**Manual Unit Tests**

1807132

Contents

[*SignUp.java* 2](#_Toc58843394)

[*Login.java* 3](#_Toc58843395)

[*MainActivity.java* 4](#_Toc58843396)

[*Library.java* 5](#_Toc58843397)

[*Profile.java* 7](#_Toc58843398)

# *SignUp.java*

Testing Email EditText:

* **Test purpose**: To check if EditText mEmail can handle unexpected inputs provided by the user
* **Test actions**: Enter text in mEmail EditText, click the Button register.
* **Expected response**: The text entered should be not null and in [example@example.com](mailto:example@example.com) format.
* **Test passed**? **Yes**
* **Actual Results**: Notifies if email entered is null, notifies if email entered is valid

Testing username EditText:

* **Test purpose**: To check if EditText mUsername can handle empty value provided by the user.
* **Test actions**: Enter text in mUsername EditText, click the Button register.
* **Expected response**: The text entered should not be null.
* **Test passed**? **Yes**
* **Actual Results**: Notifies user if he has not entered username

Testing password EditText:

* **Test purpose**: To check if EditText mPassword can handle empty value provided by the user.
* **Test actions**: Enter text in mPassword EditText, click the Button register.
* **Expected response**: The text entered should not be null.
* **Test passed**? **Yes**
* **Actual Results**: Notifies user if he has not entered password.

Testing Register Button:

* **Test purpose**: To check if upon clicking the Register Button relevant data is written to sharedPreferences
* **Test actions**: To click on Register Button after data in the required fields has been entered.
* **Expected response**: User data has been written into sharedPreferences for further offline authentication.
* **Test passed**? **Yes**
* **Actual Results**: Email from EditText is written to sharedPreferences to indicate current user and to provide offline access if needed, email and username pair is written to sharedPreferences for offline authentication, email and password is written to sharedPreferences for offline authentication.

Testing Firebase Authentication:

* **Test purpose**: To check that register button onClick method creates a user in Firebase authentication system.
* **Test actions**: Click on Register button after entering your details
* **Expected response**: Toast message that tells if the operation was successful and user was created in Firebase.
* **Test passed**? **Yes**
* **Actual Results**: Toast received which means the operation was successful, a new user appeared in the Firebase Console.

Testing Room Database User Table:

* **Test purpose**: To check if register button onClick method creates a User table in the database
* **Test actions**: Click on the register button after entering user credentials
* **Expected response**: User table was created in ROOM Database with populated user, username, password columns. User column should represent a Primary Key and the value should be email from EditText mEmail, username column should have value of entered username from EditText mUsername, password should have a value of entered password in EditText mPassword.
* **Test passed**? **Yes**
* **Actual Results**: User table is successfully created in ROOM Database with columns: user (Primary Key) – represents user email, password – represents user password, username – represents username, total\_games – represents a record count of user’s total number of games in his library, completed\_games – represents a record count of user’s total number of completed games in his library, profile\_image – represents a record for storing user profile image.

Testing Already have account text:

* **Test purpose**: To check if by clicking on this text it creates a new Intent and return the user to the Login screen.
* **Test actions**: register\_already\_have\_account onClick listener
* **Expected response**: new Intent to return to Login screen
* **Test passed**? **Yes**
* **Actual Results**: Clicking on the “Already have an account?” text creates a new Intent that starts a Login screen.

# *Login.java*

Testing Email EditText:

* **Test purpose**: To check if EditText mEmail can handle unexpected inputs provided by the user
* **Test actions**: Enter text in mEmail EditText, click the Sign in Button.
* **Expected response**: The text entered should be not null and in [example@example.com](mailto:example@example.com) format.
* **Test passed**? **Yes**
* **Actual Results**: Notifies if email entered is null, notifies if email entered is valid.

Testing password EditText:

* **Test purpose**: To check if EditText mPassword can handle empty value provided by the user.
* **Test actions**: Enter text in mPassword EditText, click the Sign in Button.
* **Expected response**: The text entered should not be null.
* **Test passed**? **Yes**
* **Actual Results**: Notifies user if he has not entered password.

Testing Register Button:

* **Test purpose**: To check if Register Button creates an Intent that starts Register Activity.
* **Test actions**: To click on Register Button
* **Expected response**: Register Activity is created.
* **Test passed**? **Yes**
* **Actual Results**: When Register Button is click it creates an Intent that starts Register Activity

Testing Sign in Button

* **Test purpose**: To check if authentication is successful if correct details are entered both for online and offline access, user email is rewritten in sharedPreferences in case of user change.
* **Test actions**: To click on Sign in button
* **Expected response**: Authenticates the user either trough Firebase Authentication or by using sharedPreferences for offline authentication. Rewrites email in sharedPreferences to the one specified in EditText mEmail to indicate current user. Creates new Intent and starts MainActivity.
* **Test passed**? **Yes**
* **Actual Results**: User is authorized, current user email is rewritten in sharedPreferences, new Intent is created and MainActivity is started.

# *MainActivity.java*

Testing Search Bar:

* **Test purpose**: To check API functionality when submitting search query into the search bar
* **Test actions**: To write a search query into the search bar
* **Expected response**: Populated RecyclerView with search results that were received from API in JSONArray format.
* **Test passed**? **Yes**
* **Actual Results**: API successfully sent back a JSONArray response from POST request. JSONArray was further handled to retrieve the data that is needed and then sent to an Adaptor that handled population the RecyclerView with the given data.

Testing Add Button:

* **Test purpose**: To check if the Add Button correctly gets the data from its row and inserts the data into Games table in ROOM Database. Ensure that Add Button correctly handles multiple inserts of the same data into the database.
* **Test actions**: To click on the Add Button of the search results in RecyclerView.
* **Expected response**: Game table is created and inserted to the database if no such table exist in the database. Notifies if the same object is in the database and does not include the same table multiple times.
* **Test passed**? **Yes**
* **Actual Results**: Add button correctly got the data of the current row and created Game table which further was inserted in the database. Game table has the following columns: id (Primary Key) – represents unique game id, user column (Foreign Key) - represents the user that added the game to his library, title – represents the title of the game, imageUrl – represents the url of the game cover image, hours\_played – represents number of hours played, completed – represents completion status of the game. If this Game table already exists and user tries to add it multiple times, the game is not added to the database and user is notified with Toast message that this game is already in his library.

# *Library.java*

Testing onCreate method:

* **Test purpose**: To check if onCreate method can handle getting the user’s game library and correctly deals with a case when user has no games in his library.
* **Test actions**: Create a Library activity.
* **Expected response**: If user has games in his library correctly displays them from database in the RecyclerView. If user has no games correctly handle the case and display empty RecyclerView.
* **Test passed**? **Yes**
* **Actual Results**: Data from the Game table displays correctly, and data absence is handled properly.

Testing Remove Button:

* **Test purpose**: To check if by clicking the button the Game gets deleted from database and the changes to the library can be instantly seen.
* **Test actions**: To click on Remove Button
* **Expected response**: Game is deleted from the database and from the library screen.
* **Test passed**? **Yes**
* **Actual Results**: Upon clicking Remove Button, correct row data is selected, and Game table is deleted from the database. RecyclerView is updated with new results.

Testing Completed Button:

* **Test purpose**: To check if by clicking the button the selected game is updated with new completed status and all following updates are handled appropriately. TextView representing progress of the game is changed to “Completed!”, and changed back to “In progress” if clicked the Completed Button on the game with status “Completed”
* **Test actions**: To click Completed Button
* **Expected response**: Selected Game’s completed status is updated and further updates to hours are restricted. TextView is updates to appropriate value.
* **Test passed**? **Yes**
* **Actual Results**: Game is successfully deleted from the database upon clicking on Completed Button. If the status of the game becomes “Completed” then buttons to edit hours in the game become disabled. Otherwise, if the status of the game becomes “Uncompleted” buttons to modify game hours become available. TextView is updated correctly with appropriate value based on completed status of the game.

Testing Edit Manually Button:

* **Test purpose**: To check if the Edit Manually Button correctly displays the dialog prompt to the user to input custom hour value. Correctly handles unexpected values from user input. When dialog window is submitted it should write the inputted value to the hours\_played column to Game table in the database. TextView hours should be update with the new value.
* **Test actions**: To click the Edit Manually Button.
* **Expected response**: Dialog is displayed when button is clicked, handles various inputs correctly, on dialog submit should update the hours\_played column of selected Game.
* **Test passed**? **Yes**
* **Actual Results**: When button is clicked, dialog prompt is displayed asking for a time value of format hours: mm. When dialog is submitted, input is checked for correct formatting and notifies a user if input is incorrect. When dialog has ensured that the input is correct it submits this data to update the hours\_played column. TextView hours is correctly updated with the new value.

Testing Add Hour Button:

* **Test purpose**: To check if the Add Hour Button updates the hours\_played column value. To ensure that upon clicking the button the TextView hours is updated with the new value.
* **Test actions**: To click on the +Hour Button.
* **Expected response**: 1 hour is added to the hours\_played column value in selected Game table. TextView is updated with the new value.
* **Test passed**? **Yes**
* **Actual Results**: 1 hour is added successfully to hours\_played column. The updated value is shown instantly in the TextView hours.

Testing Remove Hour Button:

* **Test purpose**: To check if the Remove Hour Button updates the hours\_played column value. To ensure that upon clicking the button the TextView hours is updated with the new value. Button should handle cases when hours\_played is 0 to avoid going to negative hours.
* **Test actions**: To click on the -Hour Button
* **Expected response**: 1 hour is removed from hours\_played column value in selected Game table. TextView is updated with the new value.
* **Test passed**? **Yes**
* **Actual Results**: 1 hour is removed successfully from hours\_played column. The updated value is shown instantly in the TextView hours.

# *Profile.java*

Testing Email and Username TextViews:

* **Test purpose**: To check if the Email TextView and Username TextView are updating accordingly depending on the current user.
* **Test actions**: To start Profile Activity
* **Expected response**: Email TextView and Username TextView are updated from sharedPreferences values.
* **Test passed**? **Yes**
* **Actual Results**: Upon starting Profile Activity username and email of the current user are taken from sharedPreferences and update corresponding TextViews accordingly.

Testing Counts TextViews:

* **Test purpose**: To check if total games TextView and completed games TextView are updated accordingly depending on the user logged in.
* **Test actions**: To start Profile Activity
* **Expected response**: total games TextView and completed games TextView are updated from selected user database.
* **Test passed**? **Yes**
* **Actual Results**: Upon starting Profile Activity total games and completed games are taken from current Users database and corresponding TextViews are updated accordingly.

Testing Edit Photo Button:

* **Test purpose**: To check if upon clicking the button Android Image Gallery is opened where user can choose an Image to use for profile picture. Ensure that profile picture remains after recreating the Activity.
* **Test actions**: To click Edit Photo Button
* **Expected response**: Image Gallery is opened where user can select profile image and the profile image would be saved to the selected user’s table. Image should remain in the ImageView upon restarting the Activity.
* **Test passed**? **Yes**
* **Actual Results**: Image Gallery is opening correctly, when choosing image, it is instantly visible in the ImageView as well as inserted in the user’s database. When restarting the activity image remains as the profile picture.

Testing Delete User Data Button:

* **Test purpose**: To check if the user data is deleted upon clicking the button.
* **Test actions**: To click the Delete User Data Button
* **Expected response**: All data that is specific to the current user is deleted from the database as well as from Firebase.
* **Test passed**? **Yes**
* **Actual Results**: All the data specific to the current user is deleted with his account. After this button is clicked user logged out.

Testing Wipe All Data:

* **Test purpose**: To check if all the database, sharedPreferences data is deleted.
* **Test actions**: To click Wipe All Data
* **Expected response**: All database and sharedPreferences data is deleted
* **Test passed**? **Yes**
* **Actual Results**: All database and sharedPreferences data is deleted

Testing Logout Button:

* **Test purpose**: To check that upon clicking the button the user is signed out from his current session
* **Test actions**: To click Logout Button
* **Expected response**: User should be signed out and returned to the Login screen. Button should also handle offline users.
* **Test passed**? **Yes**
* **Actual Results**: When the button is clicked Firebase calls a signOut and check for current user. If no current user present is starts Login activity. Offline users are also handled by sending new Intent to start Login Activity if no Internet connection is available to call a signOut from Firebase.