickable ImageViews for operations (copy, edit, delete)

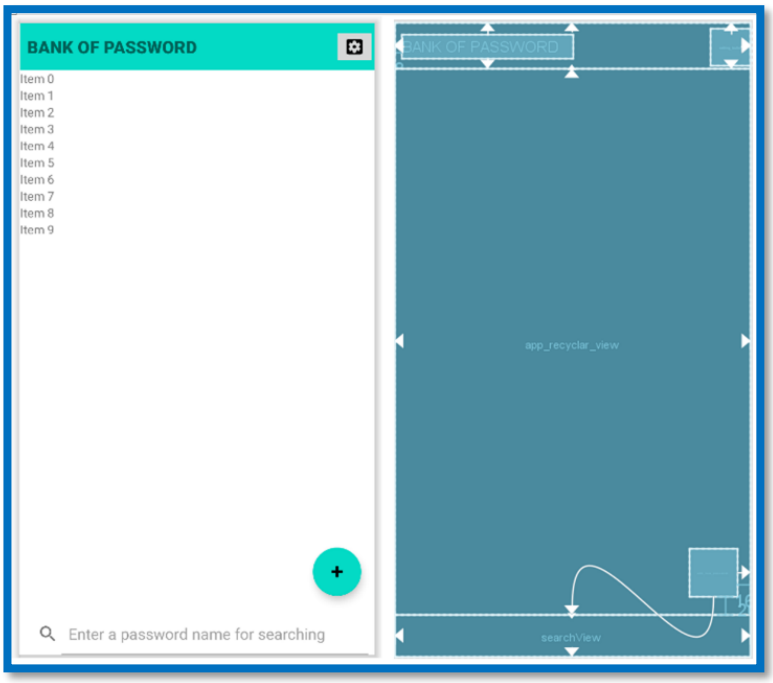

FIGURE 7. - Final layout design of the Main Activity.

4.5 Store-New-Password Dialog Design: When the user clicks on the floating button '+' in the main activity, depicted in **Figure 7**, a dialog containing a custom layout will be displayed. This dialog enables the user to add a new password. As per the user requirements mentioned in Section 2.1, each stored password should include two essential pieces of information: the name and the value. The layout design for this dialog is illustrated in **Figure 8**.

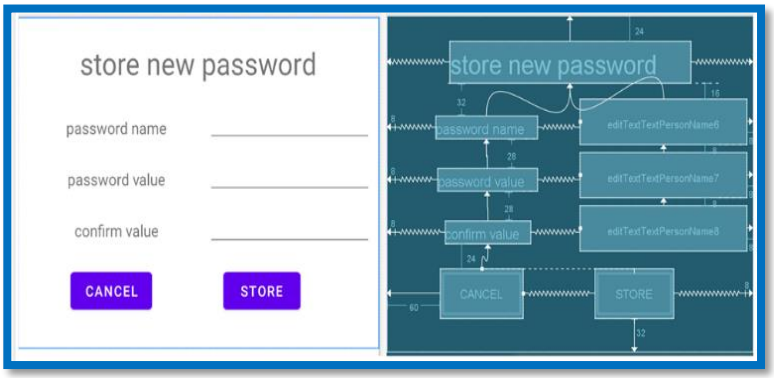

FIGURE 8: Layout design of the Store-New-Password Dialog

(This layout consists of a constraint layout for holding all components; four TextViews for labeling; three EditTexts to get the values; two buttons for actions (one for cancel and the other for adding)

4.6 Edit-Selected-Password Dialog Design: As depicted in **Figure 6**, the application provides the user with the capability to update the stored information (name or value) of any password listed by selecting the corresponding update button (pen icon) associated with that specific password. To accomplish this process, we need to design a suitable dialog that allows the user to enter the new values. The design of this dialog is presented in **Figure 9**.

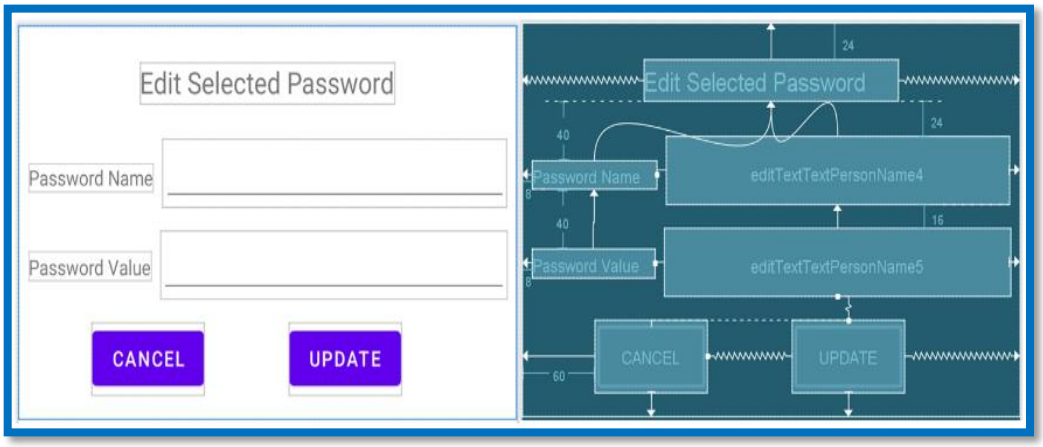

FIGURE 9: Layout design of the Edit-Selected-Password Dialog.

(This layout consists of a constant layout for containing all components; three TextView for labeling; two EditText to display the current values and edit them at the same time; two buttons one for cancellation, and the other for updating.)

4.7 Change-Login-Password Dialog Design: To enable the user to change the login password, a dialog should appear on the screen when the user selects the corresponding item in the setting menu (labeled as 'change login password' as shown in **Figure 5**). The design details of this dialog can be found in **Figure 10**.

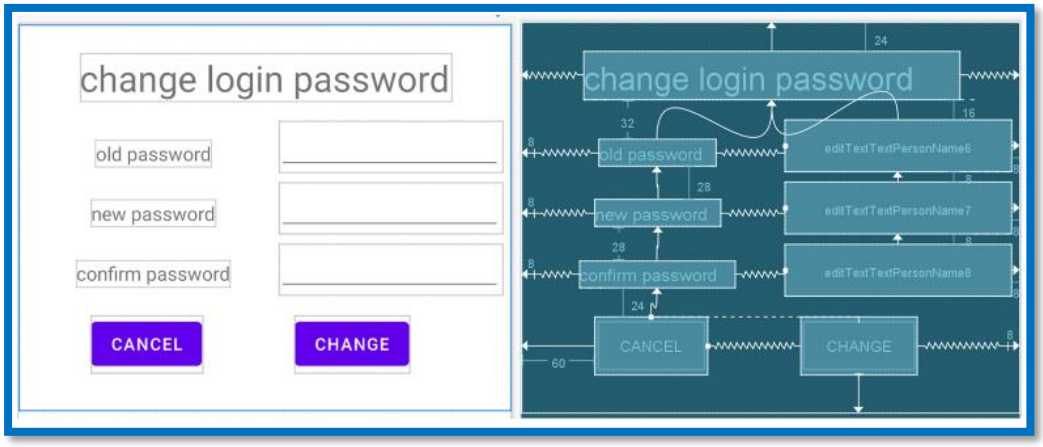

FIGURE 10: Layout design of the Change-Login-Password dialog.

(This dialog layout consists of a constant layout for containing all components; four TextView for labeling; three EditText to get the values; two buttons one for cancellation, and the other to change.)

4.8 The Rest of The Tasks: Upon revisiting the user requirements (in Sec.2.1), it becomes evident that the remaining tasks in our app (copying a password, deleting a password, resetting) do not necessitate the use of activities to fulfill their functions. These tasks no longer require data input or output. Instead, they may involve confirming or notifying dialogs (such as AlertDialog or Toast) to inform the user about their actions. The handling of these processes will be discussed in detail during the implementation of the back-end processes.

5. BACKGROUND PROCESSES IMPLEMENTATION:

Once the UI designing phase is finished, it is essential to establish the necessary connection between all application UI components and the corresponding programming logic to ensure their functionality aligns with user and technical requirements. To create a feasible roadmap for implementing the back-end processes, we propose defining a sequential flow for the execution of the application UI, starting from launch until exit. This planned sequence will serve as a guide for the implementation phase, as illustrated in **Figure 11**.

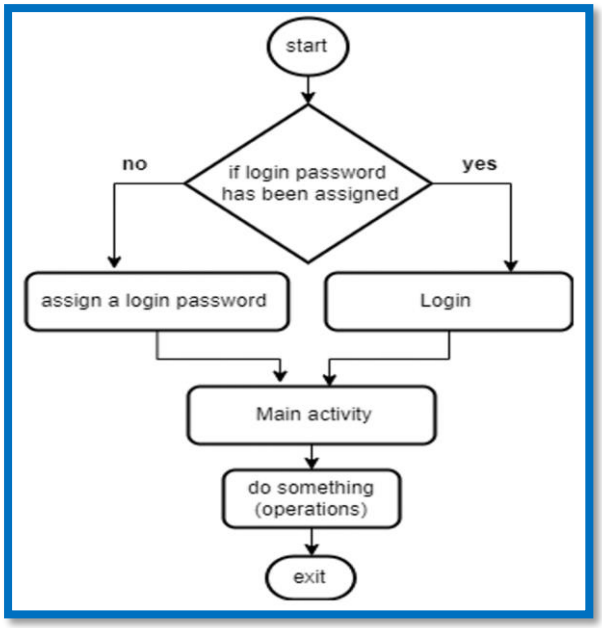

FIGURE 11: Sequential flow of application UI execution processes.

By examining the processing sequence depicted in **Figure 11**, we can deduce that the initial logic in our application involves verifying the state of the login password to determine the appropriate activity to launch. This situation encompasses two possible states: either the login password has been assigned or it hasn't. To accomplish this straightforwardly, we can create an empty activity (referred to as the Launch activity in Section 5.1) that solely performs the check to determine the true case, and subsequently deploys the corresponding activity accordingly.

5.1 Launch Activity: The Launch Activity serves as an empty activity running in a background thread. Its primary purpose is to check whether a login password has been previously saved or not, without the need for displaying any graphical user interface (GUI). This involves retrieving the stored login password, which requires determining the appropriate method of storing it within the application. In our case, the recommended approach is utilizing the SharedPreferences APIs, as suggested by the Android developers' website, as it is suitable for storing a small collection of data within our app **[10]**. Assuming the SharedPreferences APIs are employed, the role of this activity is to search the SharedPreferences for a previously stored

login password, allowing it to make the necessary decision, as illustrated in **Figure 12**. Consequently, this activity is designated as the default launch activity in our application, requiring modifications to be made to the manifest file.

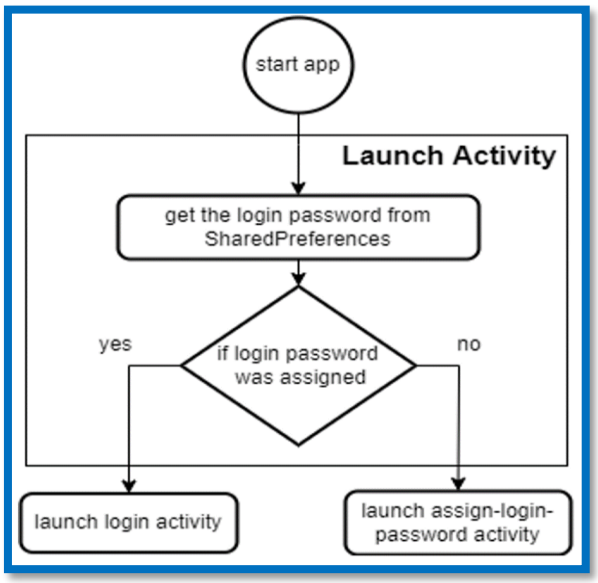

FIGURE 12: Flowchart illustrating the logic of the Launch Activity.

5.2 Login-Password Assigning Process: When the Login-password-assigning activity (in Sec. 6.2) is triggered (as depicted in **Figure 3**), the user is prompted to enter his password and proceed by tapping the 'NEXT' button. At this point, in accordance with the technical requirements (detailed in Sec. 2.2), the subsequent procedures should be sequentially executed in a background thread of the invoked activity:

1. Get password value: where two fields are holding the passwords that must be obtained (One for the password, and the other for confirming as shown in **Figure 3**).
2. Verify password value: checking the identically and length of the two values, where the length is a minimum of eight characters according to the user requirements (see Sec. 2.1). If those conditions are true, then the application can take any one of the two values as a login password.
3. Encrypt password value: In general, one-way hashing is one of the best security options in cryptography, where there are a lot of functions in this field such as MD5 and SHA256 **[11]**. In Android, the MessageDigest class provides the functionality of a message digest algorithm. Based on some recommendations to improve the security of the login process **[12]**, we intend to use the SHA-256 Hash function to generate a digest from the login password.
4. Store the generated digest: The generated digest will be stored as a Hex string in the SharedPreferences, as described clearly in **Figure 13**.

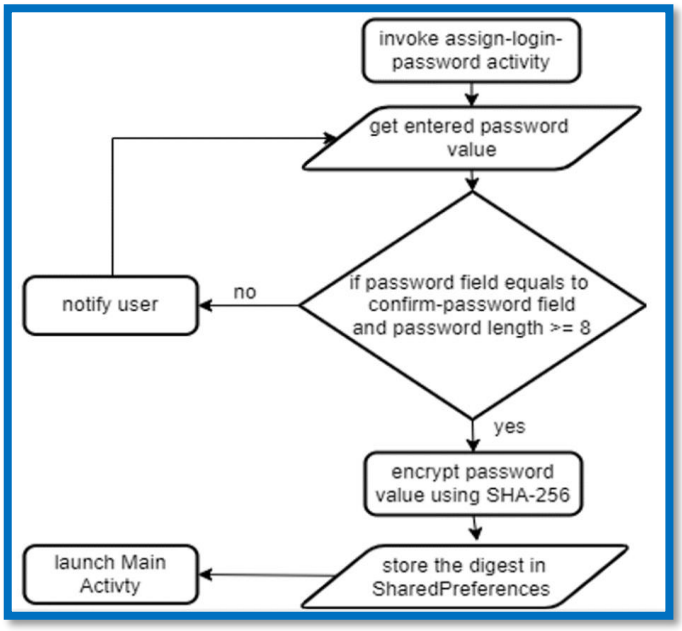

FIGURE 13: A flowchart illustrating the Login-password Assigning Process.

5.3 Login Process: Upon invocation of the login activity (as depicted in **Figure 3**), the user is prompted to enter his login password. Our objective in this stage is to generate a digest using SHA-256 from the entered value. This is achieved by converting the entered value to a Hex String as the initial step. Subsequently, the generated digest is compared with the stored digest. If the two digests are found to be identical, the login process is deemed successful, allowing the user to proceed to the main activity as outlined in the steps illustrated in **Figure 14**.

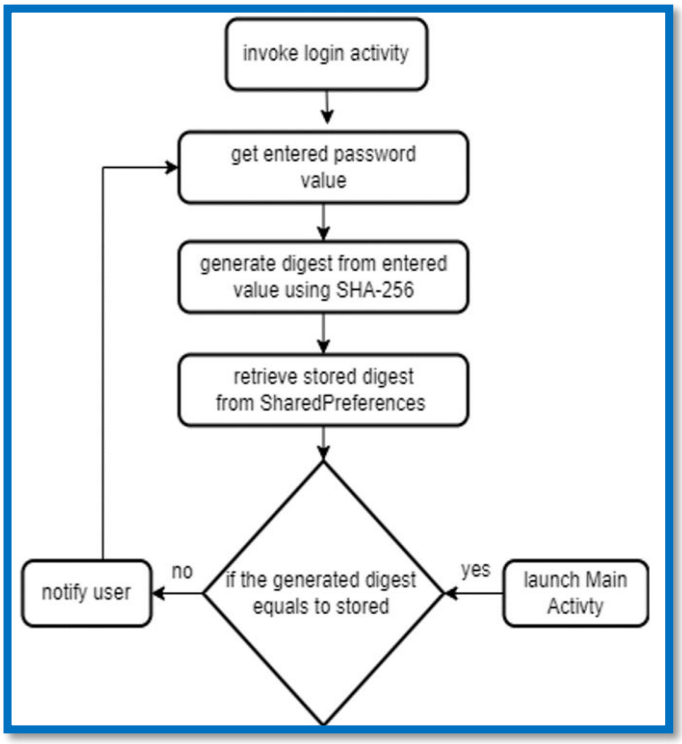

FIGURE 14: A Flowchart illustrating the login process.

5.4 Additional Login Method: In several cases, login using the traditional password method is not enough to provide the security level required to protect the data, because the login password is exposed to disclosure, which means we should employ other techniques and methods to increase the security level to protect user data (passwords). In this section, we will explore another technic that will be included with our app to enable login securely relying on user biometrics.

5.4.1 Login with fingerprint: Fingerprint technology in Android has become increasingly popular for authentication and authorization purposes, particularly in securing smartphones and protecting security-sensitive operations. It refers to biometric-based authentication and authorization using the fingerprint API. This API allows mobile apps to recognize and authenticate users based on fingerprints, providing better security for security-sensitive operations **[17]**.

To implement this feature in our app, first add the required dependencies, then get the necessary permissions:

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

To check if the user device can authenticate using fingerprint, android includes a class called `BiometricManager` which provides access to the biometric hardware and software on the device. If the checking process is complete with success (there is a sensor and stored fingerprint), it means that the device is ready for biometric authentication and waiting for the user to put his finger on the sensor. To execute the authentication task on a specific thread, we can use the 'Executor' class.

To handle the biometric authentication process, we can use the 'BiometricPrompt' class. To hold the information for the biometric prompt such as the title, the description, and the negative button text, we can use 'PromptInfo' class. Also, it is necessary to call 'onAuthenticationError' method for handling errors that occur during the authentication process, which is a useful process, especially for displaying the error message to the user. For checking the success or failure of authentication, the 'onAuthenticationSuccess' and "onAuthenticationFailed' methods are called to handle the two situations. Finally, If the fingerprint matches, the app will directly switch to the main activity, as explained clearly in **Figure 15**.

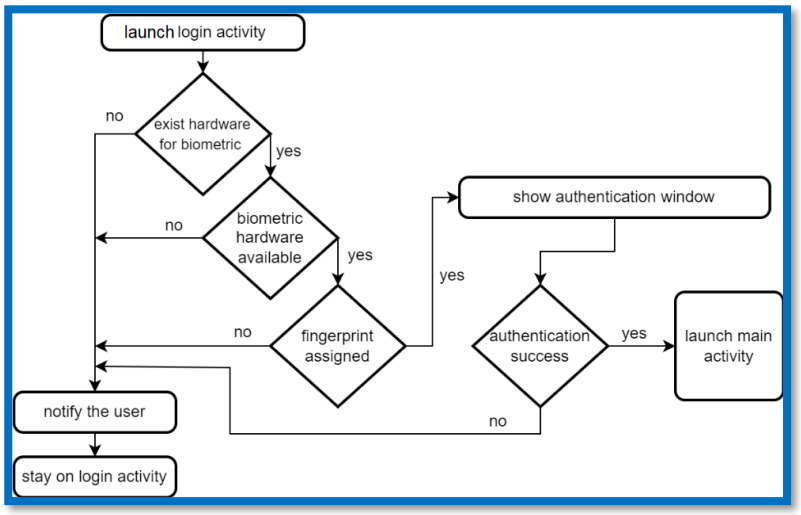

FIGURE 15: A flowchart describes the process of login using the fingerprint technique

5.5 Main Activity Processes: The main activity serves as the central component of our app, responsible for implementing the majority of its functions. The workflow within this activity starts by retrieving any stored passwords, if available, and preparing the RecyclerView to display them. To accomplish these tasks, it is crucial to establish an efficient storage mechanism for managing user passwords.

5.5.1 Passwords Storing Mechanism: To efficiently save data, the database concept is widely recognized, and in the context of Android OS, the SQLite database offers an ideal solution for managing repeated or structured data **[13]**. In our case, utilizing an SQLite database to store password information is highly suitable. Android treats the SQLite database as files stored in the device's internal storage. The necessary APIs for creating such a database can be found in the android.database.sqlite package. To construct the database, we can extend the SQLiteOpenHelper class and implement its methods. As stated in the user requirements (in Sec.2.1), each stored password consists of a name and a value. Therefore, a single table with two columns is sufficient for our needs, as depicted in **Figure 16**.

Once the implementation of the SQLite database is completed, the next step is to create an instance, either readable or writable, in the main activity. This instance will be responsible for building the database in a background thread to prevent any long-running operations. With the database successfully created, we gain the flexibility to perform various operations such as reading, inserting, deleting, or updating the database from any part of our application code.

Password_table	
Password Name *	TEXT
Password value	TEXT

FIGURE 16. – SQLite database schema for password storage

5.5.2 Storing New Passwords: When the user clicks on the '+' button as illustrated in **Figure 7**, the main activity should launch the corresponding dialog to enable the user to add new passwords as shown in **Figure 8**. Adhering to the technical requirements outlined in Section 2.2, the following procedures are performed in the background to facilitate the addition of new passwords:

1. Get and Confirm: First, get the entered name and check if other stored passwords have the same name by searching the database, then notify the user. Second, get the two entered passwords' values; and verify their identically, where this process is critical in case of users' mistype. If identical is approved, one of the values will be adopted.
2. Displaying: Add the new passwords to the RecyclerView.
3. Encrypting: For encrypting new passwords, we intend to use AES in CBC mode with 256-bit keys to provide a higher level of security and robustness against attacks. Using AES in CBC mode with a 256-bit key will ensure the data's confidentiality and integrity, making it suitable for applications requiring high-security levels **[14]**. The CBC mode has been modified to enhance the security of encrypted data by protecting against bit-flipping attacks and adding an integrity approach using the keyed-hash function **[15] [16]**. Also, using a 256-bit key in AES further strengthens the security of the encryption, making it more resistant to brute-force attacks and Grover's quantum search algorithm **[14]**. To meet technical requirements (in Sec. 2.2), the digest of the login password will be used for encrypting with this algorithm as a key, noting that the algorithm and the digest have the same length.
4. Storing: Store generated bytes from the encrypted password value with its corresponding name in the SQLite database, then dismiss the dialog (shown in **Figure 8**). Moreover, executing the encryption and storing processes in a separate thread would avoid hanging the UI thread. The sequence of the entire process is shown clearly in **Figure 17**.

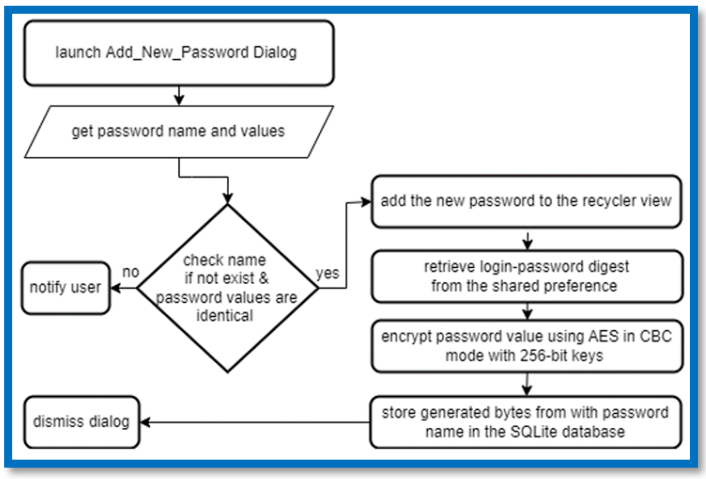

FIGURE 17: A flowchart describes the Storing-New-Passwords process

5.5.3 Displaying stored passwords: Upon launching the main activity, a checking process is initiated to verify the presence of any stored passwords within the SQLite database. If passwords are found, they are retrieved as a List containing password information such as names and values. Subsequently, the password values in the list are decrypted using a reversed encryption approach. Finally, the resulting list is utilized to populate the RecyclerView, ensuring the display of the stored passwords. The sequential workflow for this process is depicted in **Figure 18**.

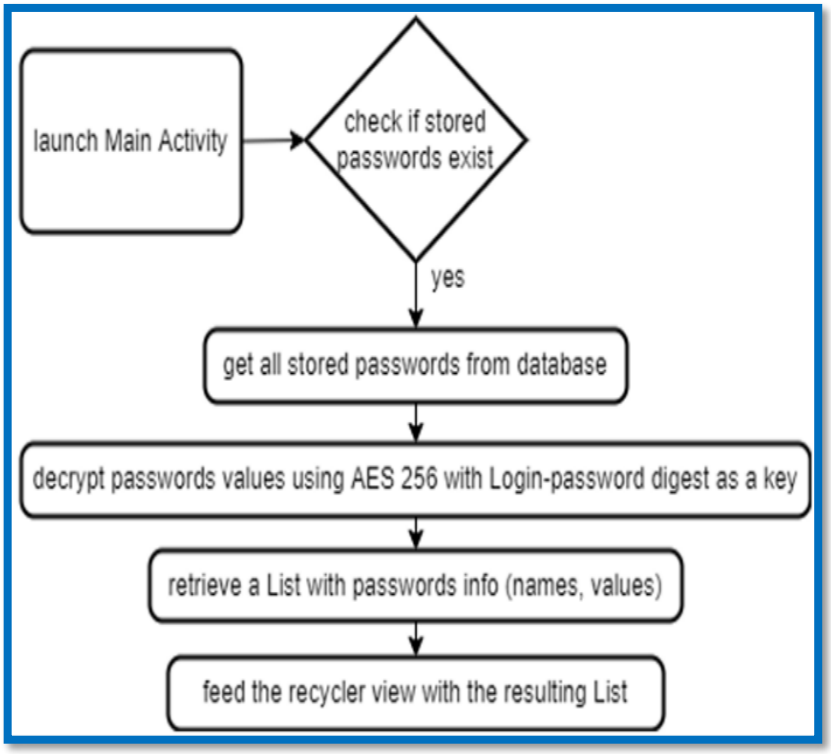

FIGURE 18A: flowchart depicts the execution process of displaying stored passwords.

Let us recall that each stored password will be accompanied by three buttons (icons) - delete, edit, and copy - as depicted in **Figure 6**. The implementation of those procedures should be incorporated within the RecyclerView adapter class, specifically in the onBindViewHolder method, as outlined below:

- Delete password: Remove the selected password from the database after receiving user confirmation, and then update the RecyclerView to reflect the changes.
- Edit password: Launch the Edit-Selected-Password Dialog (as depicted in **Figure 10**) to obtain new values from the user. Encrypt the password value and store it, along with its name, in the SQLite database. Finally, update the information of the respective password, as illustrated in **Figure 19**.
- Copy password: Copy the value of the selected password to the clipboard.

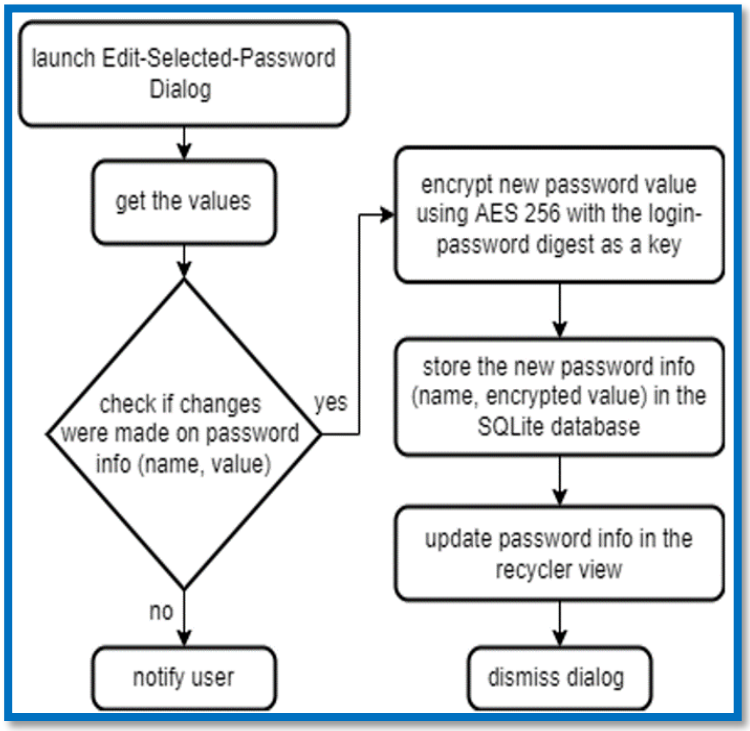

FIGURE 19: A flowchart outlining the Edit-Stored-Passwords process

5.5.4 Search for Passwords: To enable users to search for a stored password within the application, we have incorporated a SearchView widget. The SearchView located at the bottom of the main activity layout as depicted in **Figure 7**, allows users to enter a password name and view suggestions or results. When the user enters a password name in the search view, the RecyclerView is filtered based on the input text to display matching passwords, ensuring relevant results are displayed.

5.5.5 Setting Menu Items: As depicted in **Figure 5**, the setting menu consists of three items: "Change login password," "Reset," and "Exit." When the user selects any of these items, the corresponding functionality is executed in the background of the Main Activity. The implementation logic for each setting menu item is as follows:

- Change login password: To change the login password, the associated dialog (as shown in **Figure 10**) is displayed. The user must enter the valid old password along with the new password values, ensuring that the new password fields match and meet the required length conditions. The next step involves generating a new SHA256 digest from the new password value and replacing the old stored digest (in shared preference) with the new one. However, the process doesn't end here. After changing the login password and relaunching the application, the Main Activity (as shown in **Figure 18**) attempts to decrypt all password values (in the SQLite database) using the old digest as a key, resulting in incorrect plaintext. Therefore, it becomes necessary to retrieve and

re-encrypt all stored passwords with the new digest using the same AES 256 encryption approach, as described in detail in **Figure 20**.

- Reset: Resetting the app involves dropping the passwords table in the SQLite database, removing the stored login-password digest from shared preference, and returning the application to the Launch Activity, as demonstrated in **Figure 21**.
- Exit: This task entails finishing all activities and exiting the application.

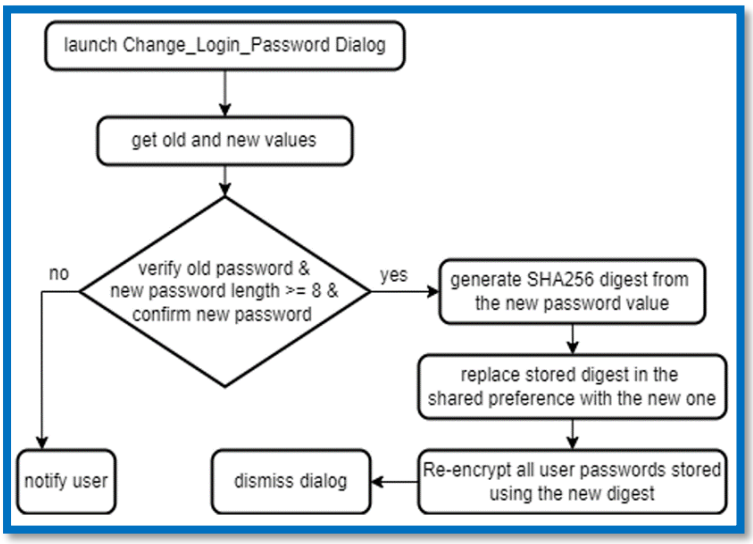

FIGURE 20: A Flowchart illustrating the Change-Login-Password process

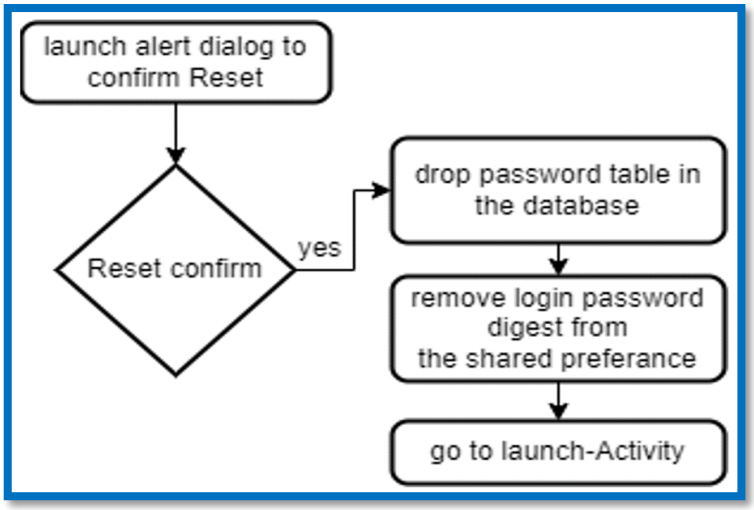

FIGURE 21: A flowchart describes how the reset process will be executed

6. TEST THE APP:

After generating the APK file, the app size was 3.91 MB. When testing the app on Android devices running with API levels 26 to 32, all the required functions worked correctly. The testing began with the first launch, where a login password needed to be assigned. Assuming the login password value was set to '12345678', the generated digest was stored directly in the SharedPreferences after encoding its bytes with Base64, resulting in the value

'73l8gRjwLftklgfdXT+MdiMEjJwGPVMsyVxe16iYpk8='. Also, The user can login using his fingerprint if it was assigned to the device, where this feature works perfectly on devices with Android 7.1 (Nougat1) and above.

During the testing of adding a new password, the password value was encrypted correctly using the stored digest as a key for the AES algorithm. For example, if a new password with the value 'pass1234' was added, the encrypted value to be stored after encoding its bytes with Base64 would be '6m0edekTY3MSbe+2OVhdaQ==='. When relaunching the app, the login procedure had to be followed by entering the correct login password. The app was then able to decrypt and display all the stored passwords in the SQLite database correctly.

In the testing phase, the login password was changed to 'computer'. The old digest was replaced directly with the new one, resulting in the value 'qpcwIVD86BFCXNhFNwKKWvvjfj8TYq1FpR1Gfhev3Jw='. The value of the previously added password "pass1234" was re-encrypted using the newly generated digest, resulting in the value 'jTf9UOE7BE/MIRpQt2oD/Q==='. However, when attempting to run the application on older devices with API level 25 or lower, it failed to launch due to the minSdk version used during development.

7. CONCLUSIONS:

As individuals, we often encounter difficulties in managing and remembering multiple passwords. The more passwords we have, the greater the risk of forgetting them and losing access to our accounts, which is a common scenario. To address this challenge, some people resort to traditional methods such as writing passwords on paper or storing them in cloud services like Google Drive. However, these approaches are flawed and can easily lead to unauthorized access by attackers or malicious individuals.

Instead of relying on traditional methods, developing an Android application specifically designed for users with Android OS devices can provide a convenient and portable solution. However, implementing such an application requires careful consideration of security aspects and usability. To ensure secure access, a login process must be enforced, requiring users to verify their identity with a password.

Storing passwords in their raw form within the device is not advisable as it poses the same risks as traditional methods. The passwords' values would be easily accessible in the database, making them vulnerable to inspection or cracking attempts. Therefore, it is crucial to store

passwords securely, and encryption is a reliable solution. Similarly, the login password should not be stored in plaintext. Hashing is an appropriate approach to make the login password unreadable.

Using the SHA256 algorithm for hashing the login password and AES for encrypting the user's stored passwords is recommended to increase the security level. Utilizing the generated digest from the login password as a key for AES encryption further strengthens the overall encryption process. By adopting these measures, users can benefit from a more secure and robust password management solution.

- **FUNDING:** None
- **ACKNOWLEDGEMENT:** None
- **CONFLICTS OF INTEREST:** The author declares no conflict of interest.

- **AVAILABILITY OF PROJECT AND CODES**
  - Our application project 'Bank of Passwords' is available on GitHub and can be accessed via: https://github.com/Hussein-abd-alkhaleq/Bank-of-Passwords
  - DOI of the project: 10.5281/zenodo.8017281

8. REFERENCES

[1] Gaw S, Felten EW. Password Management Strategies for Online Accounts. Proceedings of the Second Symposium on Usable Privacy and Security. 2006;44–55. https://doi.org/10.1145/1143120.1143127.

[2] Florêncio D, Herley C, van Oorschot PC. Password Portfolios and the Finite-Effort User: Sustainably Managing Large Numbers of Accounts. In: 23rd USENIX Security Symposium (USENIX Security 14). Aug. 2014;575–590. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/florencio.

[3] Mobile Operating System Market Share Worldwide. Statcounter GlobalStats, Sep. 03, 2023. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide . [Accessed: Sep. 03, 2023].

[4] Kujala S, Kauppinen M, Rekola S. Bridging the gap between user needs and user requirements. In: Advances in Human-Computer Interaction I (Proceedings of the Panhellenic Conference with International Participation in Human-Computer Interaction PC-HCI 2001). Typorama Publications, 2001;45–50.

[5] Wihidayat ES. Pengembangan Aplikasi Android Menggunakan Integrated Development Environment (Ide) App Inventor-2. 2017.

[6] New Android App Bundle and target API level requirements in 2021. Android Developers, Nov. 19, 2020. Available: https://android-developers.googleblog.com/2020/11/new-android-app-bundle-and-target-api.html . [Accessed: Sep. 04, 2023].

[7] Putranto BP, Saptoto R, Jakaria OC, Andriyani W. A Comparative Study of Java and Kotlin for Android Mobile Application Development. 2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI). 2020;383–388.

[8] Bose S, Mukherjee M, Kundu A, Banerjee M. A comparative study: java vs kotlin programming in Android application development. Int J Adv Res Comput Sci. 2018;9(3):41–45.

[9] Gargenta M. Main building blocks. In: Learning Android. Sebastopol, California, O'Reilly Media, Inc., 2011;28–29.

[10] Iamnitchi A, Ripeanu M, Santos-Neto E, Foster IT. The Small World of File Sharing. IEEE Transactions on Parallel and Distributed Systems. 2011;22:1120–1134.

[11] Rachmawati D, Tarigan J, Ginting A. A comparative study of Message Digest 5 (MD5) and SHA256 algorithm. Journal of Physics: Conference Series. 2018;978:012116.

[12] Ebanesar T, Suganthi G. Improving Login Process by Salted Hashing Password Using SHA-256 Algorithm in Web Applications. International Journal of Computer Sciences and Engineering. 2019.

[13] Liu H, Yang L, Wu H. Design of Embedded Data Acquisition and Management System Based on SQLite Database. 2022 11th International Conference of Information and Communication Technology (ICTech). 2022;335–338.

[14] Alslman YS, Ahmad A, AbuHour Y. Enhanced and authenticated cipher block chaining mode. Bulletin of Electrical Engineering and Informatics. 2023.

[15] Wade S. Description of Image Encryption Using AES-256 bits. International Journal for Research in Applied Science and Engineering Technology. 2023.

[16] Nugrahantoro A, Fadlil A, Riadi I. Optimasi Keamanan Informasi Menggunakan Algoritma Advanced Encryption Standard (AES) Mode Chiper Block Chaining (CBC). Jurnal Ilmiah FIFO. 2020.

[17] ElMouatez B, Karbab M, Mourad D, Debbabi A, Abdelouahid Derhab D, Djedjiga Mouheb. Fingerprinting Android Malware Packages. 2021. doi: 10.1007/978-3-030-74664-3_3.