# Secuure Android Documentation

## Introduction

Secuure is a password manager application which aims to keep track of all your passwords. This application is simple and user friendly. It allows a user to save information in a safe and encrypted way and have that information on-the-go on a mobile device. After you create an account, all you need to have is a username and remember a single master password to access your information from our database.

## Usage

Download APK, so you are able to download this app on your Android device. Make sure that your phone's driver is installed on your computer, so you can download the app directly to your device.

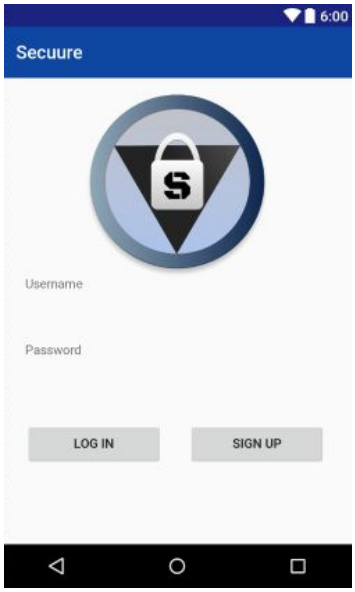
When you click on the application icon, you are brought to the main login page. If you have an account you can just login with your username and master password. If you do not have an account, you click the Sign Up button which will take you to a new page to create an account. After filling out the necessary information such as first and last name, username, password and confirming your password, you click the Sign Up button which now creates your account. A popup message will appear if you were successfully able to create an account and will take you back to the main login page.

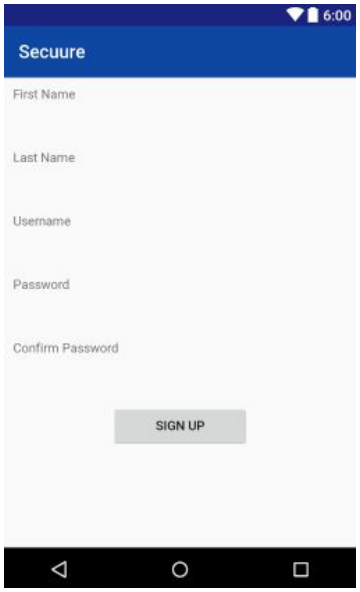
When you log in correctly, you will go to the user's page. On this page you will find two buttons a Log Out and Plus button. The Plus button will take you to another page which you can add an account and the information going with it such as title, login, password, and additional notes. On this page you will notice a few buttons: one to add the account, one to see your password, and a password generator.

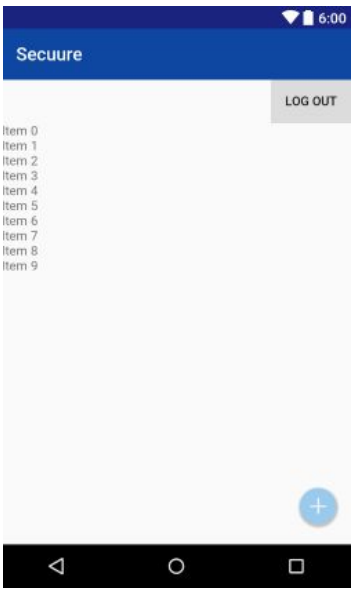
When you click on the password generator button it will take you to another page. You can choose what you want to include such as capital letters, numbers, or symbols and vary the length of the password with the seekbar. The Refresh button generates a password. If you want to keep the password press the Okay button. If you do not want to create a password, hit the Cancel button. If you want a different random password, then you click the Refresh button again.

After adding an account to your list it will update on your user page. You can view your account information by clicking on it. You can edit your account or delete it when you view your information. If you hit the Edit button you can edit the password or the additional notes. To change the other information of the account in the list, you must delete it and create a new one. If you change or delete anything it will update on your user page. Lastly, clicking the Log Out button on the user page opens a popup window asking if the user wants to log out or not. If yes is clicked then it will go back to the main login page. If no is clicked then it will close the popup.

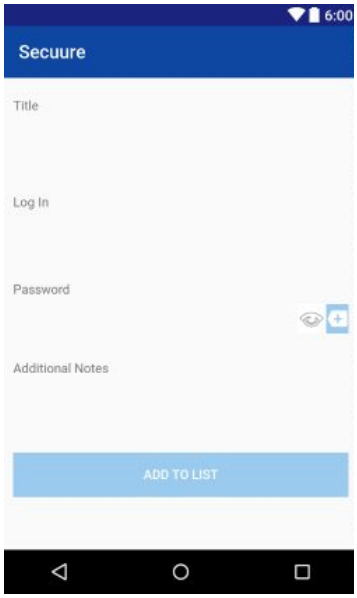
Main Login Page: Register Page: User Page:

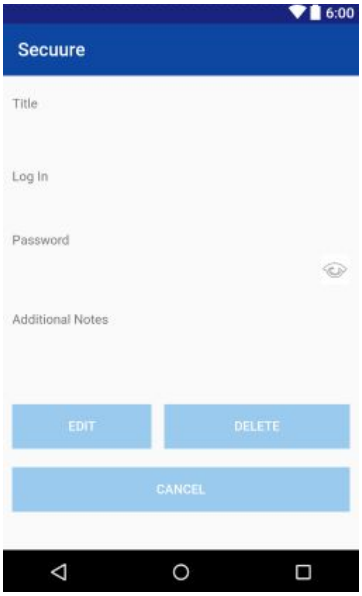


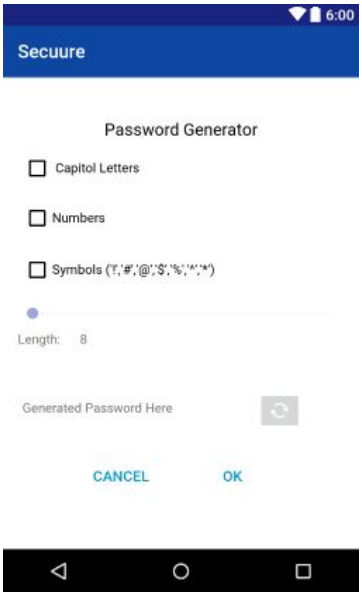




Adding Account Page: Viewing Account Page: Password Generator Page:




**Documentation**

Divided the various screens the user interacts with into different activities. The general layout for the activities were designed inside .xml files. To initialize and set up the components from those .xml files was done with the corresponding java files. With creating a new activity, we had to make sure that it was declared in the manifest file. This allowed the activity to be accessible to the system. We declared an activity by inserting the name of the activity inside "<>" under the <application> element. The manifest file also was where we had to declare permissions to access information about networks and Wi-Fi and turn on network sockets (the

internet). These were needed because our application had to be connected online to access our database (we did not implement an offline version).

MainActivity.java initialized the components from its corresponding activity file. This dealt with the main login page the user sees when opening the application. It basically calls on certain functions from different resource files such as GsonStringConverterFactory.java, User.java, and WebService.java. There was a function to log the user in. This checked if the user information typed in the fields were tied in with a valid user account found on our database. The last function checked to see if the user logged in successfully and if so switched to the next activity (Login) and displayed a successful message. If it was invalid a popup window would be displayed as a means of informing the user that it was invalid and have them try again.

Login.java initialized the components from its corresponding activity file (this was the page after logging in; called User Page in the views of the various pages above). It called on functions from some resource files checking the connection to the web server and performing a call to the server by requesting the accounts the user previously stored. There was a function which takes in a Json response from the server and populate the current table. Basically, this updated the current table shown on the page.

AddingAccounts.java initialized the components from its corresponding activity file. This had taken the information typed in for adding a password account to the database. Your list of accounts on the User Page will be updated as well.

PasswordGenerator.java initialized the components from its corresponding activity file. This basically generated a random password based on user inputs. It was broken into two functions. One function takes in flags signally if the user had checked one or more of the boxes of what to include in a password and how long. This function created a random string of characters. The second function called on that function and displayed it as a text to view on the page.

EditingAccounts.java initialized the components from its corresponding activity file. This dealt with editing the information of a password account that existed on the database. We defined that editing can only be done on either changing the password or the additional notes fields. This page also showed the information that can be viewed on an existing password account. There was a delete function which deletes the account being edited from the database. The User Page will be updated with any changes made.

Register.java initialized the components from its corresponding activity file. This saved a new user account to the database. If the username was not taken and the passwords match then it registers the new user on the server. If it was successful, we displayed a popup window saying it was successful and a message to go back to the Main Login Page.

However, not all files created were user pages, but also resource files which were used by various activities. This was done as a way of organizing the various files. Account.java created variables and function used for what account is going to be stored on the database. AccountAdapter.java helped with recycling the rows of a the password account table on the User

Page. Global.java defined global variables and functions that were going to be used in various pages. GsonStringConverterFactory.java converted strings (normal text) into Json. It also helped with preventing double quotes. User.java dealt with creating a new user account and their information. UserTable.java was used as a means to store data in a user table, but then instead used mySQL. WebInterface.java held the API calls from the server. WebService.java dealt with making sure you were connected to our online database.