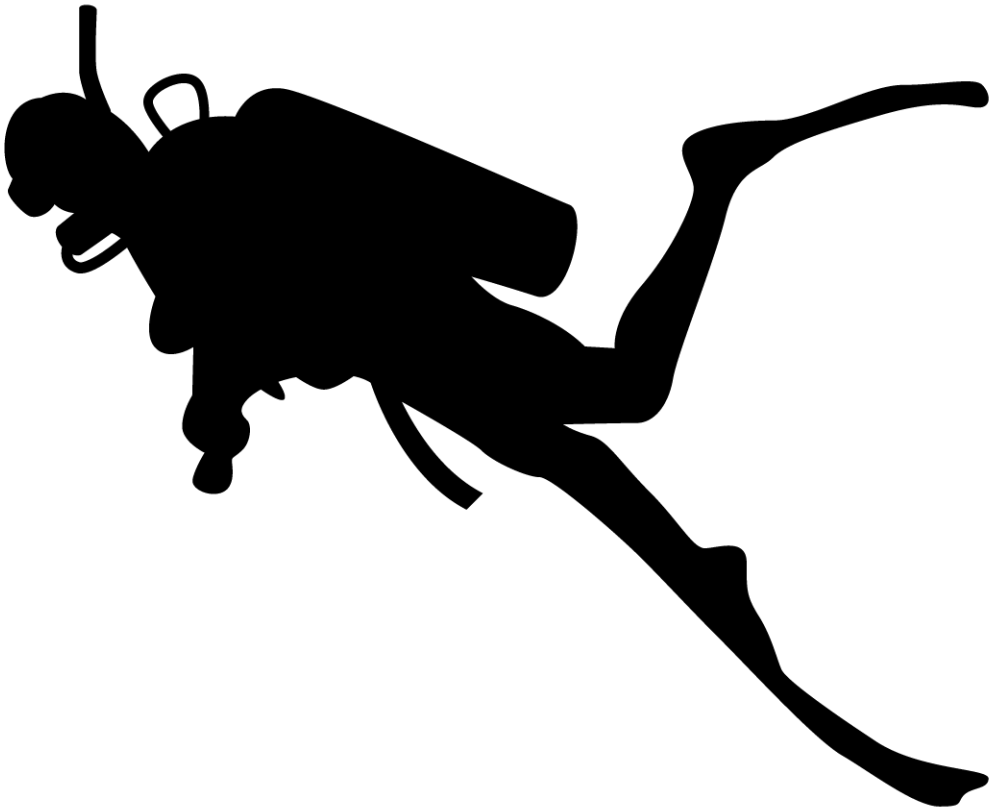


DIVE LOG APP

CS 6019
Capstone Project
Jon Hughes
MSD 2022

TABLE OF CONTENTS

INTRODUCTION

I have been a scuba diver since I was 12 years old, and prior to the Covid-19 pandemic, I was working as a scuba instructor and divemaster in the Caribbean. One of the things I noticed about my clients, most of whom were diving while on vacation, was that most did not keep logs of their dives -- or even if they normally did, they left their logbooks at home while on vacation.

WHY DIVERS SHOULD KEEP LOGS

Dive logs are really useful tools for divers. Not only are they a nice way to remember all the cool things you saw on your dive, but you can use the information in the log to help keep you safe and comfortable on future dives.

For example, in preparing for an upcoming dive, it helps to have a record of the water temperature on a particular previous dive, the diver's choice of exposure suit, whether the suit was sufficiently warm, the corresponding amount of additional lead weights needed to counteract the buoyancy of the exposure suit (e.g. a dry suit is typically more buoyant than a wetsuit), and whether the number of weights chosen was appropriate.

Weights are a big issue in diving. The goal is to be neutrally buoyant (neither sinking deeper or floating upwards) throughout the dive, unless descending or ascending by choice. The human body is almost always ever-so-slightly positively buoyant, the exposure suit is buoyant, and even the air tank can become buoyant as it becomes emptier. The amount of weight needed to counteract these buoyant forces varies greatly between individuals, depending on factors such as a person's size, body fat percentage, diving experience, etc.

Not only do the weights allow you to have enough negative buoyancy to even make the initial descent for the dive, but they are also necessary for safety by preventing uncontrolled ascents. Divers cannot ascend too quickly without risk of decompression illness, and sometimes it is unsafe to ascend due to boat traffic on the surface. So, being able to control when you ascend and descend is paramount.

However, it is difficult to remember the various permutations of the choices mentioned in the examples above unless one is diving very regularly, which is almost always not the case, especially with people who only dive while on vacation in the tropics. So, most of the time, "holiday" divers are walking into dive shops with no idea how much weight they would need for the dive and no record of previous dives to use as reference.

As a divemaster in these circumstances, I could make an educated guess as to how much weight a client might need. It was usually an overestimate because a little too much weight is safer than too little. But over-weighting has other negative consequences, like increased air consumption (moving increased mass though the water requires more exertion, and therefore more air) leading to shorter dives. Additionally, I would have to carry extra weight with me during the dive just in case I had inadvertently underestimated and the client was under-weighted.

Divers have more enjoyable dives and the divemaster's day runs more smoothly when divers know how much weight they need for a dive. The idea for his Dive Log App came from a desire to increase the keeping of dive logs.

THE PROBLEM

There are several dive log apps on the market, but many of them are over-complicated, with data fields that most divers never use. In addition to the information mentioned above (weights, exposure suit, etc.), most divers will record things like the maximum depth reached, the amount of time underwater, the date, the location, and number of the dive (divers use their number of dives as one measure of their experience level). However, most divers are not recording the starting and ending pressures of their air tanks, and since most divers today are using dive computers instead of dive tables, most have no need to record dive table data in a dive log, but the option should still be there for those that do or for training purposes.

The layout of these dive logs is also fixed, meaning that you might be scrolling past a lot of empty or useless (to you) fields just to find the one you need for the information that is relevant to you, leading to a bad user experience.

In addition, missing from the dive log apps on the market is a feature of paper dive logs that adds an element of fun to keeping dive logs: collecting stamps from various dive shops and divemasters that a person dives with. Most dive shops have unique stamps that they are happy to stamp into the dive logs of their clients. Many divers enjoy collecting these stamps in much the same way that some people enjoy collecting stamps in their passports.

BACKGROUND

EXISTING SOLUTIONS

DiverLog+ - Allows integration with some dive computers, which usually record depths, times, and temperatures, but information about weights, exposure suit, dive conditions, etc. must still be entered manually. The layout is fixed and somewhat unintuitive, potentially irrelevant fields cannot be hidden and, while there is the ability to add photos, it seems oriented towards keeping photos taken during the dive and there is not a separate section for a stamp.

Subsurface-mobile - Subsurface is an open source dive log platform that seems primarily geared to keeping dive logs on a computer rather than a phone. There is a mobile version of the app, but it is rather limited in the number of data fields available, the layout is fixed, and there is no stamp or photo section.

Dive Log - Recently added stamp functionality and has data fields that most other dive log apps have, but like the examples above, the layout is fixed and the data fields are somewhat limited (e.g. you can specify having used a wetsuit, but not its thickness).

UNIQUE FEATURES

The most recent app I used for keeping my log was DiverLog+. Since I was diving every day, I was using the app a lot and was very familiar with it. But in spite of that, I still found it rather unwieldy and eventually just started keeping my logs in a notes document on my phone. This is not the way that I would typically suggest keeping a log, but I was not in a typical situation since I was diving as part of my job.

Every day, I was diving on two of a small handful of dive sites, at roughly the same times of day, to roughly the same depths, in roughly the same conditions. My exposure suit was the same every day, and as mentioned above, I was usually diving over-weighted in case I needed to give weight to clients mid-dive. So, for the most part, my dive logs consisted of the date, the name of the dive sites, and perhaps the maximum depth if it was out of the ordinary.

Coincidentally, it turns out that what I wanted in a dive log app largely intersects with what the typical "holiday" diver needs in a dive log app. We both need something easy and customizable, so that we only see the fields in the log that are important for us, individually. Moreover, as a divemaster who wants to encourage divers to keep a log of their dives, I would hope that adding something easy and incentivizing, like collecting dive shop stamps would encourage more divers to take up keeping their logs.

DESIGN CHOICES

The first design choice was to choose a language that could simultaneously develop Android and iOS apps. After looking into some possibilities, I suggested using C# and Xamarin in my initial proposal. However, my capstone mentor (Varun) recommended Flutter, which I had previously not come across, and after looking into it, I took up the recommendation. Flutter is a cross-platform application framework using Dart. As such, this app will run on both Android and iOS.

As far as my choice for backend, I decided on Firebase. In my initial research into which backend to choose, I prioritized functionality that did not require a

continuous Internet connection. Dive sites are often in remote locations or on boats offshore and out of the range of cell towers, not to mention that if someone is traveling abroad, they may only have Internet access over the Wi-Fi of their hotel. However, I would still want the user to be able to backup their log data once they eventually did reconnect to the Internet. These thoughts initially led me towards Apache Hive, but again my capstone mentor recommended Firebase and, upon further research, found that Firebase is still readily usable while temporarily offline under the conditions which the users of my app would be. Namely, the remote data would not be changing while the user was offline.

In terms of layout choices, I decided on putting the navigation and action buttons at the top of the page in what Flutter calls the AppBar. This would keep the buttons always visible while scrolling and also when the keyboard is visible, allowing easy access to the buttons. The app itself has a very limited number of screens and actions, so the AppBar would not become cluttered.

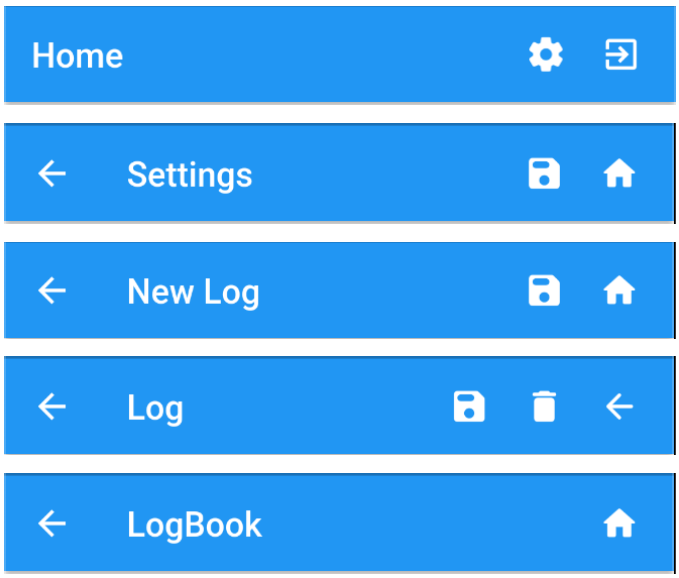
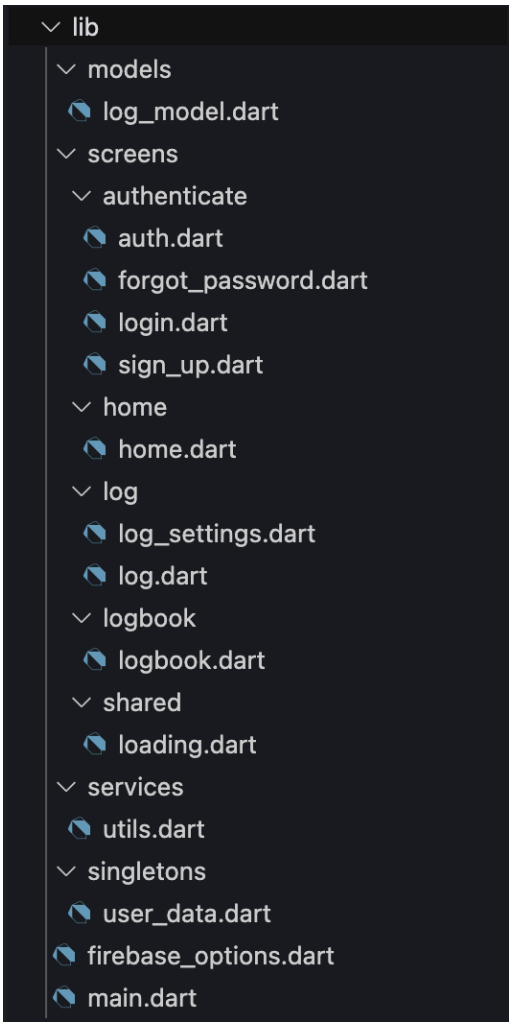


Figure 1: The AppBars of the various screens and their buttons

SOLUTION DESCRIPTION

SOFTWARE COMPONENTS



The software consists of Dart files, organized into folders and sub-folders depending on the function of the file.

The models folder holds the files for the data classes. At the moment, there is only one data class, the log_model, which is used when fetching log data from the database.

The screens folder holds the files used for each of the screens the user sees. Most of the codebase is in the screens folder. They are further subdivided by general category. The authenticate folder holds all the screens for user authentication, registration, and forgotten password. The home folder holds the home screen file. The log folder holds the files for the log screen and the settings screen. The logbook screen holds the file for generating the logbook. Finally, the shared folder holds the loading screen, which is used by several other screens when waiting for long calls to Firebase to complete.

There is a services folder which contains a utils file which contains a utility class for showing a snackbar (i.e. notification bar on the screen).

There is a singletons folder which holds the file I used to store a singleton of some user data, like the uid and an array of the order of the widgets for the log screen. This data is fetched from the database when the app starts up and the remote data is updated whenever it is changed locally, but it allows me to

reference this data from anywhere in the app without having to make repeated read calls to the database.

The last two files in the file structure are the firebase_options file, which is an auto-generated file by Firebase that allows the app to use its services, and 'main' which is where the app begins running and basically just routes the user either to the authentication screens or to the home screen if user credentials are already present.

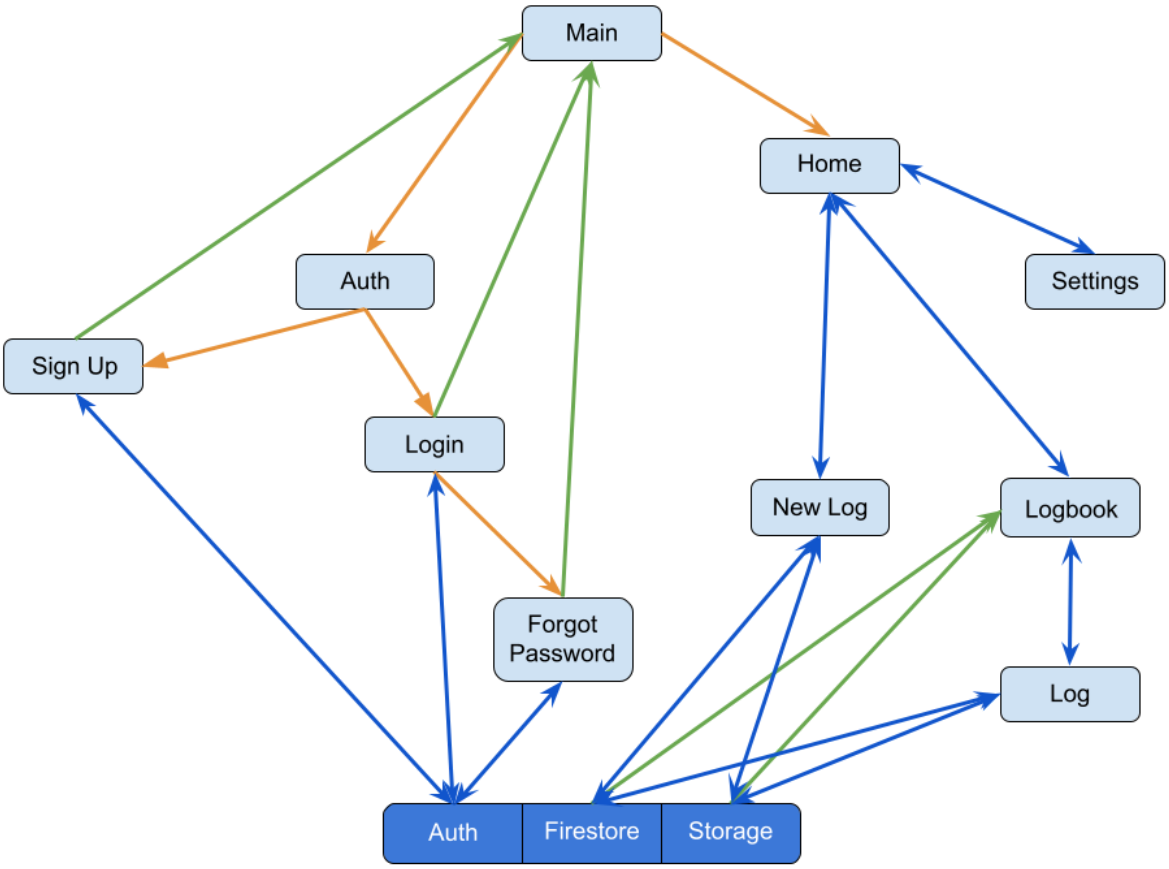
TOOLS/LIBRARIES

As mentioned earlier, for the mobile app, I am using Flutter, which allows me to develop the program in Dart, but output is compiled to Kotlin and Swift to run on Android and iOS respectively.

For the backend, I am using Firebase, which provides a number of features. Firebase Authentication easily handles user authentication, Firestore is a NoSQL database for keeping a remote backup of the dive log data, and Cloud Storage can be used to store larger files like the stamp images. Firestore has a document size limit of 1MB, so Cloud Storage was necessary for storing the stamp image files.

If there is no Internet connection, Firebase keeps a local stack of database changes, which is then applied to the remote database upon reconnection to the Internet. I was able to get this aspect working when I was only dealing with the log data, but I ran into issues when trying to do the same with the stamp image files and Cloud Storage.

SOFTWARE ORGANIZATION/DESIGN



On first use, the first screen a user is taken to is the 'authentication' screen. The user can either login or register a new user from here. The login credentials are an email address and a password. There is also the option of recovering a forgotten password here.

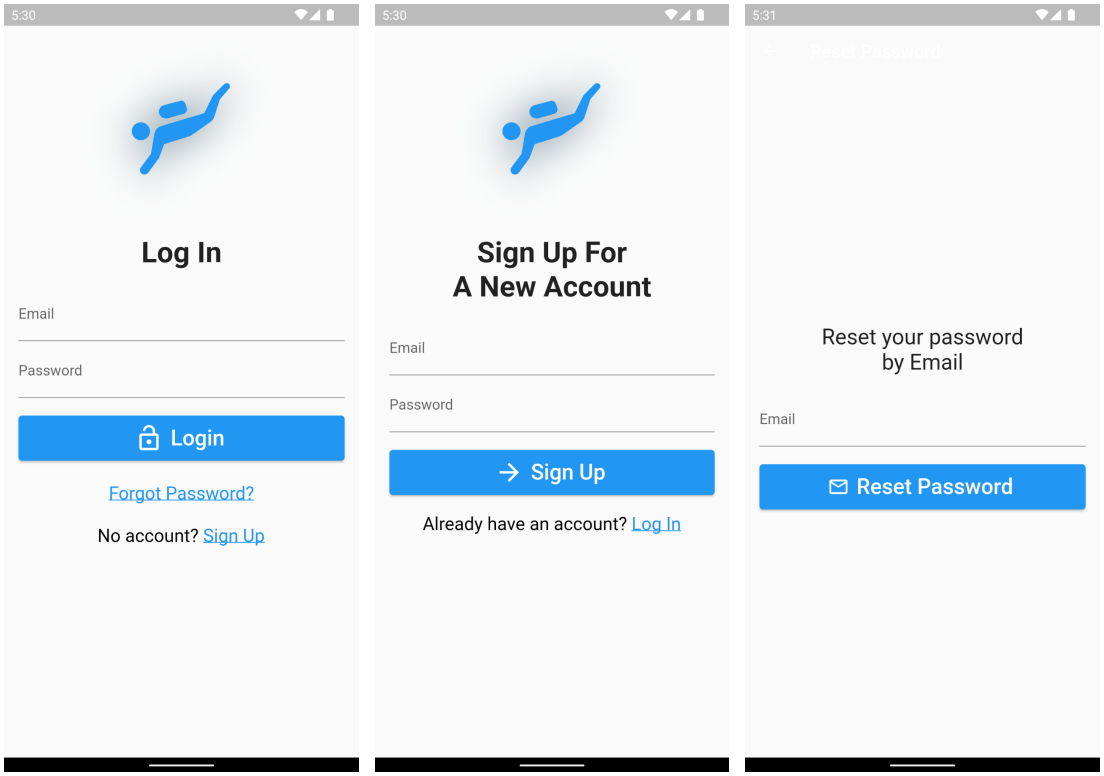


Figure 2: The authentication screens - Log In, Registration, and Forgotten Password

Once logged in, the user will not need to log in again on subsequent uses of the app unless they have logged out. On successful login, registration, or subsequent use of the app, the user will be shown the 'home' screen.

From here the user can choose to create a new dive log or view a list of previous logs in their 'logbook'. The AppBar at the top of the screen gives the user the option of changing the settings (i.e. allowing the user to choose which sections of the log will be visible and in which order they will appear), and logging out from their account, which will take them back to the login screen.

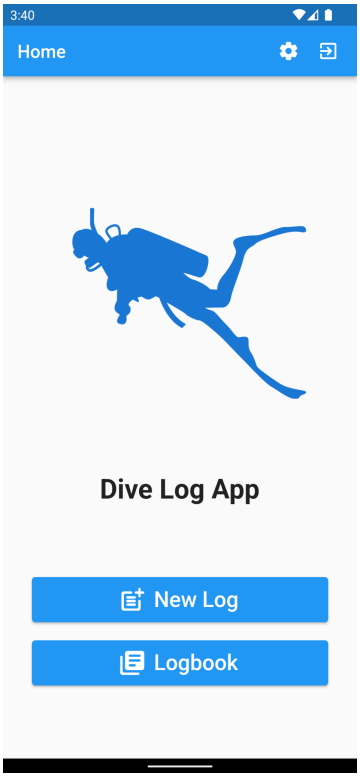


Figure 3: The Home Screen

When creating a new log, the user is shown all the sections of the log they have chosen to be visible (all are visible by default). Currently the Dive Number is calculated by incrementing the 'diveNum' variable in the singleton class, though this would likely change in future development of the app. The default 'date' value is the current date. When the user taps the 'save' button in the AppBar, the entered data is saved as a flat map to the Firestore database. Each field in the log is given a unique name making it easier to both write to and read from the map when needed.

Even if a section of the log has been hidden or no data has been entered in the field, an empty string or other appropriate data will be stored for that field, giving each log a consistent structure.

Due to Firestore being a NoSQL database, there is no requirement that all the logs be in the same format, regardless how many of the fields are filled out. However, with the way that Firebase Firestore counts reads and writes, there is no disadvantage to sending/fetching a map full of empty strings vis-à-vis a completely filled out log. Keeping a consistent format also simplifies handling the data on the front end.

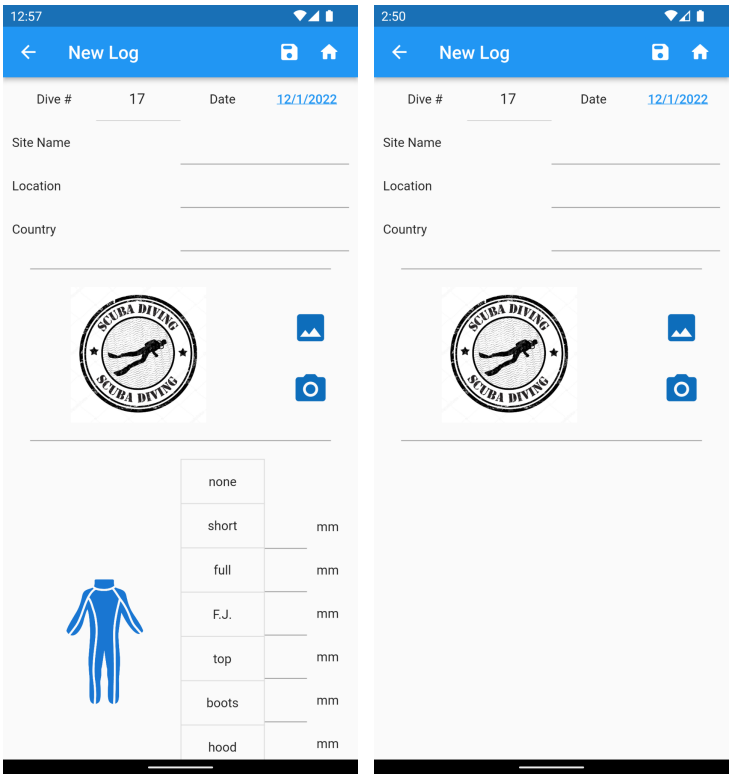


Figure 4: The New Log screen with multiple sections visible (left) and with only the stamp section visible (right). The top section (Dive#, Date, etc.) is always visible to the user.

The stamp section allows the user to choose either an existing image on the phone or to take a picture with the phone's camera. Having both the option of taking a new picture or using a previously saved image gives the user further flexibility regardless of whether their dive shop has a logo or scanned copy of their stamp available already or if the user just wants to take a picture of a stamped piece of paper.

When the log is saved, the image path is saved as part of the log map which is added to the database. The image itself is saved both locally in the Application Documents Directory, which is not readily accessible to the user, and to Firebase's Cloud Storage. The path of both of these files can be reconstructed using the path string that was added to the log map.

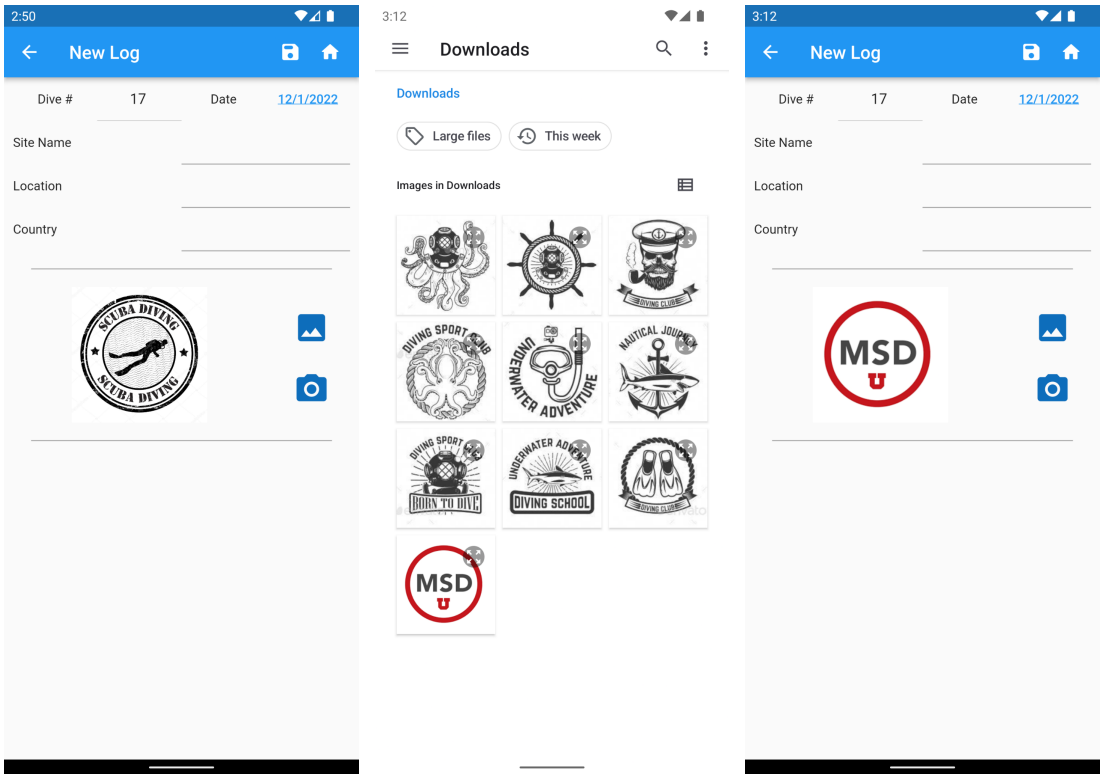


Figure 5: Left - the stamp section with a placeholder image, Center - the view of the image picker allowing the user to choose an image from the gallery, Right - the chosen image in the stamp section of the log

When viewing the logbook, the user will see a list of small tiles containing a brief summary of each dive, including the dive number, location, date and smaller version of the stamp associated with the dive (if there is one). If any of these fields has been left blank when the log is saved, the string "unknown" will be shown in its place. If no logs have been previously saved, a message indicating as such is shown. The ListView of the log tiles is generated from a stream from the Firestore database. Any changes to the database are automatically reflected on the LogBook screen.

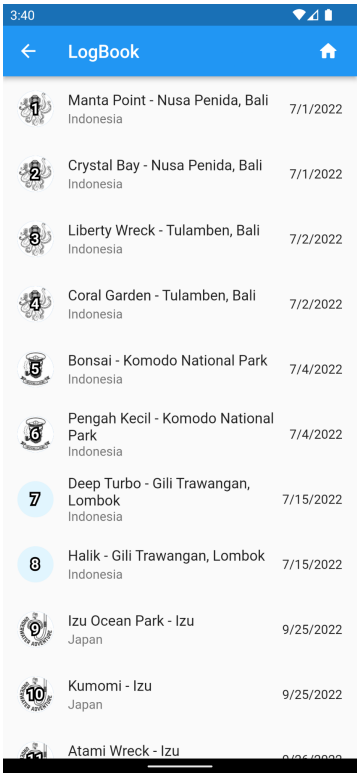


Figure 6: The LogBook screen, showing a list of previously saved logs

When the user taps on one of the logs, they will be taken to the log screen, where the fields are populated by the previously saved data for that log without the need for an additional read call to the database. The user can edit the data and tap the 'save' icon in the AppBar to update the log in the database. Additionally, the user can tap the 'trash' icon, which will delete the log from the database.

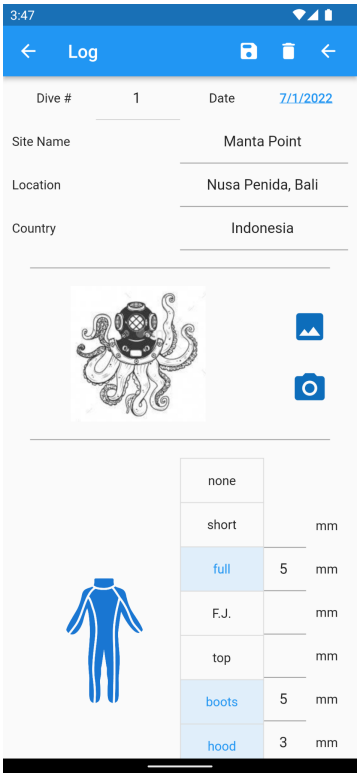


Figure 7: A previously saved log, with the option to update or delete it

From the Home screen, when the user taps the 'gear' icon in the AppBar, they are taken to the settings screen. Here the user sees a list of all the available sections of the log. The section widgets appear slightly grayed out as a visual indication that they are not looking at the normal Log screen. At the top right of each section widget is a label for that widget and an 'eye' icon indicating whether that section would be visible when viewing or editing a dive log.

Tapping on the 'eye' icon toggles the section widget as visible or invisible. Visible sections are grouped together at the top of the layout and invisible ones are grouped at the bottom. When the user makes a widget invisible, it is moved from wherever it is, to the top of the group of invisible widgets. When the user makes a widget visible, it is moved to the bottom of the visible widgets.

The first widget in the layout, which in my code I have called the 'main section', does not have an 'eye' icon, so the user cannot make it invisible. This prevents the user from seeing a completely blank log form when they go to add a new log or view a log from the logbook.

When the user long-taps on a widget, they can then drag the widget to a different place in the layout. The section label provides a natural target for the user to touch as they long-tap and drag.

There are some restrictions on which widgets the user can drag, and where. The 'main section' widget cannot be moved from the top of the layout. Any widgets that are marked as invisible cannot be moved. If the user attempts to move the 'main section' or an invisible widget, when they release the widget in its new, invalid location, the widget will simply pop back to where it was originally. Finally, any widget marked as visible cannot be moved to a position below an invisible widget. If the user attempts this, the widget simply moves to the bottom position of the group of visible widgets.

Currently, the layout settings apply to all logs. In future, it would be better to give each individual log its own layout, with some kind of master layout so that the user can define a default layout for themselves, but then alter individual logs as needed.

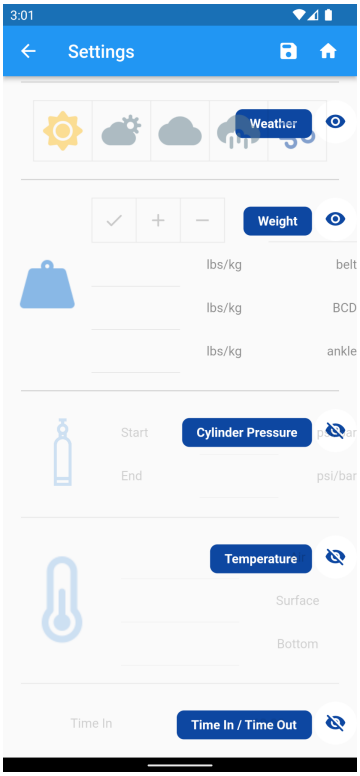


Figure 8: The settings screen with section widget labels and visible/invisible indicators

RESULTS

SUCCESSES

Both of the main goals of the project were achieved: customizable log layouts and stamp functionality. In addition, the integration of Firebase into the project allows the user-generated data to be stored remotely.

Users can add logs of their dives, which are then stored remotely in a Firebase Firestore database. The logs have a lot of flexibility in terms of their layout, allowing users to personalize them to their individual needs and preferences.

Included in the dive log entry form is the ability for the user to choose an image from their phone or to take a picture with the phone's camera and use that image

as a stamp commemorating the dive. The image is saved locally on the phone and remotely in Firebase Cloud Storage.

A list of all the user's previously saved dives can be viewed in their logbook, which is displayed as a stream from the Firestore database. If they wish, the users can edit their previous logs and tap 'save' to update the entry in the database or they can delete the log altogether.

LIMITATIONS

The most significant limitation of the app as it currently stands is that it requires internet access. With more development time, I am confident that I would have been able to more effectively handle the potentially long time some of the asynchronous calls to Firestore and Cloud Storage take. The stamp images are backed up remotely, and it is possible to download them again, but at the moment all the stamp images being shown in the logs are from the local application directory where they were saved when the log was created. Additionally, there is no local backup of the dive logs as they are displayed by a stream from Firebase. One fix for this would be to keep a local copy of the logbook that could use the stream to listen for changes and update itself accordingly instead of displaying directly from the stream.

Another limitation is that the dive number for each dive is generated at the time the log is made and cannot be changed. This was needed to allow for my initial implementation of writing and updating of logs to the database. Unfortunately, this means that it is possible for logs with higher dive numbers to be assigned dates that are previous to dives with lower dive numbers. In other words, the dives might be out of order. It also means that when a dive log is deleted, the dive numbers of the remaining logs do not adjust accordingly. To fix this, I would change the way that dive logs are added to the database to use their date instead of their dive number. I could then generate the dive numbers on the fly based on where the log falls in a chronologically ordered list of dive logs.

CONCLUSIONS

DESIGN CHOICES

The choice of using Flutter to develop the app was a good one. It is nice to be able to run the same code base on both Android and iOS phones. And with some seemingly minor changes, it would be runnable from a browser as well, giving users even more flexibility in where and how they view their dive logs. Flutter has fairly good documentation and a lot of useful and accessible libraries.

Firebase for the backend was also a good choice. It was easier to integrate into the project than AWS was into our Android development projects. One of the more difficult things about using Firebase, however, is how much and how quickly it has been changing over time. This presented a challenge when learning how to use Firebase because a lot of the tutorials and other instructional material was out-of-date.

FUTURE WORK

Going forward I would like to continue working on this app. As mentioned in the limitations section above, the first things to address would be handling the long asynchronous calls and refactoring the way that logs are stored in the database so that they are stored by date and time rather than by dive number.

With regard to additional features, in addition to forgetting to bring their logbooks, divers also very often forget their certification cards. I would like to add a feature that allows users to store a catalog of their certification cards, so they never miss out on an exciting, advanced dive because they forgot to bring the appropriate certification with them.

I would also like the user to be able to use their phone's geolocation or to put a pin in a map to indicate a dive location rather than enter it manually, which would be helpful, for example, if diving at a site that does not have a name.

Finally, in keeping with the theme of user-defined layouts, I would like to add the ability for the user to create their own widgets so that they can make their logbook as useful to them as possible and thereby encourage my own ultimate goal: more divers keeping logs of their dives.