11/1/2024

User Documentation

COUNTIFY

OPSC7312 POE

VARSITY COLLEGE

TABLE OF CONTENTS

GROUP MEMBERS:

Ethan Schoonbee (**PM**) – **ST10036509**

Leonard Bester – **ST10026396**

Ross Harper – **ST10048758**

Cameron Brooks – **ST10032014**

INTRODCUNTION

WHAT IS COUNTIFY?

Countify is a simple Android counter app developed using Kotlin in Android Studio. The app allows users to create and manage custom counters, either by registering an account with their email or by using Google SSO (Single Sign-On). Counters are persisted using Firestore Database, and the app integrates various user preferences, including notifications, theme, and language settings. The app also displays random advice fetched from a third-party API.

PURPOSE

Countify was built to provide users with a simple and intuitive way to create, edit, and track multiple counters. It enables users to increment values easily for a variety of personal or professional use cases. Additionally, the app offers settings customization for a personalized user experience, as well as integration with external APIs to enhance functionality.

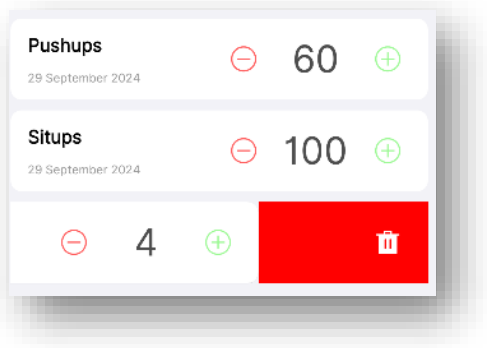
FEATURES

USER AUTHENTICATION:

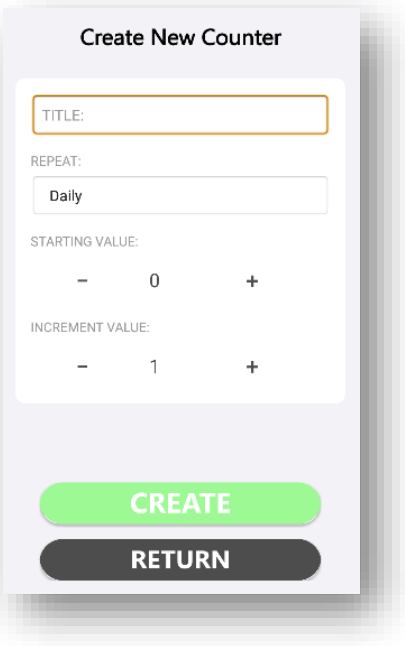
- Register using email and password or with Google SSO.

COUNTERS:

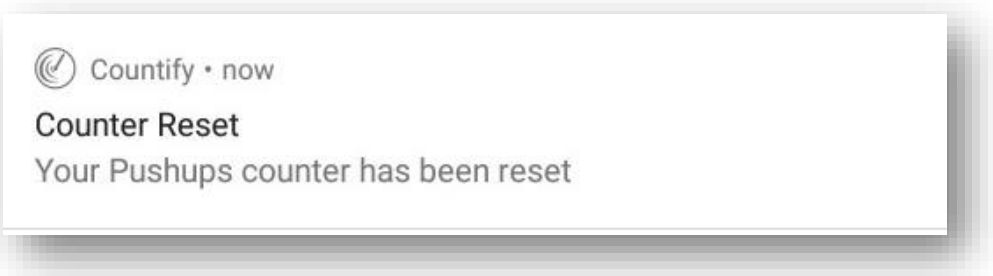
- View a list of created counters.



- Add new counters with custom titles, descriptions, and increment values.

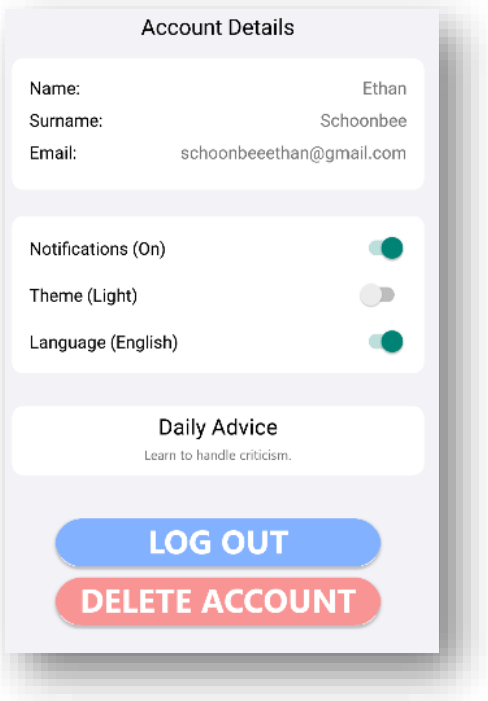


- Data is stored and retrieved from Firestore Database.
- Counters reset to their selected **starting value** after the repeat time has expired.



SETTINGS FEATURES:

- Manage account details (name, email, surname).
- Toggle settings for:
  - Notifications (On/Off).
  - Language (English/Afrikaans).
- Random Advice API integration: Displays a new random piece of advice every time the settings page is loaded.



- The delete account button will delete the user and log out *Countify*.

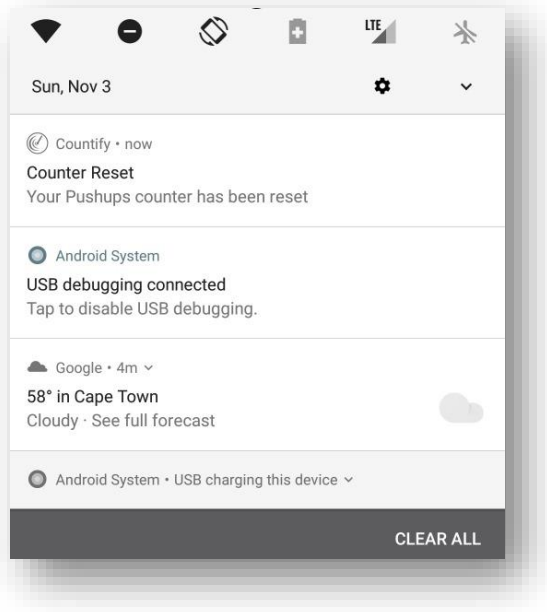
PERSISTENT DATA:

- All user preferences (settings, counters) are saved to the Firestore Database and reloaded when the application boots up.

Cloud Firestore

PUSH NOTIFICATIONS:

- In the local code base, push notifications are built and sent to the user.



Countify notification featuring in the list of notifications on Android device

DATABASE SYNCING

- Users can add a counter while being offline utilising *SQLite* to save local data. (Kreibich, 2010) Once an internet connection is established on the android device, the *SQLite* data will be merged with the **Google Firebase** cloud database.

DESIGN CONSIDERATIONS

USER INTERFACE:

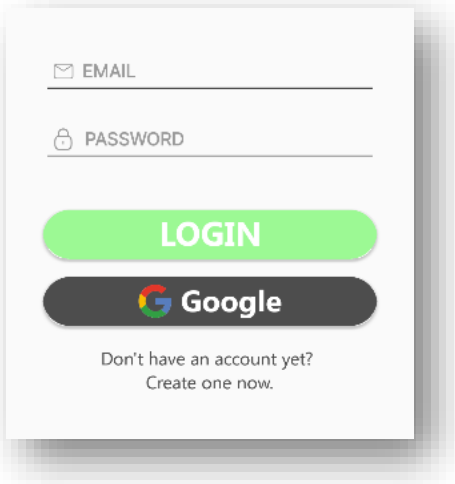
The user interface is designed to be minimalistic and intuitive, with a Floating Action Button (FAB) for creating new counters and easy navigation to the settings page.

FAB

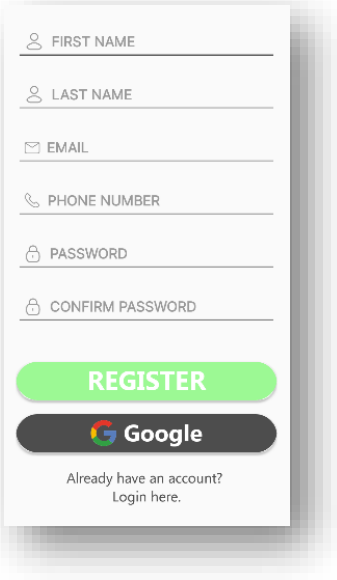
Settings Button

USER AUTHENTICATION:

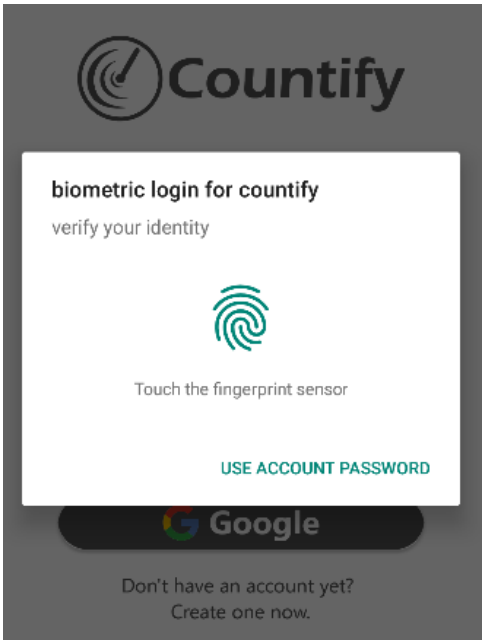
We utilized Firebase Authentication for secure login, offering both email registration and Single Sign-On (SSO) via Google for convenience and Biometrics for secure logging back in (Android). User details are stored in Firestore for easy retrieval and display in the settings.



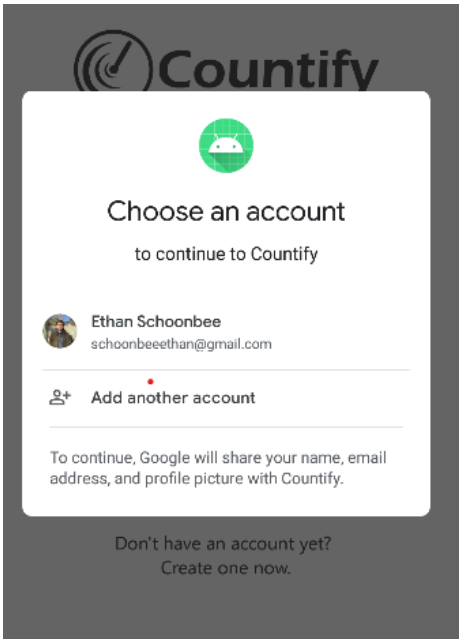
Email & Password Login



User Registration



Biometric Login

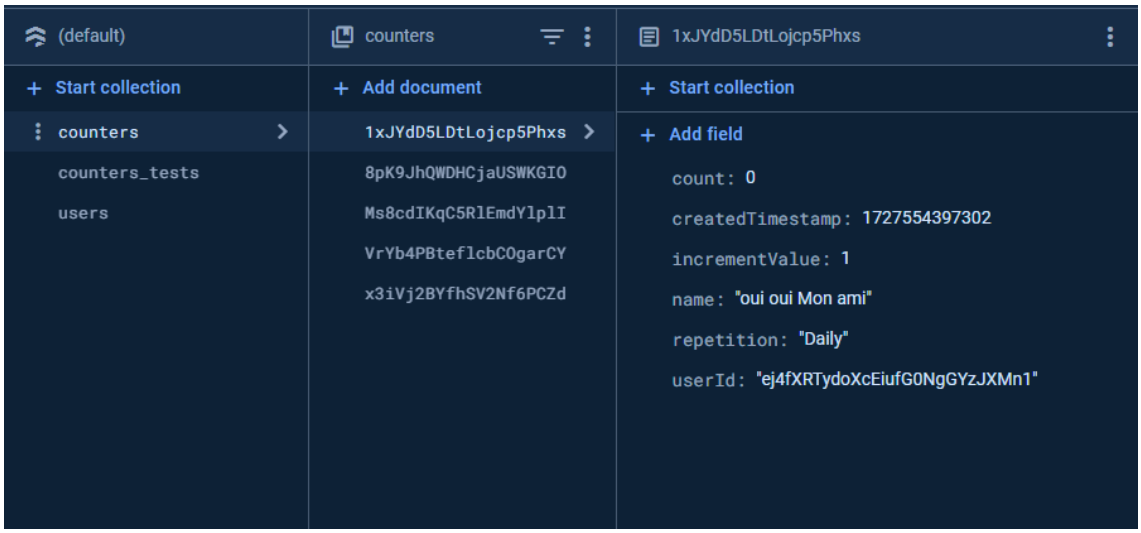


Google SSO Register and Login

FIRESTORE DATABASE:

All counters and user preferences are stored in Firestore for scalability, real-time updates, and secure storage.

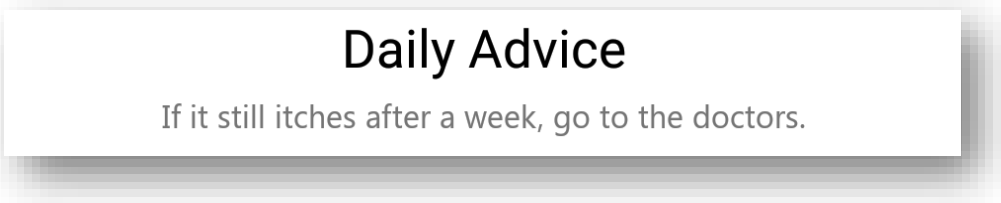
Firebase Authentication



Firestore NoSQL Database

RANDOM ADVICE API INTEGRATION:

The settings page connects to the external Random Advice API, fetching a fresh piece of advice on every load. This adds a fun, interactive element to the app.



INTERNAL DESIGNS

FRAGAMENTED PAGES:

The primary expression of forms and user interfaces for *Countify* is **fragments.** Fragments are modular forms that allow for more flexible UI design which allows for a diverse range of phone sizes to make use of the user interfaces.

Instead of creating and swapping to an entire new form, a **fragment** will populate will be opened in the main form. One could think of it as being contained inside the main form and swapping fragments allows for the same for to display while the user interface, along with the associated logic, can change to different functional pages. (WIlson, 2016)

NOTIFICATION CHANNELS & CLASSES:

The push notifications in *Countify,* makes use of a **notification channel** within android in the **Main Activity** of the application on start-up.

A **notification compact.builder** is used and structured within it's own class as a service. This class builds the required notification giving it an identification number, a title, an icon and a content message. This class then uses the **notification channel** to send to the android mobile device which will display the *Countify* notification in the notifications list. (Nudelman, 2013)

SETUP AND INSTALLATION

PREREQUISITES

- Android Studio (latest version) (Andorid, 2024)
- Kotlin configured in the Android project (JetBrains, 2024)

STEPS TO COMPILE AND RUN

1. **Clone the Repository:**
    - git clone https://github.com/ST10036509/Countify.git
    - cd countify
2. **Build the Project:**
    - Open the project in Android Studio.
    - Sync Gradle files by clicking on the "Sync Now" prompt.
3. **Run the Project:**
    - Click on the "Run" button in Android Studio or press Shift + F10.

USAGE

Register/Login:

- Open the app and sign up using an email address or log in using Google SSO. (Google, 2024)

Manage Counters:

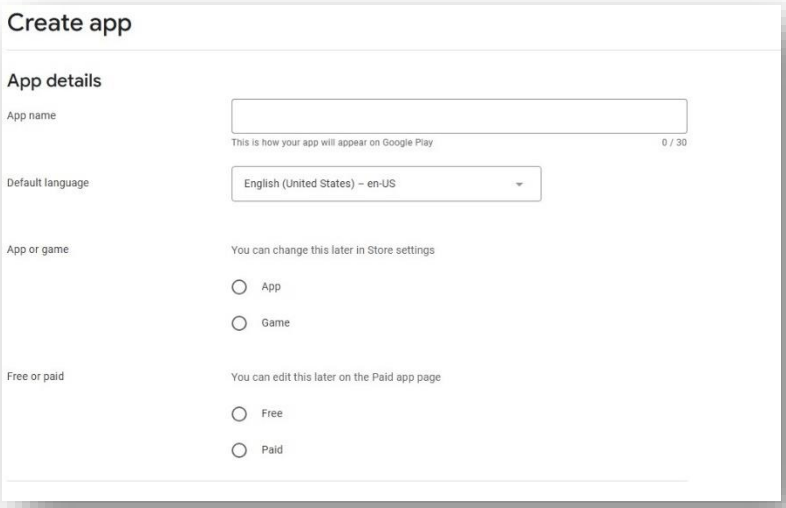
- Once logged in, you'll see a list of counters. To add a new counter, press the FAB and enter the title, description, and increment value. The counter will be saved to Firestore. Counters can be deleted but swiping left on the counter you wish to remove.

Settings:

- Navigate to the settings page to manage preferences. You can toggle notifications, change the theme, and select the language. These preferences are saved in the Firestore and applied on app restart.
- You can also view your account details (name, surname, and email) on this page.
- A random piece of advice will be fetched from the Random Advice API each time the settings page is opened.

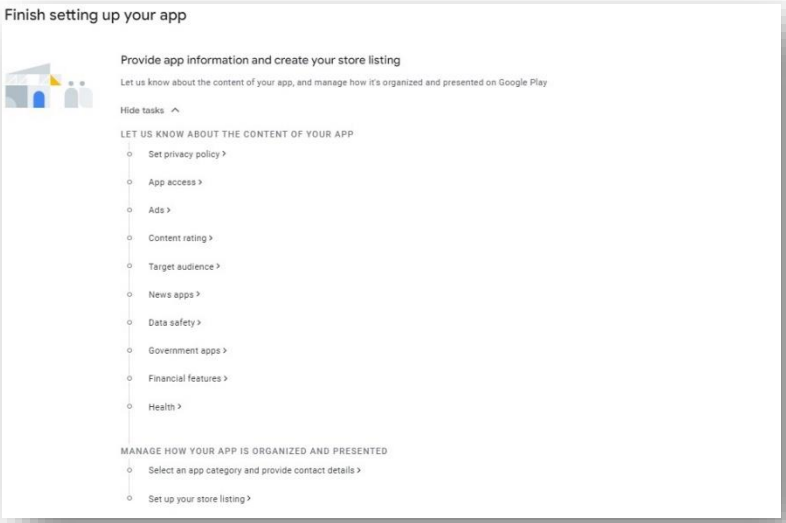
GOOGLE PLAY CONSOLE UPLOAD PROCESS

STARTING APPLICATION UPLOAD PROCESS



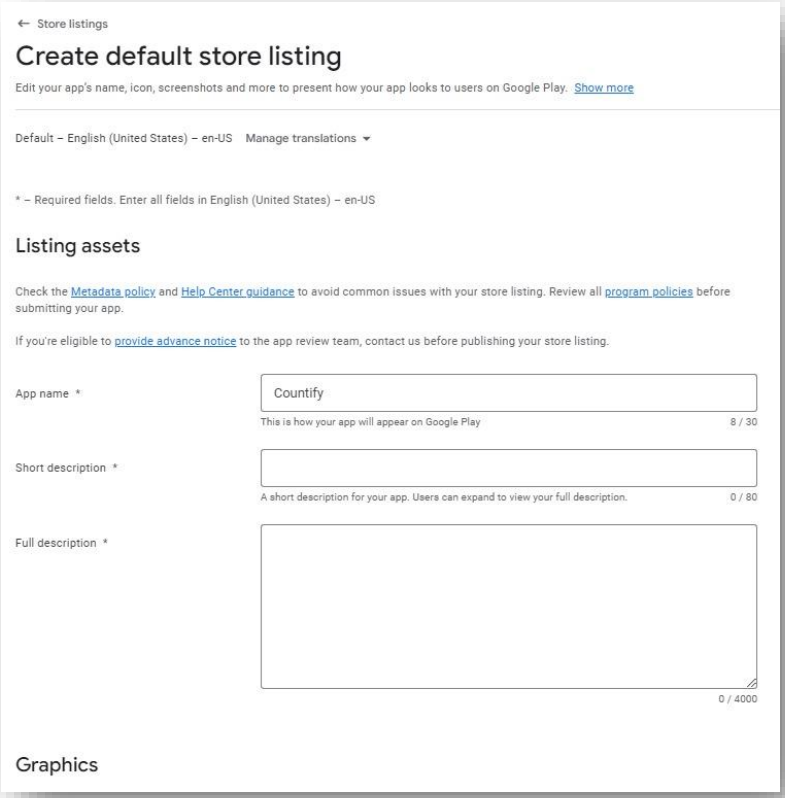
The **first step** is to initialise the upload process by filling in the application name, game status and purchase status.

INPUTTING APPLICATION DETAILS



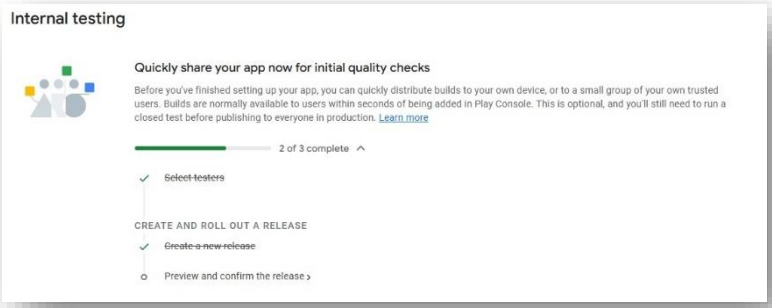
This step requires the application **details and content** like privacy policy, Content rating, targeted audience and the category of the application be entered appropriately according to the content of *Countify*.

APPLICATION DESCRIPTIONS



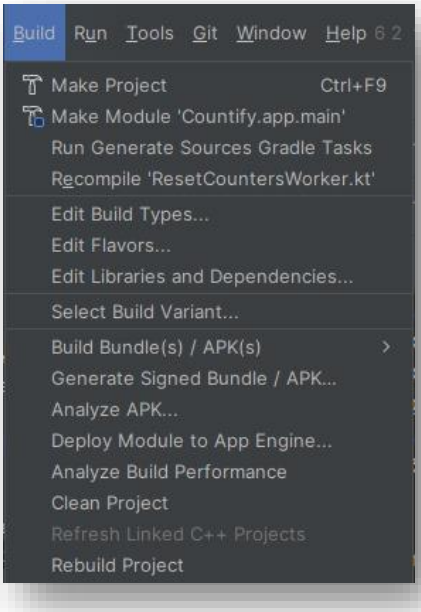
Entering the **description** of the application. Listing its **purpose and features** and explaining the functionality of *Countify*.

INTERNAL TESTING CONFIGURATION

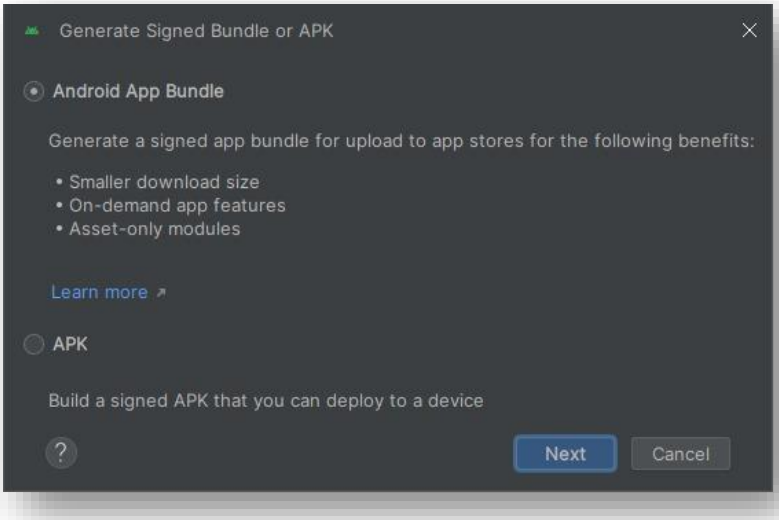


In this step, 14 **batch tester** are selected for testing the application for quality checks.

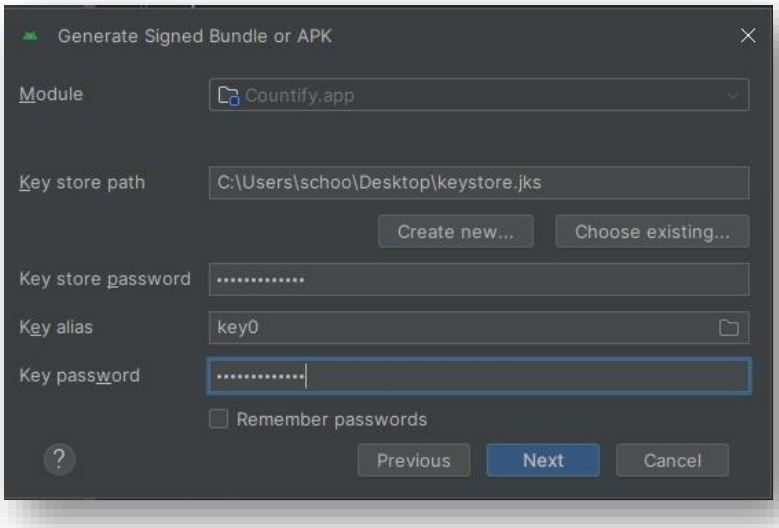
BUNDLE BUIDLING



In android studios under the build drop down, to start building a **single bundle,** the **Generate Signed Bundle** option is clicked

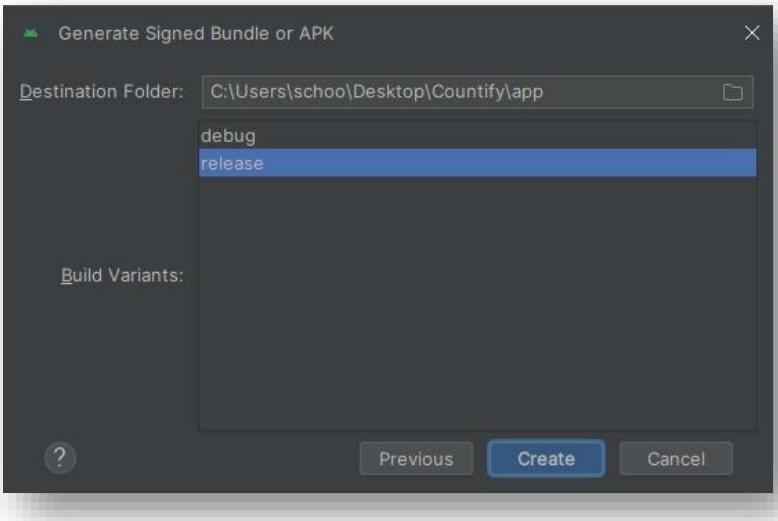


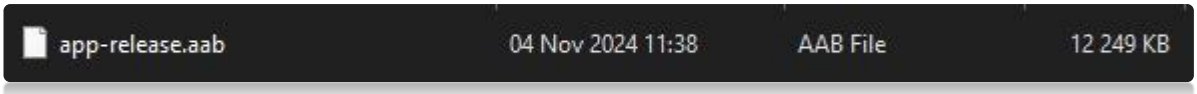
Then the **Android App Bundle** option is picked as the advantages allow for a smaller download size, On-demand app features and Asset-only modules to be enabled



Then the key is generated by entering the input fields along with a key password.

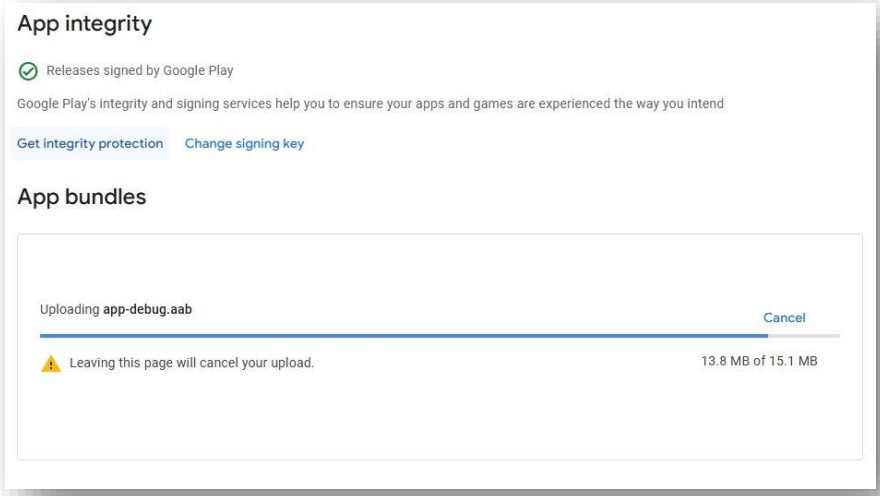
Then the last step for making a single bundle is to select the **release** option as this will be the latest running version of the application.

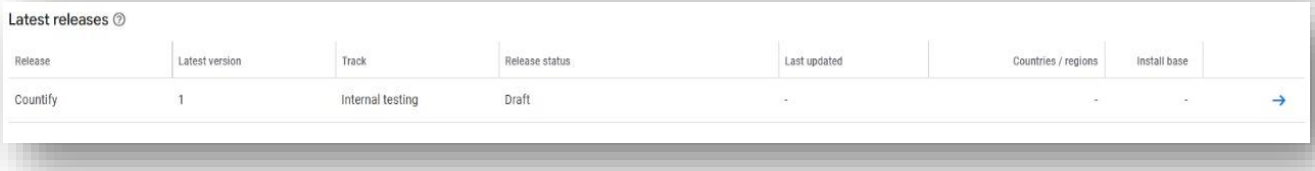


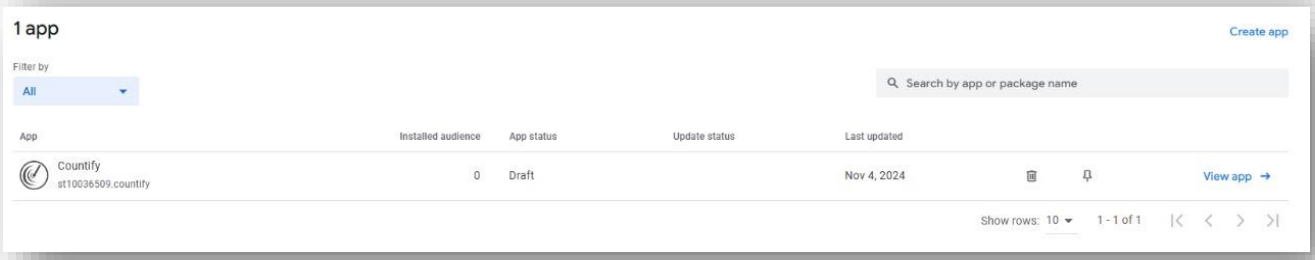


APPLICATION RELEASE

Once the application's single bundle release is ready it can be selected to be uploaded.



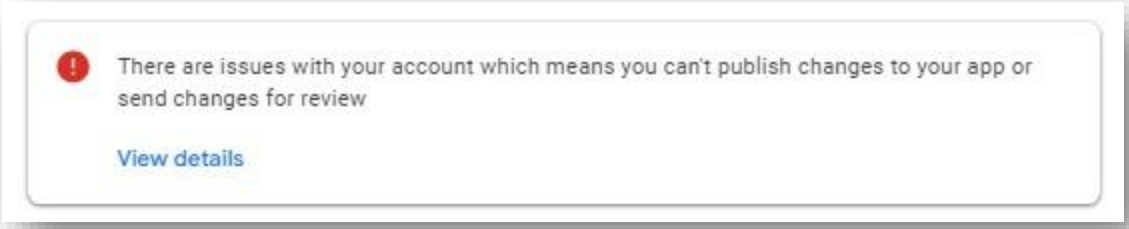


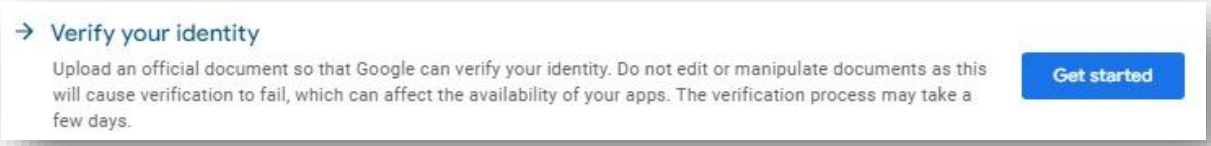


Then we can see that the application *Countify* with its unique icon, has been listed with the latest version of the source code.

FULL RELEASE ERRORS

In other circumstances, *Countify* would be on the **Google Play Store**, but the error listed below prevents this from happening. The cause of the error is the lack of a verified account and to verify the account to resolve this error would reach past the hand-in date for the **OPSC POE.**



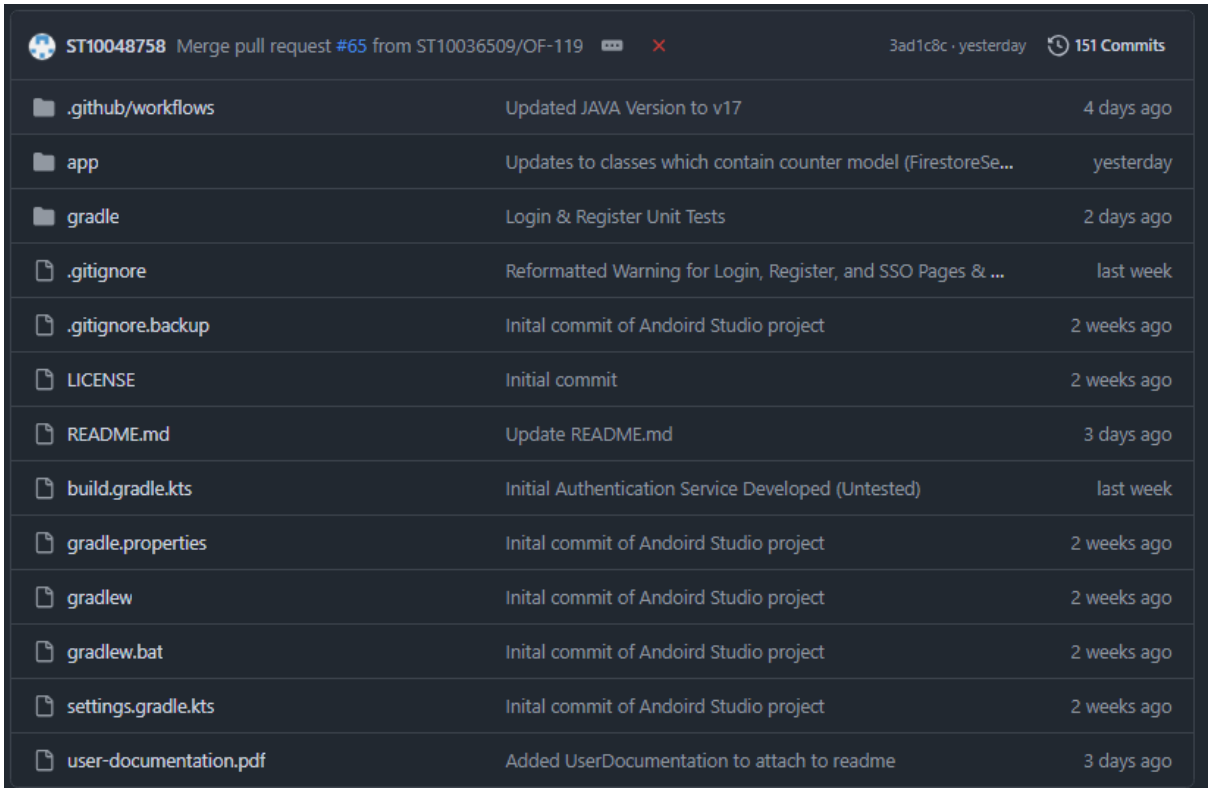


GITHUB

GITHUB USAGE:

We used GitHub for source control, allowing our team to push and pull code updates seamlessly. Branching was used to isolate features and bug fixes, ensuring that the main codebase remained stable. (GitHub , 2024)

- **Branches:** Each feature was developed on its own branch and merged into the main branch via pull requests.
- **Commits:** We maintained meaningful commit messages, documenting changes clearly.
- **Pull Requests:** Code reviews were done on pull requests to ensure high code quality and collaboration.



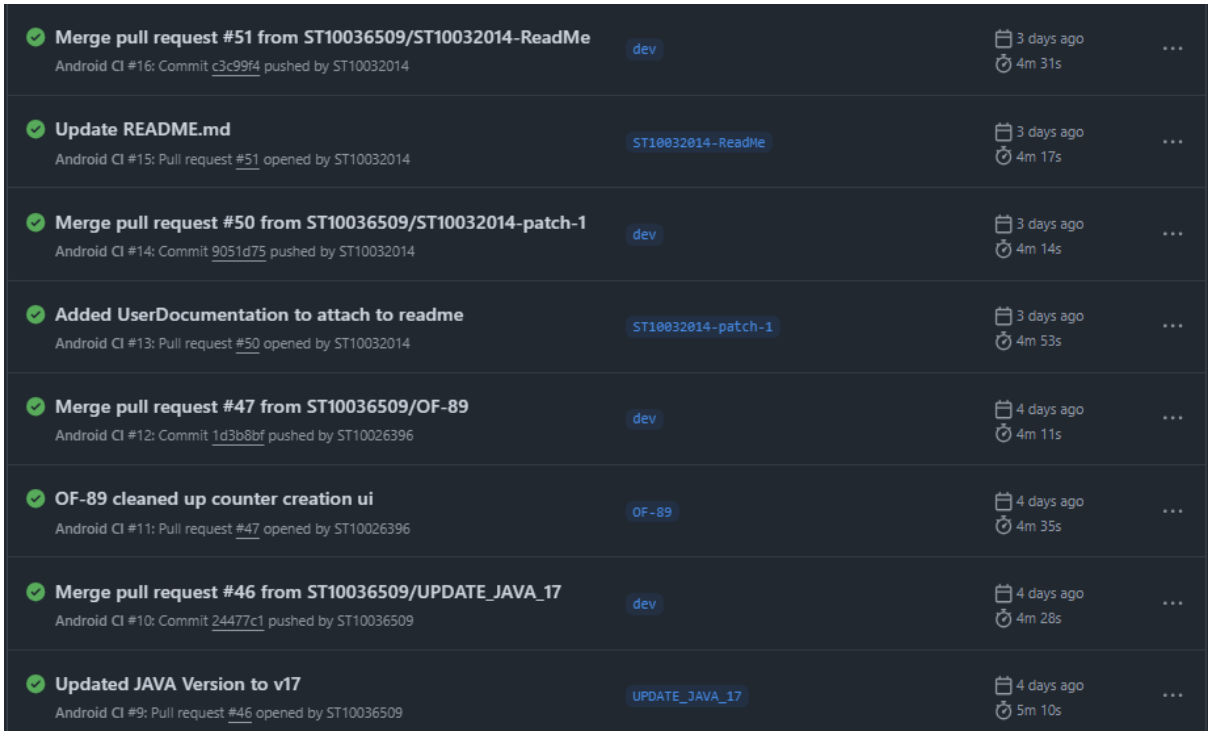
ST10048758 Merge pull request #65 from ST10036509/OF-119 3ad1c8c · yesterday 151 Commits

.github/workflows	Updated JAVA Version to v17	4 days ago
app	Updates to classes which contain counter model (FirestoreSe...	yesterday
gradle	Login & Register Unit Tests	2 days ago
.gitignore	Reformatted Warning for Login, Register, and SSO Pages & ...	last week
.gitignore.backup	Inital commit of Andoird Studio project	2 weeks ago
LICENSE	Initial commit	2 weeks ago
README.md	Update README.md	3 days ago
build.gradle.kts	Initial Authentication Service Developed (Untested)	last week
gradle.properties	Inital commit of Andoird Studio project	2 weeks ago
gradlew	Inital commit of Andoird Studio project	2 weeks ago
gradlew.bat	Inital commit of Andoird Studio project	2 weeks ago
settings.gradle.kts	Inital commit of Andoird Studio project	2 weeks ago
user-documentation.pdf	Added UserDocumentation to attach to readme	3 days ago

GITHUB ACTIONS:

We implemented GitHub Actions to automate testing and ensure code consistency. Whenever a new branch was pushed or a pull request was opened, tests would automatically run, helping to catch bugs early.

- **Automated Testing:** Unit tests were written for critical functions, and GitHub Actions would trigger these tests on every push to a branch. (GitHub, 2024)
- **CI/CD Pipeline:** The pipeline ensured that every merged change passed tests before being integrated into the main branch. (GitHub, 2024)
- As of 26/09/2024 the **Mockito** (Mockito , 2023) package used for mocking **AndroidX** fragments and classes had a depreciated package (**Byte Buddy**) (Winterhalter, 2024)which it relied on for simulating android environment with **Maven** so none of our tests could be run or pass.



Merge pull request #65 from ST10036509/OF-119 Android CI #46: Commit 3ad1c8c pushed by ST10048758	dev	yesterday 4m 48s	...
Updates to classes which contain counter model (FirestoreSer... Android CI #45: Pull request #65 opened by ST10048758	OF-119	yesterday 5m 4s	...
Merge pull request #64 from ST10036509/OF-114 Android CI #44: Commit 9fabf05 pushed by ST10048758	dev	yesterday 3m 30s	...
Changed counter model and changed CounterAdapter and C... Android CI #43: Pull request #64 opened by ST10048758	OF-114	yesterday 3m 57s	...
Merge pull request #63 from ST10036509/OF-118 Android CI #42: Commit 02aa7e4 pushed by ST10026396	dev	2 days ago 4m 11s	...
OF-118 Android CI #41: Pull request #63 opened by ST10026396	OF-118	2 days ago 4m 3s	...
Merge pull request #62 from ST10036509/OF-116 Android CI #40: Commit 58f5ab3 pushed by ST10026396	dev	2 days ago 4m 0s	...
OF-116 Android CI #39: Pull request #62 opened by ST10026396	OF-116	2 days ago 4m 6s	...

```
Byte Buddy could not instrument all classes within the mock's type hierarchy

This problem should never occur for javac-compiled classes. This problem has been observed for classes that are:
 - Compiled by older versions of scalac
 - Classes that are part of the Android distribution
        at org.mockito.internal.creation.bytebuddy.InlineBytecodeGenerator.triggerRetransformation(InlineBytecodeGenerator.java:280)
        at org.mockito.internal.creation.bytebuddy.InlineBytecodeGenerator.mockClass(InlineBytecodeGenerator.java:213)
        at org.mockito.internal.creation.bytebuddy.TypeCachingBytecodeGenerator.lambda$mockClass$0(TypeCachingBytecodeGenerator.java:47)
        at net.bytebuddy.TypeCache.findOrInsert(TypeCache.java:168)
        at net.bytebuddy.TypeCache$WithInlineExpunction.findOrInsert(TypeCache.java:399)
        at net.bytebuddy.TypeCache.findOrInsert(TypeCache.java:190)
        at net.bytebuddy.TypeCache$WithInlineExpunction.findOrInsert(TypeCache.java:410)
        at org.mockito.internal.creation.bytebuddy.TypeCachingBytecodeGenerator.mockClass(TypeCachingBytecodeGenerator.java:40)
        at org.mockito.internal.creation.bytebuddy.InlineDelegateByteBuddyMockMaker.createMockType(InlineDelegateByteBuddyMockMaker.java:391)
```

Logs showing Byte Buddy failing

ACCESS THE GITHUB REPO HERE.

FURTHER DETAILS

FIRESTORE INTEGRATION:

- Firestore was used to store both the user counters and their settings/preferences. Each user has their own document, storing counters (with titles, descriptions, and increment values) and a separate collection for their settings (notifications, theme, language). This separation makes it easy to scale and manage data.

FIREBASE AUTHENTICATION:

- Firebase Authentication was used to handle email-based login and Google SSO. Upon successful login, user details are pulled from Firebase to display in the settings page.

API INTEGRATION:

- We implemented two of Firebase's REST APIs utilising their **Firebase Authentication** service accepting either **Firebase Tokens** or **OAuth 2.0 Tokens** and their NoSQL database service **Firestore** for handling RESTful remote cloud data storage. (Google, 2024) (Google, 2024)
- The Random Advice API is a public REST API that returns a random piece of advice. We integrated it into the settings page using a simple HTTP GET request, which is triggered every time the settings page is loaded. The response is parsed and displayed in a TextView. (Kiss, 2024)
- Access the API here.

REFERENCES

Andorid. (2024). *Android Studio.* Retrieved from Developers: https://developer.android.com/studio?gad_source=1&gclid=CjwKCAjw9eO3BhBNEiwAoc0-jSipK5gLTkiI_1jT9epYd5mZyhSqpAoNOGtZe7vP0dEKQLFLkGBSgRoCjawQAvD_BwE&gclsrc=aw.ds

Android. (n.d.). *BiometricPrompt.* Retrieved from Developers: https://developer.android.com/reference/androidx/biometric/BiometricPrompt

GitHub . (2024). *About Git.* Retrieved from GitHub Docs: https://docs.github.com/en/get-started/using-git/about-git

GitHub. (2024). *Automating Projects using Actions.* Retrieved from GitHub Docs: https://docs.github.com/en/issues/planning-and-tracking-with-projects/automating-your-project/automating-projects-using-actions

GitHub. (2024). *The complete CI/CD solution.* Retrieved from GitHub: https://github.com/solutions/ci-cd

Google. (2024). *Firebase Database REST API.* Retrieved from Firebase: https://firebase.google.com/docs/reference/rest/database

Google. (2024). *Single sign-on* . Retrieved from Google Cloud: https://cloud.google.com/architecture/identity/single-sign-on

Google. (2024). *Using the Firestore REST API - Authentication and authorization.* Retrieved from Google Cloud: https://cloud.google.com/firestore/docs/use-rest-api#:~:text=For%20authentication%2C%20the%20Firestore%20REST,requests%20from%20your%20application's%20users.

JetBrains. (2024). *Kotlin - Concise. Multiplatform. Fun.* Retrieved from Kotlinlang: https://kotlinlang.org/

Kiss, T. (2024). *Advice Slip JSON API.* Retrieved from Advice Slip: https://api.adviceslip.com/

Kreibich, J. (2010). *Using SQLite.* Sebastopol: O'Reily Media, Inc.

Mockito . (2023). *How do I drink it?* Retrieved from Mockito : https://site.mockito.org/

Nudelman, G. (2013). *Android Design Patterns: Interaction Design Solutions for Developers.* Indianapolis: John Wiley & Soms, inc.

WIlson, J. (2016). *Creating Dynamic UIs with Android Fragments.* Bermingham: Packt Publishing Ltd.

Winterhalter, R. (2024). *Byte Buddy (without Dependencies).* Retrieved from Maven Repository: https://mvnrepository.com/artifact/net.bytebuddy/byte-buddy