## INTRODUCTION

Keeping grades consolidated is not easy when there is not on solution that ticks all the boxes. This leads to tutors using all sorts of different methods for keeping track of attendance, whether that is the convenience of just writing on paper but the downside of having to be digitised later or using Mylo with grade being entered electronically from the start this has the big benefits of consistent formatting and having all the grades in a central location, but with its clunky UI it is not worth for some. So, how do we take grades in a way that is convenient, but also consistent and consolidated? Well, an app of course! (Or maybe even three!).

For this app to be a viable option for tutors they need to be able to enter and remove students, quickly and easily mark the students, be able to change what type of mark they are storing in each week and be able to share this data to other applications. So, in this report I will compare and contrast the prototype mock-up and implementations of just such an app in Kotlin, Swift and Flutter made by yours truly.

## USABILITY TESTING AND USABILITY GOALS

The biggest piece of feedback I got during the user testing in the first assignment was that it was not obvious that grades lists could be clicked on the take the user to another screen to update a student's grades, But I fixed this issue in the second version of my prototype which can be seen as the first image in the visual comparison section of this report. This feedback became even more redundant when I realised that I could put the grade changing in the grade lists themselves which are much more obviously interactable than a simple row in a list. So overall the usability testing helped to show me that I was on the right track with my app and encouraged me to make some the previously mentioned efficiency improvements.

The main design goal that I focused on for all these apps is efficiency, which I think my iOS and Flutter apps both nailed. They both start on a screen that can take the user straight to the current week which is the most common use case for an app like this. They also allow grades to be edited straight from the Student and Week Details page meaning the current weeks grades can be changed in 2 clicks, one to get to the current week from the starting screen and another to change a grade on the Week Details page the user is now on. One goal that my Android app managed better was feedback. All of my apps feature feedback in most locations where it makes sense such as on the add student popup while the image is loading all of the apps inform the user to wait until the image is finished uploading and disabling the add student button for the apps that upload the image straight away so the user can't accidently interrupt the upload by force finishing the add student process. Another good example of feedback is when updating a students' details (first or last name, or student ID) in my iOS and Flutter apps (as student details can't be edited in my Android app) give feedback in the form of an alert dialog and snackbar respectively. But neither of the two previously mentioned apps give feedback on when a students' grade is updated. This feature only made its way into my Android app, making it a bit better for this particular goal. All of my apps allow offer some forgiveness in the form of a confirmation prompt when trying to delete a student, with clear indication of which student they are trying to delete to both make sure they are actually trying to delete a student, and if they are, make sure they are trying to delete the right student.

Overall, I think my flutter application is the most usable of my three app. It is the most fully featured of the three apps as it includes all the specified features in the spec unlike my previous apps and is the most bug free. My iOS app has a bug that occasionally pops up after a failing to convert the maxScore for the score grade type to a double for calculations after changing a week to this grade system. My Android app is missing a few features, including editing student details and the jump to current week screen present as the first screen in both my iOS and flutter app. It also has a problem on physical devices of rendering the grade selectors wrong in the StudentDetails screen due to the hacky way I populate the table view and also triggering some of the grade selectors as soon as the screen opens. All these problems make my Android app much less pleasant to use due to the issues with the grade tables and is missing a key efficiency feature from the other two apps with the opening screen in those two apps, overall making this to least usable in my opinion.

## PLATFORM COMPARISON

## IDE

Overall, I though all three IDEs I used for these three apps (Android Studio for Android, XCode for iOS and Visual Studio Code (VSCode) for flutter) we're all perfectly functional and fairly feature rich. My personal favourite of the three is VSCode as it is a much more general IDE and not made for a specific language or platform like Android Studio and XCode, though it is worth noting that I use VSCode for nearly all my development that doesn't require a specific IDE, so I am most definitely biased towards VSCode.

I think out of the three IDEs I think XCode was the one I had the most trouble with, though this is compounded by the fact that the iOS app was the first time I had even done serious work on a Mac before, and it was also done on an oldish dual core iMac as well which soured the experience a bit. I had some serious issues with linking my tablecells to their respective layouts in the storyboard due to XCode not autodetecting the correct classes for the tablerow and not letting me display them next to the storyboard for linking. I did eventually fix this issue but it as one of the biggest issues I've had with an IDE before. It was made more annoying by the fact that I had no issues like this with Android Studios' design view, though this could be due to Androids' xml-based layout system allowing changing things visually and in the underlying xml file. On the topic of designing the UI I think I liked the instant results of hot reloading in VSCode the most, which was kind of surprising to me. I was expecting to like Android Studios' approach of allowing both code and visual based UI design to be my favourite, with it being a sort of middle ground between XCodes' almost entirely visual based designing and flutters entirely code-based UI designing but being able to see the UI change basically instantly with the state maintained made it hands down my favourite as it was a lot more useful than XCode and Android Studio just showing what your UI looks like in theory.

I also thought that XCode didn't really make efficient use of screen space. The property panel took up to much space in my opinion

## LANGUAGE

I ended up really liking Swift paradigm of having an optional second name for a function parameter that is used to make a function sound more like a function (such as updateGrade(for: student, inWeek: week, to: 100) for my assignment). I think that this is really cool as it can help get a lot more of a feel for how a function works and make it a lot more obvious what a functions' inputs are. In a similar vein, Flutter's widget system ended up really clicking with me and I ended up enjoying using it a lot. It had a lot less boilerplate code than the other languages in my eyes (at least in the way I used it), and the official documentation for Flutter was by far the best of the three. I think something that might have helped with that is in Flutter we weren't really using any super new functionality like databinding in Kotlin and Storyboarding in Swift, so there was a lot more documentation for the features I was using.

I think one of the biggest logic differences in my three assignments was how I updated the grade average in the student lists and week lists of each app. In Kotlin I was able to just call the function straight in the class that had the average due to the hacky way I constructed my TableRows, in Swift I used a Delegate that was passed to the cell when the TableView dequeues the cell and in Flutter I just used a callback function that was passed to the ListTile. I think out of the three particular approaches I used, I think the iOS Delegate was the best approuce as it meant that I could easily add more functions by just defining them in the delegate and implementing them in the parents ViewController, where as in Flutter I would have to add an extra parameter for every function I wanted to pass in and in Android I just wouldn't recommend as it relies on my hacky table code.

## FRAMEWORK

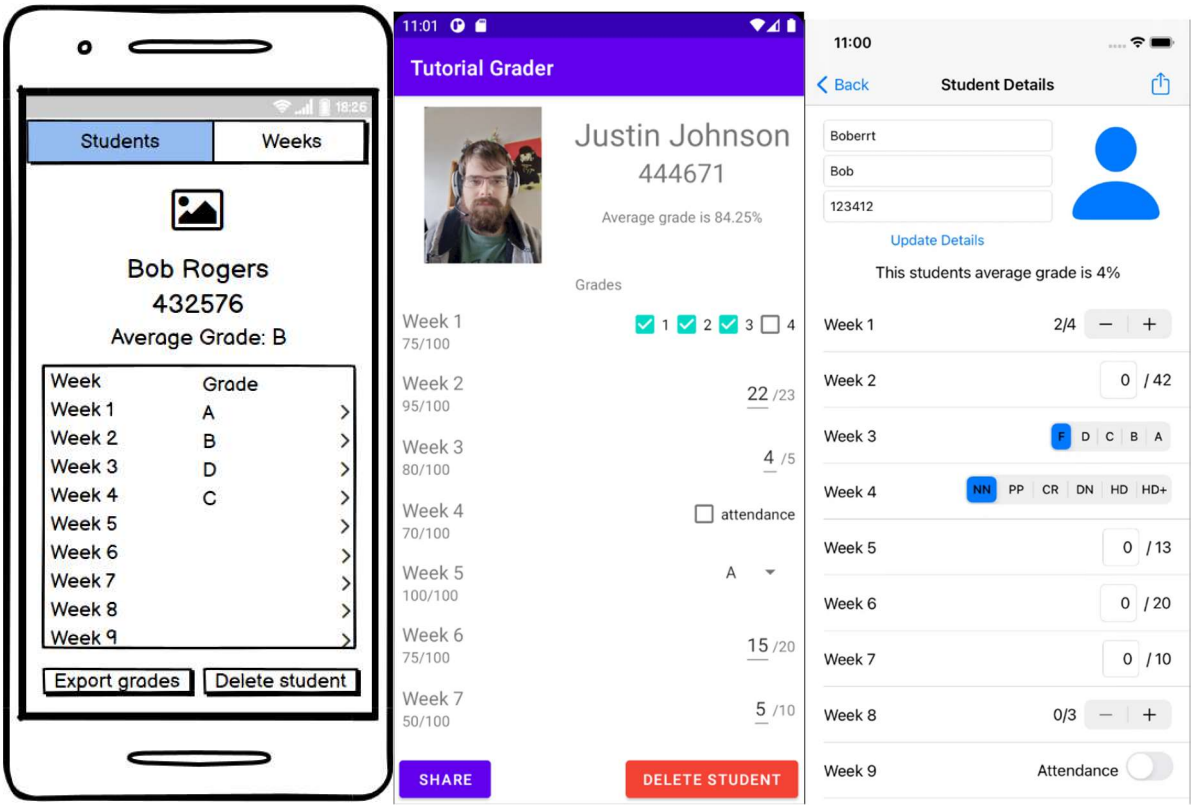
I was not the biggest fan of swifts handwavy approach to data transfer between screens with its segues. While it was perfectly functional and I got used to it by the end, it just felt it kept too much from me and as a result I don't really understand how they really work, just how to use them. On the flip side I think I liked Flutters' approach to transferring data between screens the most, with it just simply being calling the widgets constructor with the data you need to pass to it using push, and to get data back simply just having to await a result from the

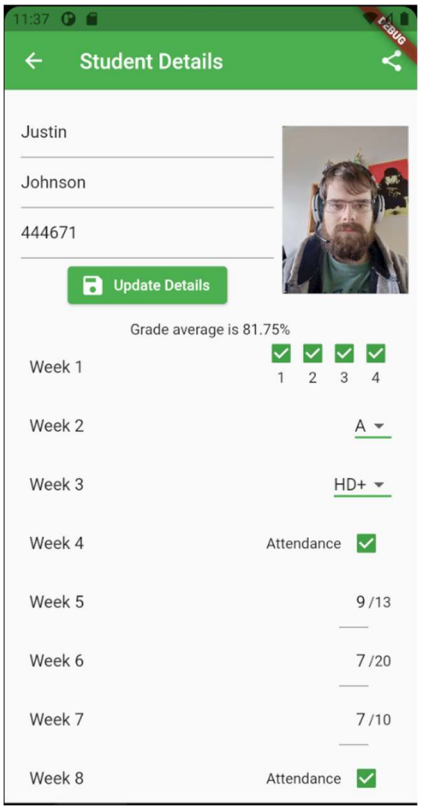
screen and then popping with the data that you need to send back. Androids' approach was pretty good as well, I just liked Flutter basically having all the steps in one unlike Android which make an intent then put the data into it then start the activity and getting data back was a bit more fiddly as well. I think all three languages had a good approach to Tables and I didn't really have any gripes with any of them, though I did like Flutters much simpler approach with a lot less boilerplate code made the code look much nicer and cleaner. Another framework issue I ran into is forcing the size of an image in Android. I had no problem with this in Flutter you can just specify the max height and width in the image constructor and in Swift you can do similar. I was also kind of annoyed that Swift doesn't have a version of a checkbox, unlike both the other languages, I ended up working around this issue, but it was annoying none the less.

## FIREBASE

After getting used to the way to think about using firebase in Kotlin, I was pretty fine in both Flutter and Swift. I don't really have any big gripes with how firebase is implemented in any of the languages. I was more of a fan of Flutters async/await approach as opposed to Kotlin and iOS' call back based implementation. This might be because I've used a bit of Javascript using a similar async/await paradigm, but I didn't hate the call back approach either, I just think await/async looks a bit cleaner and is easier to understand to me. Apart from that the changes between implementations were pretty minor in my eyes and didn't really trip me up. The sort of MVC system that we used in flutter though I think was probably a better more scalable way of using firebase compared to my implementations in my other apps as it allows any one of the segments to be changed independent of each other so code can be more easily be repurposed in other projects.

## VISUAL COMPARISON

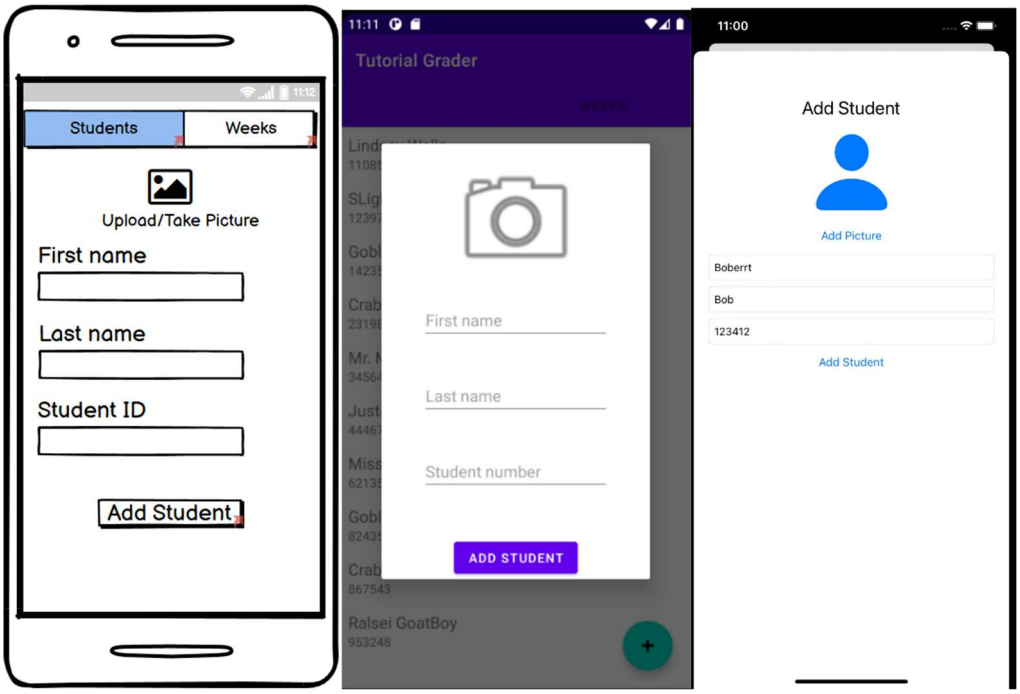


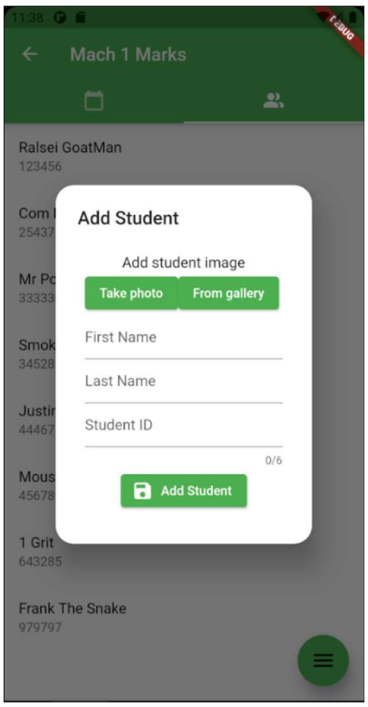




Shown from left to right, top to bottom, Prototype, Android, iOS and Flutter Student Details screens.

The Student Details screen stayed very consistent from the start of semester to the end, with some notable differences being the prototype not allowing the user to change a students' grades from this screen, instead navigating to a not pictures change grades screen, and the Android version not allowing the editing of the students' name or ID. I think overall iOS uses a much more flat design pattern which it mentions directly in their Human Interface Guide lines stating "Minimal use of bezels, gradients, and drop shadows keep the interface light and airy, while ensuring that content is paramount" (Apple, 2019) while the material design style used in both Android and my Flutter app focus more on adding depth with the use of shadowing on common parts of the interface like buttons and popups. Apple also strongly suggest only using colour sparingly to highlight important feature and indicate intractability, this difference in design philosophy can be seen with the only use of colour in the iOS app being on the back button and share icon, as well as the update details button and grade selectors. My Android and Flutter apps in comparison make a bit more liberal use of colour with both the Flutter and Android apps having coloured title/app bars as well as having much more prominent buttons in colour as well. I don't think this design paradigm makes much difference on this particular screen and is more to stay in like with other applications on these platforms to hope to align with users' mental models of how apps are supposed to function on the OS they are using. I also made much better use of the navigation/app bar in Flutter and iOS in which both have the share buttons up in the right of their respective

bars to help remove some of the clutter from the Android app of having traditional buttons beneath the grade list/table and give some more space for the grade list/table to display more grades. Out of these 4 screens I think both Flutter and iOS are tied for the best in both visual style for their platforms (with Flutter being a better example of material design than the Android app ironically) and layout in general. They both don't have anything they don't need and make good use of the space provided without sacrificing on any functionality. The original prototype wastes a fair bit of space by stacking the students' information vertically and restricts the grade list, which is the main focus of this screen. It also loses points for not having grade changing on the same screen reducing efficiency. Android isn't too far behind Flutter and iOS, but I just have some grips with it that prevent it from being a three-way tie, such as the previously mentioned poor use of the app bar and the wasted space of having extra buttons at the bottom of the screen that can be moved to different spots or moved to different screens in the case of the delete button.




Shown from left to right, top to bottom, Prototype, Android, iOS and Flutter Add Student screens.

The Add Student popup is probably one of the most consistant elements of my apps, making it a great point to comapre platform differences with. The biggest difference is the prototype being a normal screen as compared to some sort of popup, this design was mostly due to the fact that I wasn't sure what any of these languages were capable of and didn't really know if I could do custom popups. We can also see here a big difference between iOS and the other plastforms, with iOS using more a sliding sheet over the original screen comapared to Flutter and Android using greying affect on the original screen as well as a shadow on the popup to really emphasise that this popup is above the original screen. There are deffinityl an advantage to the iOS approach if you need a larger custom popup as iOS gets edge to edge comapared to Android and Flutter having significant padding on all sides of the popup.

# THE FUTURE

## IMPROVEMENTS

My iOS and Flutter applications could both use some more feedback when updating grades like my Android assignment had so the user knows if the students grade actually changed or not. This could be done in Flutter with a snack bar and in iOS I would need to find some kind of equivalent. To add on to this I could also add some forgiveness with the students grades by allowing the user to undo grade changes as they are easy to make with the grades all being changeable on the same screen so it can be easy to accidently change a grade while scrolling. This could be done by adding an undo button the previously mention snackbar and iOS equivalent much like Gmail does. I could potentially improve the efficiency of my Android app by removing the student delete button and replace it with the swipe to delete functionality I have in my iOS and Flutter apps.

## EXTRA FEATURES

- Multiple classes.
- A login system to keep track of which tutor made changes of grades and to only show classes they teach.
- Direct integration with Mylo (if possible).
- Give the option to give a name/title to each week to better indicate which tutorial it is if a tutor cannot remember which week which content was done in.

## PUBLISHING THE APPS

### ANDROID

First, I would need to set up a Google Developer account which requires a one-off fee of $25. Then if I wanted to monetise the app in any way, be it in-app purchases or make the app paid, I would need to link a merchant account to manage app sales and monthly pay outs, as well as seeing stats about the previously mentioned figures. Next, I would create the app in the in the google console, this just creates the app in my console to be properly prepared later. This step just requires the title of the app (not final at this step) and the default language of the app (D. Oragui, 2018). After this it's time to prepare the store listing. On this step we set up the store page for the app which contains a title, short and long description, an app icon and 2-8 screenshots of the app and the categories to put your app in. After this is the big moment uploading the app bundle which can be generate through Android Studio. We can choose to upload to one of three states: alpha, beta or production. Alpha means that the app can only be access by users added to an approved user list, so this can be used for private app testing to get feedback from specific users make it good to get a lot broader feedback on the app from a many more different types of users, but users are not obligated to give feedback, so it is worth thinking about if the app is ready for public testing access first. Beta means that the app is open for all to test download, but is label as a pre-release version of the app. Then there is production which is the finished or at least public ready app, publishing in production requires the app to go through a human review from google first which can take a week or two. The next step is deciding on what type of monetisation strategy is going to be used, is the app going to be free or paid, and if the app is going to be locked out of any regions. Now finally we can release the app to the big wide world.

### IOS

To publish an app to the I first would need to get an Apple Developer Account which is US$100 a year. Then we need to create a new identifier, in our case a Bundle ID. This id is recommended to follow the format "country abbreviation.com.organisation.AppName" (M. Azolin, 2019) so for example au.com.utas.Mach1Marks. Next,

we step into XCode, give our app this identifier and archive the app to start processing it for upload. Next, we got back to our developer account and add a new app, giving it a name, primary language and the bundle ID we created earlier. Now we set the subtitle of the app and other information like the categories the app belongs to and a privacy policy URL. Now we set the price (or lack there for of) of the app and which regions the app is allowed in. Now before we submit the app for its' first round of review, we have to add some screenshots, specifically we need "a 5.5" display (iPhone 8 Plus), a 6.5' display (iPhone XR) and a 12.9" display (iPad Pro)" (M. Azolin, 2019). Once all this is done, we can save and submit, after submitting most apps get reviewed within 2 days.

## CONCLUSION

To summarise this past semester, I first made a prototype of a tutorial marking app in Balsamiq and got some user feedback and made improvements after doing run-throughs with 4 different users. After this I first started with a Android app programmed in Kotlin. This app was a solid first attempt at an app having nearly all the required features but with some bugs that affected user enjoyment of the app. Then I moved on to the iOS implementation in Swift and came out the other side with a very slick app that was only missing one extra feature and had one annoying type-casting bug. Lastly, I made a Flutter app and seeing as I am now a grizzled mobile developing expert this app came out with all the base and extra features and doesn't have any critical bugs like the previous two apps. This app probably came out best due to both being my third time round the block and also being the language that clicked the most for me.

## REFERENCES


1. themanifest.com. (D. Oragui). How to Publish an App on Google Play: A Step-by-Step Guide | February 2021. [online] Available at: https://themanifest.com/mobile-apps/how-publish-app-google-play-step-step-guide [Accessed 13 Jun. 2021].
2. orangesoft.co. (A. Kharychkova). How to Publish Android App on Google Play Store in 10 Steps. [online] Available at: https://orangesoft.co/blog/how-to-publish-an-android-app-on-google-play-store [Accessed 13 Jun. 2021].
3. Apple (2019). Themes - iOS - Human Interface Guidelines - Apple Developer. [online] Apple.com. Available at: https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/.
4. MapInTime. (2013). Don Norman's Design Principles. [online] Available at: https://williamgrimes12.wordpress.com/2013/01/22/don-normans-design-principles/ [Accessed 12 June. 2021].
5. Azolin, M.F. (2020). How to: publishing an app to the App Store 2019 Guide. [online] Medium. Available at: https://blog.usejournal.com/how-to-publishing-an-app-to-the-app-store-2019-guide-1c73a582136c [Accessed 13 Jun. 2021].