

# GoGig – iOS App

COMPUTER SCIENCE PROJECT  
LEE CHILVERS

SUFFOLK ONE 19216

## Contents

<b>Section 1 – Analysis of the Problem</b> .....	3
What is the Problem? .....	3
Computational Methods and Amenability .....	3
Stakeholders.....	4
Appropriateness of Solution.....	5
Conversation with Stakeholders .....	5
Existing Solutions .....	8
DFD Analysis of Facebook Business Solution .....	10
Software and Hardware Requirements .....	11
Specification of Proposed Solution .....	12
Limitations of My Solution.....	13
Success Criteria and Objectives .....	14
<b>Section 2 – Design of the Solution</b> .....	16
Top Down Design/Stepwise Refinement .....	16
Core Module 1 - Authentication.....	16
Core Module 2 - Portfolio .....	17
Core Module 3 – Create Event (for organisers).....	18
Core Module 4 – Find Gig (for musicians).....	19
Core Module 5 – Review Activity .....	19
Usability Features .....	21
Interface Designs.....	22
Data Structures.....	31
Validation Rules .....	32
Database Data Dictionary.....	33
UML Diagrams.....	39
Algorithms .....	43
Test Strategy .....	59
Milestone Plan .....	59
Milestone Test Plans.....	61
Stakeholder Testing.....	71
<b>Section 3 – Developing the Solution</b> .....	78
Milestone 1 – Login/Signup Views and Authentication.....	78

Milestone 2 – Portfolio and Post Views.....	94
Milestone 3 – Create Event Views (for organisers) .....	105
Milestone 4 – Find Gig View (for Musicians) .....	114
Milestone 5 – Sotring Gigs by Location .....	121
Milestone 6 – Display Event in Detail View and Observe Portfolios .....	129
Milestone 7 – Account Notifications View and Send Notifications .....	135
Milestone 8 – Review Application (for organisers) .....	145
Milestone 9 – Edit Account .....	151
Milestone 10 – Edit/Delete Events .....	157
Milestone 11 – Device Push Notifications and Social Media Links.....	164
Milestone 12 – Constraints for all iPhone Devices and User Interface Finishing Touches ..	175
Adding Events to the Calendar and iOS 13 Maintenance .....	183
<b>Section 4 – Evaluation .....</b>	<b>188</b>
Stakeholder Test Plan.....	188
Questionnaire .....	196
Evaluation Against Success Criteria.....	197
Evaluation of Usability Features .....	200
Maintenance.....	201
Future Developments .....	201
Conclusion.....	202
Final Source Code .....	203

# Section 1 – Analysis of the Problem

## What is the Problem?

As a musician, who has played in a few different bands, there comes a point where you (and the band) want to start getting paid for your performances. However, I have always struggled to get a gig despite the number of people that come to watch us perform at high school and college concert events and I am sure many other young musicians would agree with me.

Getting gigs and starting out is a difficult task, even for talented musicians who have been playing for years, and I think this is because many people wanting to play have a lack of contacts to help them get the job so other musicians snatch up the job first meaning they never get the opportunity to show off their music to potential clients.

One solution to this problem is marketing the band: building a web page, creating social-media accounts and paying for advertising on these platforms. However, this plan of action can be expensive and still fail to be successful because there is no way of knowing if your adverts are reaching people that require music at an event. A computational approach to this problem could be to create a program where both musicians and event organisers use the same system to interact, so deals are created in a simple and clean way. Organisers would use this solution to input a gig in a database, which local musicians would then receive when outputted and apply for, giving them an opportunity to show off their music and work. This would solve the issue of organisers needing to write down who has applied for the job with pen and paper as it would all be recorded automatically with the use of a server and provides them with a view to easily decide who is most suitable to what they have organised.

Computers can also notify its users with e-mail/text/social-media messages; this may be applicable so that users are updated quickly when new data is pushed to a database in real-time – quicker than writing on a sheet of paper. Therefore, what I want to aim for is a computational solution which bridges the gap of contact between event organisers and musicians, allowing new musicians to get paid for their plays, and to boost an organiser's business.

## Computational Methods and Amenability

The computational methods which could help in a solution such as this:

- The use of divide and conquer to split the main problem (or a large problem) into smaller sub-problems continuously until it can be solved. Because there are two types of people involved in the problem, divide and conquer would be used in this way to present different information to the different users of the system and continue to plan and build by splitting the problem down to how these users would access and use this data differently.
- We can analyse large sets of data in a database to provide us with new information based on a hypothesis. This is called data mining and could be merged with machine learning for an intelligent response to this problem. This analysis would be a lot more accurate (and quicker) than humans manually going through all of the data collected and reaching a conclusion. In this scenario, people could get recommended to each other to remove the time and effort in searching. This is unlikely to be a method used in this project as collecting the data will be time consuming in addition to developing the solution.

- A database enables important data to be stored efficiently and effectively. Because this problem is centred around contacting people, a database opens the opportunity to store personal information privately and securely while providing the necessary data for various people to communicate with each other. A database allows various queries to be made to fetch only certain types of data when wanted, this removes human error when putting together a mailing list for example. A database also means that data does not have to be stored in the secondary storage of the device itself. If a user loses or breaks their phone or a computer crashes, their account and data is recoverable due to it not being locally stored.
- Pipelining is where instructions given to the computer system are executed in overlapping stages. The architecture of the computer allows the next set of instructions to be fetched while the processor is still completing the operations from the last set – similar to an assembly line where the first stages of a product are being performed at the same time as the end stages. This means that a computer can complete more tasks in given time – faster than pen and paper – and humans can multitask by allowing the system to complete one job while they start another resulting in them being more productive.
- Using the internet, people can send messages and data to each other on various computers. In their own home on a desktop, or on a smartphone, people can be reached and notified with a computer network. As a solution to this problem, this removes the need for advertising with flyers and posters because potential clients are reached personally and almost instantly. It also widens the circle of people that can contact each other: Without a computational approach, only very local musicians will likely be reached but with a global computer network, most people can be contacted from across the country or even the world.
- Monitors and touchscreens can display a customised user interface which is easy to use. Simple UI can encapsulate large procedures so that users do not have to be specially trained or taught to use the system but can easily learn due to how information is displayed.

## Stakeholders

### Identification of Stakeholders

I have two main stakeholders for this program at this initial stage: musicians and event organisers. Therefore, I need to target two types of end user for this project. A local musician, [Jude Mills](#), manages to play at occasions, such as weddings, with his own rock band fairly frequently - he will be my musician end user. As an event organiser, Gavin Thorrold owns CEG Productions who supply stage and PA equipment for various performances and hold their own concerts (inviting various people to play for entertainment) in order to demonstrate the quality of their equipment and services. I will ask both of them for their ideas on what features should be included in the solution and how these features would help them in their jobs. These ideas will be added to my 'Specification of Proposed Solution' and will hopefully be as useful to other people with the same job/interests elsewhere in the country. This would not be a system for a specific business; therefore, I am the creator and will be the administrator for this project, with my decisions guided by the Special Interest Groups of Jude Mills' band – Red Glass – and CEG Productions.

## Appropriateness of Solution

A computational approach is appropriate for these stakeholders because it makes managing the process of booking entertainment-related deals quicker and easier. As stated previously, text/e-mail notifications mean that these two types of end user can communicate and come to an agreement without having to meet face-to-face. With a server and database, this interaction is broadened to data being shared back and forward. Photo and video information can be shared so organisers can assess a musician's ability and a musician can preview the venue. It makes the entire process easier for these end users because paper and documents do not have to be used, reducing costs and – similar to social media – gives the user access to contacts of whoever uses the service.

## Conversation with My Stakeholders

The aim of interviewing these stakeholders is to reach a conclusion of what they (and therefore similar users) would like to see in the solution of the problem. I think a sensible approach would be to ask about their jobs: whether they've experienced a solution similar to what I am trying to achieve and any ideas of desirable features they would want in the system. I had email correspondence with both of my stakeholders as follows:

What is difficult about your job as a musician/event organiser?

Jude Mills (Musician): Sharing my band's music with people is a challenge. If people don't come and watch us play at a local event, they don't know that we have music on Spotify and Apple Music.

Gavin Thorrold (Event Organiser): When we try to book people to play at a concert, we find it difficult to find new people that are relevantly qualified, in the locality in which we are and not too expensive to hire. Even when we do decide on someone as a company, it isn't always easy to get hold of them.

Conclusion Reached: This question was asked so I can start to build my own personal ideas of how the system could help these difficulties. From this initial question, I can see that Jude wants a way to show off his band's music to people that are interested, this would mean saving the user's audio files and videos and posting them to a database so that other people can view them on another device. Gavin wants to be able to easily find and get hold of people that are suitable for the concert they are trying to host.

Have you ever marketed yourself or your business/group online before?

Jude Mills: Our band has a Facebook and Instagram page where we follow people who may be interested in our music and are likely to share our page with their friends. With social media we can also link from our page to our music on Spotify and Apple Music; people are more likely to find it from Facebook as it appears in friends of friends' Newsfeeds.

Gavin Thorrold: CEG productions have a website and are heavy in social media presence (Facebook, Instagram and Twitter). With online marketing we can view people that like our page and ultimately show our work to people in the hope that they may ask to use our equipment. We try to avoid conventional marketing as this is a very specific and niche industry and we more or less reach our target audience instantly with an online approach.

CEG is first in Google Page Rank when searching for our company, which means our website receives a lot of web-traffic.

Conclusion Reached: Asking this question gives me an idea of what products already exist to help these people achieve their marketing goals. I can research these and take what features are essential to a system such as this. It seems as though Facebook is a popular one, I know that Facebook has many business features available for its users and so might be worth researching these in depth. A key point from both responses is that they like to know who and

how many people have interacted with their pages. We can put these users in some array, store it in the database and display them through a table to the current user as part of my solution.

Did this marketing benefit you or your business/group and how? And what would make the process easier?

Jude Mills: Yes, we suspect most of our listeners on Spotify are because we have shared the link on Instagram and Facebook. In terms of where we play live, the Facebook Event feature means that the time and location of where we are going to play appears in their Newsfeed if they might be interested in the event, and also tells us how many people are planning on coming. But most importantly, it allows the people that are hosting us to see where else we have played, where we come from and how they can contact us. Facebook Messenger and Instagram Direct Message allows us to easily contact and arrange something on a mobile, without having to make many phone calls like I used to do to arrange things.

Gavin Thorrold: Lots of people contact us by email or from contact information found on our website. We have also developed our own analytics program to observe visitors to our webpage which helps us get an idea of where our customers come from. For the purpose of our company, we get a lot of business from online marketing and mainly organise over email and WhatsApp/text – occasionally we get the odd casual business message over Facebook Messenger. However, in relation to difficulty in getting acts for a concert, it would be easier if people came to us asking to play instead of us having to chase up people. We don't really want to advertise us trying to find people on Facebook, this is too public which ordinary people can see and in past experiences we have had people who are nothing like what we were looking for. Something I personally would like to see Facebook push further is letting businesses know how their followers react to a post, rather than displaying basic Like and View analytics.

Conclusion Reached: The answers provided gives me an idea of how the available products help its users. Making use of the device's clock and a map could make things easier as information is displayed in a more helpful format rather than just displaying times and locations with text. Gavin at CEG Productions wants a platform for purely musicians and organisers so that other people do not view what is going on and there is a level of privacy. I think this can also be solved with a login and logout system so that information is saved under an account and secure. Big social media names bring together their users with messaging services and is very popular for contact in business along with emails. I think that social media page links and email addresses are very important pieces of information that need to be displayed to other users in the system so that communication in that way can easily be continued if that is their preferred method.

What type of device would you like to see a computational solution to this problem on?

Why?

Jude Mills: A smartphone would be the obvious option because I take my phone with me most places (suitable for when we go and gig somewhere!) and I can reply to messages quickly from a few taps without having to boot up a desktop. Having said that, when I am in the studio editing a new track for the band, it may be beneficial to have a desktop app which does the same thing so that I don't have to reach for my phone all the time.

Gavin Thorrold: When in the office we usually do most of our work from laptop. Despite that, most of the time we are on a job and travel quite far transporting equipment; therefore we contact mainly with emails on our phone and run the Facebook page through the app on iPhone. I wouldn't expect to see a system such as the one you aim to create on anything

other than a handheld device; it keeps the frequency of interaction flowing instead of “I will reply to that message when I get home”.

Conclusion Reached: After telling them I planned to help some of their difficulties with a computational approach, their answers helped me direct what device I needed to develop on and therefore what most users around the country would prefer to use. Both end users use a laptop/desktop and a smartphone, however both say that they use their smartphone more and explain the benefits of the solution being on a handheld device. With a smartphone, developers have access to tools like the phone’s photos, location services, emails and calendar which may all be beneficial to the solution.

What key features would you want to see in a computational approach to solve this problem?

Jude Mills: I want a place to show off my band’s live performances with people that are interested in having us play for them. It would also be nice to have a link displayed so that they can find us on Spotify, so we get more listens. Ultimately though, the process of getting a gig is difficult if someone doesn’t want you from your latest performance; it would be nice to quickly find people that are looking for musicians to play and be able to seal the deal without hassle (without unrepplied messages and missed phone calls). When applying for a job I would obviously need to know the date they want me; the location, to make sure I can get there; pictures of the venue; the type of music they want me to play and the name/contact information of whoever is in charge. It would be great if dates arranged were automatically sent to my smartphone calendar so that I get a reminder a week before and can avoid any date clashes. Our band has a shared calendar so it updates on our smartphones and we can all see any changes; if the solution could use this in some way, that would be beneficial. As well as this, text notifications from anyone trying to contact me to make changes would be helpful so that I can reply quickly.

Gavin Thorrold: It would be easier if people came to us when trying to organise an event. I want a way to view all the people who have applied and choose ones we most like based on their price and their talent. We would still prefer to use emails or WhatsApp to contact these people but definitely need to be able to see what type of musician or act we are making a deal with. This means we need to be able to see their music skills to allow us time to prepare and know what to expect along with relevant information such as their genre, instrument of choice and relevant experience. It would also be good to post photos of our events so that the applicants can see what we expect from them. Notifications on my smartphone would be a good feature to let me know when someone has applied. I think everyone (or chosen people) at CEG being logged in under one account is important; that way we can monitor and see what decisions are being made between us. What you have proposed sounds like an excellent idea. For our business we would love to see this idea expand to finding new employees (like sound engineers) for CEG, or the buying and selling of various equipment!

Conclusion Reached: This is a direct question to the Stakeholders to get insight on their ideal product and confirms what features are essential and desirable. This system is music related and so (as Jude proposed) should have some connection to music streaming services such as Spotify and Apple Music. The whole concept of the program needs to reduce the difficulty of the ‘backwards and forwards’ emails and conceal a deal in a quick and easy manner.

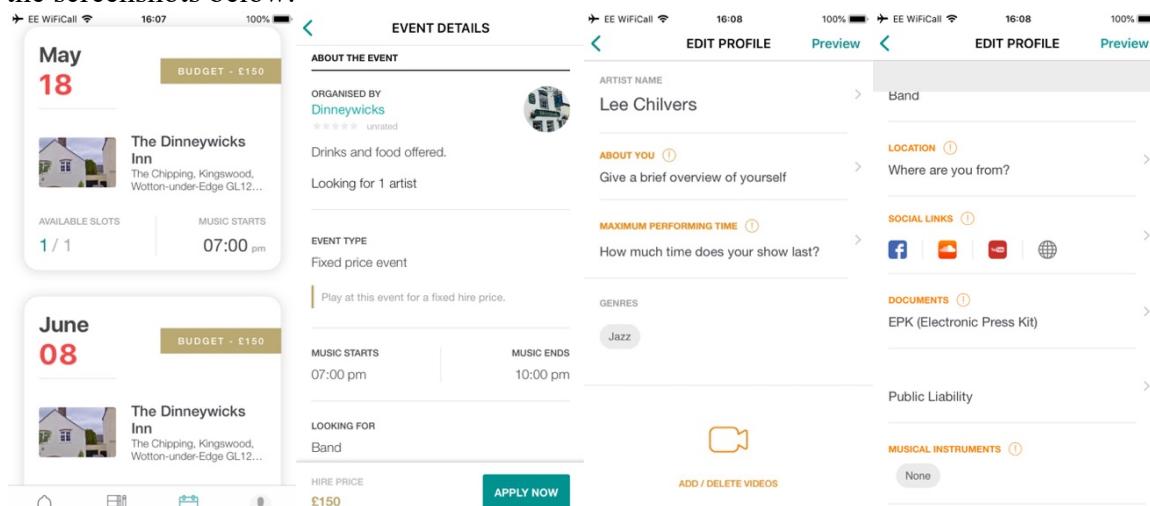
Relevant information of musicians, organisers and the events need to be displayed to the current user so that the system serves its purpose in bridging contact between these two types of end users. Jude would like to see smartphone calendar integration in the proposed system as this is his preferred method of band organisation. Perhaps these Stakeholders would like to see messaging involved in the system, however they are likely to still contact clients through other services and so this is not a high priority feature. Despite this, activity should be notified with an alert when the user is not using the system to encourage them to continue

the flow of coming to an agreement efficiently. I have in mind a portfolio style page where both user-types can post/view photos and videos which is what these people would like to see in a solution. Everything needs to be locked under a login system so that data is secure and associated with an account.

## Existing Solutions

### Melodeal (<http://melodeal.co.uk/>) [Date Accessed: 15/05/2019]

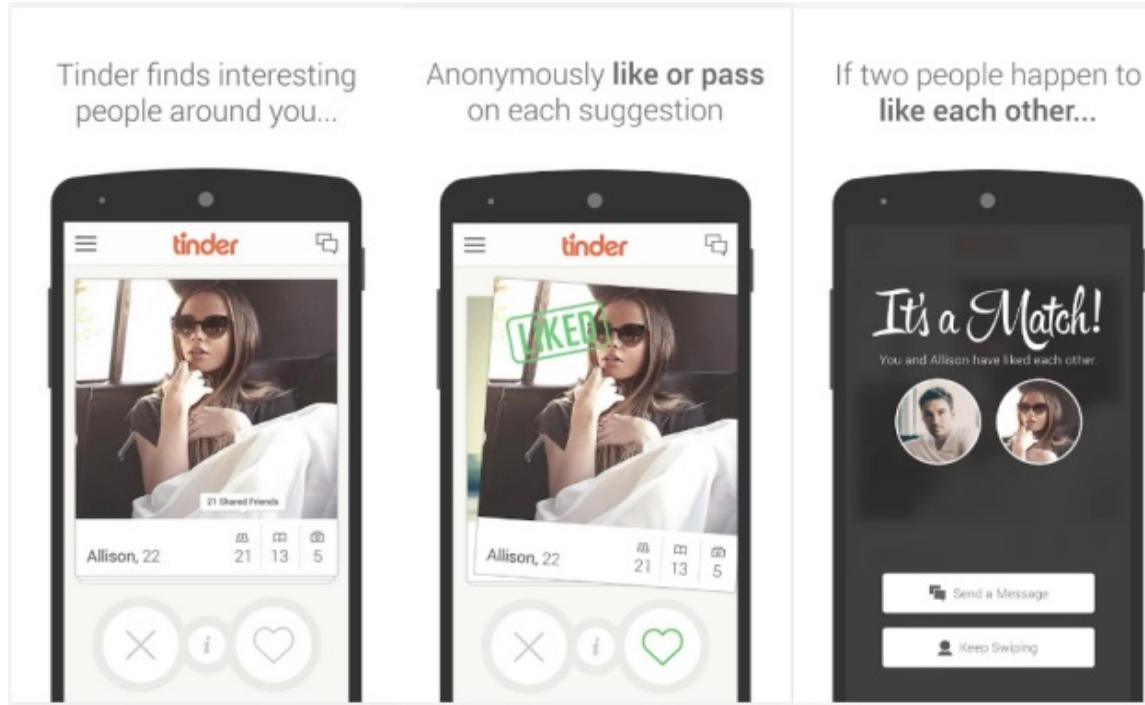
One application which already exists is called 'Melodeal'. This app, available for iOS and Android devices, is claiming to make the booking process of a gig really quick and easy. This is the ultimate result I want to achieve. Melodeal supports organisers in finding the right musician for their business or venue. The app attempts to find musicians quickly, even if one has pulled out at the last minute, and has various features such as push notifications for example when someone applies for a job; filtering methods to categorise musicians into the type of genres they play; location services (with Google Maps API) to pinpoint the location of the gig and rating systems to review musicians. Some of these features can be viewed in the screenshots below:



This is one of very few applications on the smartphone application stores which is dedicated to this exact purpose similar to my problem. Despite this, Melodeal does not have a large user base. Developed in Bristol, Melodeal only provided me with the option to apply for gigs in Bristol after signing up. Therefore, I cannot review how their sorting algorithm works in displaying gigs to a musician after they are pulled from a database. Nevertheless, I aim to use some of the useful features researched through this app, but I also want to make sure that my approach to this problem presents local gigs first to a musician rather than displaying gigs anywhere in the country. This is because I want to focus my solution on helping musicians who are starting out on their gigging career and these types of people are likely to want local opportunities before they begin to travel further.

### Tinder (<https://tinder.com/?lang=en-GB>) [Date Accessed: 15/05/19]

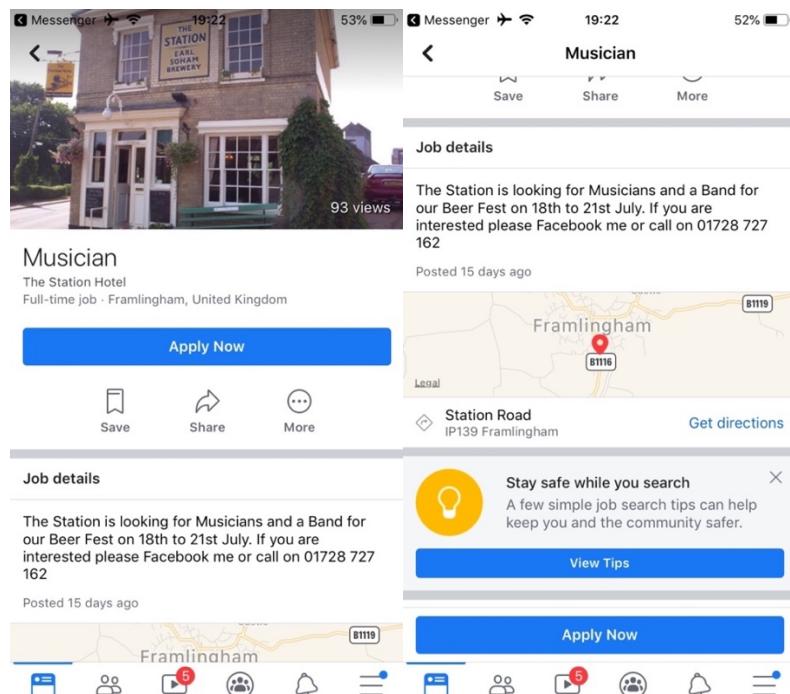
Tinder is popular dating application and a good example of an app which has one type of user interact with a profile someone else has made public. Tinder enhances their user experience with simple swipe gestures on the phone's touch screen. The Swipe Right™ gesture is where a user likes the profile that is displayed to them; therefore, they swipe right for a potential date.



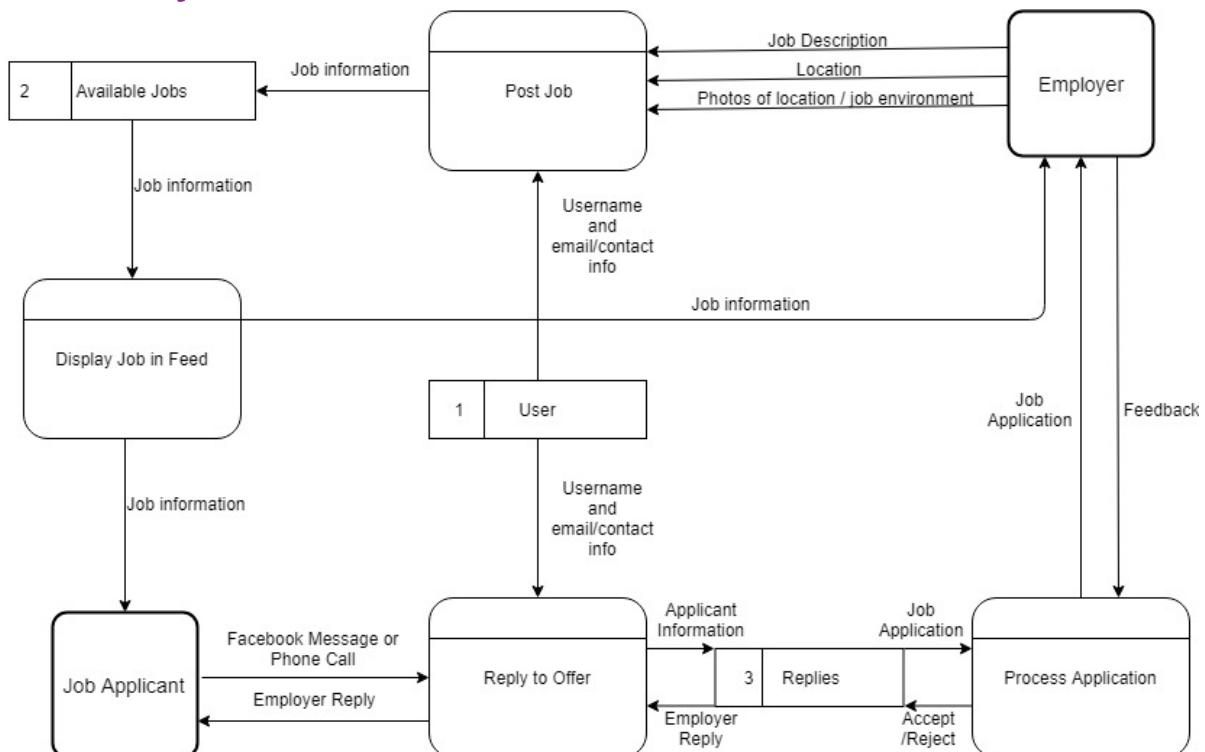
Important information is clearly seen about the user when making a decision, which is essential for any program that would follow this format. I could potentially use this concept of simple gestures to enhance the user experience, making it easier for a musician to apply for a job. In addition to this, profiles appear based on how far away they are from the current user's location. This would make use of the smartphone's location services and reinforces my idea of displaying local gigs to the user first. Like potential dates from Tinder, I think that potential gigs should be organised in a way based on the user's location.

[Facebook Business: Job Descriptions](https://www.facebook.com/business/pages/post-job) (<https://www.facebook.com/business/pages/post-job>)  
*[Date Accessed: 15/05/19]*

As a feature of Facebook, business pages can post job descriptions and pay for this job to appear as an advert on users' feeds. The format of the job description is similar to Melodeal's, although, it is more general to any type of job, not just music gigs. Facebook integrates its messaging service (Messenger) and displays all the different ways of contacting the company in order to apply for the role. Because this feature is merged into the social media platform, there is the opportunity to share the job with other Facebook users and even save the description to read later which Melodeal cannot do. Being able to save things to refer to later enhances the experience of mobile applications because the developer has to consider users that are busy and doing other tasks - operations on a mobile phone should be quick, easy and obviously simple.



## DFD Analysis of Facebook Business Solution



To conclude the research of other products, I think the solution to my problem can be approached with a mobile app. This is because potential job opportunities stay with the user wherever they go and so they can easily be notified with push notifications, making the job application and response process a quick and efficient one. I can also make the user interface really simple to use with a smartphone app, enhancing the user experience by displaying data as and when it is needed (not overwhelming them with many confusing options like a desktop application) and making use of swipe and tap gestures with the touch screen.

## Software and Hardware Requirements

Input / Output – Touch screen

Apple released iOS 11 in September of 2017 and made it so that any new apps submitted for checking on the App Store had iOS 11 as the minimum operating system. This means that in 2019 where new iOS devices can run iOS 13, the minimum hardware requirements for this app are of the iPhone 5 and iPhone 5C which cannot update to iOS 12. The minimum specs are as follows:

### iPhone

Minimum Processor Rate: Apple A6 Chip (1.3GHz dual-core processor)

Minimum RAM = 1GB RAM

Minimum Disk Space = 16 GB

Minimum Display = 4.0-inch retina

However, my end users are on an iPhone X, these are the iPhone specs as follows:

### iPhoneX

Recommended Processor Rate: Apple A11 Bionic Chip (2.39GHz Hexa-core processor)

Recommended RAM: 3GB

Recommended Disk Space: 16 GB

Recommended Display: 5.8-inch ‘super’ retina

At Apple’s Keynote Event WWDC, the Swift UI framework was introduced which makes development of user interfaces quicker. However, this framework only supports iPhones running iOS 13 and so this would not be an appropriate path for development as certain users would not be able to download the application.

This project could be developed for Android devices in the future, but because of my contacted end users’ preferences (and the time to complete this solution), I will only be developing for iOS devices.

## Specification of Proposed Solution

This is what my end users would like to see in a solution to the problem:

- **A Smartphone App:** This was the concluded device to develop on after correspondence with my end users. This enables data to be taken with the user thanks to modern technologies. Researched existing solutions such as Facebook Business are enhanced for a mobile experience and my end users believe developing for a phone will keep the contact between two clients flowing.
- **Login Authentication System:** Groups such as CEG would like to all be logged in under one account so that the activity of their employees can be monitored and so that they make decisions together. I would imagine that bands may also like to take advantage of this feature of working collectively as they try and secure a gig. This is an essential feature of a mobile app of this kind, as all data connected to a user's account needs to be secure and collected in a database to be updated and downloaded from. I am going to use Google's Firebase API to allow me to store data in a database in real time and use Firebase Authentication to aid with storing accounts and their encrypted passwords.
- **Portfolio View:** This is where users can view photos and videos of potential clients so that they know what to expect from them. Musicians require this view to show off their musical talent and sell themselves as much as they can in order to secure a job. Event Organisers may use this portfolio view to promote their venue with videos of other events they have held in the past; this would encourage musicians to apply as they have a feel for the venue they are playing at.
- **View to Create an Event (for organisers):** People such as Gavin at CEG would have a section of the proposed system only accessible by the organiser to post an advert or job description with all relevant information: payment, date, location (with Google Maps API), contact information and photos. The system needs to help the user create an advert with prompts and checks to make sure that nothing important has been left out of the job description. This aspect of the system needs to be simple and understandable in its use. An event object would again be stored in Firebase Database for musicians to access later. These created events must be open for editing or deletion in case anything was filled out incorrectly.
- **View to Apply for Gig (for musicians):** This view will only be accessible to musicians so that they can view and apply for events as they are pulled from the Firebase Database. Musicians like Jude who are still working on a local level, want gigs relevant to their location, and so potential jobs should appear in an order which has been sorted using the phone's location services. The action to apply for a job should be quick, simple and fair between all users and so I could easily take advantage of the iPhone's touchscreen gestures (similar to Tinder's Swipe Right™).
- **View to Review Activity:** Both types of users need to reference any activity which is of relevance to an event they are interested in. It may include a list of people who have applied to a posted event or a list of what events have been applied for. Any new activity which hasn't been viewed yet should trigger **Text/Email/Push Notifications** which is an essential feature both Jude and Gavin wanted to see the iPhone app make use of. These notifications would speed up the process of securing a

deal as users are told instantly when changes occur and can respond whenever they wish to.

- **User information:** Musicians want to promote their work on music streaming services and so this can be incorporated into the **Portfolio** in some way. This includes social media links, phone numbers and email addresses. Gavin at CEG would prefer to continue conversation in other messaging services such as WhatsApp and Facebook Messenger.

## Limitations of My Solution

The main limitation of my solution will be that it will not provide in-depth conversation tools between the types of user. The purpose of this system is to spark the initial contact between two users giving them new opportunities for their business or group. A direct messaging service could be developed in the future, but it will always be difficult to compete with preferred messaging services which big names like Facebook provide. Therefore, rather than reinventing the wheel, I can provide links to these services (as stated above) to continue contact.

Another limitation is a similar problem to the Melodeal app: If my app is not marketed well, the only events listed in the system will be of a similar geographical location. This creates a problem for a musician that lives far away from the area. For example, if the app is popular in Suffolk, only event organisers and musicians benefit in that county because a musician in Scotland is less likely to be hired. In addition, while the algorithm for sorting opportunities by locality may work, it still is not very useful for starting out musicians because these 'local' events are not local at all. Therefore, while a musician and hirer can promote themselves well in their portfolio, the effectiveness of the solution is dependent on its marketing and its acceptance as a good idea across the country.

The system could be developed in the future to support things such as payment between two parties as new development tools such as Apple Pay and Google Pay are becoming easier to use. This was not specifically requested by my end users and so it is not an essential feature.

Another issue which could occur is the difficulty of confirming whether a deal is genuine or not. Many companies such as eBay, Facebook and Gumtree still struggle with this issue and continue to develop solutions to this problem; I think my system could also be limited in the prevention of these illegitimate deals. However, the Portfolio feature plays a big role in this, both types of users will make an informed decision based off the contents of the portfolio (as discussed in the interview) hopefully avoiding scams.

Jude Mills: Your specification looks good and talks about all the things I would like to see in the smartphone app.

Gavin Thorrold: I think it will be a really good app if you meet all these requirements.

Looking forward to testing it out!

## Success Criteria and Objectives

These criteria should be kept in mind when developing the iOS app. I have split points into categories of ‘essential’ and ‘desired’ to plan what should (and could) be met when building the views detailed in my specification. Desired or non-essential features are listed as things which could be completed to enhance the app so that it is as good as it can be if there is still time after completing all essential tasks.

### Essential Points of Proposed System

1. **A good user experience is provided with an aesthetically pleasing user interface. This makes user interaction simple and understandable so that the solution meets its purpose of making contact between musicians and organisers quick and easy.** This is judged by whether user interface elements are obstructed or look ugly by my own opinion. It will be ultimately evaluated at the end of each milestone and end user testing.
2. **The system is easy to navigate, and any user errors are reported back and easily recoverable. All parts of the app are reachable at any time.** All developed error messages will be checked by entering invalid and erroneous data. Views are considered ‘reachable’ and there is freedom in navigation if the current user task can be paused and resumed by leaving that view and returning to it later.
3. **The system should check all input data against validation rules to avoid error, unexpected results and user confusion.** Will be measured by only correct and sensible data inputs being continued to be seen by other users. Validation rules are also checked by errors being reported back as part of success criterion 2.
4. **User can edit and review anything associated to their account at any time.** **Important data is never hidden and displayed when required.** The system never prevents the user from editing information which they have entered in the past. It will be tested by allowing the user to edit and making sure all data – whether edited or not – is displayed as expected to the current user and other people.
5. **A login authentication system holds data under an account. The user should remain logged in even when the app is not in use to receive notifications and activity updates.** Tested by checking all views reflect the correct data when logging in and out of test accounts as linked with success criterion 4. Data needs to remain constant when the app is closed and reopened while being logged in.
6. **All data is stored efficiently with Firebase Database and Storage API.** Judged purely by myself as the developer, use of the database is considered efficient if it is arranged in a logical format so retrieving data is as simple as it can be. Unused data is deleted automatically to decrease time when searching and fetching.
7. **Location services are accessed to aid with sorting data and meet stakeholders’ needs. This is optional in case it goes against users’ preferences.** Will be judged by its ability to enhance user experience in sorting gigs by locality. Considered ‘optional’ if the app works independently to the location being turned on.
8. **Able to access the iPhone camera and photo library to personalise the experience and share a portfolio with other users.** Tested by it being accessed successfully and consistently in all places it is needed and it returns an image which can be used by the user.

**9. Users can decide on the amount of information they share. Social media and contact sources are optional.** Considered ‘optional’ if the app works independently to social media links and other information being provided. Some data must be provided because otherwise it would affect the experience for other users.

### Non-Essential Points of Proposed System

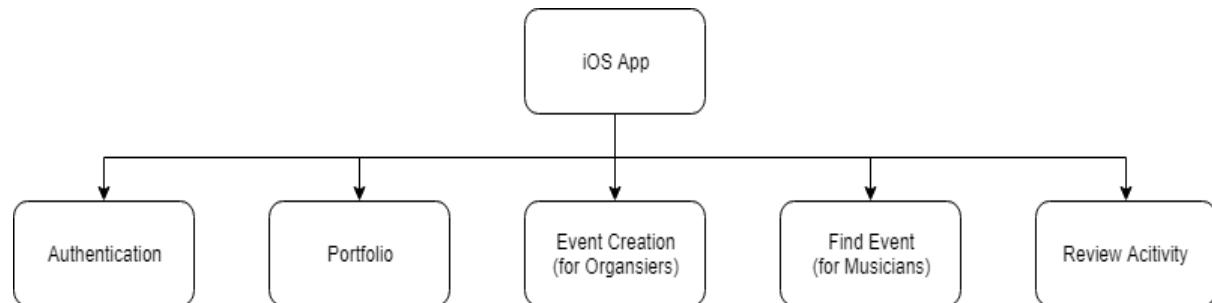
(in priority order)

- 10. Smartphone calendar integration**
- 11. Login using Facebook and Google**
- 12. Messaging included in the system**
- 13. Payment between users with bank transfers and Apple Pay**
- 14. Expansion to buying and selling of music equipment**

# Section 2 – Design of the Solution

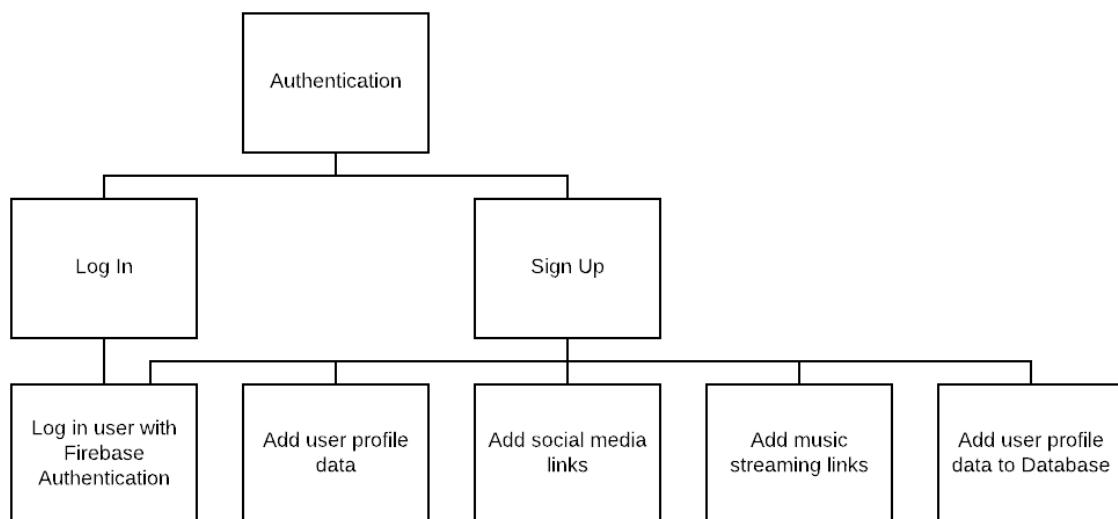
## Top Down Design/Stepwise Refinement

The iOS app is made up of the following core modules and classes:



### Core Module 1 - Authentication

As part of this core module, in all cases the user is logged in using Firebase Authentication and then taken to their portfolio as an initial view.



#### Log in

This will log in the current user. As the user's password is not stored in the Database but uses Google encryption, this enhances the app's account security. The user will be remembered as logged in so that they can resume tasks easily and continue to receive push notifications which is an essential feature. This keeps correspondence flowing as discussed with my end users and so all of the CEG team can work collaboratively as wanted by Gavin Thorrold which overall meets success criterion 5.

#### Sign up

This creates an account using an email address and chosen password. This view can be revisited to edit their account meeting success criterion 4.

#### User Profile Data

Here the user would enter a name, short biography and profile photo as well as deciding what type of user they are. This meets the requirement of providing an informative portfolio for the current user to review and edit in referring to the specification but more importantly,

display to other users. Success criterion 8 should be met here when accessing the camera and photo library as well as point 4 because this view would be reused to help edit the account later on. Point 6 is also met here because account data needs to be stored efficiently in Firebase.

### Social Media Links

In this view the user has the option to provide social media accounts which will link to other installed apps when displayed in the portfolio. Providing this option of input is what musicians, like Jude Mills, would like as it is an easy way to share their performances found on other social apps. As outlined by the success criteria, inputs for this must be entirely optional and simply be a feature which enhances the experience for both types of user. The system must check that all of these inputs are valid meeting success criteria 2 and 3. This is also optional input meeting criteria point 9.

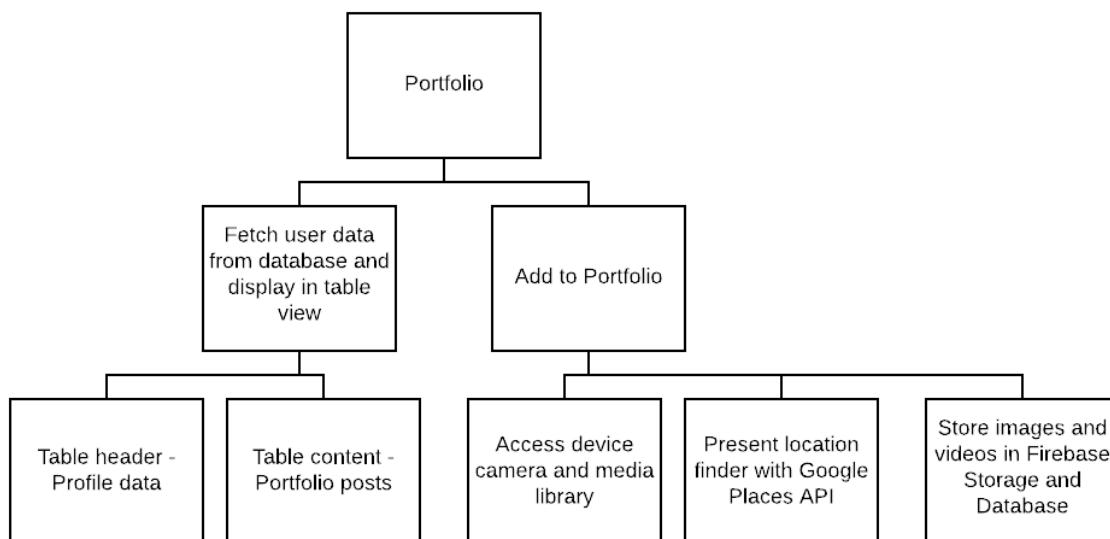
### Music Streaming Links

If a musician, you are given the option to provide links to the big streaming services such as Apple Music and Spotify. This is to support the musician users in the monetisation of their work. Again, it should be an optional input, but it will help meet the purpose of the Portfolio View represented in the specification by providing more ways to demonstrate skills.

Ultimately, it should support securing a deal quicker as displaying published, public music is impressive. Again, the system will check inputs meeting points 2 and 3 and is optional meeting point 9.

## Core Module 2 - Portfolio

This is the initial view if the user is already logged into the system. This view is reused to display other user's portfolios.



### Table Header – User Profile

The current user profile data is fetched from the database and displayed at the top of the table. Social media links would also be found in this header meeting the aim of enhancing contact between users. The profile data being displayed when wanted with all social media links in place meets success criterion 4.

## Table Content – User Portfolio Posts

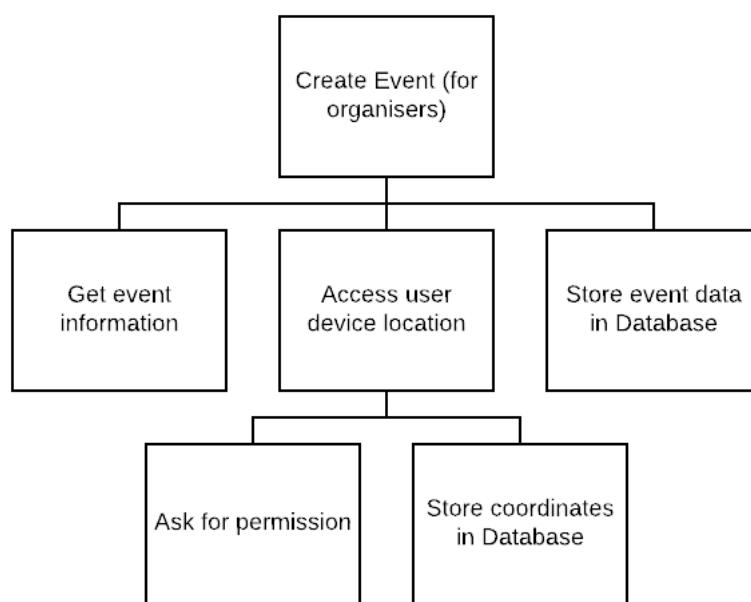
An array of portfolio posts is fetched from the database where each object in the array is displayed in a scrollable table view cell to create a feed. These images and videos are downloaded from Firebase Storage. This should meet the requirement of previewing a musician or venue before applying or confirming anything so that the Portfolio View meets its purpose as outlined in the specification. All the portfolio data will be unique to an account which follows success criterion 5. The feed needs to be aesthetically pleasing and minimal so that the media grabs the most attention, ultimately encouraging hirers to offer jobs which mirrors the most essential criterion, point 1.

### Add to Portfolio

This is a view where users would create a post object to store in the database. This means that the app needs permission to use the device's camera and access their media library. Success criterion 8 will be met in doing so. Adding to the portfolio also makes use of the Google Places API as users should be able to pin a location name to a post.

## Core Module 3 – Create Event (for organisers)

The Create Event module would be a tabbed section in which event organisers are encouraged to enter all necessary information to post their event advert. This Event object is stored in the database.



### Event Information

The user is provided various ways to input important information about the event. Doing this effectively will mean good error prevention and taking several checks before the event upload as guided by success criteria 2 and 3. UI and input methods need to be user friendly to complete the task and meet the success criteria. Input methods need to be provided to cover all the necessary information for an event listed in my specification: payment, date, location, contact information and photos.

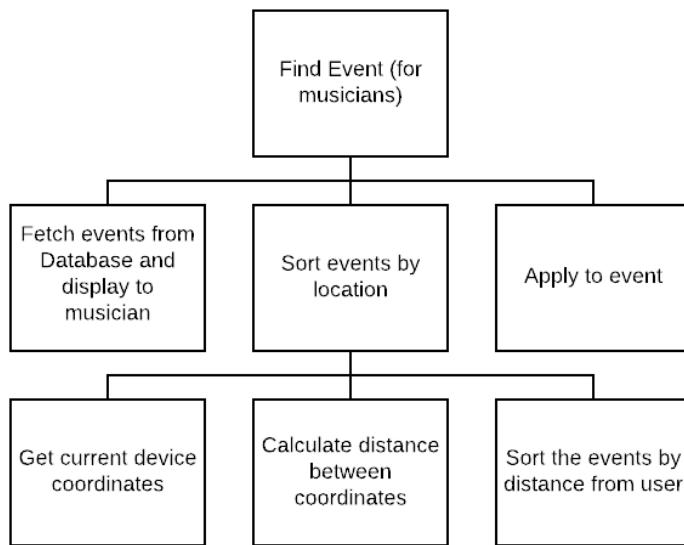
### Accessing User Location

To ensure that these events are sorted into their locality, so upcoming events are more applicable to a certain user, permission for device location services will be asked. The

coordinates of the user's device will be stored in the database so that the distance between musicians and the event venue can be calculated for the sorting process. Objective 6 will be met by making sure this location data is stored efficiently. Asking for location permission and the app working with this feature disabled will help complete criterion 7.

## Core Module 4 – Find Gig (for musicians)

The fourth module provides a way for musicians to apply to these created events.



### Sort Events

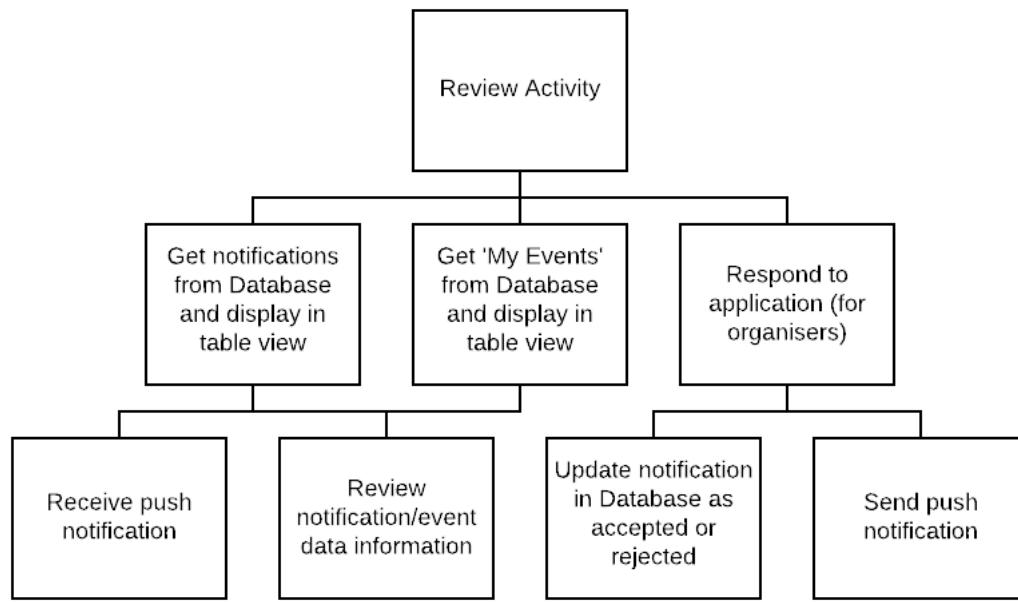
By getting the musicians current location coordinates, I can calculate the distance between the musician and the posted event coordinates. Events will therefore be more relevant to a musician when they appear as they are sorted by location. This is the second half to completing success criterion 7 where the location coordinates of both users are used to help with the sorting process. Closeness is priority guided by my stakeholders for my specification.

### Respond to Event

Applying to an event will update the database with a notification for the activity feed as well as triggering a push notification to the associated user. Making use of push notifications is an essential feature in my success criteria. Push notifications are part of objective 5 but ultimately speed up the process of securing a deal which meets point 1.

## Core Module 5 – Review Activity

The Review Activity module will include a view for all users to review information relevant to them and search through other users' profiles.



## Notifications

All updates associated with a user are fetched from the database and displayed as notifications in the form of a table. The database should be organised when meeting success criterion 6 so that fetching notifications and data is efficient. The aim of providing notifications is to speed up the process of securing a deal and making contact between two different users completing point 1. Therefore, any new messages should be flagged with a push notification to the user which was an essential feature for Gavin and Jude listed in my specification.

## 'My Events'

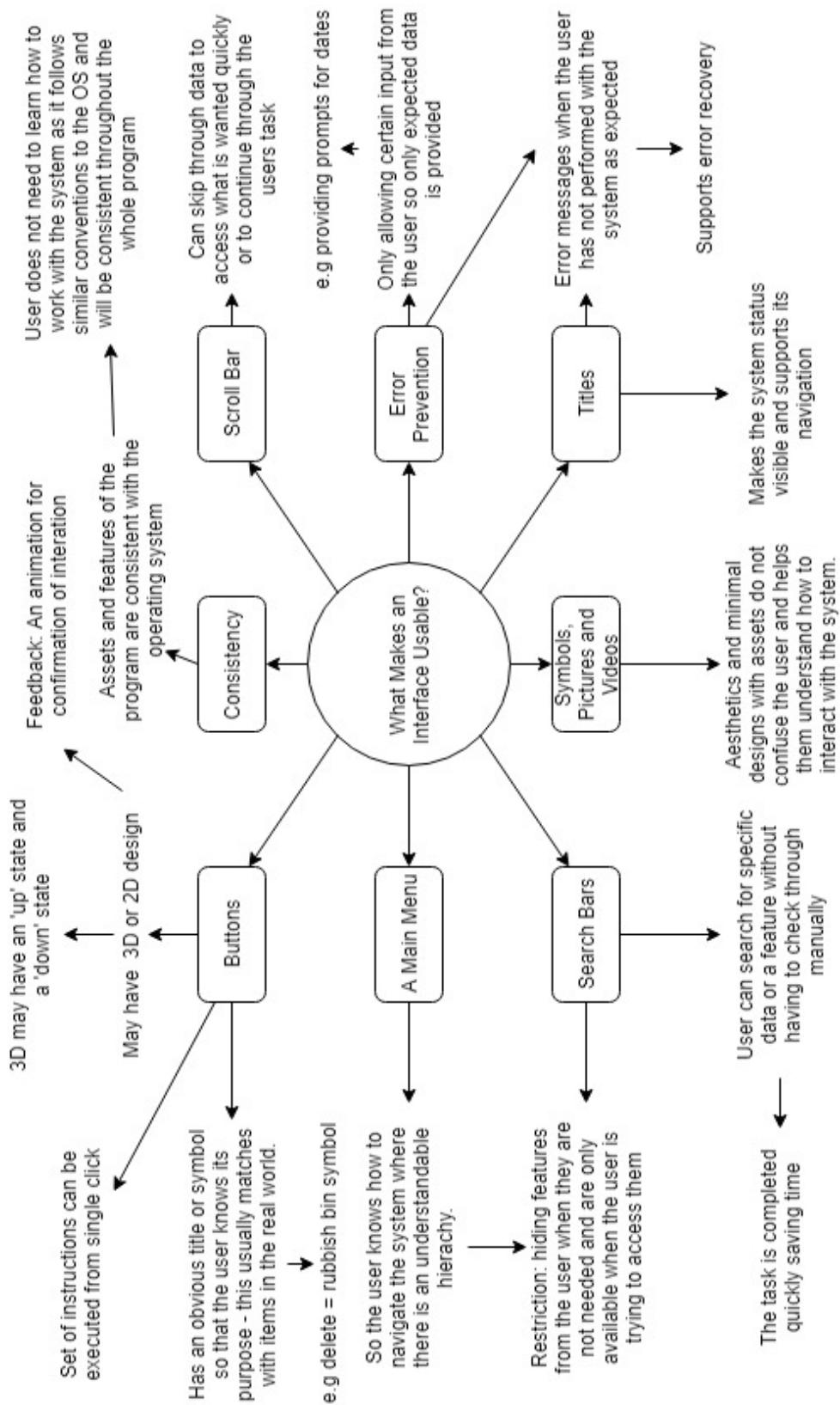
This will list all events associated with that user, whether it be their own created event or one they have applied for. Allowing them to review what they are involved with, this sub-module meets the requirement of user freedom and editing which is listed under success criterion 4.

## Respond to Application (for organisers)

Event organisers can respond to a musician's application either accepting or rejecting it. This then triggers a push notification to the associated musician meeting the most important requirement of securing a deal between the two users, which is point 1.

All of these modules should be accessible at any point in the use of the system. Even when in the middle of a task, the job should be paused so the user can navigate to the other parts of the app. Meeting the requirement of user freedom in navigation, it completes an important part of success criterion 2.

## Usability Features



## Interface Designs

(<https://www.nngroup.com/articles/ten-usability-heuristics/>) [Date Accessed: 27/06/19]

In 1994, Jakob Nielsen concluded 10 general principles for interaction design. These should be considered in the design of my UI for the proposed solution:

**Visibility of System Status:** The system should always inform its users what is going on with an appropriate method of feedback. In my iPhone application, the system status may include titles of what view a user is interacting with or loading symbols to notify people when the app is in the middle of a process.

**Match Between System and Real World:** System-orientated terms should be avoided because interacting with the system becomes easier when terms, phrases and images incorporated are familiar to the end user.

**User Control and Freedom:** Giving control means to provide a way for an 'emergency exit' or undo if necessary, so that there is not an inescapable unwanted state. For my solution, this may look something like providing the option to move back to the portfolio view while part-way through the event creation process – nothing has to be completed in order to navigate around the app.

**Consistency and Standards:** If an action repeats anywhere in the system, like triggering a database query, then it should be triggered by the same UI object to avoid confusion.

**Error Prevention:** Error messages notify the user of their mistakes, but these errors can be avoided altogether if the system limits them to only entering the expected data types. For example, a tick-box at sign up to inform whether they are a musician or not rather than writing in their user-type (which may cause potential errors) is good error prevention.

**Recognition Rather than Recall:** Instructions for the systems use should be visible when they are expected to interact with it in said way. This means that the user should not have to remember information as they progress through a task.

**Flexibility and Efficiency of Use:** Interaction can be sped up for the expert users with accelerators. These would not be seen by the inexperienced user so that the system is catered towards both independent of their experience.

**Aesthetic and Minimalist Design:** Information should all be relevant to what is needed at that time. Extra information which is less in relevance hides what is essential away from the user which could be a cause of error. More important interactable objects (such as required data-entry text boxes) could be bigger on screen than insignificant information.

**Help Users Recognise, Diagnose and Recover from Error:** If the system behaves unexpectedly then the user needs to be able return back while given feedback (in an understandable manner) of what went wrong so that it could be potentially avoided in the future.

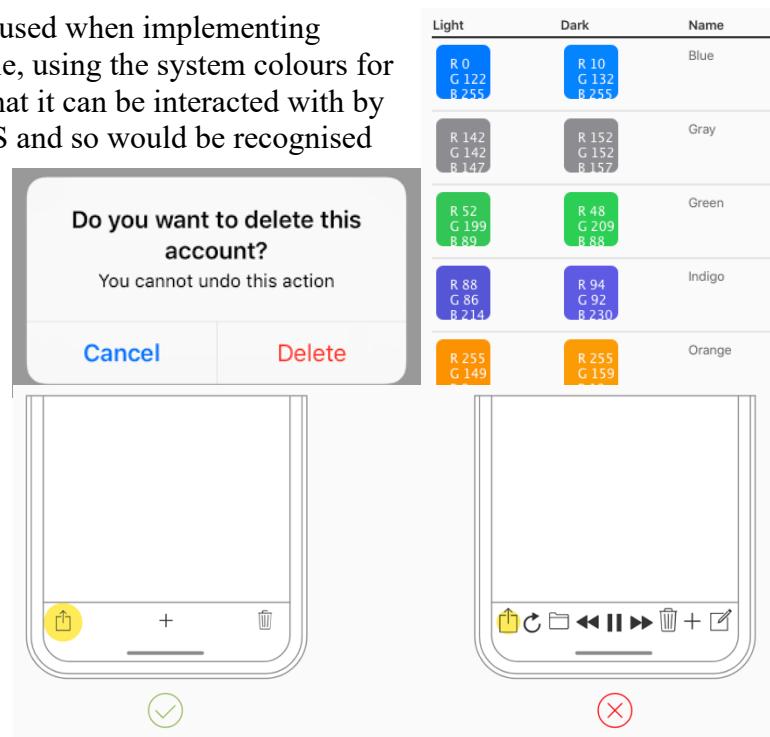
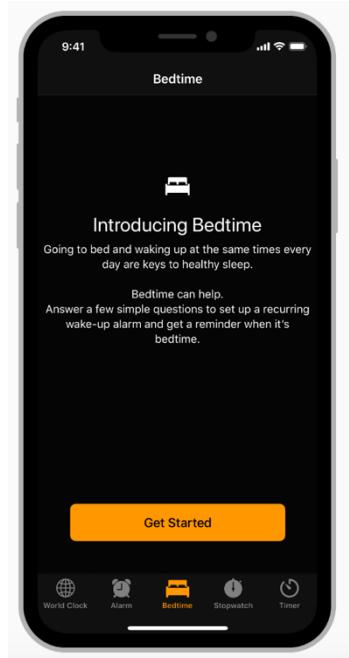
**Help and Documentation:** Help should be easily accessible to the user when necessary and should list short steps to help them complete what they want to achieve.

Following Apple Guidelines (<https://developer.apple.com/design/human-interface-guidelines/ios/visual-design>) [Date Accessed: 15/06/19]

When developing an app for iPhone, there are a few guidelines to be followed in order to create the best user experience as stated by Apple. Here is a summarised list of recommendations in visual design in association with Nielsen's 10 usability heuristics:

**System Colours:** These colours can be used when implementing various objects into an app. For example, using the system colours for a notification pop-up informs the user that it can be interacted with by the buttons. The blue is used across iOS and so would be recognised by users in this way. When using a red for a button, it catches attention so that the user thinks about the action they are taking.

**Layout and Spacing:** It is important not to overload the view with branding, symbols and features. It introduces accidental error if too many options are on screen at the same time and looks more aesthetically pleasing if there is natural space in the app.



**Terminology:** When communicating to the user, it is recommended that all words used are understandable and kept to familiar phrases. An app which appeals to everyone removes highly technical language to avoid confusion. Interface text should also be concise. By making text direct and short, this means that the user is not forced to read long passages to complete a task in the app. Interactive elements should inform at a glance what it does. For example, in the case of a button, action verbs are used: 'Connect', 'Share', 'Post'.

## Account Login and Sign Up

With my initial design, I have excluded colours as I want to ask my stakeholders what colour scheme they would like to see in the UI. The aim of the first draft is to get a feel for what UI objects would be used in the app in order to complete certain tasks.

With the Authentication View, I have gone for minimal design to encourage the user to sign up or log in. Text fields are included here for the user to fill out their email, and password. By adding text field placeholders such as 'email' that disappear when they enter text, it removes the need for titles.

I have also added symbols in these text fields along with the placeholders to confirm what information needs to be filled out. When a text field is in use, the system keyboard will appear and can be dismissed by tapping elsewhere on the screen.

Large buttons with the action verbs on them state what will happen with the information filled out in the text fields. The bottom button will change the mode of the view from Sign Up to Log In and the user is informed of this change with the use of a rhetorical question. The design of these buttons will be consistent throughout the system, so that people know it is an interactable feature.

## Account Creation

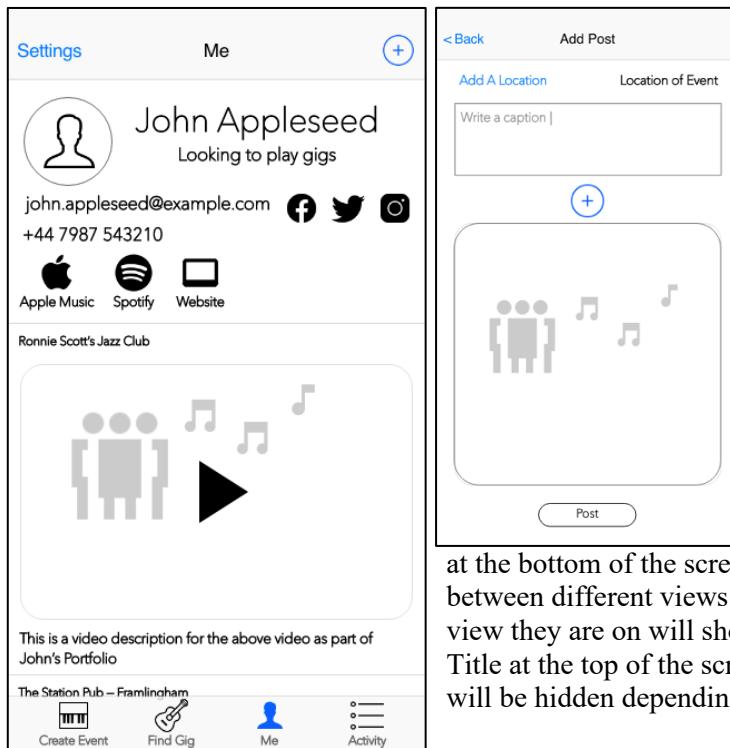
After clicking Sign Up, the user segues from the Authentication View to their account creation. Clicking 'Log In' in the previous view will bypass this.

In this view, we segregate musicians and event organisers at sign up as the tick box determines what type of user they are when the account is pushed to the database. This is a quick and fool-proof method to get this information rather than providing another text field which could contain spelling errors.

This time in the text fields, I have used placeholders as examples of what information needs to be filled out. A user biography is likely to contain more text than a name, therefore I will include a Text View which is a scrollable object for larger bodies. In this text view there should be a character limit so that content will fit on screen as expected later on.

Finally, the user is given the option to use an image from their device's library or take a photo for their profile image. A preview is presented so that they know how it will appear to other users.

## Portfolio



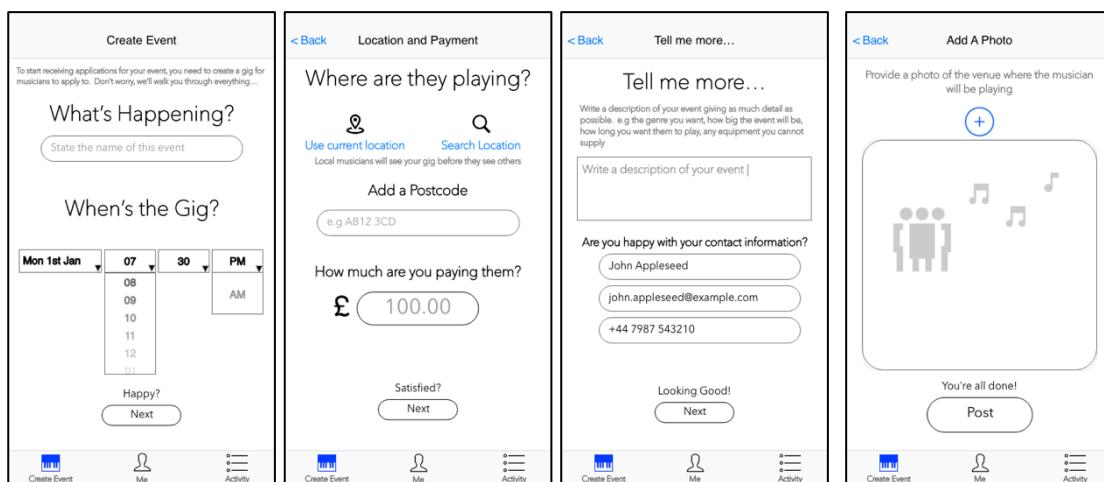
The Portfolio View will present all information at account creation along with social media and music service links. Profile information will appear in a header cell of a scrollable Table View. Following this initial cell, portfolio posts (containing images, videos, a description and location) can be viewed and scrolled through using touchscreen swipe gestures. The Portfolio View can be reviewed by the current user, and will be reused so that other users can look at their portfolio.

This view also introduces the Tab Bar at the bottom of the screen. This will help the user navigate between different views of the application where the current view they are on will show a blue icon in the Tab Bar and also a Title at the top of the screen. 'Find Gig' and 'Create Event' tab will be hidden depending on the type of user.

Adding a post to a portfolio moves to a separate view. Here there will be the option to add a location of the post (enabled through the Google Places API), a post description and a preview of the photo or video about to be posted. A button at the bottom will update the database with the post data.

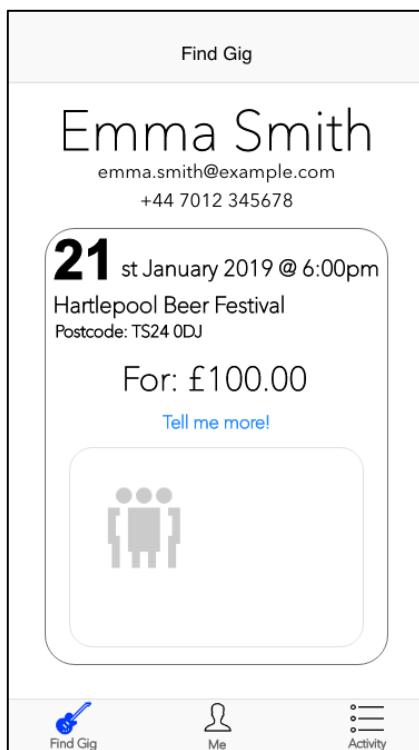
## Create Event

Under the 'Create Event' tab, event organisers are taken through a series of steps to create a gig for musicians to apply to. To speed up the process and create a good user experience, various UI objects will be included in this tab. UI objects for data input is good error prevention.



For example, drop down boxes will be used for entering a date and time. Each section of the process is prompted with a title and placeholders are used to give examples. Adding a photo at the end will be a similar process to adding a photo to the portfolio. A button at the bottom of each view (consistent in its design) will perform checks to make sure information has been filled and allow progression. This is also where the user's current device location will be used to attach coordinates to the event object. This means that when a musician pulls the event down from the database, the system will sort the events by locality. To speed up the process, some information can be filled out automatically like contact information which was stated at account creation. However autocompleted information must be editable in case the user is in disagreement.

## Find Event



[Back](#) Hartlepool Beer Festival

[Checkout Emma Smith](#)

**Monday 21st January 2019  
at 6:00pm  
for £100.**

Emma Smith:  
"We are looking for musicians and bands to play any style of music at the Hartlepool Beer Festival 2019. We are expecting over 2000 people and you will be given an hour slot. Please bring a guitar lead if required, all other equipment will be provided. Any questions please email me."

Under the 'Find Gig' tab, musicians will be able to preview all information added by the organiser about the event. The musician will be able to click a button to see the event description and, in this view, they can preview the hirer's portfolio for other information.

The gig is contained in a box which the user will be able to swipe right to accept the offer or left to reject it in a similar style to Tinder. These gig objects will be sorted so more local opportunities appear first. Their decision will update the database so that it does not appear to them again.

## Activity

Activity
<div style="display: flex; justify-content: space-between;"> <div style="flex: 1;"> <div style="display: flex; align-items: center;"> <span>You created an event: Hartlepool Beer Festival</span> </div> <div style="margin-left: 10px;"> <span style="color: blue;">Edit Event</span> <span style="color: blue;">Delete</span> </div> </div> </div>
<div style="display: flex; justify-content: space-between;"> <div style="flex: 1;"> <div style="display: flex; align-items: center;"> <span>John Appleseed applied to your event: Hartlepool Beer Festival</span> </div> <div style="margin-left: 10px;"> <span style="color: blue;">Checkout John Appleseed</span> <span style="color: blue;">Respond</span> </div> </div> </div>
<div style="display: flex; justify-content: space-between;"> <div style="flex: 1;"> <div style="display: flex; align-items: center;"> <span>You created an event: Thomas Mills Leavers Prom</span> </div> <div style="margin-left: 10px;"> <span style="color: blue;">Edit Event</span> <span style="color: blue;">Delete</span> </div> </div> </div>
<div style="display: flex; justify-content: space-between;"> <div style="flex: 1;"> <div style="display: flex; align-items: center;"> <span>Kate Lee applied to your event: Hartlepool Beer Festival</span> </div> <div style="margin-left: 10px;"> <span style="color: blue;">Checkout Kate Lee</span> <span style="color: blue;">Respond</span> </div> </div> </div>
<div style="display: flex; justify-content: space-between;"> <div style="flex: 1;"> <div style="display: flex; align-items: center;"> <span>Ben Young applied to your event: Thomas Mills Leavers Prom</span> </div> <div style="margin-left: 10px;"> <span style="color: blue;">Checkout Ben Young</span> <span style="color: blue;">Respond</span> </div> </div> </div>
<span style="color: blue;">Create Event</span> <span style="color: blue;">Me</span> <span style="color: blue;">Activity</span>

Activity
<div style="display: flex; justify-content: space-between;"> <div style="flex: 1;"> <div style="display: flex; align-items: center;"> <span>You applied to an event: Hartlepool Beer Festival</span> </div> </div> </div>
<div style="display: flex; justify-content: space-between;"> <div style="flex: 1;"> <div style="display: flex; align-items: center;"> <span>Emma Smith accepted your application: Hartlepool Beer Festival</span> </div> <div style="margin-left: 10px;"> <span style="color: blue;">Checkout Emma Smith</span> </div> </div> </div>
<div style="display: flex; justify-content: space-between;"> <div style="flex: 1;"> <div style="display: flex; align-items: center;"> <span>You applied to an event: Thomas Mills Leavers Prom</span> </div> </div> </div>
<div style="display: flex; justify-content: space-between;"> <div style="flex: 1;"> <div style="display: flex; align-items: center;"> <span>Kate Lee rejected your application: Thomas Mills Leavers Prom</span> </div> <div style="margin-left: 10px;"> <span style="color: blue;">Checkout Kate Lee</span> </div> </div> </div>
<span style="color: blue;">Find Gig</span> <span style="color: blue;">Me</span> <span style="color: blue;">Activity</span>

< Back
Respond

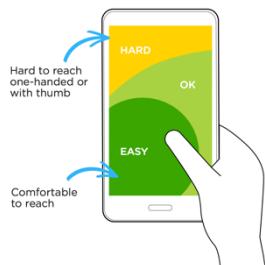
John Appleseed

Wants to play for  
Hartlepool Beer Festival

Checkout John Appleseed

The last tab will show a feed (represented in a Table View) of all activity related to the current user. This will include when an organiser has created their event, giving them the option to edit or delete it. The Activity View will also be where organisers can review who has applied for their events giving them the option to accept or reject them after reviewing their portfolio (accessible through 'Checkout musician' button). A rubbish bin symbol as a button represents deleting the event – the symbol creates a match between the system action and real life. For musicians, their activity feed would be filled with what gigs they have asked to play for and responses from employers. Using a large green tick and red cross to accept or reject a musician speeds up the process as well because the action is encapsulated into a

button press. Throughout the whole iOS application, buttons are placed at the bottom of the view because this is ergonomic, users will not have to stretch their thumb to interact because the buttons are in the natural thumb position.



## Stakeholder Feedback

I emailed my Stakeholders the initial designs (with explanations) and asked them their preferred colour scheme and for their general feedback:

**Jude Mills:** The app's design seems very ergonomic and user friendly, which appeals to me and will appeal to other musicians too. It's simple and not overcomplicated which is important when you could have a large user base. The layout is very organised: there is no complex movement to and from each section. It is clearly a very well thought through design covering all needed and required areas that a musician like me would need.

For a colour scheme, I think that purple could be a really nice gender-neutral colour that would appeal to a range of potential users. However, I really like the white on the main profile/portfolio page and think you should keep this as it is similar to Facebook and Instagram and therefore a familiar layout for a scrolling type window. With this in mind, do not overdo the purple, use it for the tabs and to bring forward any clickable features – blue buttons reminds me of the default apps already on the iPhone.

One thing I would say though is that it is not clear what tab does what with the choice of a piano and guitar. I think you should keep the guitar for the musician, but an organiser may not play an instrument and so you might need an image which relates to their job for it to make more sense to them, that is just my opinion. Also, it was not clear in the design where users can add their Facebook (or social media accounts) so that it appears in the portfolio. An appropriate place may be during the account creation or in a settings tab; you choose where it is best suited.

One idea for me that could be included is a way of putting in a radius of how far I am willing to travel for a gig, so that only gigs within a certain distance are recommended to me. In the 'Activity' tab, the listings do not appear to be in chronological order either. Notifications about someone applying to gig appears before the notification of the gig creation has even come through. This is assuming that the notification at the top of the table is the most recent item, which I think it should be so that I do not have to keep scrolling through to find the latest update of whether my band was accepted or rejected or not.

**Gavin Thorrold:** Your design of the application is very clear and would be easy to navigate which is good for new users of the system. It follows a clean structure which is typical to a phone app with sections at the bottom like most social medias.

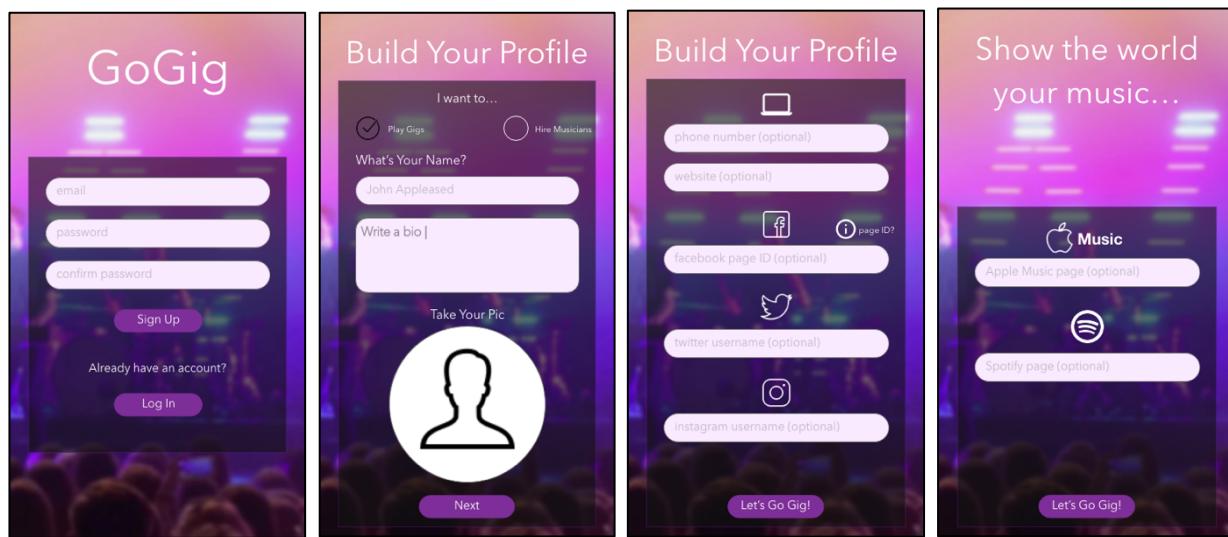
Be careful not to put too many symbols in places though. For example, if you were aiming for a minimal log in page, I think that the pictures in the text fields ruins it. You should already be able to tell what information should be entered from the placeholders. Also, a lot of information is packed into the portfolio page, but it is necessary. To clean things up a bit, you could remove the titles under the tabs and make the tabs bigger just to get rid of a few words.

For a colour scheme, I would want to see solid blocks of colour used. I will let you decide the colour, but I would recommend something bright while avoiding its use too much so that the app doesn't become too flashy. Too many apps of a similar format decorate with the colour blue, therefore try and stay clear of it.

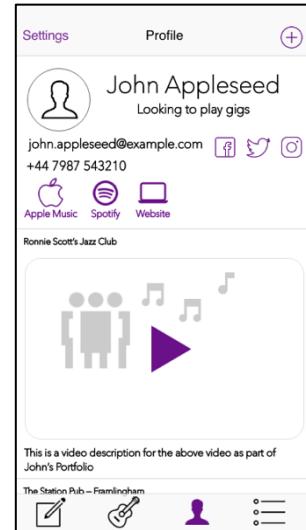
I did notice the use of drop-down boxes when creating a gig. Will this mean I will have to scroll through for a while to find the correct date for each box individually? If you look at the calendar and reminder apps on iPhone there is a neat tool which lets you pick a date on an animated wheel. Perhaps look into this, I feel that would be a quicker process.

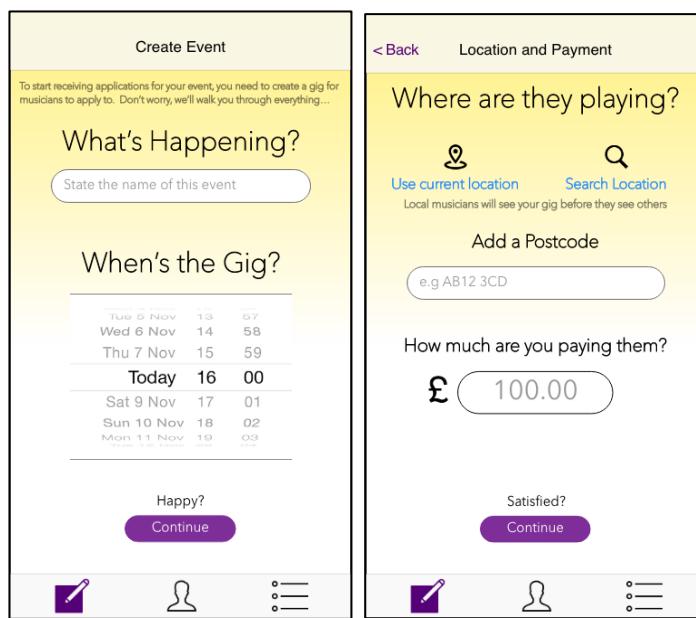
## 2<sup>nd</sup> Set of Designs

With the second set of designs for the system, I took up the recommendation of the purple colour scheme and tweaked the areas suggested:



All interactable features, such as buttons, are purple in colour and this will be consistent throughout the app. I removed the symbols from the text fields to avoid clutter but added a blurred background photo of a gig as an 'inspiration' image to users as they sign up. In addition, two more views have been added where the user can input their social media accounts – this is all optional information and can be accessed later by a 'settings' button on the main portfolio page. The portfolio page title was changed from 'me' to 'profile' so that it is a bit more obvious that this tab will be visible to other people.



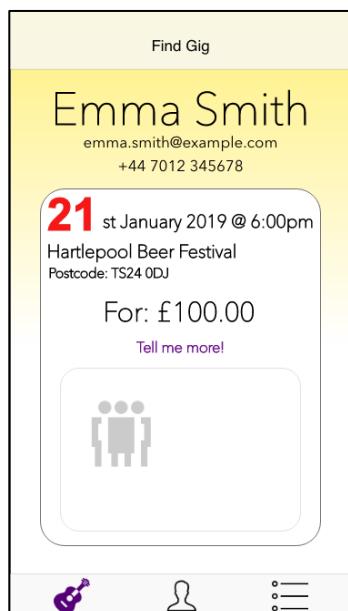


As seen, the tabs at the bottom now have no titles with them because it should be obvious from the symbols as to what section is what. Again, the blue tabs have been changed to purple and the piano image which was used for the Create Event view has been changed to a notepad, as suggested, which I think is more suitable for an organiser's role.

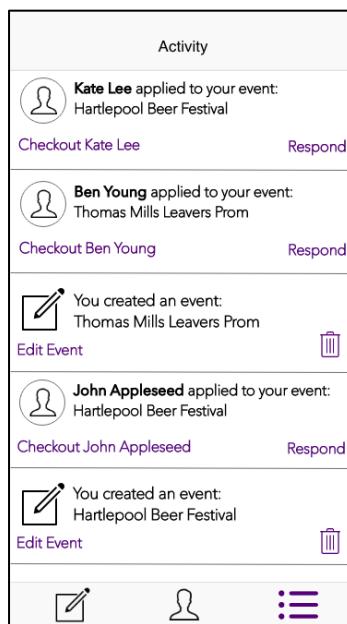
When creating a gig, I have added a yellow gradient background (purple's complimentary colour) to provide some contrast in the application. As suggested, I researched Apple's UI Date Picker object which makes

picking a date easier with touch screen swipe gestures.

In picking a location, I have kept the buttons blue because this will connect the phones location services and will likely require a privacy confirmation at the development stage. Therefore, because this feature relates to the device, I have stuck with default blue here.



There is a similar look for the musician Find Gig view. The swipe-able box will have a date in red because this is consistent with the iPhone calendar design.



The Activity tab has also been organised so that most recent notifications appear at the top of the table view so that having to scroll through everything to get to the latest update is avoided.

## Stakeholder Feedback

To conclude the design of my UI, I emailed my second set of designs to my stakeholders:

**Jude Mills:** After looking at your improved designs, I was pleased to see that the purple colour scheme I had suggested had been incorporated into the app. I think that the way this has been done is very suitable and professional, and the white being kept on the main profile/portfolio page also helps the purple from becoming overpowering. Again, the change of symbols between musician and event organiser is a really simple but useful change to help users differentiate between tabs. You have also included the ability to add social media accounts in a suitable place as you set up. Finally, I am pleased to see that the notifications have been kept in reverse chronological order, which is a really handy feature, that will be very beneficial in the long run.

**Gavin Thorrold:** This looks like a good finished product I would be happy to use with my team at CEG. The date picker in Create Event is more efficient than drop down boxes so I am happy that you have put that in.

You have used colour wisely and applied purple only to where it's necessary which is good and makes the app unique compared to other social apps on the App Store. I like the background photo you have used for the login page too; it reminds me of the Spotify authentication page and looks professional and appropriate for the target audience. Overall it looks great and I can't wait to see the finished app!

## Data Structures

Posts, notifications and gig opportunities will be stored in each of their own Swift arrays (similar to an ArrayList in Java and other languages). The reason for using this data structure is that I will not know how many post, notification and gig objects the user has stored in their account. For this reason, the data structure needs to be dynamic and in the example of a notification being added or deleted the Swift array needs to accommodate for these changes. Using this data structure means that I can iterate through it to display an object's data in a scrolling feed and sort it so a user sees particular items first, meeting success criteria point 1.

Before updating Firebase Database, I will store all inputted data from the user into a Swift Dictionary (a hash table data structure). This is the expected way to upload data to Firebase Database meeting success criteria point 6. When editing an account or gig, it will be beneficial to retrieve data and store in a dictionary due to fast access to the entries it contains.

I will store the state of being logged in under isLoggedIn. This key variable will be a Bool data type stored as a default so when the app first launches the variable will already have a value as set previously and is not assigned a value at runtime. Using isLoggedIn will allow me to keep the user logged in when they open and close the app meeting success criteria point 5.

## Validation Rules

Any input from the user needs to be validated to see if the data added is correct for the system to work as expected. Any problems should be reported back and simple to recover from to meet success criteria 3.

### Creating an Account and Logging In

When creating account some information is essential and needs to be filled out (like name, type of user and email) while other information is optional and will not have to be provided (such as social media and music links). I will need to write an algorithm of presence checks for all essential information but not for optional input. However, if an optional field has input, this still needs to be checked against rules to make sure the data is valid. Emails entered need to be already associated to an account when trying to log in and when trying to signup needs to follow mailbox@domain format. When entering a URL for a website, I will add a check to make sure the URL exists (if they have volunteered one) before the user can continue.

The UIKit front-end framework allows me to associate a type of keyboard for a specific field. For example, a keypad will appear when entering a phone number so that the string only contains numbers and no letters. This should help reduce the amount of invalid inputs for a better user experience meeting success criterion point 3.

Any error that the app comes across will report back to the user what they have done wrong by displaying an alert. I will use a UIAlertController with a title and message so that the user is familiar as to what is happening.

### Creating an Event (organisers)

Only organisers should be able to access this view and so should only show this tab when a user of this type is logged in, preventing error.

This will have similar validation techniques as the account creation process using special keyboards when required and reporting back error by displaying an alert. However, this feature will have less optional inputs as there is a required amount of information when creating a public event. A musician needs to be well informed when they apply to the event so that their user experience does not suffer because of someone else.

### Applying to Gigs (musicians)

Only musicians should be able to access this view and so should only show this tab when a user of this type is logged in, preventing error.

The validation rules involved in this view will be to check that a musician has not already interacted with the gig in front of them. I will keep a record in Database of what users have already interacted with the event in an array. If the user is in this array, then the system should not add the object to the local array so that it is never shown to the musician again.

### Replies to Applications (organisers)

Only organisers should be able to access this view and so should only show this view when a user of this type is logged in, preventing error.

The error of the user input will be prevented in this view by providing two options with buttons: Yes or No.

### Deleting an Event

I will provide an option to delete an event and this will mean different things depending on the user.

If the user is a musician and they delete an event they are booked to play for, it will only remove their association with the event. If the user is an organiser, I will write an algorithm where deleting an event means deleting their association with it and deleting the public object from the Database completely so no one can access it. This meets success criterion point 6 as data in the database is efficiently stored and point 4 where the user has complete control over what is associated to their account.

## Database Data Dictionary

The Firebase Realtime Database is stored as JSON objects and is best thought of as a ‘cloud hosted JSON tree’. Because this is not a SQL database there are no tables or records.

```
{
  "events" : {
    "UNIQUE_IDENTIFIER" : {
      "appliedUsers" : {
        "CREATOR:UNIQUE_IDENTIFIER" : true
        "UNIQUE_IDENTIFIER" : Bool
      },
      "description" : String,
      "email" : String,
      "eventID" : String,
      "eventPhotoURL" : String,
      "latitude" : Double,
      "longitude" : Double,
      "name:" : String,
      "payment" : Double,
      "phone" : String,
      "postcode" : String,
      "timestamp" : String,
      "title" : String,
      "uid" : String
    }
  },
  "users" : {
    "UNIQUE_IDENTIFIER" : {
      "activity" : {
        "UNIQUE_IDENTIFIER" : {
          "description" : String,
          "notificationID" : String,
          "reciever" : String,
          "relatedEventID" : String,
          "sender" : String,
          "senderName" : String,
          "timestamp" : Double,
          "type" : String
        }
      },
      "auth" : {
        "email" : String,
        "provider" : "Firebase"
      },
      "events" : [String],
    }
  }
}
```

```
        "posts" : {
            "UNIQUE_IDENTIFIER" : {
                "caption" : String,
                "isImage" : Bool,
                "location" : String,
                "postID" : String,
                "postURL" : String,
                "thumbnailURL" : String,
                "timestamp" : Double
            },
            "profile" : {
                "FCMToken" : String,
                "appleMusic" : String,
                "bio" : String,
                "email" : String,
                "facebook" : String,
                "gigs" : Bool,
                "instagram" : String,
                "name" : String,
                "phone" : String,
                "picURL" : String,
                "spotify" : String,
                "twitter" : String,
                "website" : String
            }
        }
    }
}
```

## Events under a unique Identifier:

Field	Data Type	Description	Example
appliedUsers	Dictionary<String, Bool>	Keep record of who created this event and which musicians have interacted with it so that it does not appear to them again. True is applied, false is ignored.	“0646DF0C-6537-4A19-97AF-E408B3AEB2E3” : true
description	String	Detailed description all about the event inputted by the organiser	“I would like a Jazz set lasting 2 hours”
email	String	The email of the organiser for contact reasons	“testuser1@gogig.com”
eventID	String	To delete the event later	“0646DF0C-6537-4A19-97AF-E408B3AEB2E3”
eventPhotoURL	String	To download the photo from Firebase Storage	““https://firebasestorage.googleapis.com/v0/b/gogig-db55e.appspot.com/o/3FAiK8aul5YZuTOt””

			JdOwN3dOv0P2%2Fevents%2F0646DF0C-6537-4A19-97AF-E408B3AEB2E3?alt=media&token=2f9dae3e-f2c7-4f09-9bd4-936d4892a03d"
latitude	Double	To create location of user for gig sort	52.399565
longitude	Double	To create location of user for gig sort	0.740887
name	String	Name of organiser who created the event	“Lee Chilvers”
payment	Double	Amount musician is getting paid for playing at the gig	150.00
phone	String	The phone number of the organiser for contact reasons	“01234 567890”
postcode	String	To provide location information to musician when they apply for the event	“AB12 CD3”
timestamp	String  Format: yyyy-MM-dd HH:mm:ss Z	To provide date and time information to the musician when they apply for the event	“2019-12-25 10:00:35 +0000”
title	String	Title of the event	“Framlingham Jazz Festival”
uid	String	Unique Identifier of event’s organiser	“3FAjk8aul5YZuTQtJdOwN3dOv0P2”

## User under a unique identifier:

### Activity under a unique identifier:

Field	Data Type	Description	Example
description	String	The information provided as part of the notification	“Applied for the event: Framlingham Jazz Festival”
notificationID	String	To delete notifications	“24AC6C99-0D3C-4FC3-B15C-81196C3043A2”
picURL	String	URL to download image that goes with notification from Storage	“https://firebasestorage.googleapis.com/v0/b/gogig-db55e.appspot.com/o/3FAjk8aul5YZuTQtJdOwN3dOv0P2%2FprofilePic%2F80571”

			3D4-31C7-4597-9699- ACA77076F856.jpg?alt=media&token=8a 366cff-0eb2-40e1-8b69-995d448aa8f6"
reciever	String	The uid of user that notification is going to	“3FAjk8aul5YZuTQtJdOwN3dOv0P2”
relatedEventID	String	Reference to the event object associated with the notification	“9EF33036-21D9-4D8D-887B-AC44C0838845”
sender	String	The uid of user that sent the notification	“3FAjk8aul5YZuTQtJdOwN3dOv0P2”
senderName	String	Name of user that sent the notification	“Lee Chilvers”
timestamp	Double	To sort notifications in reverse chronological order	1571918827.255651
type	String	Type decides who sees what notification	“personal”, “applied”, “reply”

### Auth:

Field	Data Type	Description	Example
email	String	Email associated with that user	“testuser1@gogig.com”
provider	String	How they have signed up to Firebase Authentication Service	“Firebase”

### Events:

Field	Data Type	Description	Example
events	[String]	Array of events associated with that user. Organiser – Events they have created, Musician – Events they are booked to play for	["34B95A33-B055-4F7F-A4B0-549C4EDF2853", "86FEBA50-0378-4AC4-9944-B2A3A72DCFBF", "9EF33036-21D9-4D8D-887B-AC44C0838845"]

### Posts under a unique identifier:

Field	Data Type	Description	Example
caption	String	Caption user pinned to their post	“My band playing for Framlingham Jazz Festival”
isImage	Boolean	To check if post is image or video	true
location	String	Location the post was taken	“Framlingham”
postID	String	For deletion	“57168096-2140-4757-9480-6D6D8D0D19A9”
postURL	String	To download image or video from Firebase Storage	“ <a href="https://firebasestorage.googleapis.com/v0/b/gogig-db55e.appspot.com/o/3FAjK8aul5YZuTQtJdOwN3dOv0P2%2FportfolioPost%2F57168096-2140-4757-9480-6D6D8D0D19A9.jpg?alt=media&amp;token=2ab0c7fe-bc29-47e0-9ff1-674bb4b84a6e">https://firebasestorage.googleapis.com/v0/b/gogig-db55e.appspot.com/o/3FAjK8aul5YZuTQtJdOwN3dOv0P2%2FportfolioPost%2F57168096-2140-4757-9480-6D6D8D0D19A9.jpg?alt=media&amp;token=2ab0c7fe-bc29-47e0-9ff1-674bb4b84a6e</a> ”
thumbnailURL	String	To download a thumbnail image from Storage if post is a video	“ <a href="https://firebasestorage.googleapis.com/v0/b/gogig-db55e.appspot.com/o/3DlbsH875Pce10CQ5YaNtxB8PX23%2FportfolioThumbnail%2F3E04BF61-AD52-4E05-976D-E1722E400930.mov?alt=media&amp;token=ad938ec-7af7-426e-9b97-b80c66b369d7">https://firebasestorage.googleapis.com/v0/b/gogig-db55e.appspot.com/o/3DlbsH875Pce10CQ5YaNtxB8PX23%2FportfolioThumbnail%2F3E04BF61-AD52-4E05-976D-E1722E400930.mov?alt=media&amp;token=ad938ec-7af7-426e-9b97-b80c66b369d7</a> ”

### Profile:

Field	Data Type	Description	Example
appleMusic	String	Link to Apple Music Profile	“ <a href="https://music.apple.com/us/artist/cody-fry/289832557">https://music.apple.com/us/artist/cody-fry/289832557</a> ”
bio	String	User biography, description about themselves	“Play drums for a Jazz band”
email	String	Email of the user	“testuser1@gogig.com”
facebook	String	Link to Facebook profile	“ <a href="https://www.facebook.com/lee.chilvers">https://www.facebook.com/lee.chilvers</a> ”
gigs	Boolean	Type of user they are. <u>gigs = true</u> is a musician.	true
instagram	String	Link to Instagram profile	“ <a href="https://www.instagram.com/l_chilvers/?hl=en">https://www.instagram.com/l_chilvers/?hl=en</a> ”
name	String	Name of the user	“Lee Chilvers”
phone	String	Phone number of user for contact	“01234 567890”

picURL	String	To download profile image from Storage	"https://firebasestorage.googleapis.com/v0/b/gogig-db55e.appspot.com/o/Two9DrH5X4UXMH6f3JxMhxQw5yv1%2FprofilePic%2F70BBA359-3385-4C85-A475-3E3914B5B405.jpg?alt=media&token=ea79a1e6-0ae8-416e-86fb-b2a4a7e99ccc"
spotify	String	Link to Spotify profile	"https://open.spotify.com/artist/7dOCnyDR2oEa1hQlvTXvdT"
twitter	String	Link to Twitter profile	"https://twitter.com/chilvers_lee?lang=en"
website	String	Link to user's own website	"chilly-designs.com"

## UML Diagram

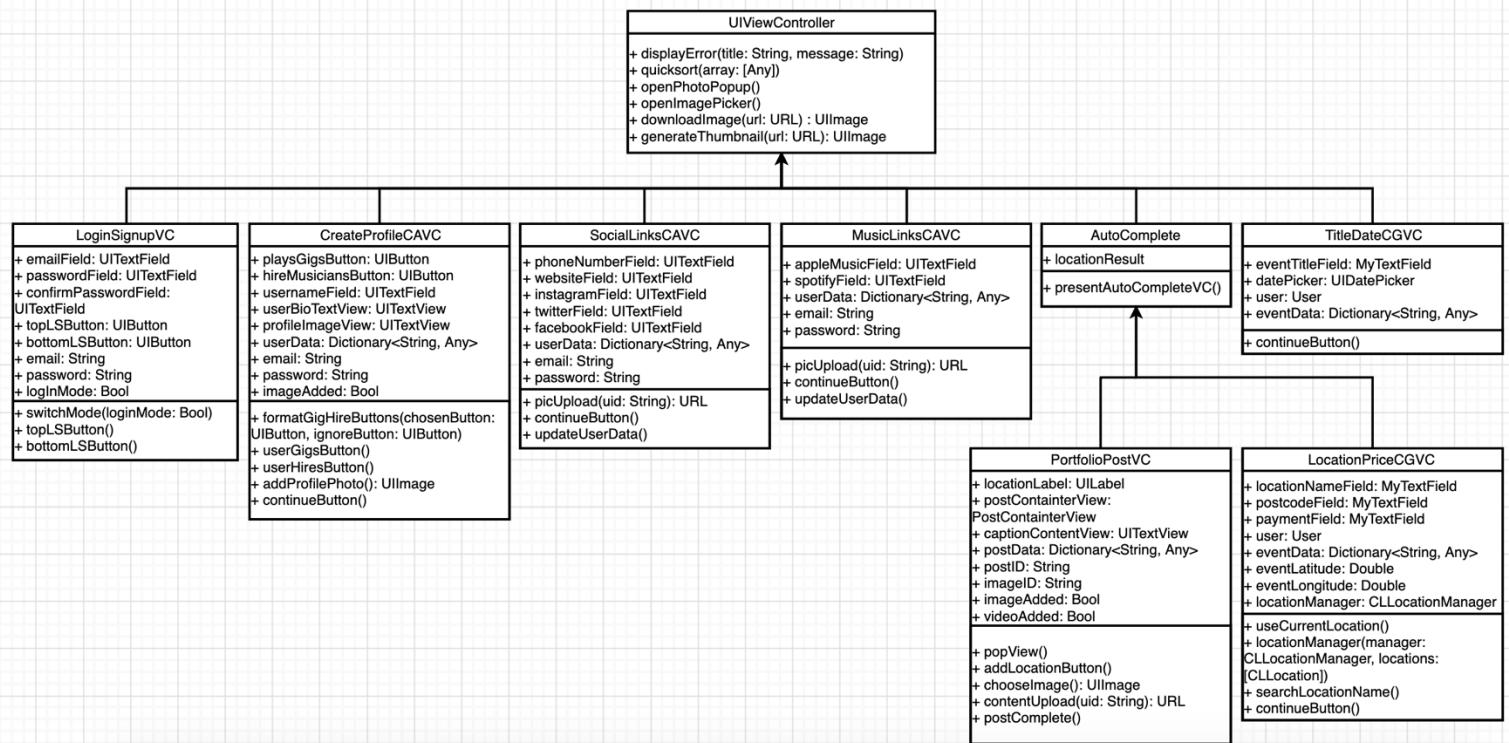
Developing in the Swift language, I want to use the Object-Oriented Paradigm. I will be trying to use the Model View Controller design pattern.

**Model:** maintains application data.

**View:** is the user interface which uses the model to display data.

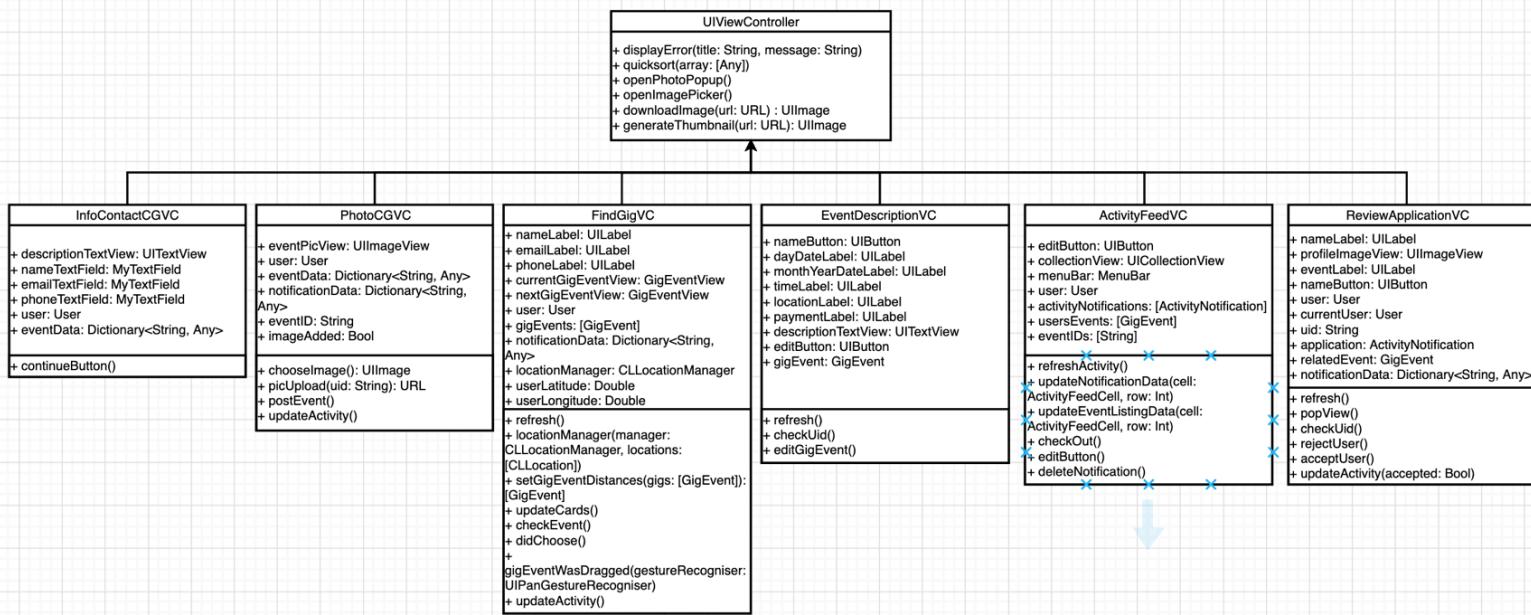
**Controller:** responsible for handling the users' requests to render the appropriate View with the correct Model data.

### View Controllers (1)



Each VC class will inherit from **UIViewController** where I have added some important extension functions. These extension functions report back error (success criteria 2) and display the camera and photo library (point 8). **LoginSignupVC** will be the initial page for users to login or signup to the app. When choosing to sign up, they will progress through **CreateProfileVC**, **SocialLinksCAVC** and **MusicLinksCAVC** to create and edit their account. As detailed in my specification, an account is essential for a mobile app of this kind. **AutoComplete** is part of the Google Places API and help with searching locations. Both **PortfolioPostVC** and **LocationPriceCGVC** will inherit from this class as they involve finding locations and will remove repeated code when meeting success criterion 7. **TitleDateCGVC** is the first view controller as part of the create event process for organisers.

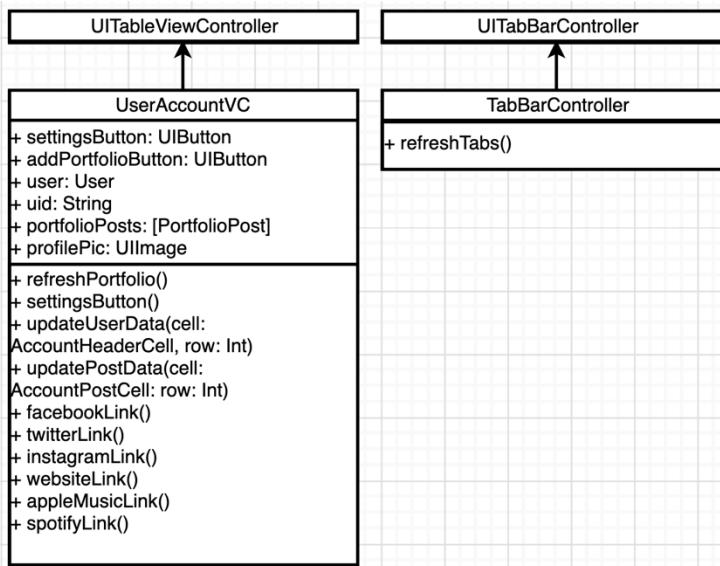
## View Controllers (2)



InfoContactCGVC and PhotoCGVC continue the create gig process ensuring all inputs are as expected when validated (point 3). Splitting up the gig creation process into various view controllers makes sure that all important information for an event, listed in the specification, are covered. FindGigVC will be responsible for presenting available opportunities to musicians and handling their responses. This view controller is key for handling communication between both user types which completes success point 1.

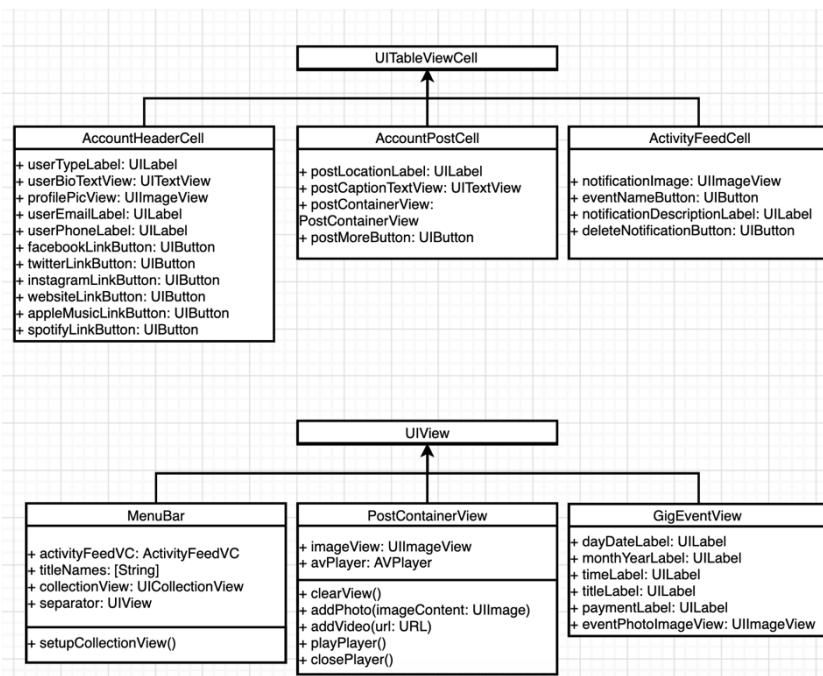
EventDescriptionVC will collect all information entered about that event and display it informatively. Both ActivityFeedVC and ReviewApplicationVC are controllers which bridge contact between the two types of users, displaying notifications and handling responses to musician applications, again meeting success criterion 1 and the purpose of these views described in the specification.

## View Controllers (3)



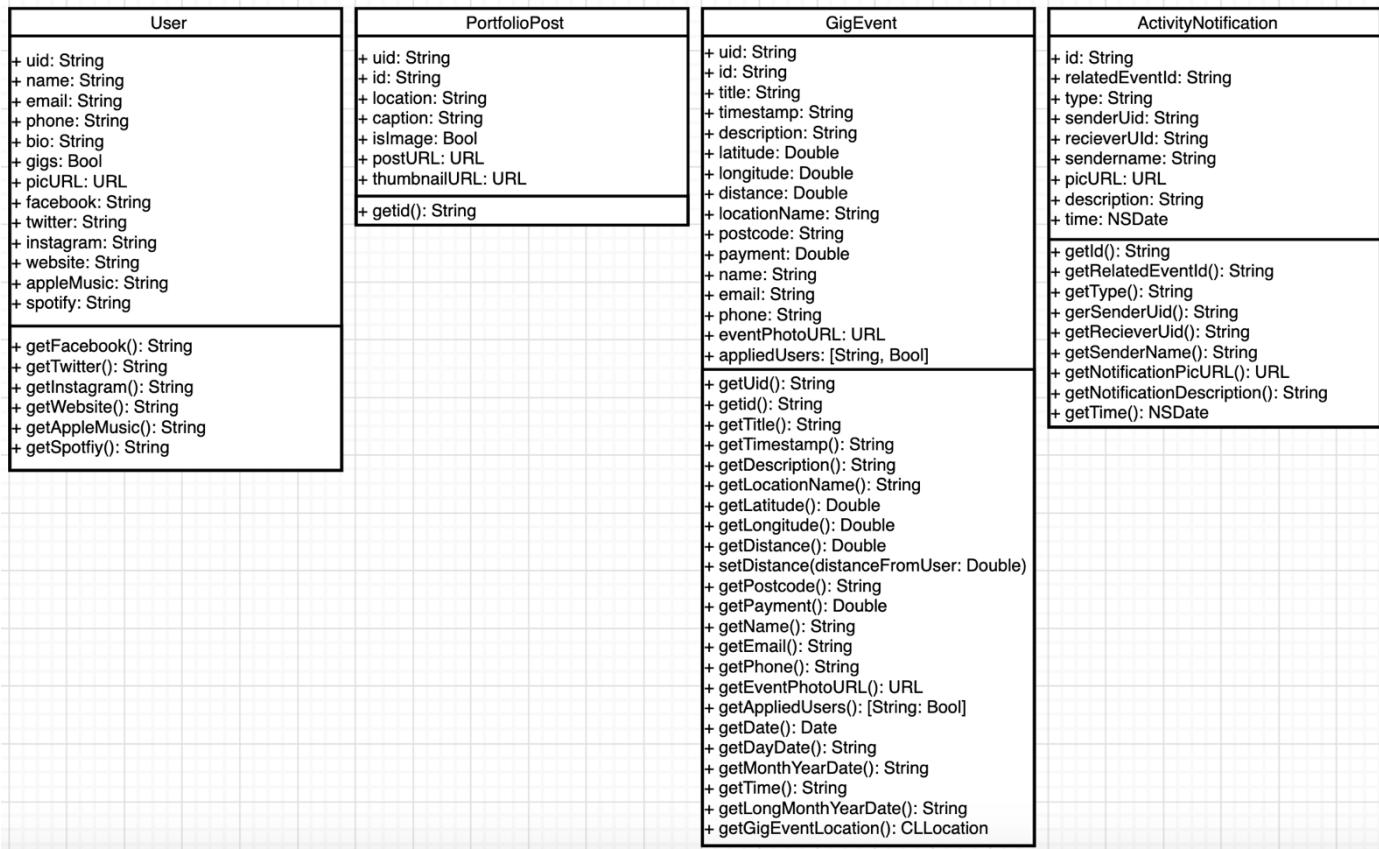
The UserAccountVC is the main portfolio page and the central tab of the UITabBarController which inherits from UITableViewController allowing me to access functions for a scrolling feed. TabBarController will control what tabs (at the bottom of the screen) present to what user. To complete objective 2, it is responsible for an essential part of the app's navigation because tabs allow various views to be accessed whenever requested.

## View



navigation in a collection view as part of ActivityFeedVC. PostContainterView will allow me to display both images and videos in the same space. GigEventView I plan to be a draggable ‘card’ which will display information about the event and can be dragged right to apply or left to ignore. These techniques keep user interaction simple meeting success point 1.

## Models



Inheriting from UITableViewCell allows me to make custom cells for the table view so that I can make sure it is aesthetically pleasing when completing objective 1. AccountHeaderCell will be a single cell shown by the UserAccountVC to display account data. AccountPostCell will be a repeated view showing each of the user’s posts in the same table view. Both make up the Portfolio view detailed in the specification. ActivityFeedCell will show notification information presented by the ActivityFeedVC.

MenuBar is a custom class allowing easy and understandable

These models will be used to instantiate objects out of the data from Firebase Database. This allows me to encapsulate all the data about the current User, display each PortfolioPost object's data in a feed, present all the data corresponding to a GigEvent object in a draggable card and display all notification data in a feed with ActivityNotification. An approach like this will help me keep track of the fetched data in data structures helping in maintainability and future development.

## Services

AuthService	DataService
<pre>+ isLoggedIn: Bool + userEmail: Bool</pre> <pre>+ registerUser(email: String, password: String) + loginUser(email: String, password: String)</pre>	<pre>+ REF_BASE: DatabaseReference + REF_USERS: DatabaseReference + REF_EVENTS: DatabaseReference + REF_ST: StorageReference</pre> <pre>+ createDBUser(uid: String, Dictionary&lt;String, Any&gt;) + updateDBUserProfile(uid: String, Dictionary&lt;String, Any&gt;) + getDBUserProfile(uid: String): User + updateDBPortfolioPosts(uid: String, postID: String, postData: Dictionary&lt;String, Any&gt;) + deleteDBPortfolioPosts(uid: String, postID: String) + getDBPortfolioPosts(uid: String): [PortfolioPost] + updateDBEvents(uid: String, eventID: String, eventData: Dictionary&lt;String, Any&gt;) + deleteDBEvents(uid: String, eventID: String) + getDBEvents(uid: String): [GigEvent] + updateDBUserEvents(uid: String, eventID: String) + deleteDBUserEvents(uid: String, eventID: String) + getDBUserEvents(uid: String): [String] + updateActivityFeed(uid: String, notificationID: String, notificationData: Dictionary&lt;String, Any&gt;) + deleteDBActivityFeed(uid: String, notificationID: String) + getDBActivityFeed(uid: String): [ActivityNotification] + sendPushNotification(to: String, title: String, body: String) + updateSTPic(uid: String, directory: String, imageContent: UIImage, imageID: String) + updateSTVid(uid: String, directory: String, vidContent: URL, imageID: String) + getSTURL(uid: String, directory: String, imageID: String): URL + deleteSTFile(uid: String, directory: String, fileID: String)</pre>

These two classes will hold commonly used functions to update, retrieve and delete data from the Firebase services Storage, Database and Authentication. My view controller classes will be cleaner and easier to read due to the fact dealing with Firebase is as simple as a function call. The functions also ensure that data gets stored efficiently and deleted when chosen by the user completing success criterion point 6.

## Algorithms

### DataService class

#### Upload to Database:

```

1: Procedure updateDBData(data)
2:   DatabaseReference.updateChildValues(data)
3: Endprocedure

```

This procedure simply uploads data to a path in Firebase Database. `.updateChildValues()` will not overwrite any data at the specified location. To meet success criterion 4, I can use `.setValue()` to overwrite the data when editing account or event information. This in turn meets criterion 6 with efficient database usage.

#### Delete from Database:

```

1: Procedure deleteDBData(object_ID)
2:   DatabaseReference.child(object_ID).removeValue()
3: EndProcedure

```

To further meet criterion 6 with efficient database storage, this method deletes at a specified location with an ID of the JSON object.

#### Fetch user profile from Database:

```

1: Function getDBUserProfile(uid)
2:
3:   profileSnapshot = DatabaseReference.child(uid).child("profile").observeSingleEvent()
4:
5:   profileData = profileSnapshot.value
6:   currentUser = new User(from: profileData)
7:
8:   RETURN currentUser
9:
10: Endfunction

```

This function returns a User object instantiated from profile data grabbed from the database. The data fetched will be at the specified location of “profile” under the user’s unique identifier (uid). The object will help with displaying data later on meeting success criterion 4 and will be used in various algorithms later on when getting profile data. Examples include: displaying profile data in the portfolio, displaying names and profile images in notifications and displaying who has applied for events to organisers.

#### Fetch user’s portfolio posts from Database:

```

1: Function getDBPortfolioPosts(uid)
2:   portfolioPosts = [PortfolioPost]
3:
4:   snapshot = DatabaseReference.child(uid).child("posts").observe()
5:
6:   postsSnapshot = snapshot.children
7:
8:   FOR snap in postsSnapshot
9:     postData = snap.value
10:    post = new Post(from: postData)
11:    portfolioPosts.append(post)
12:   NEXT snap
13:
14:   RETURN portfolioPosts
15: Endfunction

```

This function does a similar thing and gets post data. However, in this case I am iterating through all of the fetched JSON post data, each time instantiating a Post object and storing them in an array. The array is then returned. Posts are stored under the user's unique identifier (uid) so posts are efficiently fetched directly from the user in Database without having to iterate through all posts and run a check to see which belongs to who (efficient storage met – point 6).

### Get events created from Database:

```

1: Function getDBEvents()
2:   gigEvents = [gigEvent]
3:
4:   snapshot = DatabaseReference
5:   gigEventSnapshot = snapshot.children
6:
7:   FOR snap in gigEventSnapshot
8:     eventData = snap.value
9:     event = new gigEvent(from: eventData)
10:    gigEvents.append(event)
11:  NEXT snap
12:
13:  RETURN gigEvents
14: Endfunction

```

Again, I am fetching gig JSON data and instantiating objects for an array to be returned. However, this time, the DatabaseReference is not a path to the user but is global so that all the different musicians can access the available gigs later on. This, again, makes more sense than iterating through all users to see which are organisers and which have created events. Hence, it meets efficient database usage (point 6). The structure provides the optimal way to get back GigEvent objects.

### Get activity notifications from Database:

```

1: Function getDBActivityFeed()
2:   activityNotifications = [ActivityNotification]
3:
4:   snapshot = DatabaseReference.child(uid).child("activity").observe()
5:   activitySnapshot = snapshot.children
6:
7:   FOR snap in activitySnapshot
8:     notificationData = snap.value
9:     notification = new ActivityNotification(from: notificationData)
10:    activityNotifications.append(notification)
11:  NEXT snap
12:
13:  RETURN activityNotifications
14: Endfunction

```

Both this function and the previous one call .observe() when getting the data snapshot which is different to .observeSingleEvent() in that whenever the state of Database changes in that location it will return the new data added. It is good for notifications as I only want to fetch and display new ones rather than reloading all of them each time. These fetched notifications will be displayed in the activity feed as shown in a later algorithm.

### Upload pictures to Storage:

```

1: Procedure updateSTPic(uid, directiory, imageContent, imageID)
2:
3:   imageData = imageContent.jpegData()
4:   StorageReference.child(uid).child(directiory).child(imageID).putData(imageData)
5:
6: Endprocedure

```

Firebase Storage is an SDK used to store files such as images and video data. Here I convert images to JPEG data to then store at a specified location. The algorithm shown comes into play when adding a profile image in CreateProfileCAVC, images for gigs in PhotoCGVC and posting for PortfolioPostVC.

### Upload videos to Storage:

```

1: Procedure updateSTVid(uid, directory, vidContent, imageID)
2:
3:   StorageReference.child(uid).child(directory).child(imageID).putFile(vidContent)
4:
5: Endprocedure

```

This is a similar thing but am using .putFile() to upload the video. Storing each file under the imageID (a unique identifier) will allow me to delete the files later for efficient database usage. In uploading videos and images, the camera or photo library is accessed which completes success criterion 8. The only case when a video will be uploaded is when posting to the portfolio in PortfolioPostVC.

### Get the download URL of files in Storage:

```

1: Function getSTURL(uid, directory, imageID)
2:
3:   ref = StorageReference.child(uid).child(directory).child(imageID)
4:
5:   url = ref.getDownloadURL()
6:
7:   RETURN url
8:
9: Endfunction

```

This function simply returns the URL to download the file from a reference to where it is stored. The URL is used whenever an image or video appears in the view (for example for every post cell in the portfolio).

## AuthService Class

### Register and log in a user to Authentication:

```

1: Procedure registerUser(email, password)
2:   Auth.createUserWithEmail: email, password: password)
3: Endprocedure
4:
5: Procedure loginUser(email, password)
6:   Auth.signInWithEmail: email, password: password)
7: Endprocedure

```

These two procedures use Authentication SDK to register a user and log them in with an email and password which they would have inputted. Using the SDK means that Google will handle encryption and is likely to be more secure. Because an account is being created and securely maintained, success criterion 5 is being met. The unique identifier which corresponds to each registered user is used massively throughout the system. Data is organised in the Database under the identifier and therefore holds all the data for that account according to it. It would make no sense to store account data any other way and therefore success criterion 6 is met.

## Helper Function

Download an image:

```

1: Function downloadImage(url)
2:   data = URLSession.shared.dataTask(with: url)
3:   IF error THEN
4:     DISPLAY error
5:     RETURN placeholder image
6:   ELSE
7:     downloadedImage = new UIImage(from: data)
8:     RETURN downloadedImage
9:   ENDIF
10: Endfunction

```

Returning a UIImage, the function will generally use the download URL returned from Storage (shown before) to return some image data. This image data will be used to instantiate the UIImage object. The helper function will be used in various places (returning profile picture, posts, events images) and so is an extension function of UIViewController so that I can call it from all classes inheriting from UIViewController. If anything goes wrong in getting the image data, it displays an error and returns a placeholder image meeting success criterion 2.

## LoginSignupVC class

Logging in and signing up:

```

1: INPUT emailField
2: INPUT passwordField
3: INPUT confirmPasswordField
4:
5: Procedure topLSButton()
6:
7:   IF user is signing up THEN
8:     IF emailField.count > 3 and emailField.contains "@" and emailField != "" THEN
9:       IF passwordField.count > 6 and passwordField != "" THEN
10:         IF confirmPasswordField == passwordField AND != "" THEN
11:          (userData = data to be confirmed at account creation
12:           userData["email"] = emailField
13:           userData["password"] = passwordField
14:           segue to account creation
15:         ELSE
16:           displayError("Passwords do not match")
17:         ENDIF
18:       ELSE
19:         displayError("Choose a password more than 6 characters")
20:       ELSE
21:         displayError("Please enter a valid email address")
22:       ENDIF
23:     ELSE
24:       call: loginUser(emailField, passwordField)
25:       segue to portfolio
26:     ENDIF
27:
28: Endprocedure

```

The user's input is used to log them in or start the account creation process depending on what the top button displays (log in or sign up). I check the input against validations rules to make sure that everything has been entered correctly (success criterion 3) and when signing up add the input to a userData dictionary. During the account creation process, any input from the user will be assigned to a key in this dictionary. The reason why I have not registered a user to Authentication before they add their account details is that they may change their mind halfway through and so I sign them up at the end of their account creation.

It needs to be structured like this, so that there is freedom in navigation (success criterion 2) and the user can edit previously entered information before changes are made public (success criterion 4). An error is displayed when the user input does not match with the rules of the IF statements meeting criterion 2.

## CreateProfileCAVC class

```

1: INPUT playsGigs
2: INPUT usernameField
3: INPUT userBioTextView
4: INPUT profileImageView
5:
6:
7:
8: Procedure continueButton()
9:   IF usernameField.count >= 2 THEN
10:
11:     IF imageAdded and playsGigs != nil THEN
12:
13:       userData["name"] = usernameField
14:       userData["bio"] = userBioTextView
15:       userData["gigs"] = playsGigs
16:
17:       segue to SocialLinksCAVC
18:
19:     ELSE
20:       DISPLAY "Please provide all necessary information"
21:     ENDIF
22:   ELSE
23:     DISPLAY "Please enter your name"
24:   ENDIF
25: EndProcedure

```

This procedure continues account creation. User input is checked against validation rules to make sure that everything is inputted correctly and not left out (if it has, error is reported back) meeting success criteria 3 and the input is assigned to userData key. The data inputted here will appear in the UserAccountVC portfolio view and contact fields when creating a gig (InfoContactCAVC) will automatically be filled out with the name entered here. That feature meets a good user experience to meet success point 1.

## SocialLinksCAVC class

```

1: INPUT phoneNumberField
2: INPUT websiteField
3: INPUT instagramField
4: INPUT twitterField
5: INPUT facebookField
6:
7: Procedure continueButton()
8:   continueFine = true
9:
10:  IF (websiteField.contains(".")) and websiteField.count > 5) or websiteField == "" THEN
11:    userData["website"] = websiteField
12:  ELSE
13:    DISPLAY "Please enter a valid website (optional)"
14:    continueFine = false
15:  ENDIF
16:
17:  IF (phoneNumberField.count > 10 or websiteField == "") THEN
18:    userData["phone"] = phoneNumberField
19:  ELSE
20:    DISPLAY "Please enter a valid phone number (optional)"
21:    continueFine = false
22:  ENDIF
23:

```

```

24:  IF (NOT instagramField.contains("@") and NOT instagramField.contains(" ")) or
  instagramField == "" THEN
25:    userData["instagram"] = instagramField
26:  ELSE
27:    DISPLAY "Please enter a valid Instagram username (optional)"
28:    continueFine = false
29:  ENDIF
30:
31:  IF (NOT twitterField.contains("@") and NOT twitterField.contains(" ")) or twitterField
  == "" THEN
32:    userData["twitter"] = twitterField
33:  ELSE
34:    DISPLAY "Please enter a valid Twitter username (optional)"
35:    continueFine = false
36:  ENDIF
37:
38:  IF (facebookField.count > 5) or facebookField == "" THEN
39:    userData["facebook"] = facebookField
40:  ELSE
41:    DISPLAY "Please enter a valid Facebook name (optional)"
42:    continueFine = false
43:  ENDIF
44:
45:  IF continueFine THEN
46:    IF playsGigs THEN
47:      segue to music Links
48:
49:    //register organiser
50:    ELSE
51:      call: registerUser(email: emailField, password: passwordField)
52:      call: loginUser(email: emailField, password: passwordField)
53:
54:      uid = Auth.auth().getCurrentUser()
55:      url = picUpload(uid: uid)
56:
57:      userData["picURL"] = url
58:      updateDBData(data: userData)
59:
60:      segue to portfolio
61:    ENDIF
62:  ENDIF
63: EndProcedure

```

Due to social links being an optional input, I have used various IF statements to do the required validation checks (success criterion 3) if they have inputted anything. If anything is inputted incorrectly, the user is not allowed to continue, and they are notified of their error to correct it (success criterion 2). If they are allowed to continue, a user who selected to be a musician will segue to MusicLinksCAVC however an organiser is done with the account creation and so is registered and logged into Authentication at this point. Their profile picture is also uploaded along with all the userData dictionary to Database to stick with efficient usage (success criterion 6).

## MusicLinksCAVC class

```

1: INPUT appleMusicField
2: INPUT spotifyField
3:
4: Procedure continueButton()
5:  continueFine = true
6:
7:  IF (appleMusicField.contains(".")) and appleMusicField.count > 5 or appleMusicField ==
  "" THEN
8:    userData["appleMusic"] = appleMusicField
9:  ELSE
10:    DISPLAY "Please enter a valid Apple Music URL (optional)"
11:    continueFine = false
12:  ENDIF
13:
14:  IF (spotifyField.contains(".")) and spotifyField.count > 5 or spotifyField == "" THEN

```

```

15:     userData["spotify"] = spotifyField
16:     ELSE
17:         DISPLAY "Please enter a valid Spotify URL (optional)"
18:         continueFine = false
19:     ENDIF
20:
21:     IF continueFine THEN
22:
23:         call: registerUser(email: emailField, password: passwordField)
24:         call: loginUser(email: emailField, password: passwordField)
25:
26:         uid = Auth.auth().getCurrentUser()
27:         url = picUpload(uid: uid)
28:
29:         userData["picURL"] = url
30:         updateDBData(data: userData)
31:
32:         segue to portfolio
33:     ENDIF
34: EndProcedure

```

This algorithm only runs with musician users and is similar to the social links procedure. By restricting access to the view for certain users in various points in the app, as a developer, I can predict and plan for an expected outcome easier. Explained in the specification, the experience varies between user types to structure the app as wanted, this algorithm is an early example. A musicians input is checked and they always segue to the portfolio view when account creation has been completed. Success criterion 5 should be completed by this point because both types of users will have an account and profile data which will be displayed throughout the application. The data inputted is the data which is fetched from the database to instantiate a User object as demonstrated in the getDBUserProfile() function.

## UserAccountVC class

Displaying and deleting user's posts in UITableView:

```

1: uid = Auth.auth().getCurrentUser()
2: portfolioPosts = [PortfolioPost]
3:
4:
5: Procedure refreshPortfolio ()
6:
7:     user = getDBUserProfile(uid: uid)
8:     portfolioPosts = getDBPorfolioPosts(uid: uid)
9:
10:    DISPLAY user in AccountHeaderCell
11:    FOR post in portfolioPosts
12:        DISPLAY post in AccountPostCell
13:        NEXT post
14:
15: Endprocedure
16:
17:
18: Procedure deletePost()
19:
20:     INPUT choice at row
21:     IF choice is true THEN
22:         call: deleteDBData()
23:         call: deleteSTFile()
24:         portfolioPosts.remove(at: row)
25:         call: refreshPortfolio()
26:     ENDIF
27:
28: Endprocedure

```

As part of the portfolio, I get the current User object from Database to display in the AccountHeaderCell by calling getDBUserProfile() defined earlier. Using the user's uid as a

parameter, an array of portfolio posts is returned for me to iterate over and display each of the `PortfolioPost` objects in `AccountPostCell` (fetched by `getDBPortfolioPosts()` shown earlier). As part of each `AccountPostCell`, the current user will have the option to delete the post (meeting success criterion 4) which will remove the post data from Database, delete the file from Storage and remove the post from the local array which in turn meets efficient database usage for objective 6. The portfolio then needs to be refreshed to reflect this change giving the user feedback which is one of Neilson's Heuristics for recognition rather than recall and is also outlined by displaying the right thing in success criterion 4.

## PortfolioPostVC class

### Uploading post from user input:

```

1: INPUT locationLabel
2: INPUT postContainerView
3: INPUT captionContentView
4: uid = Auth.auth().getCurrentUser()
5:
6: Procedure uploadPost()
7:
8:   postID = unique string
9:
10:  IF imageAdded THEN
11:    call: updateSTPic(uid: uid, directory: "portfolioPost", imageContent: imageContent,
imageID: postID)
12:  ELSE IF videoAdded THEN
13:    call: updateSTVid(uid: uid, directory: "portfolioPost", vidContent: imageContent,
imageID: postID)
14:  ELSE
15:    DISPLAY "Please choose something to post"
16:  ENDIF
17:
18:  postURL = getSTURL(uid: uid, directory: "portfolioPost", imageID: postID)
19:
20:  postData = ["postID": postID, "isImage": imageAdded, "postURL": postURL, "caption": captionContentView, "location": locationLabel]
21:
22:  updateDBData(postData)
23:
24: Endprocedure

```

This algorithm uploads a post using the `DataService` procedure defined earlier. It does a validation check to make sure that an image or video has been added and reports back an error if they haven't. This meets success criteria 2 and 3. If something has been added then the image or video will be uploaded to Storage and data about the post is stored in the `postData` data structure which is uploaded to Database.

## TitleDateCGVC class

```

1: INPUT eventTitleField
2: INPUT datePicker
3:
4: eventData: Dictionary<String, Any>
5: uid = Auth.auth().getCurrentUser()
6:
7: Procedure continueButton()
8:   timestamp = datePicker.date
9:
10:  IF eventTitleField != "" and eventTitleField.count <= 60 THEN
11:
12:    eventData["uid"] = uid
13:    eventData["title"] = eventTitleField
14:    eventData["timestamp"] = timestamp
15:
16:    segue to LocationPriceCGVC

```

```

17:
18:   ELSE
19:
20:     DISPLAY "Please add the title of your event to continue"
21:   ENDIF
22: EndProcedure

```

The event creation process for organisers is a similar series of algorithms to the create account process. Input data is collected across classes in the eventData dictionary. This class, users will input their event title and the date in a wheel picker. When they choose to continue, their title length is checked. Their date picked is also assigned as a timestamp (format: yyyy-MM-dd HH:mm:ss +zzzz) which will allow me to manipulate the date for algorithms later on. The input is added to the dictionary.

## LocationPaymentCGVC class

```

1: INPUT locationNameField
2: INPUT postcodeField
3: INPUT paymentField
4:
5: eventLatitude = 0.00
6: eventLongitude = 0.00
7:
8: Procedure useCurrentLocation()
9:   IF currentLocationOn == false THEN
10:     call: startUpdatingLocations()
11:     currentLocationOn = true
12:   ELSE
13:     call: stopUpdatingLocations()
14:     currentLocationOn = false
15:   ENDIF
16: EndProcedure
17:
18: Procedure startUpdatingLocations()
19:   eventLatitude = userLocation.latitude
20:   eventLongitude = userLocation.longitude
21: EndProcedure
22:
23: Procedure stopUpdatingLocations()
24:   eventLatitude = 0.00
25:   eventLongitude = 0.00
26: EndProcedure
27:
28: Procedure continueButton()
29:
30:   IF locationNameField.count > 2 THEN
31:     IF postcodeField.count == 7 or postcodeField.count == 8 THEN
32:
33:       eventData["latitude"] = eventLatitude
34:       eventData["longitude"] = eventLongitude
35:       eventData["locationName"] = locationNameField
36:       eventData["postcode"] = postcodeField
37:       eventData["payment"] = Double(paymentField)
38:
39:       segue to InfoContactCGVC
40:
41:     ELSE
42:       DISPLAY "Please enter the correct 7 character postcode of the event"
43:     ENDIF
44:   ELSE
45:     DISPLAY "Please search for or type in a location"
46:   ENDIF
47: EndProcedure

```

To meet success criteria 7 and use location services, the user selects a button which will start updating location. Global variables which hold the user's coordinates are stored in the dictionary when continuing, or if the user turns location usage off again, these coordinates are

set to (0.00, 0.00). This is to allow freedom as to what information a user shares, so if they change their mind against location sharing, their coordinates need to be reset before uploading the data (success criterion 9). The postcode is checked against validation rules to make sure they are the correct length and added to the eventData dictionary if everything is added correctly (success criterion 3). These coordinates will be used when sorting the GigEvent object in displaying it to the musician in FindGigVC. Again, all entered event information is added to the dictionary like the previous algorithm.

## InfoContactCGVC class

```

1: INPUT descriptionTextView
2: INPUT nameTextField
3: INPUT emailTextField
4: INPUT phoneTextField
5:
6: nameTextField = user.name
7: emailTextField = user.email
8: phoneTextField = user.phone
9:
10: Procedure checkPhoneField()
11:   IF phoneTextField.count > 7 THEN
12:     eventData["phone"] = phoneTextField
13:   ENDIF
14: EndProcedure
15:
16: Procedure continueButton()
17:   IF nameTextField != "" THEN
18:     IF descriptionTextView.count > 10 and descriptionTextView != "" THEN
19:
20:       eventData["name"] = nameTextField
21:       eventData["description"] = descriptionTextView
22:
23:       IF emailTextField.contains("@") and emailTextField.contains(".") and
emailTextField.count >= 5 THEN
24:
25:         eventData["email"] = emailTextField
26:
27:         call: checkPhoneField()
28:
29:         segue to AddPhotoCGVC
30:
31: ELSE IF phoneTextField.count >= 7 THEN
32:
33:   eventData["phone"] = phoneTextField
34:
35:   segue to AddPhotoCGVC
36:
37: ELSE
38:
39:   DISPLAY "Please enter a valid email address or phone number"
40:
41:   ENDIF
42: ELSE
43:   DISPLAY "Please enter a description or your event outlining the suggested points"
44:
45:   ENDIF
46: ELSE
47:   DISPLAY "Please enter your name"
48: ENDIF
49: EndProcedure

```

This algorithm starts by auto filling out the user's details in the contact fields using the user object. It speeds up the process and enhances the user experience meeting success criteria 1. A name is essential but either an email or phone number can be provided. My algorithm runs a check for an email input, then calls a function which decides if they have entered a phone number as well. It also checks for just a phone without an email so that all three cases are met. If neither have been entered (or entered incorrectly) I display the error to the user

completing criterion 2. This data will be displayed in the FindGigVC when fetched and instantiated from the database. Again, all entered event information is added to the dictionary like the previous algorithm.

## AddPhotoCGVC class

```

1: Procedure postEvent()
2:   IF imageAdded THEN
3:
4:     eventID = unique String
5:
6:     eventData["eventID"] = eventID
7:
8:     url = picUpload(uid: uid)
9:     eventData["eventPhotoURL"] = url
10:    event["appliedUsers"] = ["CREATOR:" + uid : true]
11:
12:    call: updateDBData(data: eventData)
13:    notificationData = updateActivity()
14:    call: updateDBData(data: notificationData)
15:
16:    go to activity tab
17:
18:  ELSE
19:    DISPLAY "Please take or add a photo of the venue to post event"
20:  ENDIF
21: EndProcedure

```

This algorithm is similar to adding a photo ready to post. However, this time only an image can be selected as input data and still meets criterion 8. The procedure will check the image has been added and add the unique image id to the dictionary. After uploading it to Storage, event[“appliedUsers”] will be an array of the musicians that have interacted with the event. I have listed the creator at index 0 of this array so that the Database has some value for this key when we fetch it. The ‘appliedUsers’ array will control whether a musician can access and see a GigEvent object or not in FindGigVC. Ensuring the correct data is displayed for a user is important in completing success criterion 4. The eventData dictionary which was being added to is finally uploaded. I then trigger a notification to be sent and the organiser is segued to the activity tab after they have finished to see their personal notification of the event created. By segueing the user, it provides feedback and creates a good understandable user experience (success criterion 1). It also demonstrates how the views in the specification link together in their purpose to the user.

## Upload notification object to Database:

```

1: Function updateActivity()
2:   notificationID = unique String
3:   senderUid = user.uid
4:   receiverUid = senderUid
5:   senderName = "You"
6:   notificationPicURL = user.picURL
7:   notificationDescription = "Created the event: " + eventData["title"]
8:   timestamp = Date()
9:
10:  RETURN notificationData = ["notificationID": notificationID, "relatedEventID": eventID,
11: "type": "personal", "sender": senderUid, "receiver": receiverUid, "senderName": senderName,
12: "picURL": notificationPicURL, "description": notificationDescription, "timestamp": timestamp]
11:
12: EndFunction

```

This function updates the database with a personal notification for the organiser to receive about them creating the event. Therefore, the receiver of the data is also the sender. The data

of the notification is returned to be uploaded to the Database. The notification data sent is what is used to instantiate a personal notification object in ActivityFeedVC. The notification updates are important according to the specification because it allows organisers to review their own activity.

## FindGigVC class

Displaying gig opportunities to musicians sorted by locality:

```

1: uid = Auth.auth().getCurrentUser()
2: gigEvents = [GigEvent]
3: user = getDBUserProfile(uid: uid)
4:
5: Procedure refreshGigs()
6:
7:   call: startUpdatingLocation()
8:   gigEvents = getDBEvents(uid: uid)
9:
10:  gigEvents = setGigEventDistances(gigs: gigEvents)
11:
12:  call: updateCards()
13:
14: Endprocedure
15:
16:
17: Function setGigEventDistances(gigs)
18:
19:   userLocation = CLLocation(from: user.latitude and user.longitude)
20:
21:   FOR gig in gigs
22:     gigEventLocation = gig.getGigEventLocation()
23:     distance = CALCULATE distance between gigEventLocation and userLocation
24:
25:     gig.setDistance(distanceFromUser: distance)
26:   NEXT Gig
27:
28:   RETURN gigs.sortedByLocation()
29:
30: Endfunction
31:
32: Procedure updateCards()
33:
34:   IF gigEvents.count >= 1 THEN
35:     currentGigEvent = gigEvents[0]
36:     DISPLAY currentGigEvent data in currentGigEventView
37:
38:     IF gigEvents.count > 1 THEN
39:       nextGigEvent = gigEvents[1]
40:
41:       DISPLAY nextGigEvent data in nextGigEventView
42:     ENDIF
43:   ELSE
44:     DISPLAY view as no gigs around
45:   ENDIF
46:
47: Endprocedure

```

This algorithm, for musicians, first gets the array of all the events the musician can apply for. This for gig in the array, the distance between the musician's current location and the location of the event are calculated and sorted by this value. This meets success criterion 7 by making gigs more relevant in the use of device's location services. The sorted array being returned is then displayed in the card views. I cover the cases in the procedure when there is only one opportunity, two or more and none at all. If there are no gigs, then I display a message to inform the user that nothing has been returned (error reported back – success criterion 2).

## Musician applying and swipe animation:

```

1: INPUT interactedGigEvent
2:
3: Procedure didChoose(applied)
4:
5:   gigEventAppliedUsers = interactedGigEvent.gigEventAppliedUsers()
6:   gigEventAppliedUsers[uid] = applied
7:
8:   updateDBData(data: gigEventAppliedUsers)
9:
10:  IF applied THEN
11:    call: updateActivity()
12:  ENDIF
13:
14:  gigEvents.remove(at: 0)
15:
16:  call: updateCards()
17: Endprocedure
18:
19: Procedure gigEventWasDragged(gestureRecogniser)
20:
21:   translation = gestureRecogniser.translation in the view
22:   theView = gestureRecogniser.view
23:   theView.center = new Point(from: view.width / 2 + translation in x and view.height/2 +
translation in y)
24:   xFromCenter = theView.center.x - view.width/2
25:   rotation = CALCULATE rotation angle with xFromCenter
26:   scale = min(absolute(100/ xFromCenter), 1)
27:   stretchRotation = rotation.scaledBy(scale)
28:   theView.transform = stretchRotation
29:
30:  IF user finishes dragging the card THEN
31:    IF theView is left of the screen THEN
32:      call: didChoose(applied: false)
33:    ELSE IF theView is right of the screen THEN
34:      call: didChoose(applied: true)
35:    ENDIF
36:
37:    rotation = 0
38:    stretchRotation = rotation.scaledBy(1)
39:    theView.transform = stretchRotation
40:    theView.center = new Point(from: view.width/2 and view.height/2)
41:  ENDIF
42: Endprocedure

```

The input in this algorithm is how the musician has interacted with the gig card (left or right). For the dragging animation, gigEventWasDragged() is called when the touch screen is being used (is a gesture recogniser function call). The animation involves a translation which returns a vector of where the user drags to. The view's centre is set in the middle of the screen plus the translation on each coordinate axis. It works out a distance from the centre and uses this distance to calculate the angle of rotation. This rotation is assigned to the view. When the user finishes dragging the card the IF statements decide if the view is left or right of the screen to then call a function to respond to the event's organiser. When didChoose() is called, the musician is appended to the applied users dictionary so the gig does not appear to them again (displaying the correct thing – success criterion 4). updateActivity() is called to send a notification to meet success criterion 1 so that contact between two users is made quickly. The dragging feature should be the main attraction to the view capturing what was wanted by my end users in the specification.

## Upload notification object to Database:

```

1: Procedure updateActivity()
2:
3:   notificationID = unique string
4:   relatedEventID = interactedGigEvent.getID()
5:   senderUid = user.uid
6:   receiverUid = interactedGigEvent.getuid()
7:   senderName = user.senderName
8:   notificationPicURL = user.notificationPicURL
9:   notificationDescription = "description of notification
10:  timestamp = new Date()
11:
12:  notificationData = [{"notificationID": notificationID, "relatedEventID": relatedEventID,
13: "type": "applied", "sender": senderUid, "receiver": receiverUid, "senderName": senderName,
14: "picURL": notificationPicURL, "description": notificationDescription, "timestamp": timestamp}]
15:
16:  returnedUser = getDBUserProfile(uid: receiverUid)
17:  call: sendPushNotification(to: returnedUser)
18:
19:  notificationData["senderName"] = "You"
20:  notificationData["receiver"] = senderUid
21:  notificationData["type"] = "personal"
22:
23:  call: updateDBData(data: notificationData)
24:
25: Endprocedure

```

This procedure is similar to the last updateActivity() algorithm, however this time, the notification is being sent to the uid of the organiser that created the event. In addition, a personal notification is sent as well. Both users are notified of the musician's actions to create an effective view to review activity hence following the specification.

## ActivityFeedVC class

### Displaying notifications and events in UICollectionView:

```

1: uid = Auth.auth().getCurrentUser()
2: user = getDBUserProfile(uid: uid)
3:
4: Procedure refreshActivityFeed()
5:
6:   activityNotifications = getDBActivityFeed(uid: uid)
7:   activityNotifications = activityNotifications.sortedByTimestamp()
8:
9:   //Notifications Section
10:  FOR notification in activityNotifications
11:    DISPLAY notification data in ActivityFeedCell
12:  NEXT notification
13:
14:   //My Events Section
15:   eventListings = [GigEvent]
16:
17:   eventIDs = getDBUserEvents(uid: uid)
18:   FOR eventID in eventIDs
19:     returnedGigEvent = getDBSingleEvent(uid: uid, eventID: eventID)
20:
21:     IF returnedGigEvent != nil THEN
22:       eventListings.append(returnedGigEvent)
23:       DISPLAY returnedGigEvent data in ActivityFeedCell
24:     ELSE
25:       index = eventIDs.index(of: eventID)
26:       eventIDs.remove(at: index)
27:     ENDIF
28:   NEXT eventID
29:
30: Endprocedure

```

The procedure consists of getting all the ActivityNotification objects in an array and sorting them, so they are in reverse chronological order. These objects are instantiated with the notification data added to the database shown in the previous algorithm. Looping through this array, they are displayed in a table view feed cell each time. Then, an array of all the gig ids associated to the user are returned. For each id, I return the GigEvent object and append it to an array which gets displayed in a separate section of the activity feed. If I try and fetch a GigEvent object from an id but it returns as nil, then it no longer exists in the database. Therefore, for efficient Database usage, I delete the id in the database and remove it from the local array.

## Deleting notifications and events:

```

1: Procedure deleteNotification()
2:   row = row of ActivityFeedCell
3:
4:   IF notifications section THEN
5:
6:     call: deleteDBData()
7:     activityNotifications.remove(at: row)
8:   ELSE
9:
10:    IF user is organiser THEN
11:      call: deleteDBData()
12:      call: deleteSTFile()
13:    ENDIF
14:
15:    eventIDs.remove(at: row)
16:    eventListings.remove(at: row)
17:  ENDIF
18: Endprocedure

```

When deleting a notification, if could be from either section in the table view feed. If it is a notification, then remove it from the database and from the local array. However, if it is all the events related to the user, and the user is the event's organiser, the object can be deleted from the database and the photo file deleted from Storage. Both users remove their association to the event by deleting the id under their user object in Database. This allows users to edit and review things associated to their account (criterion 4) and manage the Database efficiently (6).

## ReviewApplicationVC class

### Displaying musicians application to the organiser:

```

1: application = clicked notification
2: user = new User(from: getDBUserProfile(uid: application.getSenderUid()))
3: currentUser = new User(from: Auth.auth().getCurrentUser())
4: Procedure refresh()
5:   DISPLAY user data in view
6:
7:   relatedEvent = getDBSingleEvent(uid: user.uid, eventID: application.getRelatedEventId())
8:
9:   DISPLAY relatedEvent data in view
10: Endprocedure

```

The application is the notification object sent by the musician and received by the organiser. I instantiate a User object from the uid of the musician (shown in getDBUserProfile()), and also instantiate a User object from the uid of the current user (the organiser of the event). As part of this algorithm, I display everything about the applying user and the event data in the view for the organiser to make a decision, hence meeting success criterion 4. This makes

contact easy and understandable as part of success criterion 1. The organiser should be able to see the portfolio here too to get all the musician information relevant which was wanted by my stakeholders and included in the specification.

## Upload notification object to Database of their response:

```

1: Procedure updateActivity(accepted)
2:   notificationID = unique string
3:   senderuid = currentUser.uid
4:   recieveruid = user.uid
5:   senderName = currentUser.senderName
6:   notificationPicURL = currentUser.picURL
7:   relatedEventTitle = relatedEvent.getTitle()
8:   relatedEventID = relatedEvent.getId()
9:
10:  notificationDescriptpion = ""
11:  IF accepted THEN
12:    notificationDescription = "hired you for the event: " + relatedEventTitle
13:
14:    call: sendPushNotification(to: recieverUid)
15:  ELSE
16:    notificationDescription = "declined you for the event: " + relatedEventTitle
17:
18:  ENDIF
19:
20:  timestamp = Date()
21:
22:  notificationData = [{"notificationID": notificationID, "relatedEventID": relatedEventID, "type": "reply", "sender": senderuid, "reciever": recieveruid, "senderName": senderName, "picURL": notificationPicURL, "description": notificationDescription, "timestamp": timestamp}]
23:
24:  call: updateDBData(notificationData)
25:  notificationData["senderName"] = "You"
26:  notificationData["reciever"] = senderUid
27:  notificationData["type"] = "personal"
28:  notificationData["description"] = "hired " + user.name + " for the event: " + relatedEventTitle
29:  call: updateDBData(notificationData)
30:
31: Endprocedure

```

Responding to the musician's application involved sending a notification. Depending on whether the organiser has accepted or rejected them, a notification object is sent to the database for the musician to view. A push notification is also sent when the musician does not have the app open which was an important reason to remain logged in as part of success criterion 5. Similar to other `updateActivity()` algorithms, a personal notification is sent to the current user so they can keep track of who has been hired for their event.

## TabBarController class

Determine what tabs appear to musicians and event organisers:

```

1: userGigs: Boolean
2: user: User
3: tabs = [UIVIEController]
4: Procedure refreshTabs()
5:   uid = Auth.auth().getCurrentUser()
6:
7:   //user is logging in
8:   IF userGigs == nil THEN
9:     user = getDBUserProfile(uid: uid)
10:    userGigs = user.gigs
11:  ENDIF
12:
13:  IF userGigs == true THEN
14:    tabs.remove(at: 0)
15:  ELSE
16:    tabs.remove(at: 1)

```

```

17:  ENDIF
18:
19: Endprocedure

```

This final algorithm controls what users see what tabs as part of the TabBarController. There is a global variable tabs which holds an array of all 4 view controllers (or tabs) as part of the tab bar. If the user is logging into the app, then I set assign userGigs a Boolean to check if the user is an organiser or a musician. However, if they are resuming the app (or first launching it and is already logged in), userGigs will already have a value. Either way, if userGigs is true (they are musician), remove tab at index 0; the create event tab. If userGigs is false (they are organiser), remove tab at index 1; the find gigs tab. As stated previously, restricting access to various views and tabs depending on user type holds the structure and purposes set out by the specification. It also eliminates the possibility of incorrect data being accessible and on display (success criterion 4).

## Test Strategy

### Overall Approach to Testing

To ensure that the end users' needs are fully met, I will start by alpha testing (myself as the programmer) to ensure that the program works as I expect it to. Alpha testing will involve white box testing which includes testing each part of my algorithms. To fix encountered bugs I will use breakpoints to examine the state of the variables during my code's execution. The second part of alpha testing is black box where I will focus on the input and output of the system wherever it applies rather than testing the working of my algorithms. My end users will not see the inner workings of my app and therefore black box testing is very important to simulate results my end users would expect to see on the front-end.

Following this, I will conduct beta testing where I load the app onto each of my stakeholder's phones and explain what is new and what has changed. Then, after letting them play around with the latest version, I will ask them if it is as they intended and if there are any bugs I have looked over. I will then ask for their overall feedback as part of acceptance testing where we discuss changes that are required and improve the system.

## Milestone Plan

### Milestone 1 – Login/Signup Views and Authentication

The first milestone will be creating Login and Signup pages and setting up Google Firebase Authentication API to create and manage secure accounts. I will also set up Firebase Database to store user profile data.

### Milestone 2 – Portfolio and Post Views

The second milestone will be to complete a view where musicians and event organisers can build a portfolio by posting photos and videos in a scrolling table view feed. I will need to setup post data in the database and store image/video data with Firebase Storage.

### Milestone 3 – Create Event Views (for organisers)

Milestone 3 will allow organisers to add all necessary details for their public event in a guided process. Attaching a location to this event will be added in Milestone 5, as the purpose of Milestone 3 is to create an object for musicians to receive and test it works.

## Milestone 4 – Find Gig View (for musicians)

This milestone lets musicians pull new events from the database to apply for with a swipe gesture.

## Milestone 5 – Sort Gigs by Location

Enable feature to add a location when creating an event and use a sorting algorithm to arrange gigs by distance from this location when the musician pulls from the database.

## Milestone 6 – Display Event in Detail View and observe other Portfolios

Milestone 6 will be to complete the view where a musician can read what they are applying for in more detail. Also, reuse the Portfolio view to display another user's profile.

## Milestone 7 – Account Notifications Views and Send Notifications

I will go back and create a notification object to store in the database every time a user performs a task which needs to notify someone else. Create a view to collect all the notifications and display them in a table view feed.

## Milestone 8 – Review Application View (for organisers)

This milestone will be to complete a view where the event organiser can review an incoming applicant from a notification and reply with a button tap of 'accept' or 'reject'.

## Milestone 9 – Edit Account

Enable feature so all users can edit their account data changing what is seen on their portfolio.

## Milestone 10 – Edit/Delete Events

Enable feature so organisers can edit their events (or delete them completely) and allow musicians to remove their association with an event.

## Milestone 11 – Device Push Notifications and Social Media Links

Milestone 11 is about enhancing contact between two users. Send a push notification with Google Firebase Cloud Messaging API whenever a notification object is created and enable links to navigate to social media apps and websites.

## Milestone 12 – Constraints for all iPhone Devices and User Interface finishing touches

The final milestone will involve adding the finishing touches so the UI matches my designs. I will add constraints so that the layout is the same for all size of iPhone devices (from minimum of iPhone 5 to maximum of iPhone XS)

## Data Sets for Testing

My app requires various types of user input due to some forms to create accounts and create an event. I have therefore shown the test data inputted for my milestone alpha testing:

## Milestone Test Plans

### Milestone 1 – Login/Signup Views and Authentication

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Testing valid inputs when creating an account.	3 – Validation rules	["bio": "Biography of the Test User", "instagram": "", "website": "", "appleMusic": "", "spotify": "", "phone": "", "gigs": false, "email": "testuser1@gogig.com", "name": "Test User", "picURL": String_URL, "twitter": "", "facebook": ""]	When adding emails, passwords and profile information, the program should display an error if the input is unexpected, does not meet requirements or if nothing was entered.	Valid and Invalid test data
2	Check account creation.	5 – Authentication system 6 – Efficient storage in Database	Button	Checking account was created with Firebase Auth and entered profile data was added to Firebase Database.	Valid
3	Make sure user can swap between options to sign up and log in	1 – Understandable user interface	Button	If user already has an account, the mode needs to be swapped when requesting to log in. User will resume their account rather than create a new one as expected.	Valid
4	Ensure user can pick a profile picture from their camera or library	8 – Access camera and media library	Camera or Device Photo Library App	After taking or choosing an image, the returned image should be displayed to the user and stored in Firebase Storage when user finished account creation at Sign up.	Valid
5	Check user remains logged in when leaving or	5 – Remain logged in	Firebase Authentication	When the app is opened, login page is displayed if the user is using the app without a current associated account.	Valid

	closing the app				
--	-----------------	--	--	--	--

## Milestone 2 – Portfolio and Post Views

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Check user profile data is displayed as expected.	4 – Data displayed when required	Firebase Database	All data entered at account sign up should be grabbed from the database and displayed at the top of the portfolio.	Valid
2	User should be able to log out of their account	2 – Easy navigation 5 – Login authentication	Button	When requesting to log out, the Login view is displayed to the user.	Valid
3	Check navigation to Post View	2 – Easy navigation	Navigation Button	The Post View is displayed to the user when they request it. They should be able to go back, and view should dismiss anyway when they complete a post.	Valid
4	Make sure Google Places API returns a location string when searched to attach to the post	7 – Optional location usage	Google Places API	The Google Places Autocomplete View should display and return a location name when searching with the search bar.	Valid
5	Access camera, image library and video library	8 – Personalisation of portfolio	Camera or Device Photo Library App	After taking or choosing an image, the returned image should be displayed to the user and stored in Firebase Storage when user completes their post.	Valid
6	Test valid inputs when creating a post	3 – Validation rules 6 – Data stored efficiently	["postURL": String_URL, "postID": String_Id. "caption": "This is the post caption", "location": "Location", "isImage": true/false]	If user has not added an image or video when trying to post, app should display an error. When post is complete, the object should be added to the Database.	Valid and Invalid test data

7	Make sure the post is displayed in a scrolling table view feed.	1 – Good user experience and user interface	Firebase Database	Portfolio posts (and their data) should be displayed in a scrolling table view feed. A new post should be added automatically without the user having to manually refresh the feed.	Valid
8	Check user can delete a post from the feed and from the database	4 – User can edit their account data 6 – Data stored efficiently	Button	When clicking a button, the table is refreshed removing the selected post from the feed. In the backend, the post object is deleted from the database and the image (or video) is deleted from Firebase Storage. All other posts should be unaffected.	Valid

### Milestone 3 – Create Event Views (for organisers)

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure all navigation buttons work as expected	2 – Easy to navigate	Navigation Buttons	All 'continue' buttons and 'back' buttons should take the user to expected parts of the Create Event tab.	Valid
2	Test valid inputs when creating an event	3 – Checking validation rules 2 – Report back errors	<p>["eventPhotoURL": String_URL, "longitude": 0.0, "uid": String_Uid, "name": "Test User", "locationName": "Suffolk One", "eventID": String_Id, "title": "Test Event 1: Jazz Night", "timestamp": "yyyy-MM-dd HH:mm:ss +zzzz", "description": "Looking for Jazz bands to perform a two hour set for our event at Suffolk One. The aim of this event is raise money for charity and inspire music students.", "latitude": 0.0, "email": "testuser1@gogig.com", "payment": 500.0, "phone": "01473 556601", "postcode": "IP8 3SU", "appliedUsers": ["CREATOR:String_Creator_Uid: true]]</p>	If user has not entered required information about the event, app should display an error and not progress.	Valid and Invalid test data
3	Make sure Google Places API returns a location string when searched to	7 – Optional location	Google Places API	The Google Places Autocomplete View should display and return a location name when searching with the search bar.	Valid

	attach to the event				
4	Access camera, image library and video library	8 – Access device camera and media library	Camera or Device Photo Library App	After taking or choosing an image, the returned image should be displayed to the user and stored in Firebase Storage when user completes their post.	Valid
5	Check gig event creation	6 – Efficient Database storage	Button	The event should be added publicly to the database with all correct data	Valid

### Milestone 4 – Find Gig View (for musicians)

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure we can grab all events as gig opportunities	6 – Efficient data storage in database	Firebase Database	Query should return an array of Events to be displayed to the musician	Valid
2	Test swipe gesture to apply for gig	7 – Good user experience 7 – Simple Interaction	Touch screen gesture	Data about an event is displayed in a draggable card. When swiped to the left or right, database should update listing the musician as an interacted user.	Valid
3	Check that gig opportunities already interacted with do not appear again.	3 – Check against rules to avoid error and confusion	Firebase Database	If the user has already interacted with the event. The program should block the display of that event object.	Invalid

### Milestone 5 – Sort Gigs by Location

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure we can fetch the current location of the device	7 – Location services accessed	Device location services	When requested, location coordinates are printed to the console and updated when changing location.	Valid
2	Enable optional feature to	9 – Users decide on the amount of	Button	If user pins their current location to the event, database should store their	Valid and Borderline

	attach these coordinates to an event when creating one.	information they share.		coordinates under the event object. The program should mark coordinates as 0.00, 0.00 if location services is disabled by user.	
3	Check program can calculate distance to event location	7 – Location accessed	Firebase Database, Device location services	Program should be able to calculate distance between the musician's coordinates and the event coordinates.	Valid
4	Make sure sorting the event array by location works	7 – Data is sorted by locality.	The GigEvent objects	I will check that the sorting algorithm works by manually inputting coordinates from Google Maps and making sure they appear to the musician as more local first.	Valid

### Milestone 6 – Display Event in Detail and Observe Other Portfolios

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Check navigation to view works	2 – Easy to navigate.	Navigation Button	Program should navigate to new view with the event data ready to be displayed. Should also navigate to the portfolio view with associated user data ready to be displayed.	Valid
2	All event data is displayed correctly	3 – Avoid unexpected results	Firebase Database	The view should display all data about the event to the user in a logical format.	Valid
3	Portfolio view is reused and refreshed to display another user	1 – Making contact between users easy 4 – Users can only edit things associated to their account	Firebase Database	Program should be able to access another user's profile and post data to display in the same the same way as the current user's portfolio. Options to add/delete posts should be hidden.	Valid

### Milestone 7 – Account Notification View and Send Notifications

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure notification	5 – Notifications		Whenever an action is taken which involves another user	Valid

	object is created and added to database	and activity updates		(musician applied or organiser responded to application), app should create a notification object associate to the user to store in the database.	
2	Check notifications are added to a scrolling table view feed	1 – Good user experience and user interface.	Firebase Database	Notification objects (and their data) should be displayed in a scrolling table view feed. A new notification should be added automatically without the user having to manually refresh the feed.	Valid
3	Check events associated with that user are displayed in 'My Events' scrolling table view feed.	3 – Check validation rules to get expected results 4 – User can review anything associated to their account	Firebase Database	Events associated to a user (and their data) should be displayed in a scrolling table view feed. An event should be added automatically to this section when an organiser has created a listing and when a musician is confirmed to play for that event.	Valid

### Milestone 8 – Review Application View (for organisers)

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure that only organisers can navigate to this view	2 – Easy navigation 4 – User can edit anything associated to their account only	Table view feed cell	App should check the user is an organiser and the notification clicked on is a job application to navigate to this view.	Valid
2	Portfolio view is reused and refreshed to display another user.	1 – Making contact between users easy 4 – Users can only edit things associated to their account	Firebase Database	To review the event applicant, the program should refresh the portfolio view again to display the musician's work. Options to add/delete posts should be hidden.	Valid
3	Check a notification is sent back to musician confirming	5 – Receive notifications and activity updates	Button	When the organiser accepts the musician's application, by clicking a button, a notification object should be added to the database to	Valid

	that they are playing for that event.			inform the user they got the job. This in turn must add the event listing to the musicians 'My Events' feed.	
--	---------------------------------------	--	--	--	--

### Milestone 9 – Edit Account

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure user can navigate to the edit account views.	2 – Easy navigation	Navigation buttons	From the portfolio view, the user must be able to navigate to the same views as account creation to edit their data.	Valid
2	Text fields are automatically filled with their current profile data.	1 – Good user experience 1 – Simple interaction	Firebase Database	The app should display their current account data, so they <u>edit</u> their data rather than input it again entirely. Their current data should be fetched from the database.	Valid
3	Test valid inputs when editing.	3 – Validation rules avoid error	<p>["bio": " Edited biography of the Test User", "instagram": "", "website": "", "appleMusic": "", "spotify": "", "phone": "", "gigs": false, "email": "testuser1@gogig.com", "name": "Test Use Editedr", "picURL": String_New_URL, "twitter": "", "facebook": ""]</p>	Similar to account creation, the same validation rules should be applied when editing their account.	Valid and Invalid test data
4	Check account data is updated	6 – Efficient database storage	Button	The database should be updated with the new profile data and all displayed information in the portfolio should be refreshed to reflect these changes.	Valid

### Milestone 10 – Edit/Delete Events

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure only organisers can navigate to views to	2 – Easy navigation 4 – User can edit anything associated to	Navigation buttons	If current user is a musician, the program should block the ability to edit the event.	Invalid

	edit their events	their account only			
2	Text fields are automatically filled with the event information	1 – Good user experience 1 – Simple interaction	Firebase Database	The app should display the current event data, so they <u>edit</u> their data rather than input it again entirely. Their current data should be fetched from the database.	Valid
3	Test valid inputs when editing an event.	3 – Validation rules avoid error	<pre>["eventPhotoURL": String_New_URL, "longitude": User_Lat, "uid": String_Uid, "name": "Test User", "locationName": "Suffolk One Edited", "eventID": String_Id, "title": "Test Event 1: Jazz Night Edited", "timestamp": "yyyy-MM-dd HH:mm:ss +zzzz", "description": "This is an edited event.", "latitude": 0.0, "email": "testuser1@gogig.com", "payment": 900, "phone": "01473 556601", "postcode": "IP8 3SU", "appliedUsers": ["CREATOR:String_Creator_Uid: true]]</pre>	Similar to event creation, the same validation rules should be applied when editing the event.	Valid and Invalid test data
4	Check event data is updated	6 – Efficient database storage	Button	The database should be updated with the new event and all displayed information in the 'My Events' section should be refreshed to reflect these changes.	Valid
5	Check deleting an event takes appropriate action corresponding to type of user.	6 – Efficient database storage	Button	When deleting an event in the 'My Events' section, if the user is a musician, then it should just delete their association with that event. However, if the event's creator, it should delete the event publicly and remove it entirely from the database.	Valid
6	Check user can delete a notification from the feed and the database	4 – User can edit anything associated to their account	Button	When clicking a button, the table is refreshed removing the selected notification from the feed. In the backend, the notification object is deleted from the database. All other notifications should be unaffected.	Valid

### Milestone 11 – Device Push Notifications and Social Media Links

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid

1	Send a test notification from Firebase Website to specific devices	5 – Receive notifications when app is not in use	Firebase Cloud Messaging  <b>Title: Test Notification</b> <b>Message: This is a test notification</b>	This is to check that a device can receive a push notification when one is sent. The push notification should come through with custom text, appear on the lock screen and when the app is not in use. The push notification should not be received when the app is in use.	Valid
2	Send test notification from one device to another.	5 – Receive notifications when app is not in use	<b>#1 Title: Application Pending</b> <b>Message: Sender has applied for your event</b>  <b>#2 Title: You got the gig!</b> <b>Message: Sender has hired you for the event:</b> <i>eventName</i>	This is to check that a device can send and receive a push notification. The push notification should come through with custom text, appear on the lock screen and when the app is not in use. The push notification should not be received when the app is in use.	Valid
3	Check push notification is sent whenever necessary.	5 – Activity updates	Device	The program should send a push notification to the correct device whenever the database is updated with a new notification object.	Valid
4	Make sure the app can be opened from the push notification.	1 – Good user experience	Push notification	The app should launch after opening from the push notification on the lock screen or the device home page.	Valid
5	User can add social media links and check against validation rules.	1 – Make contact between users 3 – Check against rules to avoid error 9 – Social and contact links	Phone: +44 7496243062 Website: chilly-designs.com Instagram: l_chilvers Twitter: chilvers_lee Facebook: 1837812439827573 Apple Music: <a href="https://music.apple.com/gb/artist/cody-fry/289832557">https://music.apple.com/gb/artist/cody-fry/289832557</a> Spotify: <a href="https://open.spotify.com/artist/7dOCnyDR2oEa1hQlvTXvdT">https://open.spotify.com/artist/7dOCnyDR2oEa1hQlvTXvdT</a>	The app should compare entered information against validation rules to ensure that the social media link will likely work when displayed on the portfolio page.	Valid and Invalid test data

<b>6</b>	Check we can open social media apps and web URLs from GoGig.	2 – Easy navigation	Button	Social apps corresponding to the button pressed should be opened and navigate to the correct account. If apps are not installed, then it should open a website in Safari web browser. If entered information is invalid, the app should notify the user that this link does not exist rather than throw error.	Valid and Invalid
----------	--	---------------------	--------	--	-------------------

## Milestone 12 – Constraints for all iPhone Devices and User Interface Finishing Touches

<b>Test No</b>	<b>Test Explanation</b>	<b>Success Criteria Point/s</b>	<b>Input Dataset/s</b>	<b>Expected Outcome</b>	<b>Valid/Invalid</b>
<b>1</b>	Compare features to the app design	1 – Aesthetically pleasing UI		The look of the program should match the layout and design of the solution, unless improvements are decided in development and testing.	Valid
<b>2</b>	Check auto-layout constraints work as expected	1 – Aesthetically pleasing UI	Auto-layout constraints	Running the app on the minimum iPhone specified and upwards should show a similar layout. No features should be inaccessible on a particular device.	Valid

## Stakeholder Testing

### Post Development Test Plan

Test No	Task for End User	Success Criteria Point/s	Justification	Valid/Invalid
1	Open the app from the iPhone home screen	2 – Easy navigation	Users should be able to find the app and search for it from the phone home screen.	Valid
2	Login view swaps between Sign up and Log in	1 – Simple and understandable interaction 2 – Easy navigation	So that a user can choose to login or create a new account at any time.	Valid
3	Can navigate to account creation process and you can enter your information with ease	1 – Simple and understandable interaction 2 – Easy navigation 8 – Personalise their experience	So that users can start creating their account	Valid
4	Try entering invalid data (leaving a field blank or adding a name more than 50 characters)	2 – User errors are reported back and easily recoverable	Check to see if the app prevents invalid data entry and prevent progression if anything wrong is entered.	Invalid
5	Is the action sheet shown when you add a profile picture? And is an image returned when you use the camera or pick from the image library?	8 – Able to access the iPhone camera and library	Allows the user to add a picture for their profile to personalise the experience.	Valid
6	Add social links when creating an account, does the app report back if you have entered something wrong?	2 – Report back error 9 – Social media links	This test ensures that the system reports back anything wrong with the social links that they provided.	Valid and Invalid test data
7	Progress with 'Continue' without adding any social information. This feature should be optional.	2 – Navigation 9 – User decides on information associated to account	This test ensures that the social links are optional and an account can be created without them.	Valid
8	Only a musician can navigate to the music links view	2 - Navigation	This test ensures that only musicians have the option to add a music links as only they can navigate to it.	Valid

9	App reports back when music links are added incorrectly and can even progress without adding any at all as this is an optional feature.	2 – Report back error 9 – User decides on information associated to account	The test makes sure that music links are optional and a musician account can be created without them.	Invalid
10	Do you get taken to the portfolio view where you can see your account data just entered?	2 – Navigation 5 – Data is held under account	Test ensures the app takes them to the portfolio view automatically.	Valid
11	Press 'Settings' and choose to log out. Is the login page presented and can you log into your account again. Your data should be seen same as before. Login to each other's accounts to make sure Authentication works.	5 – Working authentication system	Test ensures a user can log in and out of their account and even access their account on a different device with Authentication.	Valid
12	Double tap the iPhone home button and flick the app up. This closes the app. Now press the app icon again to launch it. You should be taken straight to your portfolio.	5 – User is remained logged in until they choose to log in	Test ensures that they remain logged in ready to receive push notifications later on.	Valid
13	Navigate to 'Add a Post' page, and choose a post from your camera, image library and video library.	2 – Navigation 8 – Able to access camera and library	Test ensures that the user can post whatever they like to be shown in their portfolio which potential employers will see.	Valid
14	If you click 'Add Location' is a search view presented (however searching will not work)	7 – Location features	This test ensures that the Google API view gets presented to the user despite the fact they can't use it due to the fact it needs to be paid for.	Valid
15	Try posting without choosing an image (or video)	2 – Report back error	The app should stop the user making a post without actually adding an image or video.	Invalid
16	After making a post, you should be taken	5 – Post Data held under account	Test ensures that a portfolio is built with all the posts' data	Valid

	back to your portfolio and your posts should be in the feed in reverse chronological order.	6 – Post data stored efficiently	shown in the table view cells. Other users will see this later.	
17	Clicking the three dots by the post you can delete one of your posts.	4 – Users can edit anything associated to their account 6 – Efficient database storage 9 – Users decide on amount of information they share	This test will make sure that a post is removed from the database and from the portfolio view so it cannot be seen by any user.	Valid
18	Make sure only three tabs are seen at the bottom of the screen. Musician should see a guitar and an organiser should see a notepad. Is this consistent when you close the app and log in and out?	2 – Easy to navigate 3 – Avoid error	This test will make sure that users can only navigate to the areas of the app which belong to that type of user.	Valid
19	Click on every social link in your portfolio.	9 – Social Media links	This test ensures that the links entered do in fact open up the correct apps and show the user's profile.	Valid
<b>Gavin Thorrold (organiser)</b>				
20	Click on the left-most (notepad) tab.	2 – Easy navigation	This checks that the organiser can navigate to create a public event.	Valid
21	Try and enter some invalid data (miss out a field and try to select a date back in time)	3 – Avoid error	Making sure that only valid information about the event being created can be entered.	Invalid
22	Press 'Use current location' does the compass icon come up next to your battery percentage?	7 – Location services are accessed	This makes sure that the device location can be accessed and used when creating an event to sort by distance later.	Valid
23	Choose an image for the event.	8 – Camera and photo library access	This test ensures that an image URL is attached to the event before posting so it will be shown to the musician later.	Valid
24	Click 'Post Event'. Are you taken to your activity feed with the most recent	2 – Navigation 3 – Check all input against validation	This test will tell the organiser their event has been created.	Valid

	notification being “You created the event: eventName”?	4 – User can review anything associated to their account		
25	Click ‘My Events’ in the menu bar, the event you just created should be seen there.	4 – User can review anything associated to their account	Test ensures that the event can be viewed and edited by the event’s organiser.	Valid
26	Click on the notification in the ‘My Events’ section	2 – Navigation 4 – User can review anything associated to their account	Test ensures that both users can see all the information about the event in detail.	Valid
27	Click ‘Edit’. Are you taken to the same page as when you created the event? Try changing some of the values you entered and edit the event.	4 – User can edit anything associated to their account at any time.	Test makes sure that the organiser is able to change features of their event. It also checks that information is auto filled out so that if anything is not changed, it keeps the same value.	Valid and Invalid test data
28	Close the app	N/A	Gavin is ready to receive push notifications from Jude Mills	N/A
<b>Jude Mills (musician)</b>				
29	Gig opportunities appear created by Gavin Thorrold, click “Tell me more”. Click “Check out Gavin Thorrold” to view his portfolio.	1 – Contact made between musicians and organisers 2 – Easy navigation	This test makes sure that the musician can navigate to read the event in full detail. It also tests that they can go and view the other user’s portfolio before making a decision. Here they can click on social links and view the portfolio like they did their own before.	Valid
30	Navigate back to the previous view and drag the card right. There will either be more gig opportunities created by Gavin or you will be notified that there are no more to apply to. When there are no more, press the refresh button.	1 – Simple and understandable interaction. 3 – Avoid error to avoid confusion	This test makes sure that musicians can apply to events and that they do not appear again when the view is refreshed as they have already been interacted with.	Valid
31	Close the app.	N/A	Jude Mills is ready to receive push notifications from Gavin	N/A
<b>Gavin Thorrold (organiser)</b>				
32	You should have a push notification come through from	1 – Good user experience 1 – Contact made	This test is to make sure that an organiser receives a push	Valid

	Jude Mills applying to your event. Go to notification centre and open the app through the notification.	2 – Easy navigation 5 – Receive notifications and activity updates	notification when a musician applies to their event.	
33	Click on the notification in the activity feed telling you that Jude has applied. Click “Check out Jude Mills” to view his portfolio.	1 – Contact made between musicians and organiser 2 – Easy navigation 4 – Review anything associated to their account. 5 – Activity updates	This test will make sure that the activity feed updates that Jude has applied and will also test if the organiser can view a musician’s portfolio and click on all his social links.	Valid
34	Navigating back, accept Jude’s application by clicking the big tick.	1 – Contact made 1 – Good user experience	Test ensures that organiser is sent back to the activity feed after replying to an application.	Valid
<b>Jude Mills (musician)</b>				
35	You should receive a push notification telling you that Gavin accepted you to play. Open the app from the notification.	1 – Good user experience 1 – Contact made 5 – Activity updates	This test ensures that the musician receives push notifications too letting them know if they were rejected or accepted.	Valid
36	You should see a notification in the feed telling you that you that you are hired.	1 – Contact made 5 – Activity updates	Test ensures that Jude gets a notification telling him he is booked to play for Gavin’s event.	Valid
37	This means that if you click “My Events”, you can view all the events you are booked to play for here and view them in detail.	2 – Easy navigation 4 – Can review anything associated to their account at any time.	Test ensures that the gig object has been associated to the musician for them to refer back to.	Valid
<b>Both</b>				
38	Click the date of the event in “My Events” section and navigate to the calendar app on the iPhone.	Non-essential point covered: 10 – Smartphone calendar integration.	This test ensures that the event and its description is added to the device calendar so that the user can receive alerts when it’s closer to the time.	Valid
39	Navigate back to your portfolio, click “settings” and “edit profile”. You are taken to the same screen as when you first created your account. Change a	2 – Easy navigation 2 – Report back error. 3 – Check all input against validation rules to avoid error and confusion	This test will check to see if the user can edit their account information and this change will be reflected everywhere necessary in the app. Allowing the user to edit their information gives them freedom in the app’s usage.	Valid and Invalid test data

	few items such as your profile picture.	4 – User can edit anything associated to their account 6 – Efficient data storage in Firebase 8 – Access camera and photo library 9 – Decide on amount of information that they share. Social media is optional.		
--	---	---	--	--

## Acceptance Testing – End User Questionnaire

1. On a scale of 1 – 10, how much did you like the design of the app and its user interface?
2. On a scale of 1 – 10, how satisfied were you with the user experience?
3. Were there any stages when you didn't know what to do as part of the testing session?
4. Did you spot any errors or bugs when you used the app? If so, what were they?
5. On a scale of 1 – 10, how satisfied were you with the error messages for your invalid data input?
6. On a scale of 1 – 10, how much user freedom in the app's navigation did you feel you had?
7. On a scale of 1 – 10, how much did you feel that you decided how much information you share?
8. On a scale of 1 – 10, how satisfied were you with the personalisation options of your account and user experience?
9. Are you comfortable with the app using your current location to make results more relevant? If not, why?
10. Are you happy with how I have tackled the problem and satisfied with my solution? If not, how could it be improved?
11. On a scale of 1 – 10, how much do you feel this app is a good solution to helping musicians and organisers get in contact and organise music events quickly?

Question 1 focusses on how satisfied my end users will be with my final design after development of the user interface measuring how successful I was in meeting the specification of building a smartphone app. Answers given towards a 10 suggest success criterion 1 was met in building an aesthetically pleasing user interface.

Question 2 focusses on how satisfied my end users will be with the user experience of the finished app. If a high answer is given, then I have been successful in creating a good user experience for all aspects detailed in the specification and met success criterion 1.

Question 3 will reveal if my design and solution was understandable and easy to use completing success criterion 1 if they agree I did a good job. I can also find out from the question if each of my views and authentication system detailed in the specification were completed successfully.

Question 4 has the purpose of checking my alpha testing was thorough and I truly met the success criteria and specification at the end of each milestone.

Question 5 will evaluate how successful I was in meeting criteria 2 and 3 in trying to prevent errors with validation routines and error alerts. By preventing errors and getting the user to correct and recover from them, it ensures the best chance for all the views to work correctly according to the specification.

Question 6 may further confirm my completion of success criterion 2 in how well I meet the specification while keeping freedom in user navigation.

Question 7 will reveal if I met success criterion 9 well or not. If the users feel they have plenty of choice, and the purpose of each point in my specification works well independently of these choices, I will count the criterion as completely met.

Question 8 focusses on if users like personalisation abilities like setting their profile picture or adding unique images to their portfolio. This question is about whether I achieved a portfolio view which feels like it's a user's own space, which Gavin and Jude both wanted in my research interview.

Question 9 will allow me to get a feel if users like the iPhone using location services if it means data is more relevant to them. If they do like it, it opens up more possibilities in improving user experience for future development.

Question 10 will ultimately evaluate how well I stuck to the specification in completing a solution my users wanted and asked for.

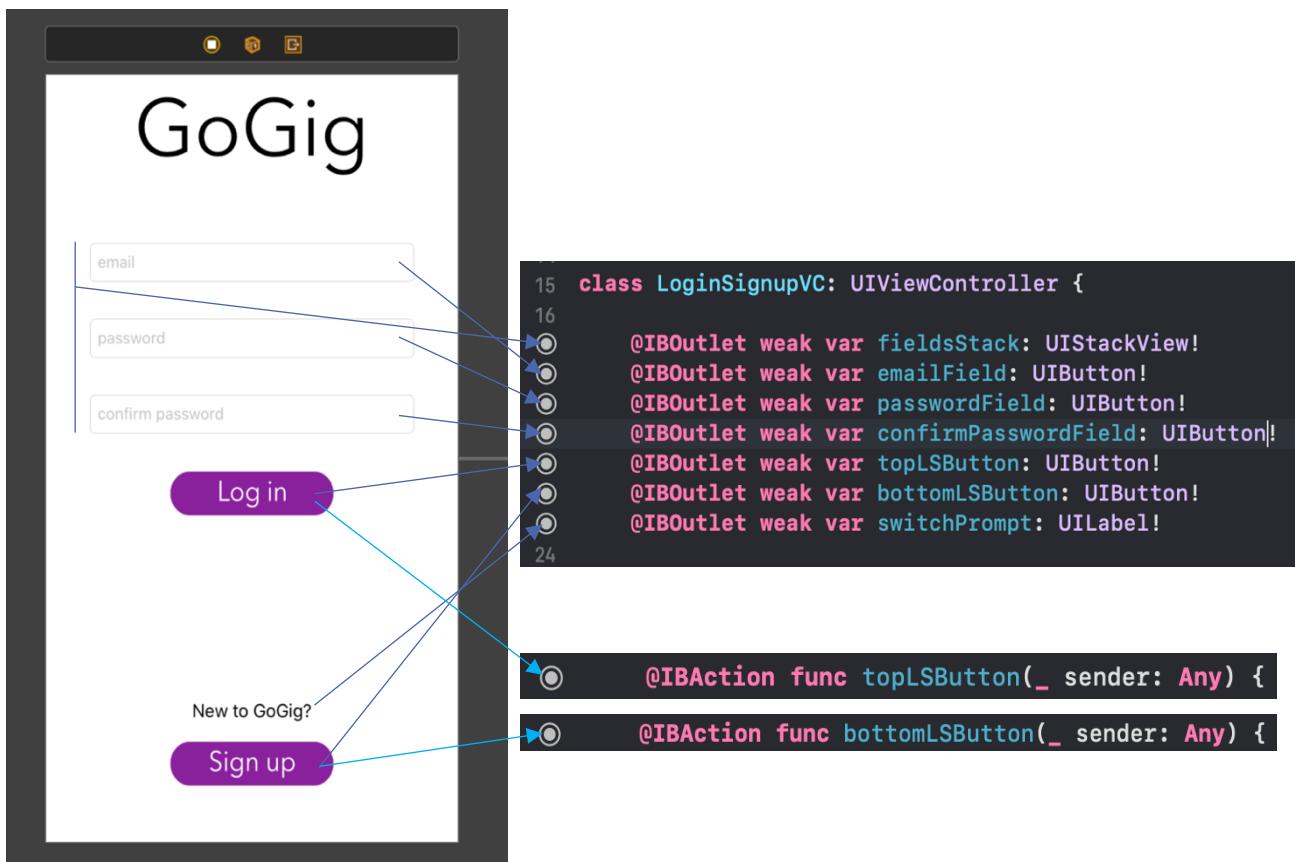
Question 11 will also evaluate if I completed the most important part of success criterion 1: **“The solution meets its purpose of making contact between musicians and organisers quick and easy.”**

# Section 3 – Developing the Solution

## Milestone 1 – Login/Signup Views and Authentication Development Log

To begin the project, I recognised that each view would need its own subclass of the `UIViewController` class. The first thing I wanted to focus on was building a login page which changed its state when a ‘swap’ button was pressed at the bottom. The login page is the first stage of the Authentication System detailed in my specification. The initial state of this view is login, and when the bottom button is pressed the view is set out for signup.

Here is the view in the Interface Builder with connected outlets and actions for `LoginSignupVC`:



I created a function which swaps the state of the view around when the bottom button is pressed. With a variable which records the current state of the view, when the function is called the `confirmPasswordField` should be hidden for login and the two button images swap over. The placeholder of the password field should change to “password” in login mode rather than “create password” in signup mode, so there are detailed instructions making the system understandable (success criterion 1). I made the `switchPrompt` label change too, so the user knows what the bottom button is swapping to. The switch prompt makes the user interface understandable as it gives the user some guidance as part of my success criteria.

When the user presses the topLSButton, depending on the state of the view, the user is either logged in or taken to account creation pages. With **if let userEmail = emailField.text {}** a constant is only created when the emailField.text has a value, preventing progression without required information, preventing error (success criterion 3). This is similar to the passwordField and confirmPasswordField. If this information is not filled out, we display an error to meet the success criteria of error recovery.

This error is displayed as a **UIAlertController** called with a function as follows:

```

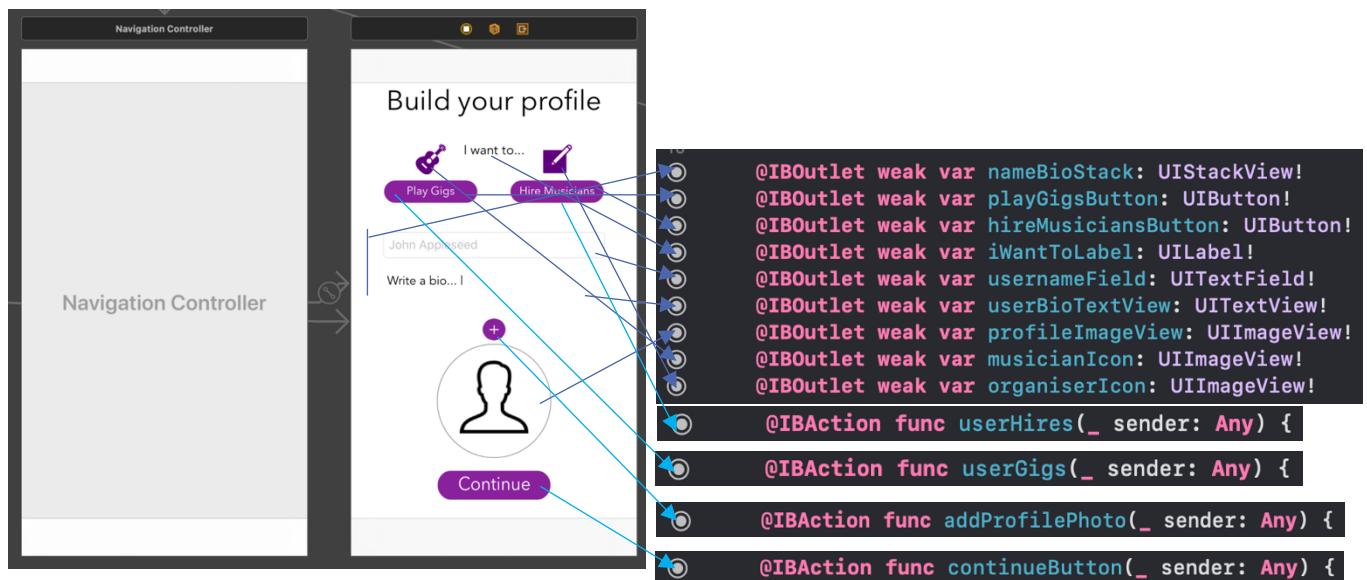
36  func displayError(title: String, message: String) {
37      //alert controller with a title and message and 'OK' dismiss button
38      let alertController = UIAlertController(title: title, message: message, preferredStyle: .alert)
39      alertController.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
40      //present the alert
41      self.present(alertController, animated: true, completion: nil)
42  }

```

The function is defined in an extension file of UIViewController. This means that any future class which inherits from UIViewController can call this function to display the error. The parameters of the UIAlertController include a title and a message which will change depending on the validation rule that was broken. This will be tested later. An “OK” button dismisses this alert so the user can put right the error they have made.

In order to test the login feature, I needed to create an account, therefore I needed to implement account signup first. At this point, when the user requests to sign up, a segue is performed to take them to the CreateProfileCAVC view controller. This class is where the user will put together a profile for an informative portfolio view which helps musicians and hirers secure deals as guided by my specification. Originally, I subscribed the user up to the Firebase Authentication service when pressing ‘Sign Up’. However, I realised that the user’s email should be subscribed at the end of the account creation process, in case they close the app without adding all of their account details. This would cause Firebase to return the error of trying to subscribe the same email twice before the user has the opportunity to add their details. Therefore, the approach I needed to take was when the user navigates through the signup process, data is collected from each view controller and stored in a dictionary when they segue to the next view. These series of view controllers are embedded in a UINavigationController to achieve this dictionary being passed between view controller classes. The UINavigationController is a container view controller that follows a stack-based scheme to navigate hierarchical content. This meets success criterion 2 in user freedom in navigation as they can go back to the previous view if they want to change their entered details which in turn also covers success criterion 4 by allowing them to change inputted data at any time with no restrictions.

The CreateProfileCAVC with connected outlets and actions:



In this view the user decides what type of user they are signing up to be. After clicking on playGigsButton or hireMusiciansButton I wrote a function, so the opposite button is disabled so that it is obvious which choice has been made. After this choice, a userGigs variable is updated with a Boolean value where a musician is equal to true. The usernameField and userBioTextView are implemented in a similar way to the email and password text fields on the login page. My main task of this view is to access the camera and photo library for the profile UIImage which gets previewed to the user in the profileImageView.

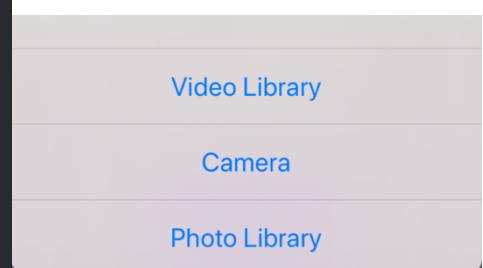
This of course means giving the user an option to take or choose a profile photo. Therefore, I used a UIAlertController again to display these choices. However, it is given .actionSheet property for the preferredStyle parameter in order to list these options. This action sheet will need to be called in other view controllers, therefore I have defined this function again in the extension file. Here, I have created access to the camera and library (success criterion 8) in an understandable and simple way which completes success criterion 1.

```

69 func openPhotoPopup(imagePickerController: UIImagePickerController, title: String, message: String){
70     //The UIAlertController
71     let photoPopup = UIAlertController(title: title, message: message, preferredStyle: .actionSheet)
72     //First Choice - Camera
73     let cameraAction = UIAlertAction(title: "Camera", style: .default) { (buttonTapped) in
74
75         do {
76
77     }
78     //Second Choice - Photo Library
79     let photoAction = UIAlertAction(title: "Photo Library", style: .default) {
80         (buttonTapped) in
81
82         do {
83
84     }
85
86     }
87     //Third Choice - Video Library
88     let videoAction = UIAlertAction(title: "Video Library", style: .default) { (buttonTapped) in
89
90         do {
91
92     }
93
94     }
95     photoPopup.addAction(videoAction)
96     photoPopup.addAction(cameraAction)
97     photoPopup.addAction(photoAction)
98     present(photoPopup, animated: true, completion: nil)
99 }

```

When called, it presents this action sheet:



The problem is that in this specific case, we do not want to provide the option to access the video library for the profile photo. Therefore, I edited the function so that the display of “Video Library” can be controlled with a video parameter:

To open the camera and photo library, I instantiated an object from UIImagePickerController called imagePicker which is passed as a parameter when choosing a source on the action sheet. Depending on the source provided, the camera or photo album view is presented, defined in the openImagePicker() function.

```

69  func openPhotoPopup(video: Bool, imagePicker: UIImagePickerController, title: String, message: String){
70      //The UIAlertController
71      let photoPopup = UIAlertController(title: title, message: message, preferredStyle: .actionSheet)
72      //First Choice - Camera
73      let cameraAction = UIAlertAction(title: "Camera", style: .default) { (buttonTapped) in
74
75          do {
76
77      }
78      //Second Choice - Photo Library
79      let photoAction = UIAlertAction(title: "Photo Library", style: .default) {
80          (buttonTapped) in
81
82          do {
83
84      }
85
86  }
87
88 //So choosing videos is only sometimes an option
89 if video {
90
91     //Third Choice - Video Library
92     let videoAction = UIAlertAction(title: "Video Library", style: .default) { (buttonTapped) in
93
94         do {
95
96         }
97
98         photoPopup.addAction(videoAction)
99     }
100
101 photoPopup.addAction(cameraAction)
102 photoPopup.addAction(photoAction)
103 present(photoPopup, animated: true, completion: nil)
104 }
```

```

122 func openImagePicker(imagePicker: UIImagePickerController, source: UIImagePickerController.SourceType) {
123     //imagepicker is the user Photo Library/Camera/Video Library
124     imagePicker.sourceType = source
125
126     if source == .savedPhotosAlbum {
127         //Limit library to videos
128         imagePicker.mediaTypes = ["public.movie"]
129     } else {
130         //Limit library to photos
131         imagePicker.mediaTypes = ["public.image"]
132     }
133     //Present View Controller
134     present(imagePicker, animated: true, completion: nil)
135 }
```

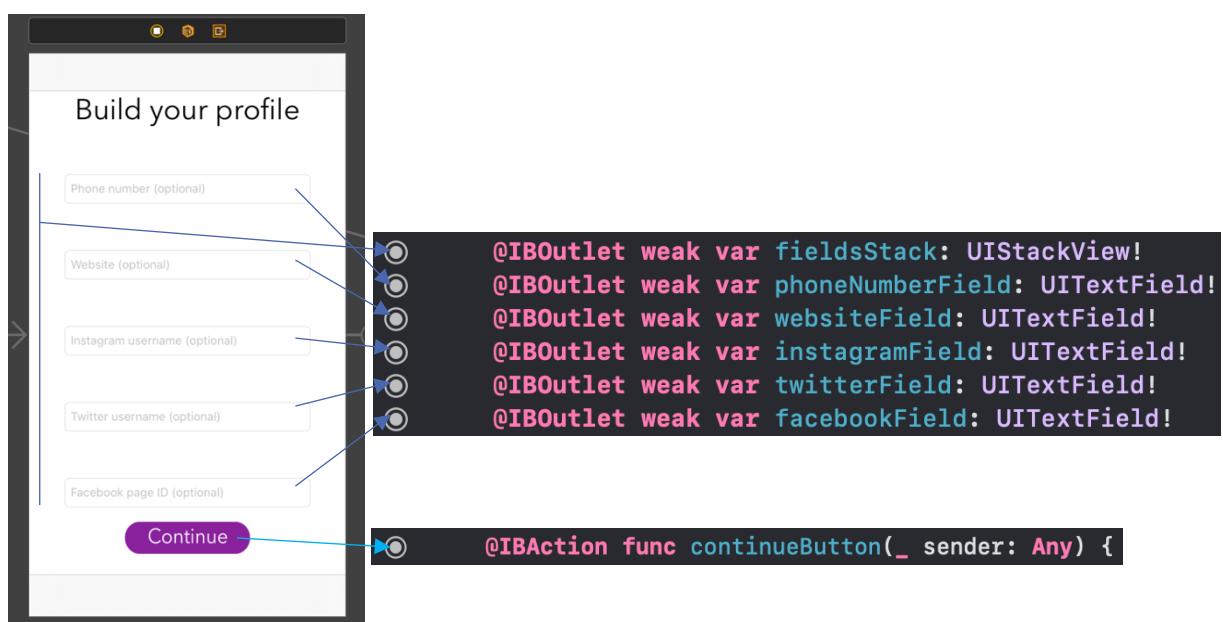
```

83
84     //First Choice - Camera
85     let cameraAction = UIAlertAction(title: "Camera", style: .default) { (buttonTapped) in
86         do {
87             //open the camera
88             self.openImagePicker(imagePicker: imagePicker, source: .camera)
89         }
90     }
91 }
```

When the user takes or picks an image, it is returned as a UIImage which can be displayed in the profileImageView. Using NSUUID().uuidString the image is given an ID which is always unique which will allow us to store it in Firebase Storage without error.

Progressing to the next account creation view meant I stored all inputted data about the user into the dictionary and copied the value of the dictionary over to the SocialLinksCAVC when

the segue is performed. I originally decided to upload the profile image to Firebase Storage before performing the segue, however if the user navigates back and decides to change their profile picture, there is an unwanted image in Firebase Storage hence meeting efficient database usage (criterion 6) and allowing users to change their data if they want (criterion 4). Therefore, to meet the criteria and store data efficiently, this will be uploaded when the user has finished creating their account (similar to subscribing their email to Authentication). The SocialLinksCAVC with connected outlets and actions:



I will revisit the main purpose of this view in milestone 11. For an event organiser, progressing from this view will do three things: subscribe the user to Firebase Authentication, upload the account data dictionary to Firebase Database and upload the profile photo to Firebase Storage.

To complete these tasks, I needed to add the Firebase SDKs to the app. I used CocoaPods for the installation of the libraries by adding the pods of the products into the project's podfile:

**pod 'Firebase/Core'**  
**pod 'Firebase/Auth'**  
**pod 'Firebase/Database'**  
**pod 'Firebase/Storage'**

Running **\$ pod install** in the terminal adds the SDKs to the project.

```

9 import Foundation
10 import FirebaseAuth
11 import FirebaseDatabase
12 import FirebaseStorage
13
14 //location references for Database and Storage
15 let DB_BASE = Database.database().reference()
16 let ST_BASE = Storage.storage().reference()
17
18 //handles for observing Database changes
19 var postsHandle: DatabaseHandle?
20 var activityHandle: DatabaseHandle?
21 var eventsHandle: DatabaseHandle?
22
23 class DataService {
24
25     //using a singleton
26     static let instance = DataService()
27
28     //MARK: DATABASE
29
30     //private for safety
31     //specific location references
32     private var _REF_BASE = DB_BASE
33     private var _REF_USERS = DB_BASE.child("users")
34     private var _REF_EVENTS = DB_BASE.child("events")
35
36     //closures to get references and
37     //ensure we dont change values
38     var REF_BASE: DatabaseReference {
39
40         return _REF_BASE
41     }
42
43     var REF_USERS: DatabaseReference {
44
45         return _REF_USERS
46     }

```

I defined the methods to complete these tasks in singleton classes of DataService and AuthService to interface with the Firebase services. The DataService class imports the SDKs just installed. The global constants of DB\_BASE holds the reference for the root of the Firebase Database. This is the base URL of the server:



#### Realtime Database

<https://gogig-db55e.firebaseio.com>

The singleton design pattern initialises the class within itself a single time with the static property. This means the class instance can be shared globally which is suitable for commonly used methods to communicate with Firebase.

Closures are used to create public variables from the private variables in order to set values and access information.

Within the DataService class I defined a method to update the profile data in the database under the child of

“users”. The dictionary of collected account data will be passed as a parameter into this function. A completion handler returns when the upload is complete.

```

62     func updateDBUserProfile(uid: String, userData: Dictionary<String, Any>,
63                             handler: @escaping (_ completion: Bool) -> ()) {
64         REF_USERS.child(uid).child("profile").updateChildValues(userData) {
65             (error:Error?, ref:DatabaseReference) in
66             if error != nil {
67                 handler(false) //handle error if couldn't update db
68             } else {
69                 handler(true) //add the user data under 'profile'
70             }
71         }

```

To store the profile image in Storage, I added these variables and the following method to the DataService class:

```

524     private var _REF_ST = ST_BASE
525
526     var REF_ST: StorageReference {
527
528         return _REF_ST
529     }
530
531     func updateSTPic(uid: String, directory: String, imageContent: UIImage, imageID: String,
532         uploadComplete: @escaping (_ status: Bool, _ error: Error?) -> ()) {
533
534         //Converting the imageData to JPEG to be stored
535         if let imageData = imageContent.jpegData(compressionQuality: 0.1) {
536
537             //Uploading the image with unique string ID
538             REF_ST.child(uid).child(directory).child(imageID).putData(imageData, metadata: nil, completion:
539                 { (metadata, error) in
540                     if error != nil {
541
542                         uploadComplete(false, error) //handle error if couldn't update storage
543                         return
544                     }
545                     uploadComplete(true, nil) //success
546                 }
547             )
548         }
549     }

```

The AuthService class is setup in a similar way with the following method to register a user with Authentication:

```

17     //subscribe the user to Firebase Authentication
18     func registerUser(withEmail email: String, andPassword password: String, userCreationComplete:
19         @escaping (_ status: Bool, _ error: Error?) -> ()) {
20
21         //adds user to Auth list
22         Auth.auth().createUser(withEmail: email, password: password) { (authResult, error) in
23             //if there is no result of creating user
24             guard let user = authResult?.user else {
25                 //complete as error
26                 userCreationComplete(false, error)
27
28             }
29
30             //adds user to the database
31             let userData = ["provider": user.providerID, "email": user.email] //provider =
32                 Firebase/Facebook/Google/Email
33             //add this userData to Database
34             DataService.instance.createDBUser(uid: user.uid, userData: userData as Dictionary<String, Any>)
35             //complete as no error
36             userCreationComplete(true, nil)
37         }

```

And to log in a user with Authentication:

```

38     //Log the user in with Firebase Authentication
39     func loginUser(withEmail email: String, andPassword password: String, loginComplete: @escaping (_
40         status: Bool, _ error: Error?) -> ()) {
41         //sign them in
42         Auth.auth().signIn(withEmail: email, password: password) { (authResult, error) in
43             //if there is an error signing in
44             if error != nil {
45                 //complete with error
46                 loginComplete(false, error)
47                 return
48             }
49             //complete with no error
50             loginComplete(true, nil)
51         }

```

In both cases the completion handler will return an error if Firebase has an issue signing the user up or logging them in. Firebase

automatically returns the kind of error which is useful to me as a programmer and to the user in reporting back error meeting success criterion 2. I had difficulty using these closures because they are completed asynchronously so the main thread does not wait for it to complete. I had to learn to define empty variables global of that scope, and then assign its value within the closure. This is my code to sign up a user to Authentication and an example of how to solve the issue:

```

143 //Sign the user up
144 AuthService.instance.registerUser(withEmail: email!, andPassword: password!,
145     userCreationComplete: { (success, error) in
146     if error != nil {
147         Has to log in user from
148         within the closure
149         because otherwise it gets
150         called before the user
151         has completed
152         registering. Firebase
153         returns an error because
154         it attempts to log in a
155         user that doesn't exist!
156     } else {
157         //Display a UIAlertController if something goes wrong
158         self.displayError(title: "There was an Error", message:
159             error!.localizedDescription)
160     }
161     //Successfully registered and added to database
162     //Now log user in
163     AuthService.instance.loginUser(withEmail: self.email!, andPassword:
164         self.password!, loginComplete: { (success, nil) in
165             self.updateUserData()
166         }
167     )
168 }

```

The signup process is complete if the user has chosen to be an event organiser at this point. After subscribing to Authentication, the database is updated with the user data from the dictionary:

```
["bio": "Biography of the Test User", "instagram": "", "website": "", "appleMusic": "", "spotify": "", "phone": "", "gigs": false, "email": "testuser@gogig.com", "name": "Test User", "picURL": "https://firebasestorage.googleapis.com/v0/b/gogig-db55e.appspot.com/o/3FAjK8aul5YZuTQtJd0wN3d0v0P2%2FprofilePic%2F8C82BE42-DCBA-4EFD-AB23-4E6023C87F0D.jpg?alt=media&token=2ad3cd94-4351-47cd-a06e-5df9150329fd", "twitter": "", "facebook": ""]
```



I originally stored the image ID in the database and planned to use a method to query Storage by navigating the directory for that image file. However, this was a lot more code than necessary because if I fetch and store the download URL of the profile photo as soon as it is uploaded instead, getting the image back later is as simple as downloading the content of the URL.

First draft of uploading profile image and storing image ID in database:

```

66     //Put the profile pic in firebase storage
67     func picUpload(uid: String, handler: @escaping (_ url: URL) -> ()) {
68
69         if let userPic = profileImage {
70             //Upload the picture to Firebase
71             DataService.instance.updateSTPic(uid: uid, directory: "profilePic", imageContent: userPic,
72                 imageID: imageID, uploadComplete: { (success, error) in
73                     if error != nil {
74                         self.displayError(title: "There was an Error", message: error!.localizedDescription)
75                     }
76                     //Image uploaded
77                     //Store imageID in dictionary to put in database
78                     self.userData!["imageID"] = self.imageID
79                 })
80             }

```

imageID: "4JKhjse8tut23UIL6Vuv9LK.jpg"

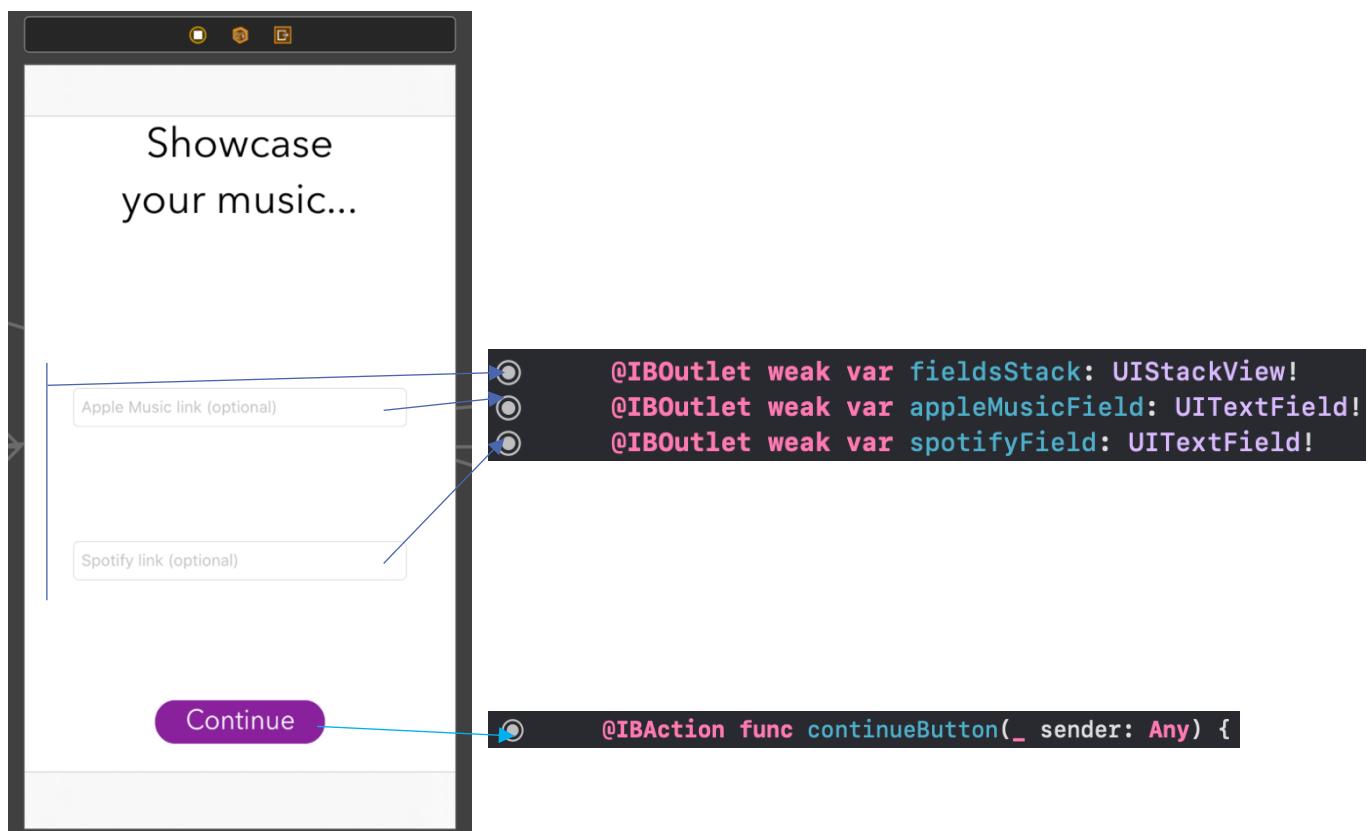
Reviewed draft of uploading profile image and storing download URL in database:

```

66    //Put the profile pic in firebase storage
67    func picUpload(uid: String, handler: @escaping (_ url: URL) -> ()) {
68
69        if let userPic = profileImage {
70            //Upload the picture to Firebase
71            DataService.instance.updateSTPic(uid: uid, directory: "profilePic", imageContent: userPic,
72                imageID: imageID, uploadComplete: { (success, error) in
73                    if error != nil {
74                        self.displayError(title: "There was an Error", message: error!.localizedDescription)
75
76                    } else {
77                        //Get the URL of the stored image, to store in the database
78                        DataService.instance.getSTURL(uid: uid, directory: "profilePic", imageID: self.imageID)
79                            { (returnedURL) in
80
81                                handler(returnedURL)
82                            }
83                    }
84                }
85            }
86        }
87    }
88
89    picURL: "https://firebasestorage.googleapis.com/v0/b/gog..."
```

This approach will be taken whenever I upload images or video files for future milestones.

I used selection to check if the user chose to be a musician previously. I made it so the musician's profile and user data are not uploaded at this point, but they have the opportunity to add music streaming profile links, to meet the requirements Jude Mills wanted for his band. Testing these links will be completed in milestone 11, but this is the interface of the MusicLinksCAVC view controller:

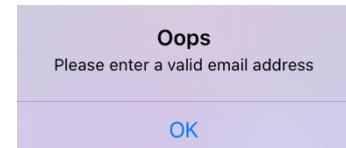


This view is very similar to the SocialLinksCAVC and will upload the musicians profile picture and data to Storage and Database. After both users have finished creating their

account, I performed a segue to take them to their portfolio view. In addition to this, the same segue is performed when the user logs in from the LoginSignupVC.

## Validation Routines

These validation rules are checked when the user progresses from a view after entering information. To meet my success criteria of avoiding error, and easy recovery, whenever input did not meet the required rules, I displayed a helpful error with UIAlertController. I will cover the validation routines of the social media and music streaming text fields in milestone 11.



IBoutlet	Rules	Recovery
emailField	<ul style="list-style-type: none"> <li>User has entered an email, not equal to empty string</li> <li>String enters contains "@" and "."</li> <li>String length is more than 3</li> <li>Character limit is 62</li> <li>Firebase Authentication returns error if email is not valid, or email has not been subscribed when logging in</li> </ul>	<u>Display Error</u> Title: "Oops" Message: "Please enter a valid email address"  <u>Firebase Error</u> Title: "Couldn't Log In" Message: "The email is not associated with an account", "The email is already associated with an account", "The email is not valid"
passwordField	<ul style="list-style-type: none"> <li>User has entered a password, not equal to empty string</li> <li>Password length is more than 6</li> <li>Character limit is 30</li> </ul>	<u>Display Error</u> Title: "Oops" Message: "Please enter a password"  Title: "Password Length" Message: "Choose a good strong, password more than 6 characters."
confirmPasswordField	<ul style="list-style-type: none"> <li>User has entered a password confirmation, not equal to empty string</li> <li>Password confirmation is equal to the password</li> </ul>	<u>Display Error</u> Title: "Passwords" Message: "The password confirmation does not match"
playsGigsButton & hireMusiciansButton	<ul style="list-style-type: none"> <li>User has selected a button and chosen type</li> </ul>	<u>Display Error</u> Title: "Oops" Message: "Please provide all necessary information"

usernameField	<ul style="list-style-type: none"> <li>Username has entered an email, not equal to empty string</li> <li>Username length is more than or equal to 2</li> <li>Character limit of 50</li> </ul>	<u>Display Error</u> Title: "Oops" Message: "Please enter your name"
userBioTextView	<ul style="list-style-type: none"> <li>No validation rules, optional input</li> </ul>	N/A
profileImageView	<ul style="list-style-type: none"> <li>User has added a profile image</li> </ul>	<u>Display Error</u> Title: "Oops" Message: "Please provide all necessary information"

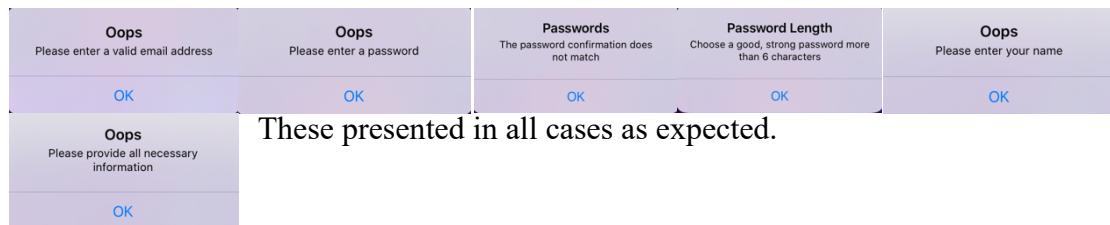
## Test Plan

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Testing valid inputs when creating an account.	3 – Validation rules	["bio": "Biography of the Test User", "instagram": "", "website": "", "appleMusic": "", "spotify": "", "phone": "", "gigs": false, "email": "testuser1@gogig.com", "name": "Test User", "picURL": String_URL, "twitter": "", "facebook": ""]	When adding emails, passwords and profile information, the program should display an error if the input is unexpected, does not meet requirements or if nothing was entered.	Valid and Invalid test data
2	Check account creation.	5 – Authentication system 6 – Efficient storage in Database	Button	Checking account was created with Firebase Auth and entered profile data was added to Firebase Database.	Valid
3	Make sure user can swap between options to sign up and log in	1 – Understandable user interface	Button	If user already has an account, the mode needs to be swapped when requesting to log in. User will resume their account rather than create a new one as expected.	Valid

4	Ensure user can pick a profile picture from their camera or library	8 – Access camera and media library	Camera or Device Photo Library App	After taking or choosing an image, the returned image should be displayed to the user and stored in Firebase Storage when user finished account creation at Sign up.	Valid
5	Check user remains logged in when leaving or closing the app	5 – Remain logged in	Firebase Authentication	When the app is opened, login page is displayed if the user is using the app without a current associated account.	Valid

## Test 1 – Valid inputs

Using nested ‘if’ statements and the validation routines stated above, I checked each of the user inputs. If a requirement was not met, using ‘else’ I displayed an error with a message.



On the other hand, I wanted to make the user biography text optional, which meant different logic before progressing. Before, if the user did not enter a bio, “Please provide all necessary information” would be displayed like every other essential piece of data.

```

166 //Check Bio, it is optional
167 if let userBio = userBioTextView.text {
168     var userBio = userBio
169     //So if the text is the default
170     if userBio == "Write a bio... |" {
171         //then just store in database as an empty string
172         userBio = ""
173     }

```

As well as this optional feature, I had the issue that the UITextField class does not have a placeholder property. This meant that the user would start

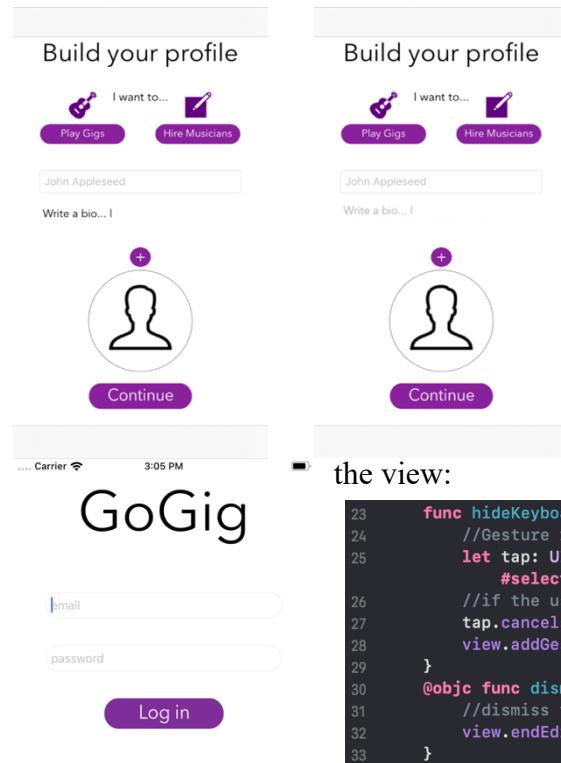
editing after “Write a bio...|”. To get around this issue, I had to create a custom class which changed the colour of pre-set text to look like a placeholder, and then cleared the field when the user began to edit. I therefore replaced the subclass of userBioTextField from

UITextField to MyTextField

```

34 //To allow access to the editing delegate methods
35 class TextViewDelegate: NSObject, UITextViewDelegate {
36
37     var placeholder = ""
38
39     //Clear the placeholder
40     func textViewDidBeginEditing(_ textView: UITextView) {
41         if textView.textColor == UIColor.lightGray {
42             textView.text = nil
43             textView.textColor = UIColor.black
44         }
45     }
46     //If empty add the placeholder
47     func textViewDidEndEditing(_ textView: UITextView) {
48         if textView.text.isEmpty {
49             textView.text = placeholder
50             textView.textColor = UIColor.lightGray
51         }
52     }
53 }

```



## Test 2 – Swap Log in to Sign up

On iOS, when tapping on a text field the device keyboard is presented. The keyboard remains in the view unless a method is written to dismiss it. The view is therefore not free to navigate. This was causing issues when inputting an email, and then deciding to change state of the view as the 'Sign Up' button is now inaccessible:

Therefore, I needed this function to dismiss the keyboard when the user taps anywhere else in

the view:

```

23     func hideKeyboard() {
24         //Gesture recogniser so we call method when the screen is tapped
25         let tap: UITapGestureRecognizer = UITapGestureRecognizer(target: self, action:
26             #selector(UIViewController.dismissKeyboard))
27         //if the user taps inside the keyboard, ignore
28         tap.cancelsTouchesInView = false
29         view.addGestureRecognizer(tap)
30     }
31     @objc func dismissKeyboard() {
32         //dismiss the keyboard (end editing)
33         view.endEditing(true)
34     }

```

The function was again defined in the Extensions file because it will be needed in any view controller that requires the keyboard. The fix ensures user freedom and easy navigation.



In

addition to the keyboard, upon switching states of the view, confirmPasswordField did not show, and the images for the topLSButton and bottomLSButton did not switch. When in signup mode, the user did not know they had to confirm their password creation and may believe they are logging in when in actual fact they are signing up. I therefore did not format the buttons properly after switching mode with the bottom button. These fixes took the right progression path depending on the mode as expected and is now clear on how the user needs to interact with the view.

```

74     //Show everything for Sign up
75 } else {
76
77     //Reset input when changing mode
78     emailField.text = ""
79     passwordField.text = ""
80     passwordField.placeholder = "create password"
81     confirmPasswordField.text = ""
82
83     //show the confirm password field
84     confirmPasswordField.isHidden = false
85
86     //swap the images for the buttons
87     topLSButton.setImage(UIImage(named: "signupButton"), for: .normal)
88     bottomLSButton.setImage(UIImage(named: "loginButton"), for: .normal)
89
90     switchPrompt.text = "Already have an account?"
91
92     //return logInMode as false
93     return false
94 }

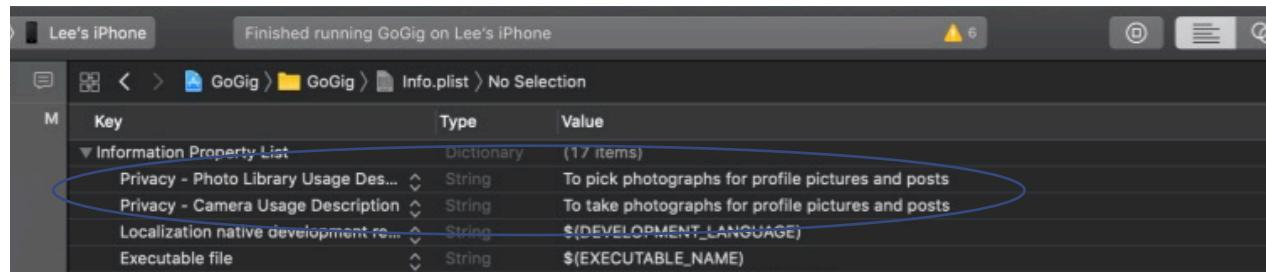
```

## Test 3 – Profile Picture from Camera or Library

Initially a fatal exception error was thrown because as the app developer, I needed to ask the user for permission to access the camera and photo library. To solve the issue, and hence ask for permission, I needed to provide a usage explanation message in the info.plist file for the key NSCameraUsageDescription and NSPhotoLibraryUsageDescription. The info.plist is a

XML structured file containing keys and values which contain information about the app and how it is configured.

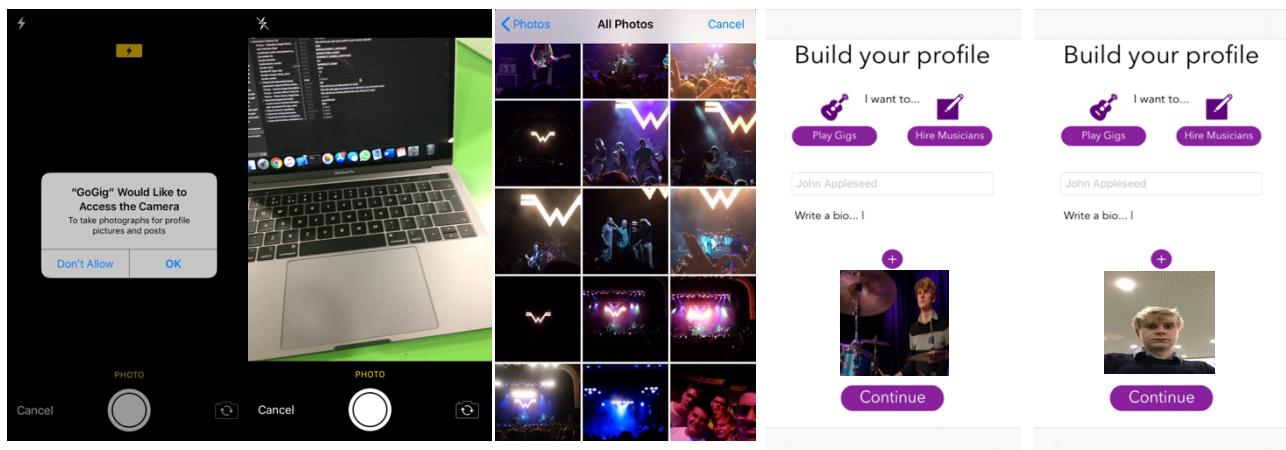
Therefore, after adding these to the file:



A screenshot of the Xcode interface showing the Info.plist file. The table lists several keys under the 'Information Property List' section, including 'Privacy - Photo Library Usage Description', 'Privacy - Camera Usage Description', and 'Localization native development region'. The 'Privacy - Photo Library Usage Description' and 'Privacy - Camera Usage Description' rows are circled in blue.

M	Key	Type	Value
<b>▼ Information Property List</b>			
	Privacy - Photo Library Usage Description	String	To pick photographs for profile pictures and posts
	Privacy - Camera Usage Description	String	To take photographs for profile pictures and posts
	Localization native development region	String	\$(DEVELOPMENT_LANGUAGE)
	Executable file	String	\$(EXECUTABLE_NAME)

I could access what I wanted after the alerts were shown and an image was returned from both sources.



The dimensions and the frame of the returned image will be fixed later for finishing touches. This meets the success criteria of personalising the experience with the users own photographs (point 8). Requesting permission also meets the requirement of users choosing how much information they share.

## Test 4 & 5 – Account Creation and Remain Logged In

I had no problems signing the user up to Firebase after they had created their account as seen my developer log. As part of test 5, however, I needed to make sure that the user can log in successfully with this created account and remain logged in when returning to the home screen. To do this, I set up a temporary view with a button which signs the user out.



I figured out for the user to remain logged in, this view would have to be the initial view when the app finishes launching. Then I instantiate and present the LoginSignupVC if the user is not logged in. In the AppDelegate.swift file, in the didFinishLaunchingWithOptions method, I query Authentication to see if there is a current user and present the login view if not. This did not present the login view when the user pressed 'Sign Out' though. To do this I had to present the view again after the user signed out of Authentication.

```

17 @UIApplicationMain
18 class AppDelegate: UIResponder, UIApplicationDelegate {
19
20     var window: UIWindow?
21
22
23     func application(_ application: UIApplication,
24                      didFinishLaunchingWithOptions launchOptions:
25                      [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
26         //Configure app with Firebase
27         FirebaseApp.configure()
28
29         //If we are logged out, then present the LoginVC
30         if Auth.auth().currentUser == nil {
31             print("No current user")
32             let storyboard = UIStoryboard(name: "Main", bundle: Bundle.main)
33             let loginVC =
34                 storyboard.instantiateViewController(withIdentifier:
35                 "LoginSignupVC")
36             window?.makeKeyAndVisible()
37             window?.rootViewController?.present(loginVC, animated: true)
38         }
39
40         return true
41     }
42
43 }

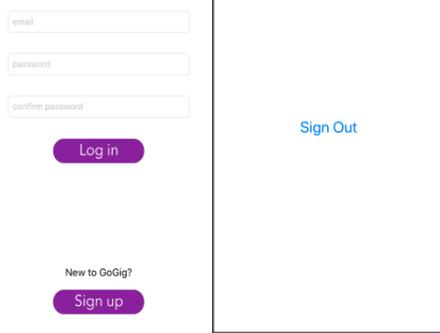
```

```

try Auth.auth().signOut()
let loginVC = self.storyboard?.instantiateViewController(withIdentifier:
    "LoginSignupVC") as? LoginSignupVC
self.present(loginVC!, animated: true, completion: nil)

```

## GoGig



The app did the following things in testing:

- Launched the app to the initial view if there was a current user.
- Presented the LoginSignupVC after launching the app if there is no current user.
- Presented the LoginSignupVC when the user taps 'Sign Out'.
- Performed segue to the main view after logging in and signing up.
- Resumed both views when closing the app and reopening again.

All these results were as expected and so meets success criterion 5 in the user remaining logged in unless requested otherwise. At this point, I have followed my specification in creating a successful Authentication System but will have to see if this is still the case as the app unfolds when data and logic is implemented considering accounts.

## Milestone Review

This milestone went fairly well and was not too hard to implement the features to meet the success criteria. The main bugs I encountered was the camera permission crash and the switch login mode keyboard restriction.

A few times I encountered a fatal error when the app unexpectedly found nil value. This usually came from not setting the value of the userData dictionary properly when performing the segues at account creation. This was fixed by adding a single line in the prepareForSegue method which changes the values of variables in other view controllers just before a segue is performed.

Apart from this, the validation checks went really well and will hopefully avoid error as I develop the app further. I met the success criteria by checking the inputs against rules, storing data in the database efficiently, accessing the camera, reporting back error and giving the user freedom in their data and navigation.

My end users checked my work so far:

Gavin Thorrold: “The look does not quite fit the design, but I know you will fix this later on. The login page works well because you have followed common practices of showing a popup when I have not entered something correctly. I believe this will work well as our team at CEG can be logged in under one account to receive notifications together.”

Jude Mills: “You have provided a way to build a profile which is easy to use. With this, and adding social media later, information under my account should hopefully promote my band to help me get more gigs.”

At the end of this milestone, my end users agreed I met objectives 2, 3, 5, 8, 9 from the success criteria so far.

## Milestone 2 – Portfolio and Post Views

### Development Log

For this milestone, I focussed on the central tab of the app where both users can prepare their profile and portfolio to meet the specification of creating a personalised space for other users to view. As part of my design, to access this view controller, the user taps on an icon in a tab bar at the bottom of the view. Therefore, I needed to embed the view controller in a UITabBarController which manages the tab bar for me. The portfolio view (UserAccountVC class) is a subclass of UITableViewController which allowed me to display the account data and portfolio post data in a scrolling table view feed, similar to social medias like Facebook and Instagram.

I created two classes for the cells of the UITableView, AccountHeaderCell and AccountPostCell of subclass UITableViewCell. These classes get instantiated to reuse identifiers for multiple rows of the table view. This is for performance reasons.

UserAccountVC outlets and actions:

The image shows the Xcode interface with the following components:

- Storyboard:** Shows a Tab Bar Controller with a single tab selected. The tab's content is a UITableViewController with a prototype cell. The prototype cell displays a user profile with a placeholder image, a 'Looking to' text area, and contact information (email and phone). It also includes a 'Location' section and a 'More' button.
- UserAccountVC Code:**

```

21 class UserAccountVC: UITableViewController {
22
23     @IBOutlet weak var settingsBarButton: UIBarButtonItem!
24     @IBOutlet weak var addPortfolioBarButton: UIBarButtonItem!
25
26     @IBAction func settingsButton(_ sender: Any) {
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
        }
    
```
- AccountHeaderCell Code:**

```

11 class AccountHeaderCell: UITableViewCell {
12
13     @IBOutlet weak var userTypeLabel: UILabel!
14     @IBOutlet weak var userBioTextView: MyTextView!
15     @IBOutlet weak var profilePicView: UIImageView!
16     @IBOutlet weak var userEmailLabel: UILabel!
17     @IBOutlet weak var userPhoneLabel: UILabel!
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
        }
    
```
- AccountPostCell Code:**

```

11 class AccountPostCell: UITableViewCell {
12
13     @IBOutlet weak var postLocationLabel: UILabel!
14     @IBOutlet weak var postCaptionTextView: MyTextView!
15     @IBOutlet weak var postImageView: UIImageView!
16     @IBOutlet weak var postMoreButton: UIButton!
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
        }
    
```
- User Model Class:**

```

13 class User {
14
15     var uid: String
16     var name: String
17     var email: String
18     var phone: String
19     var bio: String
20     var gigs: Bool
21     var picURL: URL
22     private var facebook: String
23     private var twitter: String
24     private var instagram: String
25     private var website: String
26     private var appleMusic: String
27     private var spotify: String
28
29     //instantiate User object
30     init(uid: String, name: String, email: String, phone: String, bio: String, gigs: Bool, picURL: URL,
31           facebook: String, twitter: String, instagram: String, website: String, appleMusic: String, spotify:
32           String) {
33         self.uid = uid
34         self.name = name
35         self.email = email
36         self.phone = phone
37         self.bio = bio
38         self.gigs = gigs
39         self.picURL = picURL
40         self.facebook = facebook
41         self.twitter = twitter
42         self.instagram = instagram
43         self.website = website
44         self.appleMusic = appleMusic
45         self.spotify = spotify
46     }
47 }
    
```

To begin the development of this milestone, I focussed on grabbing the user profile data from the database to display in the AccountHeaderCell. With the MVC approach, I created a User model class to store all data about the user with an object.

I then query the database with the path of “profile” using a single observer to grab it one

time only when called. Using nested “if let” statements a User object is not instantiated if one of the properties are not found which avoids error and crashes completing success criterion 3.

```

73 func getDBUserProfile(uid: String, handler: @escaping (_ user: User) -> ()) {
74
75     //Grab user profile data from the database...
76     REF_USERS.child(uid).child("profile").observeSingleEvent(of: .value, with: { (profileSnapshot) in
77
78         //... and cast as a NSDictionary
79         let profileData = profileSnapshot.value as? NSDictionary
80         //Get every value from every key of the dictionary
81         if let currentUserName = profileData?["name"] as? String {
82             if let currentUserEmail = profileData?["email"] as? String {
83                 if let currentUserGigs = profileData?["gigs"] as? Bool {
84                     if let currentUserBio = profileData?["bio"] as? String {
85                         if let currentUserPicURLStr = profileData?["picURL"] as? String {
86
87                             if let currentUserPhone = profileData?["phone"] as? String {
88                                 if let currentUserFacebook = profileData?["facebook"] as? String {
89                                     if let currentUserTwitter = profileData?["twitter"] as? String {
90                                         if let currentUserInstagram = profileData?["instagram"] as? String {
91                                             if let currentUserWebsiteURLStr = profileData?["website"] as? String {
92                                                 if let currentUserAppleMusicURLStr = profileData?["appleMusic"] as? String {
93                                                     if let currentUserSpotifyURLStr = profileData?["spotify"] as? String {
94
95
96                                         let currentUserPicURL = URL(string: currentUserPicURLStr)
97
98                                         //instansiate a new user object from the data grabbed
99                                         let currentUser = User(uid: uid, name: currentUserName, email: currentUserEmail, phone: currentUserPhone, bio:
100                                         currentUserBio, gigs: currentUserGigs, picURL: currentUserPicURL!, facebook: currentUserFacebook, twitter:
101                                         currentUserTwitter, instagram: currentUserInstagram, website: currentUserWebsiteURLStr, appleMusic:
102                                         currentUserAppleMusicURLStr, spotify: currentUserSpotifyURLStr)
103
104
105                                         //return the user
106                                         handler(currentUser)
107                                     }
108                                 }
109                             }
110                         }
111                     }
112                 }
113             }
114         }
115     }
116 }

```

With a user instance, I outputted the data to the AccountHeaderCell outlets by accessing the object’s properties. One property is the download URL of the profile photo. Here is the method to download the content and return a UIImage.

```

131     func downloadImage(url: URL, handler: @escaping (_ returnedImage: UIImage) -> ()) {
132
133         //Get's the data of the URL
134         let task = URLSession.shared.dataTask(with: url) { data, response, error in
135             if let error = error{
136                 //print the error if one
137                 print(error.localizedDescription)
138             } else {
139
140                 //So picture if first when UI loads
141                 DispatchQueue.main.async() {
142
143                     //Converts the image data to a UIImage
144                     if let downloadedImage = UIImage(data: data!) {
145
146                         handler(downloadedImage)
147                     }
148                 }
149             }
150         }
151         //Resumes the task after image is set
152         task.resume()
153
154     }

```

DispatchQueue.main.async() performs the task asynchronously (performs on a difference FIFO queue) and so doesn’t block the app’s main thread improving the performance when creating a profile image out of the data downloaded for a good user experience (point 1).

I could then display the profile data in the header cell with a refresh function.

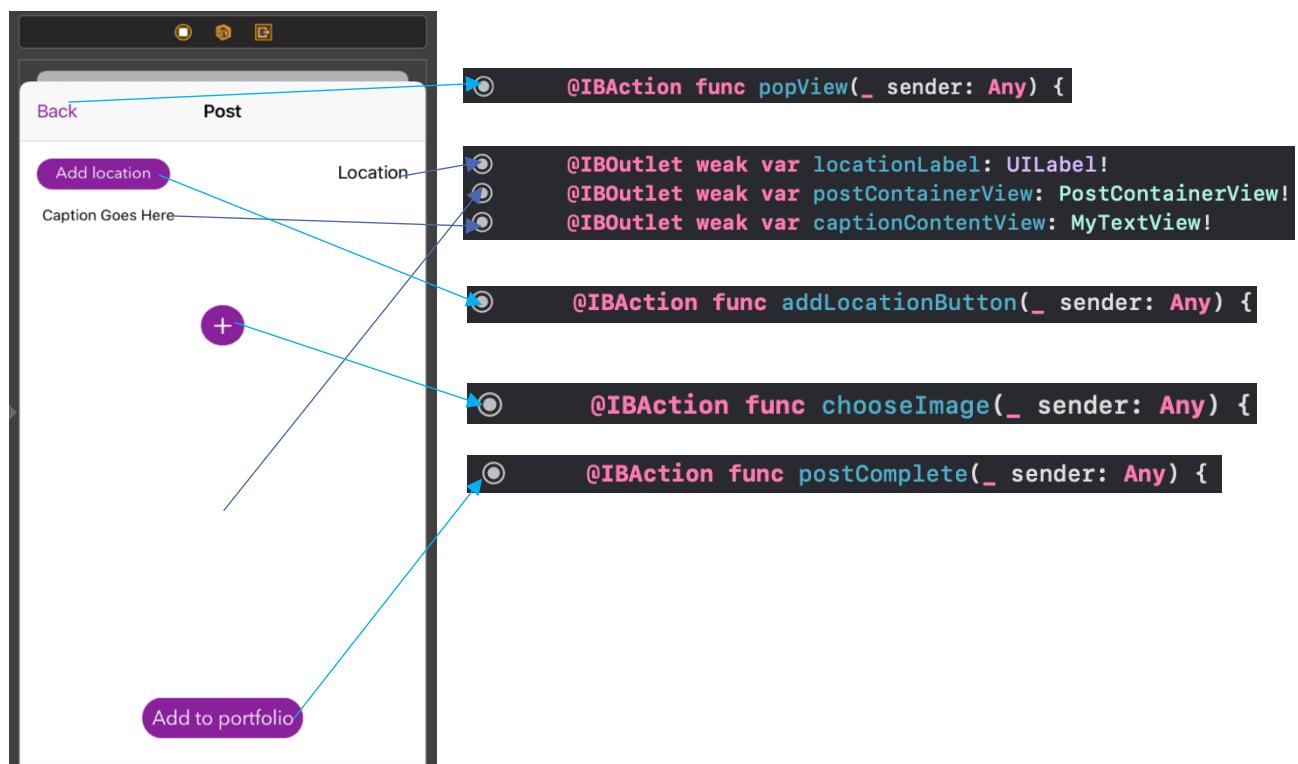
```
76     DataService.instance.getDBUserProfile(uid: uid!) { (returnedUser) in
77         self.user = returnedUser//get profile data and set user who owns portfolio
78         self.downloadImage(url: returnedUser.picURL) { (returnedProfileImage) in
79             self.profilePic = returnedProfileImage//set the profile picture
80             self.tableView.reloadData()//show all data in table view
81     }
```

When `tableView.reloadData()` is called, the header cell is updated when the `cellForRowAt` delegate table view method is called.

```
28     //We access the row using indexPath.section instead of indexPath.row
29     override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
30
31         //FIRST CELL IS USER PROFILE
32         if indexPath.section == 0 {
33             let headerCell = tableView.dequeueReusableCell(withIdentifier: "AccountHeaderCell", for: indexPath) as!
34             AccountHeaderCell//instantiate a reusable header cell
35
36             updateUserData(cell: headerCell) //update the cell with the user data
37
38             return headerCell
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
```

```
157     var profilePic = UIImage(named: "icons8-user") //Have a placeholder image
158     func updateUserData(cell: AccountHeaderCell){
159         //Set the navigation bar title
160         self.navigationController?.navigationBar.topItem?.title = user?.name
161
162         //Set the account header cell outlets
163         cell.userBioTextView.text = user?.bio
164         if user?.gigs == true {
165             cell.userTypeLabel.text = "Looking to play"
166         } else {
167             cell.userTypeLabel.text = "Hiring entertainment"
168         }
169
170         cell.profilePicView.image = profilePic
171         cell.userEmailLabel.text = user?.email
172         cell.userPhoneLabel.text = user?.phone
```

To show posts in the other cells of the table view, I needed to create the PortfolioPostVC for the user to add photos and videos to their portfolio.



When the user taps chooseImage method, the action sheet gives them the option to choose from the video library this time, as well as the camera and photo library sources. A video is better for demonstrating musical skill which is what Jude wanted detailed in my specification. However, I cannot return a video and display it in a UIImageView, therefore I created a subclass PostContainerView of UIView which adds an AVPlayer to the view if a video is returned and adds a UIImageView if the user takes or chooses a picture. The subclass is used to preview to the user before the post (so the user can review their own data – criterion 4) and is used to display the post itself in the portfolio table view. I then needed to add the feature to pin a location name to the post. I had difficulty with this because I would need a whole set of places to search through with a search bar. To save time, I used the Google Places API which presents its own view with a search bar and returns the location name selected. I installed the **‘GooglePlaces’** pod with cocoapods and provided the API key after registering to the GMSPlacesClient.

The Google view controller presented is the AutoComplete subclass of UIViewController, this therefore meant I could make the PortfolioPostVC a subclass of AutoComplete and present the view when the user tapped the addLocationButton.

The data prepared for posting was collected in the dictionary postData similar to the userData dictionary when creating an account and is uploaded using a similar method in the DataService class. When uploading a video to storage I used the following method:

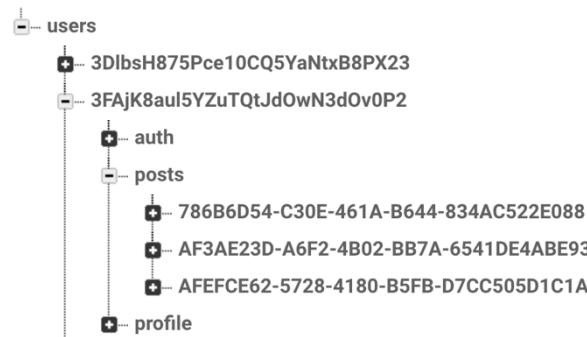
```
551 func updateSTVid(uid: String, directory: String, vidContent: URL, imageID: String, uploadComplete: @escaping (_
552   status: Bool, _ error: Error?) -> ())
553
554   //Put the video local URL in storage
555   REF_ST.child(uid).child(directory).child(imageID).putFile(from: vidContent, metadata: nil, completion: {
556     (metadata, error) in
557     if error != nil {
558
559       uploadComplete(false, error)
560       return
561     }
562     uploadComplete(true, nil)
563   })
564 }
```

When uploading a post, the database was updated with the post data from the dictionary under the unique key of postID.

```
[{"postURL": "https://firebasestorage.googleapis.com/v0/b/gogig-
db55e.appspot.com/o/3FAjK8auI5YzuTQtJd0wN3d0v0P2%2FportfolioPost%2FAF3AE23D-A6F2-4B02-BB7A-
6541DE4ABE93.jpg?alt=media&token=9ac43443-db2d-40e3-96f5-924a2a299bdc", "postID": "AF3AE23D-
A6F2-4B02-BB7A-6541DE4ABE93", "caption": "This is the post caption", "location": "Location",
"isImage": true}]
```

I included a UIActivityIndicator which shows a spinner and disables interaction with the screen while the content is being uploaded. This prevents the error of uploading the same post twice by clicking more than once. I have done this to report back to the user that content is being uploaded so they understand what the app is doing meeting success criteria 1 and 2.

Posts are then added to the database under the user’s uid.



These posts are fetched with a similar method to getting the user profile data. However, this time when a post object is instantiated, it is appended to an array to display in each cell of the portfolio table view. I created a post model for this purpose.

```
12 class PortfolioPost: Comparable {
13
14     var uid: String           //user that owns post
15     private var id: String    //unique identifier of post
16     var location: String     //name of posts location
17     var caption: String      //caption of post
18     var isImage: Bool         //decide if post is image/video
19     var postURL: URL         //image download URL
20
21     //instantiate object
22     init(uid: String, id: String, location: String, caption: String, isImage: Bool, postURL: URL) {
23         self.uid = uid
24         self.id = id
25         self.location = location
26         self.caption = caption
27         self.isImage = isImage
28         self.postURL = postURL
29     }
30 }
```

And updating the table view with the post data:

## Validation Routines

The only validation rule I needed to check against in this milestone was that the user had added everything correctly to make a post. Any errors would again be reported back with a UIAlertController

<u>IBOutlet</u>	<u>Rules</u>	<u>Recovery</u>
captionContentView	<ul style="list-style-type: none"> <li>Optional input</li> </ul>	N/A
postContainerView	<ul style="list-style-type: none"> <li>User must add a photo or video to post anything</li> </ul>	<u>Display Error</u> Title: “Oops” Message: “Please add a photo or video to post”

## Test Plan

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Check user profile data is displayed as expected.	4 – Data displayed when required	Firebase Database	All data entered at account sign up should be grabbed from the database and displayed at the top of the portfolio.	Valid
2	User should be able to log out of their account	2 – Easy navigation 5 – Login authentication	Button	When requesting to log out, the Login view is displayed to the user.	Valid
3	Check navigation to Post View	2 – Easy navigation	Navigation Button	The Post View is displayed to the user when they request it. They should be able to go back, and view should dismiss anyway when they complete a post.	Valid
4	Make sure Google Places API returns a location string when searched to attach to the post	7 – Optional location usage	Google Places API	The Google Places Autocomplete View should display and return a location name when searching with the search bar.	Valid
5	Access camera, image library and video library	8 – Personalisation of portfolio	Camera or Device Photo Library App	After taking or choosing an image, the returned image should be displayed to the user and stored in Firebase Storage when user completes their post.	Valid
6	Test valid inputs when creating a post	3 – Validation rules 6 – Data stored efficiently	[{"postURL": "String_URL", "postID": "String_Id", "caption": "This is the post caption", "location": "Location", "isImage": true/false}]	If user has not added an image or video when trying to post, app should display an error. When post is complete, the object should be added to the Database.	Valid and Invalid test data
7	Make sure the post is displayed in a scrolling	1 – Good user experience and user interface	Firebase Database	Portfolio posts (and their data) should be displayed in a scrolling table view feed. A new post should be added automatically without the	Valid

	table view feed.			user having to manually refresh the feed.	
8	Check user can delete a post from the feed and from the database	4 – User can edit their account data 6 – Data stored efficiently	Button	When clicking a button, the table is refreshed removing the selected post from the feed. In the backend, the post object is deleted from the database and the image (or video) is deleted from Firebase Storage. All other posts should be unaffected.	Valid

## Test 1 – User Profile Data is Displayed Correctly

For this test I needed to check that the correct user data was shown in the header cell depending on what user was logged in. To test it, I had to log in and out of different accounts. Initially, when the user first creates their account or the app launches already logged in, the header cell loads correctly because the refresh was called in ViewDidLoad method which gets called the first time the view is shown only. However, I needed to call the refresh function again when the user logs into another account. To solve the problem, I used a NSNotification which can call a function from another view controller. Because the portfolio is embedded in a UITabBarController, I called this notification whenever the UITabBarController view appears with ViewDidAppear. This method is only called when the login page is dismissed and so refreshes the portfolio with the profile (and posts) corresponding to the user just logged in.

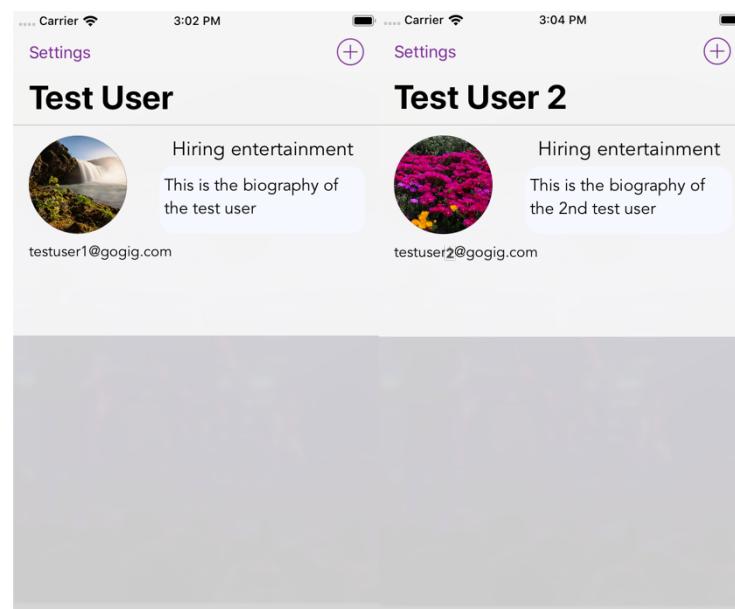
Before:

```
33     override func viewDidLoad() {
34         setupView()
35
36         refreshPortfolio()
37
```

After:

```
33     override func viewDidLoad() {
34         setupView()
35         //refresh from the TabBarController
36         NotificationCenter.default.addObserver(self, selector: #selector(refreshPortfolio), name:
37             NSNotification.Name(rawValue: "refreshPortfolio"), object: nil)
38
39     override func viewDidAppear(_ animated: Bool) {
40         //refresh from the TabBarController
41         NotificationCenter.default.post(name: NSNotification.Name(rawValue: "refreshPortfolio"), object: nil)
42
```

The header cell then displayed the correct information corresponding to the logged in user.

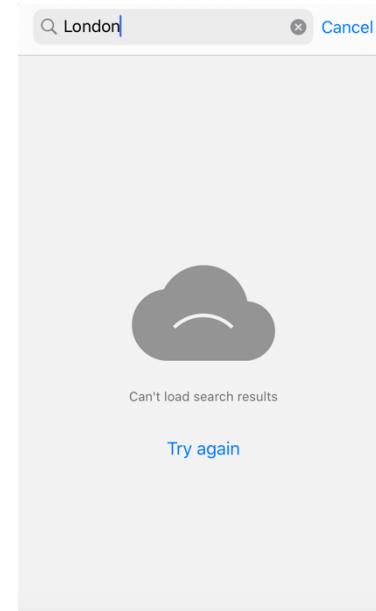


Completing test 1 also achieved test 2 as I could log in and out successfully and it presented the login view as expected.

### Test 3 & 4 – Navigation with Post View & Using Google Places API

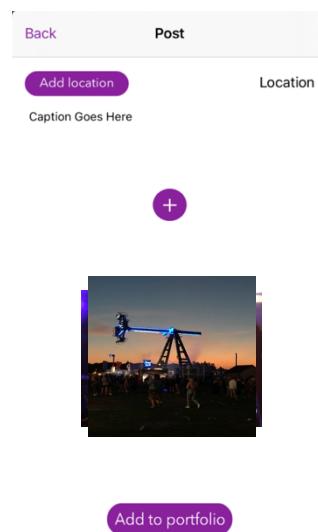
The portfolio post view is presented when the user taps the plus button in the top right corner, the view is dismissed when the user requests it and also when the post has been uploaded to Storage and Database. I did however have an issue with the Google Places API as it requires payment in order to complete a search.

If I were to pay for this service, I could access the returned location name as a string data type. Because this API would work, I will leave this feature as it is, as it only means that “location” is displayed with every post rather than an accurate name.



### Test 5 – Access to camera, photo and sources

Because I reused the same action sheet and methods from the create account process, I had no problem accessing these sources again so the user can select what they want to post completing success criterion 8. The issue I did have was with the PostContainerView class. When adding an image or a video, and then deciding to change the post, it just added a new layer on top of the current preview. Previously I had this issue occurring in the UIView:



As shown, when reselecting a new post, a new layer is created. This means if it is a different orientation, it is visible and if the layer underneath is a video, then the audio of the AVPlayer is still audible. To fix this issue, I needed to clear the PostContainerView whenever something new was added to it.

```

47 func clearView(fit: Bool){
48     //Clear the container view if there is anything
49     //If there is a video, close it
50     if avPlayer != nil {
51         closePlayer()
52     }
53     //If there's an image remove it
54     if self.subviews.count > 0 {
55         imageView.removeFromSuperview()
56     }

```

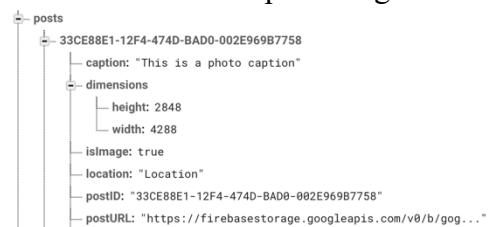
Returning an image ready for posting then worked as expected in all cases.

Test 6 was also completed as an error is reported when expected:

## Test 7 – Posts Displayed in a Scrolling Table View Feed

I had most difficulty with test 7 as a number of bugs occurred.

The first issue was that the cells of the table view did not display and return with a height value. The height is normally set automatically depending on the content within the cell, however the program does not know the height of the cell at runtime because the post image is added to the cell after the view appears as part of database query. To get around this issue, I stored the dimensions of the post in the database (scaled down) so that it fits in the PostContainerView and using the HeightForRowAt delegate method, the table view knows what height each cell needs to be.

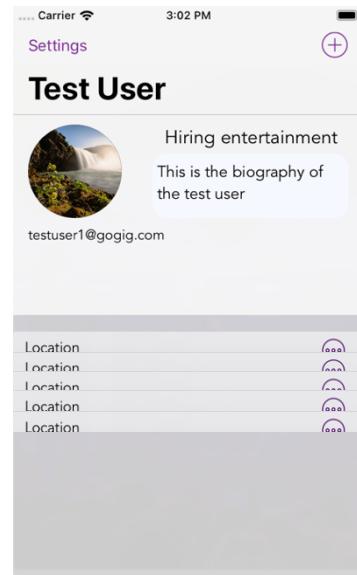


```

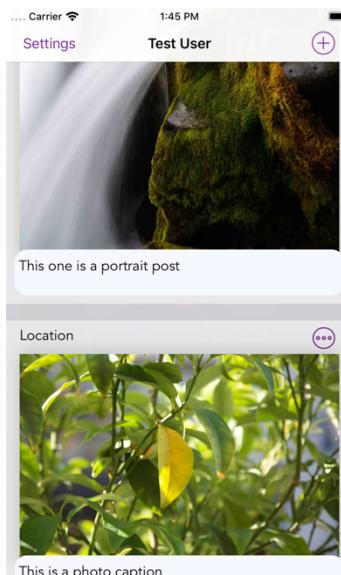
86     //Get ratio of how much to shrink the image by by using the UIView
87     //(width is set using constraints)
88     //We haven't set a height, because that's what we're changing
89     let ratio = self.frame.size.width / imageView.frame.size.width
90     //Change the height of the UIView by setting it to the new height of the imageView
91     self.frame.size.height = imageView.frame.size.height * ratio

```

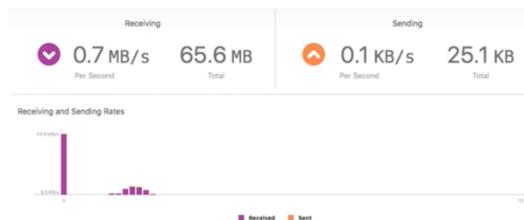
Before:



After:



The second problem I encountered was that because the cells are reused, whenever the user scrolls, the images are redownloaded from Storage continuously. On a mobile device unnecessary mobile data is used. The solution was a method that caches commonly used images.



It is very similar to `downloadImage` method but this time, it checks for an existing cache first and if not then downloads it.

I had an issue with this extension function because within it, I created a video thumbnail image for the feed after it had been downloaded. The scrolling of the feed was horrendous and really slow because as a cell is reused, the video about to be displayed had to be downloaded and have a thumbnail generated. It was simpler to generate a thumbnail before it is uploaded and store the image in Firebase Database and Storage too. Test 7 was a necessary test to provide a better user experience and hence meet success criterion 1 for use of the portfolio view so that it achieves everything it is supposed to in my specification.

```

157 //Storing images as cache reduces the network usage of the app
158 func loadImageCache(url: URL, isImage: Bool, handler: @escaping (_ returnedImage: UIImage) -> ()) {
159
160     let urlString = url.absoluteString as NSString
161     //Check for a cached image under that URL
162     if let cachedImage = imageCache.object(forKey: urlString) {
163
164         handler(cachedImage)
165
166     } else {
167         //Get's the data of the URL
168         let task = URLSession.shared.dataTask(with: url) { data, response, error in
169             if let error = error {
170
171                 print(error.localizedDescription)
172
173             } else {
174
175                 //So picture if first when UI loads
176                 DispatchQueue.main.async() {
177
178                     //Converts the image data to a UIImage
179                     if let downloadedImage = UIImage(data: data!) {
180
181                         imageCache.setObject(downloadedImage, forKey: urlString)
182
183                         handler(downloadedImage)
184
185                     }
186
187                 }
188             }
189             //Resumes the task after image is set
190             task.resume()
191
192         }
193
194     }
195
196     func generateThumbnail(url: URL) -> UIImage {
197
198         do {
199             let asset = AVURLAsset(url: url)
200             let imageGenerator = AVAssetImageGenerator(asset: asset)
201             imageGenerator.appliesPreferredTrackTransform = true
202
203             //Grab an image right at the start of the video
204             let cgImage = try imageGenerator.copyCGImage(at: .zero,
205                 actualTime: nil)
206
207             //Return the image
208             return UIImage(cgImage: cgImage)
209         } catch {
210             print(error.localizedDescription)
211             //Return placeholder if there is an error
212             return UIImage(named: "second")!
213         }
214
215     }
216
217 }
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
15
```

dimensions of the image or video and constraints in the interface builder for it to work. The performance of this feed still is not as good as I want it to be, because you can see the posts loading on the first time you scroll initially. I am confident that image caching was the correct approach to take as it enhances the user experience in not using up mobile data which would then discourage app downloads if it was published. I have also given full control over what posts are associated to an account, so they can decide what things are seen by other people and can reverse any mistakes made. You can still log out of the app and navigate to all areas as expected at any time which meets my criteria of complete user freedom. Most importantly, milestone 2 has meant I've completed a portfolio view for other users to see allowing them to make informed decisions when making contact with each other.

My end users said the following:

Gavin Thorrold: "The portfolio has a good design as it clearly shows who I am looking at with the name as a large title and a profile picture. Watching videos in the feed is a good experience and it feels like a good place to review musicians before hiring them. It is unfortunate that you cannot search for a location with Google without paying for it, but if it is just a location name as you said, do not worry too much about it because they can type this out in the caption if they really have to."

Jude Mills: "The feed is really similar to apps like Instagram, so I know how to navigate through my portfolio and therefore other people's when you get to including that feature. I want you to reuse this screen when viewing other people's portfolios because I like the idea of reviewing my own profile as someone else would see it. One thing I really like is how you have asked the user what type of media they want to post before they select it ('camera', 'photo' and 'video') and I can preview it so I can double check what employers will see."

At the end of this milestone, my end users agreed I met objectives 1, 2, 4, 6, 8 from the success criteria.

## Milestone 3 – Create Event Views (for organisers)

### Development Log

Milestone 3 is a similar process to the account creation. Embedded in a UINavigationController are a series of view controllers which collect input from the user about the event they want to advertise, stores it in a dictionary and uploads the dictionary to the database.

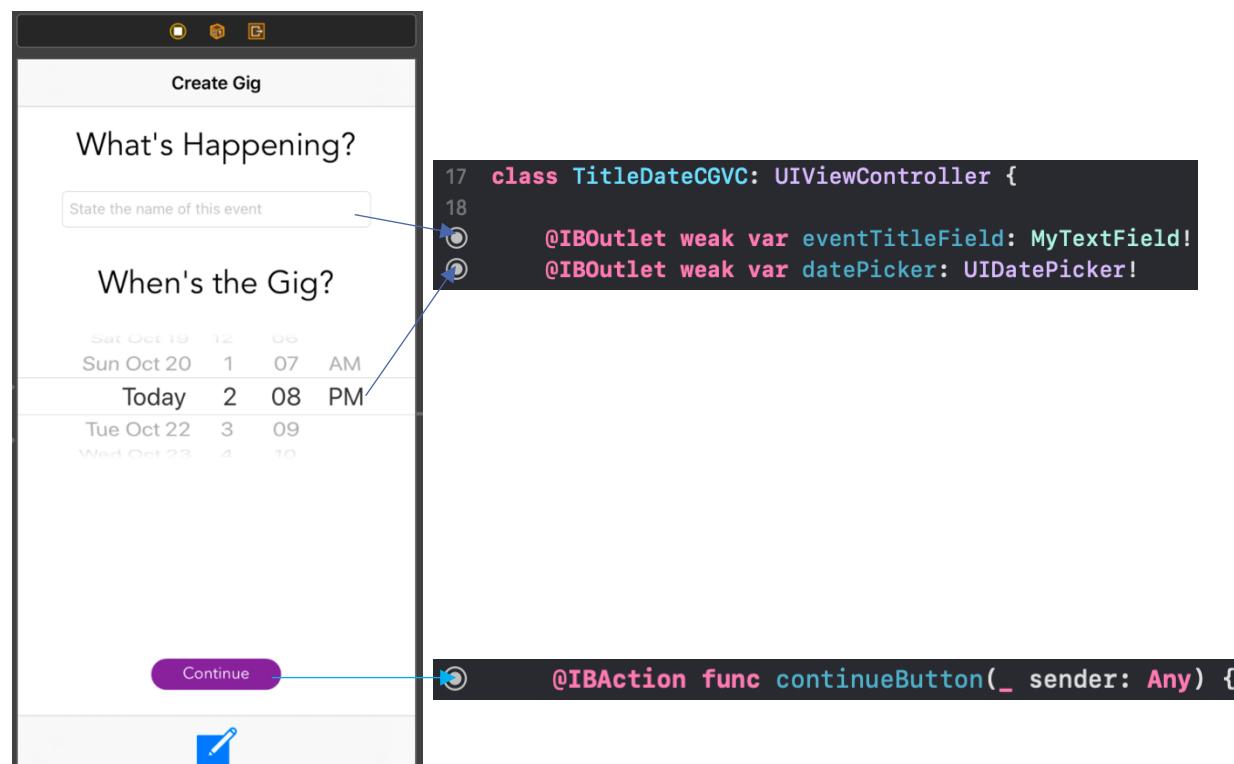
To begin with, the ‘Create Event’ tab should only be accessible by event organisers as stated in the specification. I approached it by storing all tabs in an array and removing the correct one once we can confirm if the user is a musician or not.

```

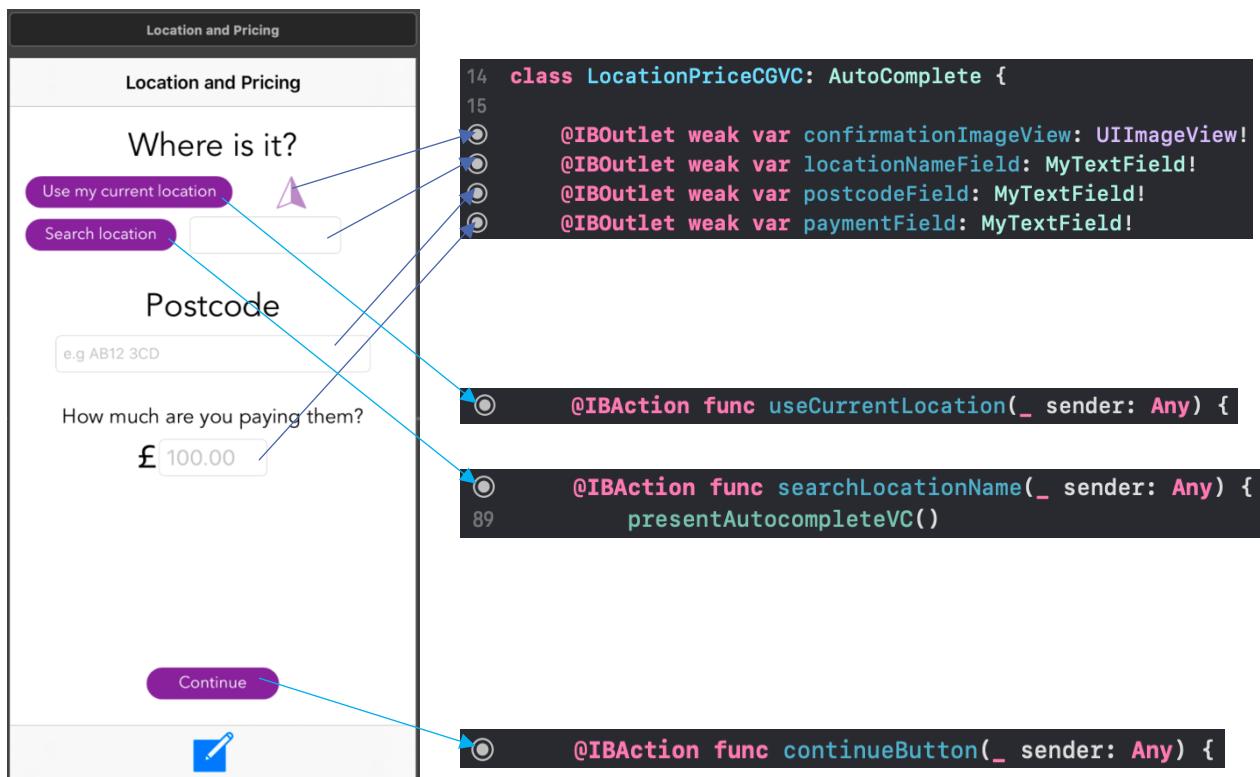
33  @objc func refreshTabs(){
34      print("Tabs have been refreshed")
35      //Set the original state of the tabs (all four)
36      if tabGateOpen {
37          tabs = self.viewControllers!
38
39          //Remove the tabs that shouldn't be seen by musician/organiser
40          if let uid = Auth.auth().currentUser?.uid {
41
42              //IF USER IS LOGGING IN
43              if userGigs == nil {
44                  DataService.instance.getDBUserProfile(uid: uid) { (returnedUser) in
45                      if returnedUser.gigs == true {
46                          self.viewControllers?.remove(at: 0) //Remove create event tab
47                          print(self.viewControllers!)
48                      } else {
49                          self.viewControllers?.remove(at: 1) //Remove find gig tab
50                      }
51                  }
52
53              //IF THE USER IS SIGNING UP FOR THE FIRST TIME
54          } else {
55              if userGigs == true {
56                  self.viewControllers?.remove(at: 0) //Remove create event tab
57                  print(self.viewControllers!)
58              } else {
59                  self.viewControllers?.remove(at: 1) //Remove find gig tab
60              }
61          }
62      }
63  }

```

TitleDateCGVC with connected outlets and actions:

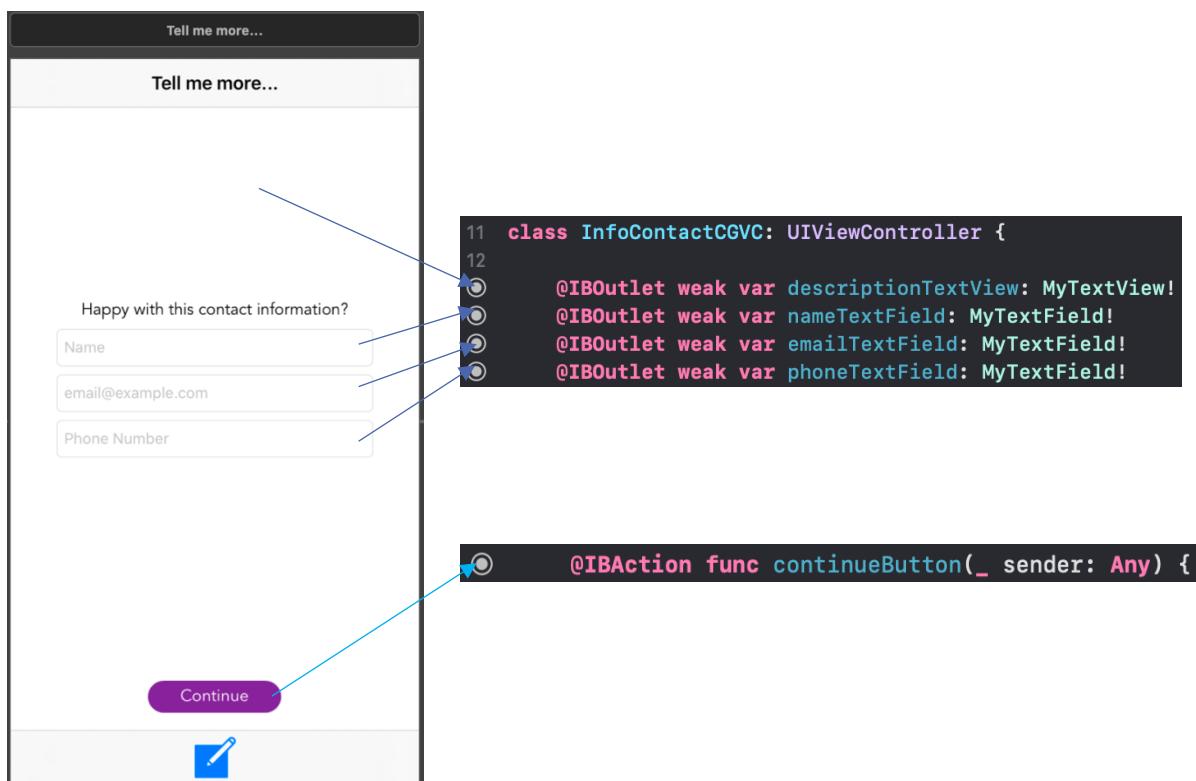


LocationPcieCGVC with connected outlets and actions:

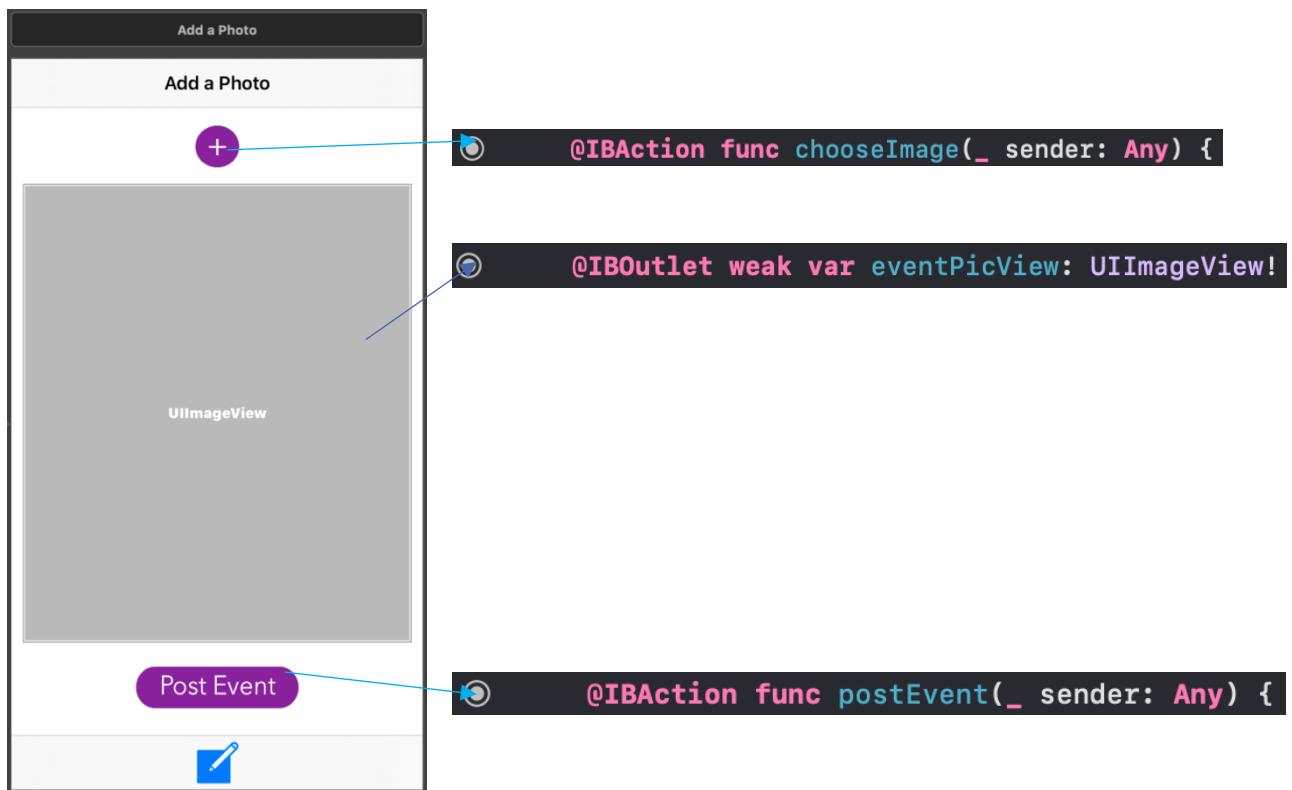


(useCurrentLocation will be covered in milestone 5)

InfoContactCGVC with connected outlets and actions:



PhotoCGVC with connected outlets and actions:



searchLocationName button presents the Google Places API again because LocationPriceCGVC is a subclass of Autocomplete as before. For InfoContactCGVC, the user's email and name are automatically inputted into the contact information fields from their profile to save time and for a better user experience (point 1). However, there is the option to change these if the user wishes completing criterion 9. Depending on what information is required in the form, to avoid error, different keyboards are used. For example, when inputting a payment amount, a number keyboard is presented rather than the standard; the user is forced to enter a Double data type rather than a String.

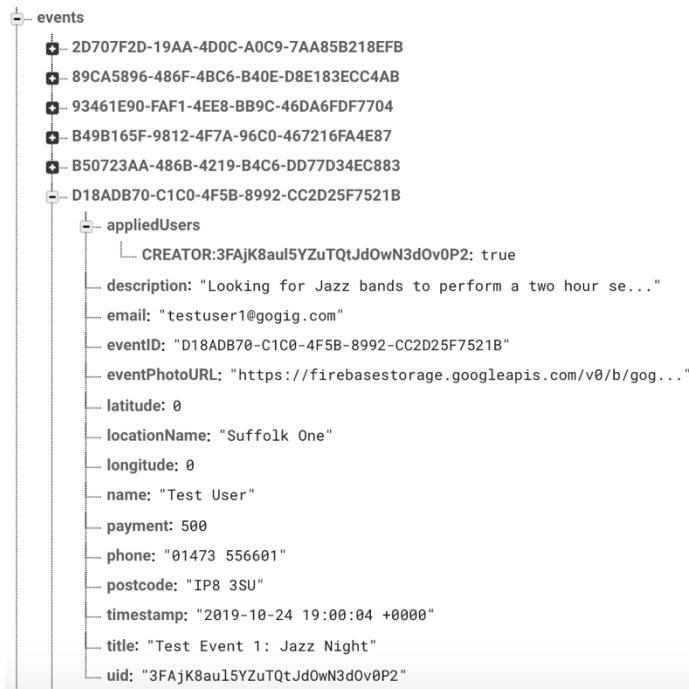
I discussed with my end users whether attaching a photo to the event advert should be optional or not. Because the event is in the future, the organiser may not have photographic evidence of that venue. Nevertheless, we discussed how companies like eBay forces its users to select an image when selling an item like I did in my section 1 research. It also ensures nothing important is left out, which was discussed in my specification too. By making my users attach a photograph, it confirms that the gig is legit for musicians applying and therefore they can assess if they are comfortable with what they are interacting with. The image is chosen from the two sources of camera and photo library and is uploaded to Storage under the folder of “events”.

The collected data is stored in the dictionary eventData and stored in Database under its own child of “events” because all musicians will need to access them later on. The key “appliedUsers” is an array of musicians so that we can keep track of who has already interacted with the gig advert.

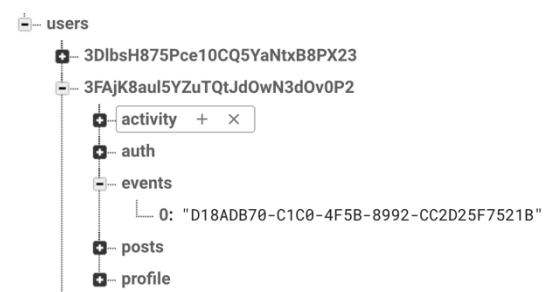
```

115 self.eventData!["appliedUsers"] = ["CREATOR:(self.user!.uid)": true]
["eventPhotoURL": "https://firebasestorage.googleapis.com/v0/b/gogig-
db55e.appspot.com/o/3FAjk8aul5YZuT0tJd0wN3d0v0P2%2Fevents%2FD18ADB70-C1C0-4F5B-8992-
CC2D25F7521B?alt=media&token=c87b1d2a-f591-48b9-862b-1254045e72ab", "longitude": 0.0, "uid": 
```

```
Optional("3FAjk8aul5YzuTQtJd0wN3d0v0P2"), "name": "Test User", "locationName": "Suffolk One", "eventID": "D18ADB70-C1C0-4F5B-8992-CC2D25F7521B", "title": "Test Event 1: Jazz Night", "timestamp": "2019-10-24 19:00:04 +0000", "description": "Looking for Jazz bands to perform a two hour set for our event at Suffolk One. The aim of this event is raise money for charity and inspire music students.", "latitude": 0.0, "email": "testuser1@gogig.com", "payment": 500.0, "phone": "01473 556601", "postcode": "IP8 3SU", "appliedUsers": ["CREATOR:3FAjk8aul5YzuTQtJd0wN3d0v0P2": true]]
```



In addition to storing the event under the “events” child, I stored the eventID in an array under the “user” child in case we need to reference what events belong to an organiser later on.



## Validation Routines

IBoutlet	Rules	Recovery
eventTitleField	<ul style="list-style-type: none"> <li>Maximum Length of 50 characters</li> <li>User has entered title, not equal to empty string</li> </ul>	<u>Display Error</u> Title: “Add a Title” Message: “Please add the title of your event to continue”
datePicker	<ul style="list-style-type: none"> <li>Cannot choose a timestamp before the current date and time.</li> <li>Cannot choose date more than 10 years in advance</li> </ul>	Automatically scroll back to the current date and time
locationNameField	<ul style="list-style-type: none"> <li>User has entered location name, not equal to empty string</li> <li>Length is bigger than 2</li> </ul>	<u>Display Error</u> Title: “Location” Message: “Please search for or type in a location”

postcodeField	<ul style="list-style-type: none"> <li>Postcode has to be equal to length 7 (or 8 with a space)</li> </ul>	<u>Display Error</u> Title: "Postcode" Message: "Please enter the correct postcode of the event"
paymentField	<ul style="list-style-type: none"> <li>Has to enter a Double amount</li> <li><u>Can</u> be equal to 0.00</li> </ul>	<u>Display Error</u> Title: "Payment" Message: "Please enter the chosen amount in a suitable format"
descriptionTextView	<ul style="list-style-type: none"> <li>User has entered a description, not equal to empty string</li> <li>Description length must be more than 10 characters</li> </ul>	<u>Display Error</u> Title: "Event Description" Message: "Please enter a description of your event outlining the suggested points"
nameTextField	<ul style="list-style-type: none"> <li>Name is not equal to empty string</li> </ul>	<u>Display Error</u> Title: "Contact Information" Message: "Please enter the name of the event's organiser"
emailTextField	<ul style="list-style-type: none"> <li>User has entered an email, not equal to empty string</li> <li>email contains “.”</li> <li>email contains “@”</li> <li>email length is more than or equal to 5</li> </ul>	<u>Display Error</u> Title: "Contact Information" Message: "Please enter a valid email or phone number"
phoneTextField	<ul style="list-style-type: none"> <li>Either email, phone or both has to be entered</li> <li>In case of phone being entered, length is more than or equal to 7</li> </ul>	N/A (phone is optional)
eventPicView	<ul style="list-style-type: none"> <li>User has added an event image</li> </ul>	<u>Display Error</u> Title: "Oops" Message: "Please take or add a photo of the venue to post the event"

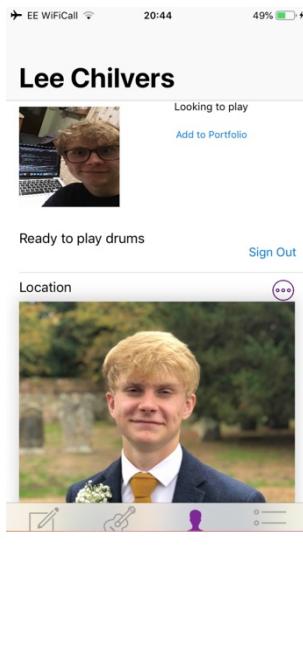
## Test Plan

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure all navigation buttons work as expected	2 – Easy to navigate	Navigation Buttons	All 'continue' buttons and 'back' buttons should take the user to expected parts of the Create Event tab.	Valid
2	Test valid inputs when creating an event	3 – Checking validation rules 2 – Report back errors	<pre>["eventPhotoURL": String_URL, "longitude": 0.0, "uid": String_Uid, "name": "Test User", "locationName": "Suffolk One", "eventID": String_Id, "title": "Test Event 1: Jazz Night", "timestamp": "yyyy-MM-dd HH:mm:ss +zzzz", "description": "Looking for Jazz bands to perform a two hour set for our event at Suffolk One. The aim of this event is raise money for charity and inspire music students.", "latitude": 0.0, "email": "testuser1@gogig.com", "payment": 500.0, "phone": "01473 556601", "postcode": "IP8 3SU", "appliedUsers": ["CREATOR:String_Creator_Uid: true]]</pre>	If user has not entered required information about the event, app should display an error and not progress.	Valid and Invalid test data
3	Make sure Google Places API returns a location string when searched to attach to the event	7 – Optional location	Google Places API	The Google Places Autocomplete View should display and return a location name when searching with the search bar.	Valid
4	Access camera, image library and video library	8 – Access device camera and media library	Camera or Device Photo Library App	After taking or choosing an image, the returned image should be displayed to the user and stored in Firebase Storage when user completes their post.	Valid
5	Check gig event creation	6 – Efficient Database storage	Button	The event should be added publicly to the database with all correct data	Valid

## Test 1 – User Navigation

The UINavigationController meant that navigation between the various view controllers was easy and of a similar format to account creation. The navigation bugs I had in this milestone was to do with the tabs belonging to the UITabBarController. Firstly, when the app first

launches and the user is already logged in, because I query the database first to find out what type of user they are, removing the correct tabs can be very slow when the connection is not very good. The problem is that it allows users to access parts of the app they are not supposed to if they are not registered as that type of user because all four tabs remain.



The way to fix this was to store the value of the user type (Bool) by storing it locally on the device rather than fetching from the Database. Tabs are therefore set without need for internet and set quicker avoiding error. UserDefaults allows me to store a Bool data type under a string key.

```

33  @objc func refreshTabs(){
34      print("Tabs have been refreshed")
35      //Set the original state of the tabs (all four)
36      if tabGateOpen {
37          tabs = self.viewControllers!
38      }
39
40      //IF USER IS RESUMING
41      if let userGigsDefaults = UserDefaults.object(forKey: "gigs") as? Bool {
42
43          //Remove the tabs that shouldn't be seen by musician/organiser
44          if tabGateOpen {
45              if userGigsDefaults == true {
46                  print("tabs reset as expected")
47                  self.viewControllers?.remove(at: 0)
48                  print(self.viewControllers!)
49              } else {
50                  self.viewControllers?.remove(at: 1)
51              }
52
53              tabGateOpen = false
54
55              //Makes the initial tab the profile tab
56              self.selectedIndex = 1;
57      }

```

```

87  func setDefaults(userGigsCondition: Bool) {
88      if userGigsCondition == true {
89          //remove the create tab and view
90          self.viewControllers?.remove(at: 0)
91
92          //write value to device
93          UserDefaults.set(true, forKey: "gigs")
94      } else {
95          //remove the find tab and view
96          self.viewControllers?.remove(at: 1)
97
98          //write value to device
99          UserDefaults.set(false, forKey: "gigs")
100     }
101
102     tabGateOpen = false
103
104     //Makes the initial tab the profile tab
105     self.selectedIndex = 1;
106 }

```

The second issue concerning tabs was that they did not reset when a user logged out so that if say a musician logged in, the state of the tab bar would be for the organiser. Fixing involved resetting the state of the tabs and setting the UserDefaults key to nil when they sign out in UserAccountVC.

```

122
123     try Auth.auth().signOut()
124     let loginVC = self.storyboard?.instantiateViewController(withIdentifier: "LoginSignupVC") as?
125         LoginSignupVC
126     self.present(loginVC!, animated: true, completion: nil)
127
128     //When the user logs out we need to return the tab bar to its original state ready for either
129     //type of user to log in
130     if let tabBarController = self.tabBarController {
131         tabBarController.viewControllers = tabs
132
133         self.uid = nil
134
135         UserDefaults.set(nil, forKey: "gigs")
136     }

```

The tabs reset and displayed as expected for the different user types when logging in and out. By going for the tab layout, an event organiser can start putting together an event and navigate through other tabs of the app and return to this one to resume the form, giving the user complete navigation freedom in the app's use which was important for success criterion 2.

## Test 2 – Valid Inputs

Displaying errors based on checks against the validation rules was a similar process to account creation and testing each scenario displayed the correct UIAlertController when expected. The main problem with this testing stage was with the UIDatePicker. I needed to set rules so that if it is scrolled back in time, or ten years in the future it set back to the current date to prevent these cases.

```

45     dateFormatter.dateFormat = "yyyy-MM-dd HH:mm:ss zzzz"
46
47     //Set restrictions on the Date Picker
48     let calendar = Calendar(identifier: .gregorian)
49     let currentDate = Date()
50     var components = DateComponents()
51     components.calendar = calendar
52
53     //Can only choose a date up to 10 years in advance
54     components.year = 10
55     components.month = 0
56     let maxDate = calendar.date(byAdding: components, to: currentDate)!
57
58     //Cannot choose date or time before current time
59     components.year = 0
60     let minDate = calendar.date(byAdding: components, to: currentDate)!
61
62     datePicker.minimumDate = minDate
63     datePicker.maximumDate = maxDate
64

```

Another bug I found with the date picker was that whatever time the user set, it returned the current Date as an hour behind. I fixed it, by adding an hour (3600 seconds).

```

//Raw value from picker - Hour behind, add an hour (3600s)
let chosenTime = datePicker.date.addingTimeInterval(3600)
//string for FIR
let timestamp = "\(chosenTime)"

```

## Test 3 & 4 – Google Places API & Camera/Photo Library

As stated before, Google Places requires payment to use. To get around this issue, I simply provided a text field to type in the location rather than search for it. I reused the same methods from the extension file to access the camera and photo library, so I had no problem there.

## Test 5 – Gig Event Creation

The function to add the eventData dictionary to the database worked correctly. The bug I had was adding the eventID to the array in the database. Whenever a new event was added, instead of appending to the array, the previous event was overwritten.

Before:

```

350     func updateDBUserEvents(uid: String, eventID: String) {
351         REF_USERS.child(uid).child("events").setValue(eventID)
352     }

```

I worked out setValue overwrites the value at that location. Therefore, I fetched the array and appended to it locally and then set the value overwriting the old array in Database.

After:

```

351     //Simply keep record of which events this user has interacted with to list in the 'My Events' section
352     func updateDBUserEvents(uid: String, eventID: String) {
353         //Get the current array of user's events
354         //And append to it to update the database with
355         getDBUserEvents(uid: uid) { (returnedEvents) in
356             var recordedEvents = returnedEvents
357             //recordedEvents.insert(eventID, at: 0)
358             recordedEvents.append(eventID)
359
360             self.REF_USERS.child(uid).child("events").setValue(recordedEvents)
361         }
362     }

```

## Milestone Review

The third milestone included many validation tests because the event organiser has to provide all necessary input to post an informative advert for musicians. I think the way I have put

together this tab is good because it splits the event creation process into view controllers and with the UINavigationController the user is guided through the process while still having freedom to go back and navigate through other parts of the app. Because I am reusing extension functions, tasks like accessing the photo library and uploading data to Firebase is easy to do. My main challenge for this milestone was resetting the tabs for when people log out of the app and also finding the optimum solution on how to set them when the app first launches (storing the value locally with UserDefaults).

I believe I have made the 'Create Event' section simple and understandable to interact with by using placeholders and large titles which meets point 1 of my success criteria. As part of objective 2, all tabs of the app can be reached at any time, even while in the middle of creating an event to post allowing complete user freedom in navigation. I debated on whether attaching an image should be optional or not, but my end users agreed that posting an advert needs photo evidence.

In the next milestones, I will include the feature to pin location coordinates to the gig event as this was not implemented as part of milestone 3. In addition, to meet the success criteria in allowing users to edit anything associated to their account, I will include the feature for organisers to edit their events or delete them completely.

My end users reported back at the end of milestone 3:

Gavin Thorold: "I like how you have put this screen together because I know what information to add as I am told. Everything is spaced out nicely as well and it is not overwhelming despite the amount of information I need to provide. I particularly like the description text box because I can put anything special that needs mentioning in here."

Jude Mills: "I understand that as a musician, I generally won't be able to use this feature because of how I sign up, but you have put together a way for people to post a job which ensures all the information I need to consider applying is collected."

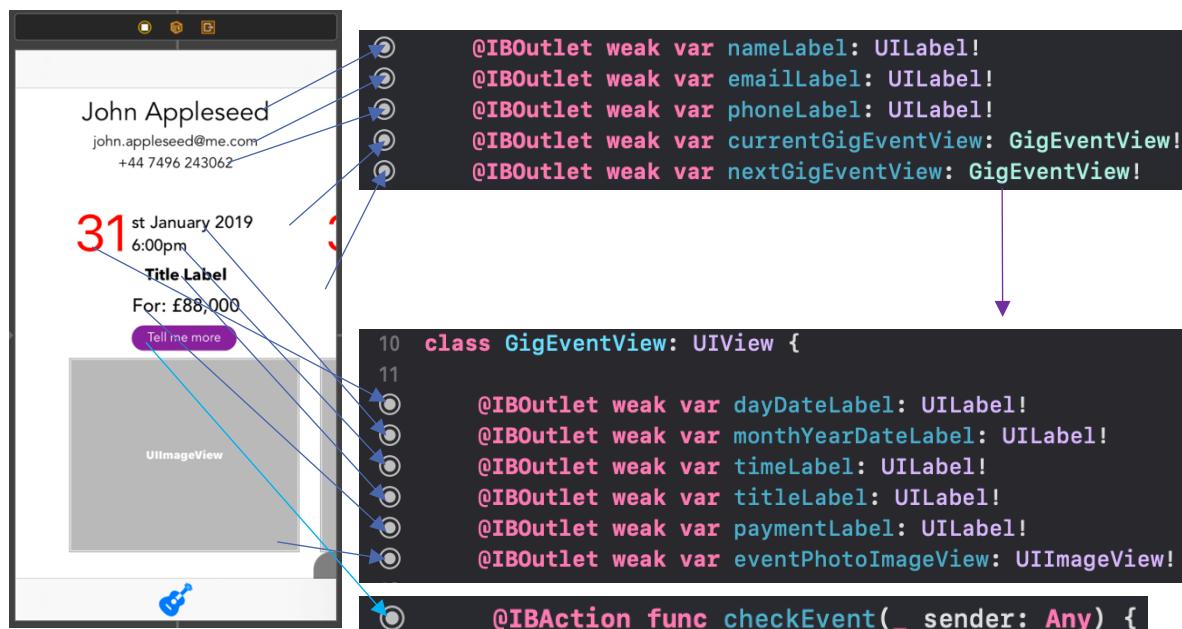
At the end of this milestone, my end users agreed I met objectives 1, 2, 3, 8 from the success criteria.

## Milestone 4 – Find Gig View (for musicians)

### Development Log

The fourth milestone included dragging gestures on the touch screen. For a musician, they can apply for a gig by swiping a card right and ignore the listing by swiping it left. I had to grab event information and display it in a UIView and make it interactable with UIPanGestureRecognizer.

FindGigVC with connected outlets and the draggable GigEventView which is a subclass of UIView:



To begin with, I create the GigEvent model class to instantiate objects and store them in an array ready to display to musicians. The method to fetch each event from firebase is similar to the user profile and portfolio posts.

```

class GigEvent: Comparable {
    //attributes
    private var uid: String
    private var id: String
    private var title: String
    private var timestamp: String
    private var description: String
    private var latitude: Double
    private var longitude: Double
    private var distance: Double
    private var locationName: String
    private var postcode: String
    private var payment: Double
    private var name: String
    private var email: String
    private var phone: String
    private var eventPhotoURL: URL
    private var appliedUsers: [String: Bool] // to keep track of what musicians have seen event

    //instantiate GigEvent object
    init(uid: String, id: String, title: String, timestamp: String, description: String, latitude: Double, longitude: Double, locationName: String, postcode: String, payment: Double, name: String, email: String, phone: String, eventPhotoURL: URL, appliedUsers: [String: Bool]) {
        self.uid = uid
        self.id = id
        self.title = title
        self.timestamp = timestamp
        self.description = description
        self.locationName = locationName
        self.latitude = latitude
        self.longitude = longitude
        self.postcode = postcode
        self.payment = payment
        self.name = name
        self.email = email
        self.phone = phone
        self.eventPhotoURL = eventPhotoURL
        self.distance = 0.0
        self.appliedUsers = appliedUsers
    }
}

```

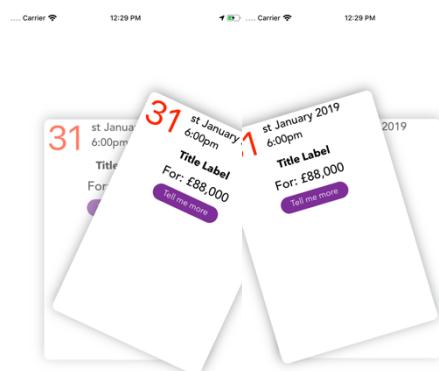
Before I displayed the gig event data in the `UIView`, I first wanted to put together the `UIPanGestureRecognizer` which calls a function whenever you drag across the screen. I wanted the animation of the `GigEventView` to be in the style of flicking away a playing card which meant that I would need to translate and rotate the `UIView`. Here is the function I put together to meet an aesthetically pleasing but understandable user gesture (success point 1):

```

291 //called method when the gesture is recognised
292 //Pan Gesture - swipe across screen
293 @objc func gigEventWasDragged(gestureRecognizer: UIPanGestureRecognizer) {
294
295     //Returns a vector of where user drags to
296     let translation = gestureRecognizer.translation(in: view)
297
298     let theView = gestureRecognizer.view!
299
300     //move the view to where the user is dragging
301     theView.center = CGPoint(x: self.view.bounds.width / 2 + translation.x, y: self.view.bounds.height / 2 +
302                             translation.y)
303
304     //calculate distance of view from the centre
305     let xFromCenter = theView.center.x - self.view.bounds.width / 2
306
307     //view will rotate as it moved further from centre (radians)
308     //will rotate less the further from the centre it goes - view won't go upside down
309     var rotation = CGAffineTransform(rotationAngle: xFromCenter / 200)
310
311     //apply the rotation to the view
312     theView.transform = rotation
313
314     //when the user finished dragging
315     if gestureRecognizer.state == UIGestureRecognizer.State.ended {
316
317         //the area at which a definite choice has been made:
318         //dragged left
319         if theView.center.x < 40 {
320
321             didChoose(applied: false)
322             confirmChoiceAnimation(applied: false)
323
324             //return the view to the centre
325             rotation = CGAffineTransform(rotationAngle: 0)
326             theView.center = CGPoint(x: self.view.bounds.width / 2, y: (self.view.bounds.height / 2) + 60)
327
328         //dragged right
329         } else if theView.center.x > self.view.bounds.width - 40 {
330
331             didChoose(applied: true)
332             confirmChoiceAnimation(applied: true)
333
334             rotation = CGAffineTransform(rotationAngle: 0)
335             theView.center = CGPoint(x: self.view.bounds.width / 2, y: (self.view.bounds.height / 2) + 60)
336
337         }
338     }
339 }

```

Which produced this effect:



I wanted the card to shrink as it got close to the edge of the screen as well though, so I changed these lines of the function:

```

310     //shrink the card as it reaches screen edge - set the scale
311     let scale = min(abs(100 / xFromCenter), 1)
312
313     //the stretch and rotation effect set by the scale
314     var stretchAndRotation = rotation.scaledBy(x: scale, y: scale)
315
316     //apply the rotation and stretch to the view
317     theView.transform = stretchAndRotation

```

And the card animated as expected.



The next thing to do was to present the gig event data in a GigEventView. Contact information should be displayed large at the top of the view and all information about the event displayed in the UIView. Nothing important should be hidden (point 4). In addition, whenever there is more than one event in the array, the next object should be shown in the nextGigEventView enhancing the effect of a stack of cards. I just updated the outlets:

```

144     func displayGigEventInfo(gigEventView: GigEventView, gigEvent: GigEvent) {
145         //show data about the event on the card
146         gigEventView.dayDateLabel.text = gigEvent.getDayDate()
147         gigEventView.monthYearDateLabel.text = gigEvent.getLongMonthYearDate()
148         gigEventView.timeLabel.text = gigEvent.getTime()
149         gigEventView.titleLabel.text = gigEvent.getTitle()
150         gigEventView.paymentLabel.text = "For: £\u00a3" + gigEvent.getPayment()
151     }

```

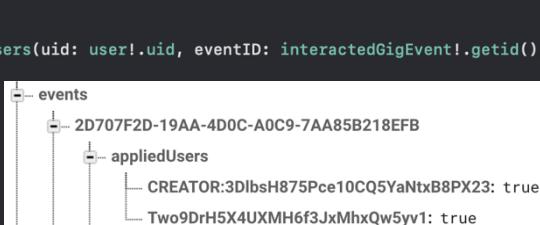
In the case where the array is more than length one, update both the currentGigEventView and nextGigEventView again displaying the correct thing – success criterion 4.

When the user makes the decision (drags left or right) update the database under applied users of their choice which keeps track of who has interacted with the event already to avoid error and hence meet success criterion 3.

```

232     //MARK: UPDATE APPLIED USERS
233
234     func didChoose(applied: Bool){
235
236         //Get the interacted users of that event
237         var gigEventAppliedUsers = interactedGigEvent?.getAppliedUsers()
238         //and add a new key with the current users uid
239         gigEventAppliedUsers![user!.uid] = applied
240
241         //update the dictionary in the database
242         DataService.instance.updateDBEventsInteractedUsers(uid: user!.uid, eventID: interactedGigEvent!.getid(),
243             eventData: gigEventAppliedUsers!)
244
245         //remove the card from the gigEvent stack
246         gigEvents.remove(at:0)
247
248         //update the UI
249         updateCards()
    }

```



I had successfully grabbed an array of events, displayed them in a draggable UIView, which animates nicely, and updated the backend based on the user's interaction with the card.

## Test Plan

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure we can grab all events as gig opportunities	6 – Efficient data storage in database	Firebase Database	Query should return an array of Events to be displayed to the musician	Valid
2	Test swipe gesture to apply for gig	7 – Good user experience 7 – Simple Interaction	Touch screen gesture	Data about an event is displayed in a draggable card. When swiped to the left or right, database should update	Valid

				listing the musician as an interacted user.	
3	Check that gig opportunities already interacted with do not appear again.	3 – Check against rules to avoid error and confusion	Firebase Database	If the user has already interacted with the event. The program should block the display of that event object.	Invalid

## Test 1 – Displaying Gig Opportunities

Displaying the gig event object in the `nextGigEventView` and `currentGigEventView` was no problem. My code even hid the `nextGigEventView` when there was only one object left in the array. However, in testing it, I found that pictures on the card did not update very quickly. I was downloading the gig image every time which would be unnecessary usage of mobile data.

```
181         downloadImage(url: currentGigEvent.getEventPhotoURL()) { (returnedImage) in
182             //download the image if first in array
183             self.currentGigEventView.eventPhotoImageView.image = returnedImage
184         }
```

The image is always downloaded for `nextGigEventView` and that is fine because it is not seen due to the card on top. However, I can store the image downloaded under a variable and reuse it for `currentGigEventView` which saves having to download it again.

```
205         //this image is always downloaded
206         downloadImage(url: nextGigEvent.getEventPhotoURL()) { (returnedImage) in
207             //prepare image to display in current event card
208             self.nextGigEventView.eventPhotoImageView.image = returnedImage
209             self.nextEventImage = returnedImage
210         }

177         //get image from nextGigEventView or download one
178         if nextEventImage != nil {
179             currentGigEventView.eventPhotoImageView.image = nextEventImage
180         } else {
181             downloadImage(url: currentGigEvent.getEventPhotoURL()) { (returnedImage) in
182
183                 self.currentGigEventView.eventPhotoImageView.image = returnedImage
184             }
185         }
```

## Test 2 – Swipe Gesture

I managed to get the animation working for the `currentGigEventView` fairly easily. Although I wanted to hide everything and let the user know that there is nothing to apply to when the array returns empty, otherwise the view would make no sense.

```
214     } else {
215         //No gigs to apply for
216         nameLabel.text = "No Gigs Around"
217         emailLabel.text = "Share GoGig"
218         nextEventImage = nil
219         phoneLabel.isHidden = true
220         currentGigEventView.isHidden = true
221         nextGigEventView.isHidden = true
222         refreshButton.isHidden = false
223     }
224 }
230 func confirmChoiceAnimation(applied: Bool) {
231     var confirmationImageView: UIImageView?
232     if applied {
233         //will be a tick image
234         confirmationImageView = UIImageView(image: UIImage(named: "appliedGigEvent"))
235     } else {
236         //will be a cross image
237         confirmationImageView = UIImageView(image: UIImage(named: "ignoredGigEvent"))
238     }
239     //frame of the image view
240     confirmationImageView!.frame = CGRect.init(x: 0, y: 0, width: 100, height: 100)
241     confirmationImageView!.center = self.view.center
242     //subtle confirmation
243     confirmationImageView!.alpha = 0.5
244     view.addSubview(confirmationImageView!)
245     //grow and fade
246     UIView.animate(withDuration: 0.2, animations: {
247         //grow x1.3
248         confirmationImageView!.transform = CGAffineTransform(scaleX: 1.3, y: 1.3)
249     }) { (complete) in
250         //fade in 0.2 seconds from 0.5 opacity to 0.0
251         UIView.animate(withDuration: 0.2, delay: 0.2, options: .curveEaseOut, animations: {
252             confirmationImageView!.alpha = 0.0
253         }) { (complete) in
254             //after animation, remove it from the view
255             confirmationImageView!.removeFromSuperview()
256         }
257     }
258 }
```

I also needed to show that the musician had actually applied for a gig which I achieved with a UIImageView which animates on the screen with either a tick or a cross meeting objective 2 of the success criteria in reporting back to the user to create an understandable system and avoid confusion.

I also tested at what point was comfortable to drag the card to in order to interact with the card. Here I am fully taking advantage of the iPhone touchscreen gestures as stated in my specification

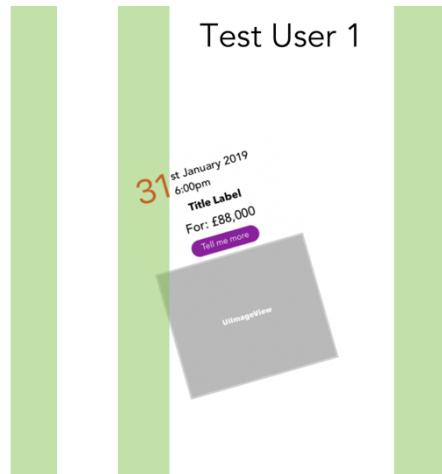
100 Pixels from each side



25 Pixels from each side



40 Pixels from each side



```
322         //the area at which a definite choice has been made:
323         //dragged left
324         if theView.center.x < 40 {
325
326             //dragged right
327         } else if theView.center.x > self.view.bounds.width - 40 {
```

I found that 100 pixels wasn't enough space to show the shrink and rotation animation, and 25 pixels you had to stretch your finger on each

side of the screen which wasn't a good user experience. This is why I chose 40 pixels from each side.

## Test 3 – Rules of Gigs Appearing

The database has recorded who has already interacted with the gig, but this did not prevent it from appearing again to the musician in the view. When I fetch the data from the database in DataService class, I check to make sure the user isn't in this array before returning the object. Before:

```
220         //Loop through them and grab data for instantiation
221         for snap in snapshot {
222
223             if let eventData = snap.value as? NSDictionary {
224
225                 if let appliedUsers = eventData["appliedUsers"] as? [String: Bool] {
226
227                     if let eventID = eventData["eventID"] as? String {
228                         if let eventTitle = eventData["title"] as? String {
229                             if let timestamp = eventData["timestamp"] as? String {
```

After:

```
220         //Loop through them and grab data for instantiation
221         for snap in snapshot {
222
223             if let eventData = snap.value as? NSDictionary {
224
225                 if let appliedUsers = eventData["appliedUsers"] as? [String: Bool] {
226                     if self.checkAppliedUsers(appliedUsers: appliedUsers) {
227                         //check to see if user has already interacted
228                         if let eventID = eventData["eventID"] as? String {
229                             if let eventTitle = eventData["title"] as? String {
230                                 if let timestamp = eventData["timestamp"] as? String {
```

Which calls:

```

276      //So that the same gig doesn't appear twice to a musician that has already seen it
277      func checkAppliedUsers(appliedUsers: [String: Bool]) -> Bool {
278
279          for (uid, _) in appliedUsers {
280              if uid == Auth.auth().currentUser?.uid {
281
282                  //print("denied access to gig")
283                  return false
284              }
285          }
286
287          //print("granted access to gig")
288          return true
289      }

```

Where returning false will not go on to instantiate a GigEvent object. Gigs which had previously been interacted with updated the database and did not appear again after the view was refreshed hence meeting success criterion 4 in displaying the correct thing.

## Milestone Review

I was happy with the complex animation I put together as part of this milestone. I battled more with optimising the solution so that it met the criteria of good user experience. My two main examples of this were preventing an image download saving mobile data (and reducing time complexity) and designing the UIPanGesture so the distance of the drag required was both visually pleasing and ergonomic.

I am really pleased with how this view looks so far with the ‘pack of cards’ feel and various animations on the view. I think it will look better when colour is added to the view though. In terms of the success criteria I felt I met a good user experience and simple interaction (point 1), the system checks that the musician has not already interacted with that event to prevent error (point 3) and data is stored efficiently in Database (point 6).

Feedback from my end users:

Gavin Thorrold: “I really like how you have encapsulated all the information I would have entered in the previous view within a small rectangle which applicants can swipe around. The one thing I am assuming is that if I click the “Tell me more” button I would be able to look at the description I inputted as it does not work at the moment. The view is a fun way to apply for a job because it involves animation rather than simply clicking a button.”

Jude Mills: “You have done a really good job at bringing the app to life with this view. The button on the cards doesn’t work though, I don’t know if you knew that. But you have designed the cards really well displaying everything in an informative way which is really helpful if I wanted to put my name out there for a gig.”

At the end of milestone 4, my end users agreed I met objectives 1, 2, 3 from the success criteria.

## Milestone 5 – Sorting Gigs by Location

### Development Log

Before I accessed the location services on the device, I needed a sorting algorithm which would sort the gigs based on distance from the user following my specification for the view closely. The Quicksort algorithm is ideal because, depending on how many people are interacting with the app, the array of events could become fairly large and Quicksort is considered to have one of the best performances in the average case for most inputs  $O(n \log n)$ .

```
func quickSort(array: [Int]) -> [Int] {
    //Base case for recursion (escape)
    if array.isEmpty { return [] }
    //Store the first element of the array to compare it with smaller or larger number
    let first = array.first!
    //first half = all values smaller or equal to first
    let smallerOrEqual = array.dropFirst().filter { $0 <= first }
    //second half = values large than first
    let larger = array.dropFirst().filter { $0 > first }
    //First and second half are recursed and inserted either side of the first value
    return quickSort(array: smallerOrEqual) + [first] + quickSort(array: larger)
}
```

This function only works if the array holds integer data types. I could write multiple functions for all the data types I could encounter or make use of **Generics**.

```
65     //Quick Sort an array of type generic
66     func quickSort<T: Comparable>(array:[T]) -> [T] {
67         //Base case for recursion (escape)
68         if array.isEmpty { return [] }
69         //Store the first element of the array to compare it with smaller or larger number
70         let first = array.first!
71         //first half = all values smaller or equal to first
72         let smallerOrEqual = array.dropFirst().filter { $0 <= first }
73         //second half = values larger than first
74         let larger = array.dropFirst().filter { $0 > first }
75         //First and second half are recursed and inserted either side of the first value
76         return quickSort(array: smallerOrEqual) + [first] + quickSort(array: larger)
77     }
```

Only the interface and type has changed so that the function can sort any type that conforms to the **Comparable** protocol. I want to sort an array of the GigEvent objects therefore the class needs to conform to Comparable.

```
161     //Quicksort based on location – nearest is first
162     static func < (lhs: GigEvent, rhs: GigEvent) -> Bool {
163         //compare greater than by distance
164         return lhs.getDistance() < rhs.getDistance()
165     }
166     //compare equal to by distance
167     static func == (lhs: GigEvent, rhs: GigEvent) -> Bool {
168         return lhs.getDistance() == rhs.getDistance()
169     }
```

With the Quicksort algorithm, I can now sort an array of GigEvent objects by distance from the user.

My plan of development for milestone 5 was to pin coordinates to an event before posting, use the musicians coordinates and find the distance between the two allowing me to then sort. When clicking useCurrentLocation button for LocationPriceGCVC start receiving coordinates in an array form and upload the most recent update to the database.

```

64    var currentLocationOn = false
65    @IBAction func useCurrentLocation(_ sender: Any) {
66        //start updating location
67        if currentLocationOn == false {
68            locationManager.delegate = self
69            locationManager.startUpdatingLocation()
70            //show it's being used
71            confirmationImageView.alpha = 1.0
72            confirmationImageView.isHidden = false
73            //stop updating the location
74        } else {
75            currentLocationOn = false
76            locationManager.stopUpdatingLocation()
77            //set coordinates back to zero if they
78            //don't want to use location
79            eventLatitude = 0.00
80            eventLongitude = 0.00
81            //show it's not being used
82            confirmationImageView.alpha = 0.3
83        }
84    }
85
86
87    var eventLatitude = 0.00
88    var eventLongitude = 0.00
89    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
90        //grab the most up to date coordinates (first in array)
91        let userLocation: CLLocation = locations[0]
92        eventLatitude = userLocation.coordinate.latitude
93        eventLongitude = userLocation.coordinate.longitude
94    }

```

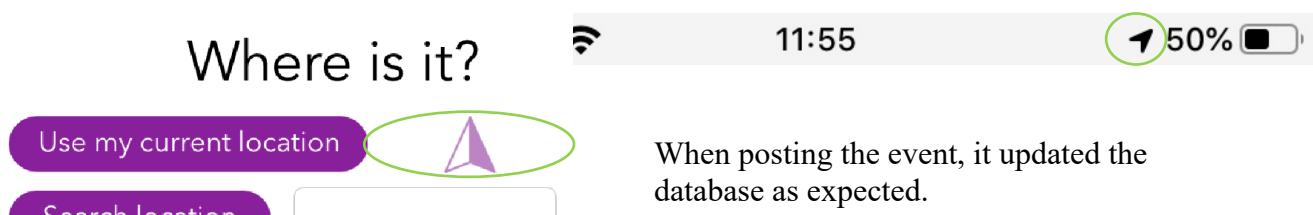
To save battery, I made the accuracy of the coordinate update to the nearest 100m.

```

27    let locationManager: CLLocationManager = {
28        let lm = CLLocationManager()
29        //To nearest hundred metres, to save battery
30        lm.desiredAccuracy = kCLLocationAccuracyHundredMeters
31        lm.requestWhenInUseAuthorization()
32        return lm
33    }()

```

The user knows that their current location is being used by both my own feedback and the iOS icon:



I had to request permission for location access with info.plist file. In the case that the user rejects the permission request, latitude and longitude update as 0.00, 0.00 and so is optional (criterion 9).

When posting the event, it updated the database as expected.

```

latitude: 37.33233141
locationName: "Location_Name"
longitude: -122.0312186

```

Next I had to request the musician's location when they enter the FindGigVC tab.

```

86     func refresh() {
87         if let uid = Auth.auth().currentUser?.uid {
88             DataService.instance.getDBUserProfile(uid: uid) { (returnedUser) in
89                 self.user = returnedUser //instantiate a User object
90                 DataService.instance.getDBEvents(uid: uid) { (returnedGigEvents) in
91                     //self.gigEvents = returnedGigEvents
92                     self.gigEvents = self.setGigEventDistances(gigs: returnedGigEvents)
93                     //request musician location and sort by it
94                     self.updateCards()
95
96                 }
97             }

```

And use these coordinates to instantiate locations and find the distance between the two to quick sort. This will hopefully complete the main feature of the View and meet what Jude wanted in the specification for local opportunities appearing first.

```

102    var userLatitude = 0.00
103    var userLongitude = 0.00
104    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
105        //Grab the coordinates
106        let userLocation: CLLocation = locations[0]
107        userLatitude = userLocation.coordinate.latitude
108        userLongitude = userLocation.coordinate.longitude
109    }
110
111    //We compare the two coordinates of the gig and the user
112    //update the distance attribute
113    //then use the distance to do a quick sort
114    func setGigEventDistances(gigs: [GigEvent]) -> [GigEvent] {
115
116        locationManager.delegate = self
117        locationManager.startUpdatingLocation()
118        //Instanitiate a location out of coordinates
119        let userLocation = CLLocation(latitude: userLatitude, longitude: userLongitude)
120
121        for gig in gigs {
122
123            //Get the location of the GigEvent object
124            let gigEventLocation = gig.getGigEventLocation()
125            //Find the distance between the two
126            let distance = gigEventLocation.distance(from: userLocation) as Double
127            //Set the distance from user of the GigEvent object
128            gig.setDistance(distanceFromUser: distance)
129        }
130        //Sort the objects by distance
131        return quickSort(array: gigs)
132    }

```

155 //MARK: get exact point location using the latitude and the longitude
156 func getGigEventLocation() -> CLLocation {
157 let gigEventLocation = CLLocation(latitude: latitude, longitude: longitude)
158 return gigEventLocation
159 }

78 func setDistance(distanceFromUser: Double) {
79 self.distance = distanceFromUser
80 }

## Test Plan

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure we can fetch the current location of the device	7 – Location services accessed	Device location services	When requested, location coordinates are printed to the console and updated when changing location.	Valid

2	Enable optional feature to attach these coordinates to an event when creating one.	9 – Users decide on the amount of information they share.	Button	If user pins their current location to the event, database should store their coordinates under the event object. The program should mark coordinates as 0.00, 0.00 if location services is disabled by user.	Valid and Borderline
3	Check program can calculate distance to event location	7 – Location accessed	Firebase Database, Device location services	Program should be able to calculate distance between the musician's coordinates and the event coordinates.	Valid
4	Make sure sorting the event array by location works	7 – Data is sorted by locality.	The GigEvent objects	I will check that the sorting algorithm works by manually inputting coordinates from Google Maps and making sure they appear to the musician as more local first.	Valid

## Test 1 – Fetch current location

I had no issues getting the device's location, and they printed to the console as expected.

```
<+37.33233141,-122.03121860> +/- 5.00m (speed 0.00 mps / course -1.00) @ 10/24/19, 12:40:08 PM British Summer Time
<+37.33233141,-122.03121860> +/- 5.00m (speed 0.00 mps / course -1.00) @ 10/24/19, 12:40:09 PM British Summer Time
<+37.33233141,-122.03121860> +/- 5.00m (speed 0.00 mps / course -1.00) @ 10/24/19, 12:40:11 PM British Summer Time
<+37.33233141,-122.03121860> +/- 5.00m (speed 0.00 mps / course -1.00) @ 10/24/19, 12:40:12 PM British Summer Time
<+37.33233141,-122.03121860> +/- 5.00m (speed 0.00 mps / course -1.00) @ 10/24/19, 12:40:13 PM British Summer Time
<+37.33233141,-122.03121860> +/- 5.00m (speed 0.00 mps / course -1.00) @ 10/24/19, 12:40:14 PM British Summer Time
<+37.33233141,-122.03121860> +/- 5.00m (speed 0.00 mps / course -1.00) @ 10/24/19, 12:40:15 PM British Summer Time
<+37.33233141,-122.03121860> +/- 5.00m (speed 0.00 mps / course -1.00) @ 10/24/19, 12:40:16 PM British Summer Time
```

I decided to make the location accuracy to the nearest 100m to save battery as we do not need the exact value as we are not dealing with a situation such as satellite navigation. I had a bug where leaving the view (say looking at the portfolio) continued to use the location and update coordinates. There is no need for it and is draining battery, therefore, for a better user experience I stopped updating the location when ViewDidDisappear() is called.

```
44     override func viewDidDisappear(_ animated: Bool) {
45         //stop updating when view disappears to conserve battery life
46         locationManager.stopUpdatingLocation()
47     }
```

## Test 2 – Feature is Optional

Some users will not want to share their location, and if they do not accept permission (or do not attach coordinates to the event at creation) then the database is simply updated with 0.00, 0.00 (null island). In most scenarios, the distance to the musician will be too long and so is sufficient to push the gig to the back of the array after the Quicksort. It is good that it is optional and not fully dependant on the location usage, meeting point 9 of the success criteria.

## Test 3 & 4 – Can Calculate Distance and Sort by it

Calculating distance is as simple as using `.distance(from: )` as part of the `CoreLocation` swift framework. To test all distances were calculated correctly, and the Quicksort algorithm does its job, I manually adjusted coordinates in Database.

```

latitude: 52.399565
locationName: "Thetford"
longitude: 0.740887

```

As part of the test I had 6 `GigEvent` objects:

**Testing location (first)** –

Ixworth: (52.294376, 0.839577)

**Testing location (2)** –

Thetford: (52.399565, 0.740887)

**Testing location (3)** –

Watton: (52.559817, 0.833555)

**Testing location (4)** –

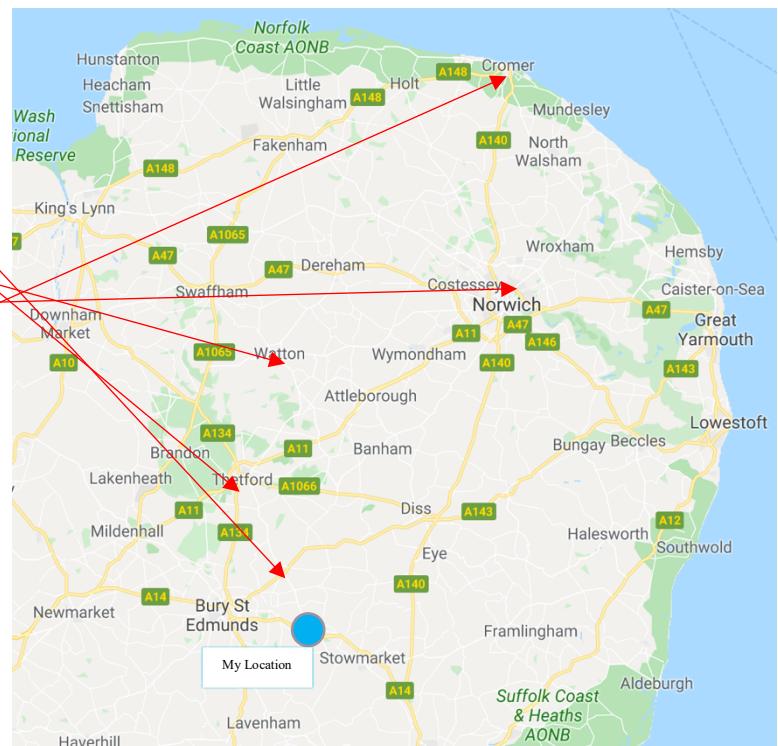
Norwich: (52.653596, 1.290593)

**Testing location (furthest)** –

Cromer: (52.928920, 1.287789)

**Testing location (null island)** –

Null Island in the Gulf of Guinea:  
(0.000000, 0.000000)



The expected order would obviously be **[first, 2, 3, 4, furthest, null island]** from my current location south-east of Bury St Edmunds.

However, I had a failed test where null island appears first in the card stack.

Using a breakpoint and examining the values of variables, I found this was because it was calculating my current location from 0.0, 0.0 so that appeared first, and the rest was in the expected order.

```

123 //Instantiate a location out of coordinates
124 print("\(userLatitude), \(userLongitude)")
125 let userLocation = CLLocation(latitude: userLatitude, longitude: userLongitude)
126
127 for gig in gigs {
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2317
2318
2319
2320
2321
```

After:

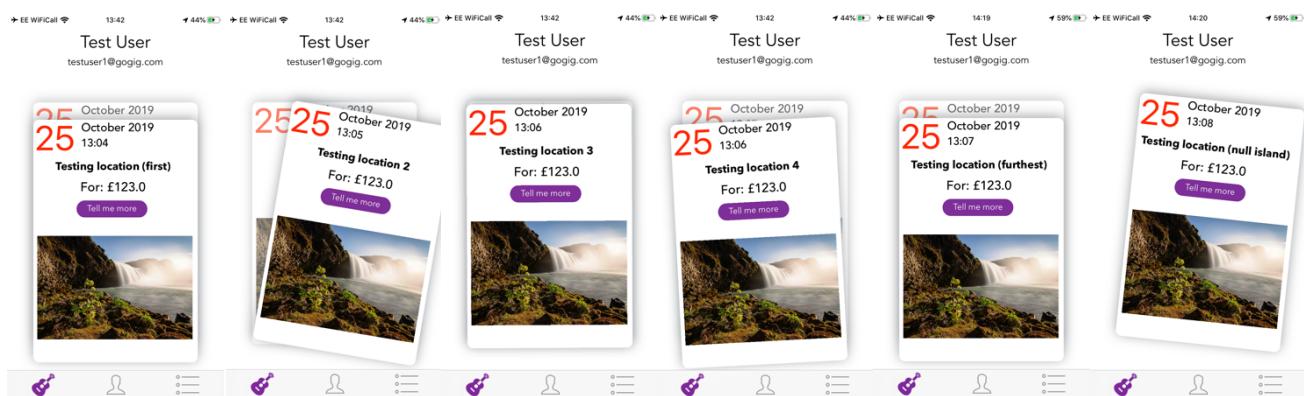
```

89         self.user = returnedUser
90
91         self.locationManager.delegate = self
92         self.locationManager.startUpdatingLocation()
93
94     DataService.instance.getDBEvents(uid: uid) { (returnedGigEvents) in
95         //self.gigEvents = returnedGigEvents
96         self.gigEvents = self.setGigEventDistances(gigs: returnedGigEvents)
97
98         self.updateCards()
99
100    }
101  }
102 }
103
104
105 //MARK: SORT GIGS BY LOCATION
106 var userLatitude = 0.00
107 var userLongitude = 0.00
108 func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
109     //Grab the coordinates
110     let userLocation: CLLocation = locations[0]
111     userLatitude = userLocation.coordinate.latitude
112     userLongitude = userLocation.coordinate.longitude
113 }
114
115 //We compare the two coordinates of the gig and the user
116 //update the distance attribute
117 //then use the distance to do a quick sort
118 func setGigEventDistances(gigs: [GigEvent]) -> [GigEvent] {
119
120     //Instanitate a location out of coordinates
121     print("\(userLatitude), \(userLongitude)") //Prints correct coordinates
122     let userLocation = CLLocation(latitude: userLatitude, longitude: userLongitude)

```

Fix: start updating the location before we grab the array of GigEvent objects from Database

The sorting algorithm worked both times, however it was updating the location that went wrong. It then worked as expected:



## Milestone Review

My main problem I came across was it took me a long time to figure out why null island was appearing first after the sort. I do not believe my solution of fixing it was the best, but it works and the cards sort by locality when location services are accessed. The only drawback I find is that when creating an event as an organiser, the current location of the device may not be at the venue of the performance (but due to time restrictions for this project, I stuck to sorting by this current location) and so if the user decides not to use current location, as it is not relevant, their advert will be one of the last index of the sorted array and appear last to musicians.

I am pleased that I have stuck to keeping it an optional feature though, testing that it works when on or off, so that I meet the success criteria of user privacy.

Reports from end users:

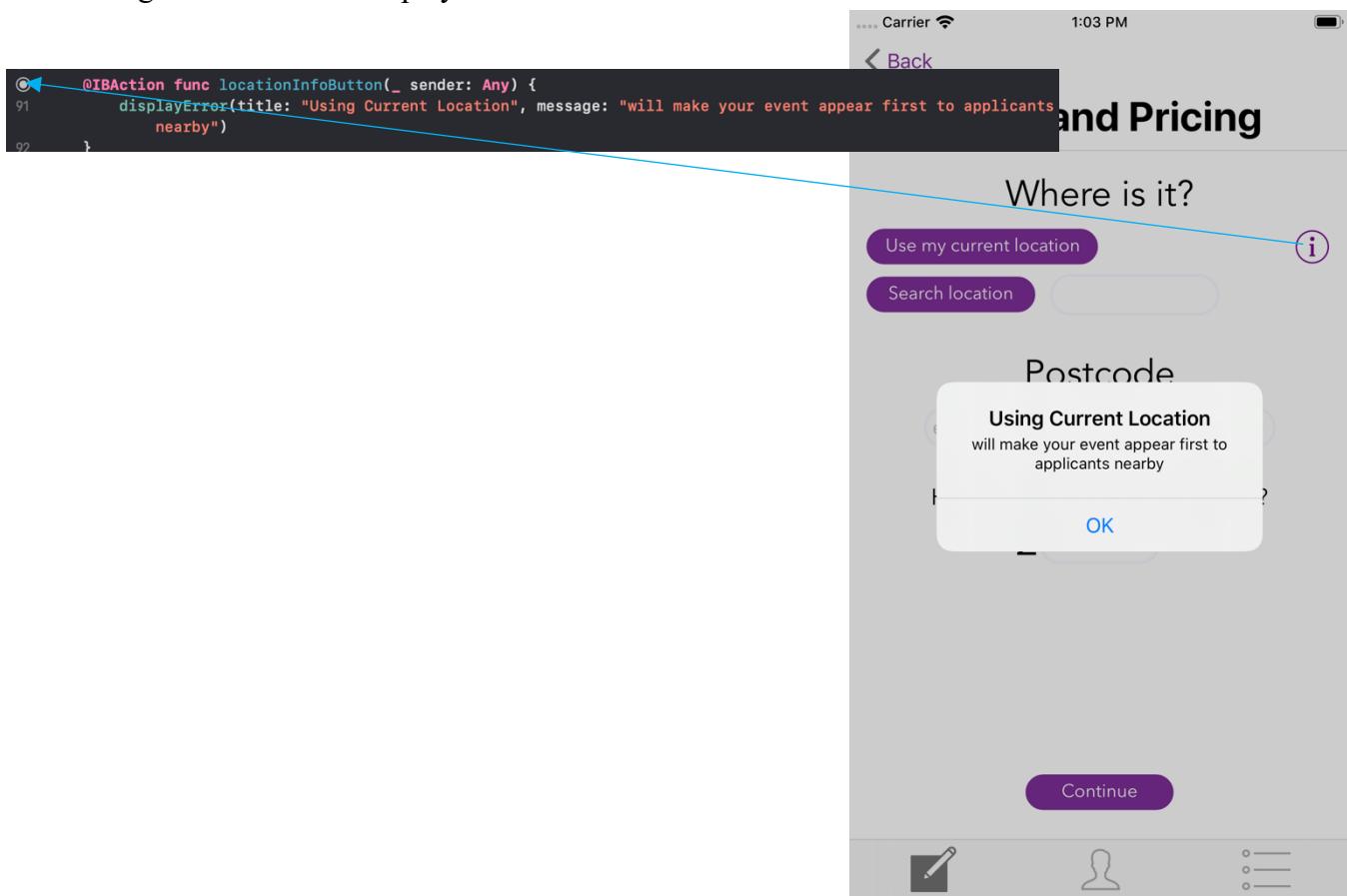
Gavin Thorrold: "This is a handy feature which should hopefully bring in local applicants. My only note is that you may want to explain to my type of user how this feature is beneficial and what difference it makes to them, as it seems to me it only makes sense to people who will apply to my advert."

Jude Mills: "That is really cool, I love how this should help me find gigs closer to home or wherever I am at the time. I also appreciate how it happens under the hood, and you don't even know that the app is arranging them by location which brings an intelligent feel to the program. Everything is laid out nicely, which I think I stated before, although the button on the cards still do not work yet."

## Adjustments from Feedback

Gavin wanted a little note to say why attaching coordinates to an event is beneficial:

Clicking this button will display a UIAlertController



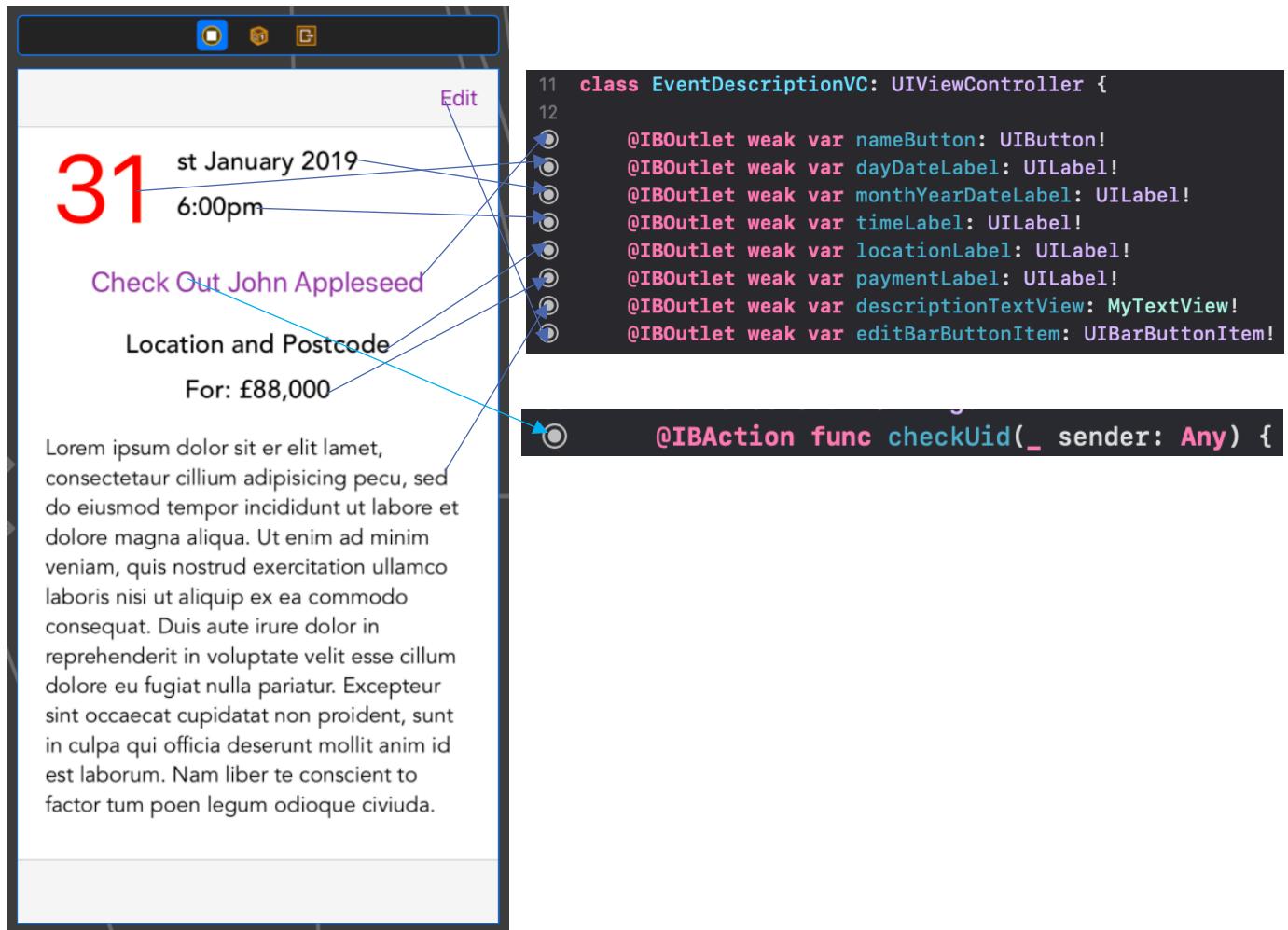
At the end of milestone 5, my end users agreed I met objectives 1, 2, 7, 9 from the success criteria.

## Milestone 6 – Display Event in Detail View and Observe Portfolios

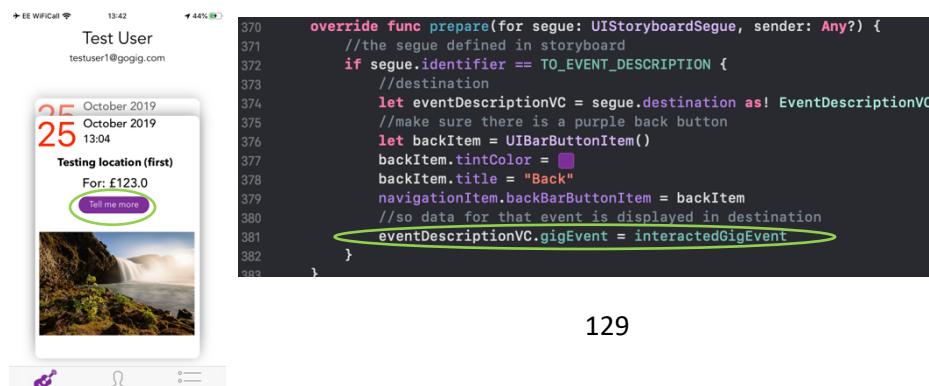
# Development Log

At the moment, not all information about the gig is accessible to the musician. Milestone 6 is all about displaying the event information in full and allowing musicians to look at the potential employer's portfolio.

EventDescriptionVC with connected outlets and actions:



Because all data about the gig is encapsulated in a `GigEvent` object, when the user taps the “Tell me more” button on one of the cards, I just change the value of `gigEvent` in the `EventDescriptionVC` class with `prepareForSegue` method.



And when displaying the data, I just set the values of the outlets with the getter methods of the GigEvent object displaying all the correct data to the user when relevant for criterion 4.

```

47     func refresh() {
48         //set all outlets with information about the gig
49         self.navigationItem.title = gigEvent?.getTitle()
50         nameButton.setTitle("Check out \(gigEvent!.getName())", for: .normal)
51         dayDateLabel.text = gigEvent?.getDayDate()
52         monthYearDateLabel.text = gigEvent?.getLongMonthYearDate()
53         timeLabel.text = gigEvent?.getTime()
54         locationLabel.text = gigEvent!.getLocationName() + " " + gigEvent!.getPostcode()
55         paymentLabel.text = "For: £\(gigEvent!.getPayment())"
56         descriptionTextView.text = gigEvent?.getDescription()

```

The musician can now see the gig in detail before swiping right to apply meeting an informative view outlined in my specification. Now I want to view the portfolio and refresh it to show the person who organised the event to successfully fulfil the purpose of the Portfolio in my specification.

```

59     //look at the portfolio of the event creator
60     var checkUid: String?
61     @IBAction func checkUid(_ sender: Any) {
62         //uid is to refresh portfolio of that user
63         checkUid = gigEvent?.getuid()
64         performSegue(withIdentifier: TO_CHECK_PORTFOLIO_3, sender: nil)

```

When the segue “TO\_CHECK\_PORTFOLIO\_3” is performed, we change the value of uid in the UserAccountVC class and refresh the portfolio to show the posts of the person we are observing.

```

77     override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
78         if segue.identifier == TO_CHECK_PORTFOLIO_3 {
79
80             let userAccountVC = segue.destination as! UserAccountVC
81             //add a back button in place of 'settings' in portfolio view
82             let backItem = UIBarButtonItem()
83             backItem.tintColor = □
84             backItem.title = "Back"
85             navigationItem.backBarButtonItem = backItem
86             //uid needed to refresh portfolio for that user
87             userAccountVC.uid = checkUid!
88             //mark a user is observing that portfolio
89             userAccountVC.observingPortfolio = true
90             //refresh the portfolio ready for the segue
91             userAccountVC.refreshPortfolio()

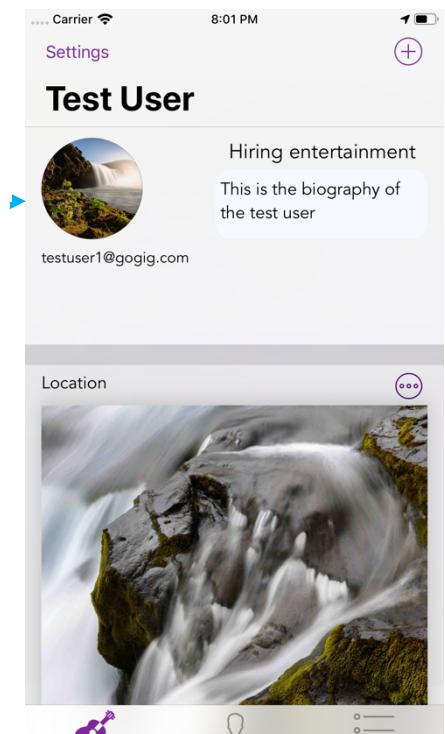
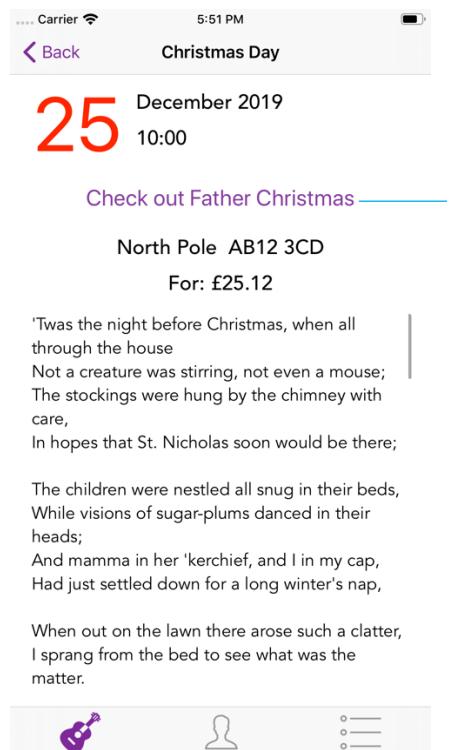
```

When the UserAccountVC loads the view, we are reusing all of my previous code, but simply refreshing it with a new user uid, fetching different data from the database.

```

59     @objc func refreshPortfolio(){
60
61         //user is looking at themself
62         //gate is needed incase user signs in and signs out again
63         if uid == nil || accountGateOpen {
64             accountGateOpen = false
65             uid = Auth.auth().currentUser?.uid
66             //user is looking at another
67         }
68         //get the profile data
69         DataService.instance.getDBUserProfile(uid: uid!) { (returnedUser) in
70             //set the user who owns the portfolio
71             self.user = returnedUser
72             //load profile picture (from cache or download)
73             self.loadImageCache(url: returnedUser.picURL, isImage: true) { (returnedProfileImage) in
74                 self.profilePic = returnedProfileImage
75                 DataService.instance.getDBPortfolioPosts(uid: self.uid!) { (returnedPosts) in
76                     //quick sort the posts by reverse chronological
77                     self.portfolioPosts = returnedPosts
78                     //show profile
79                     self.hideForLoad = false
80                     //show all the data in table view
81                     self.tableView.reloadData()
82                 }
83             }
84         }

```



## Test Plan

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Check navigation to view works	2 – Easy to navigate.	Navigation Button	Program should navigate to new view with the event data ready to be displayed. Should also navigate to the portfolio view with associated user data ready to be displayed.	Valid
2	All event data is displayed correctly	3 – Avoid unexpected results	Firebase Database	The view should display all data about the event to the user in a logical format.	Valid
3	Portfolio view is reused and refreshed to display another user	1 – Making contact between users easy 4 – Users can only edit things associated to their account	Firebase Database	Program should be able to access another user's profile and post data to display in the same the same way as the current user's portfolio. Options to add/delete posts should be hidden.	Valid

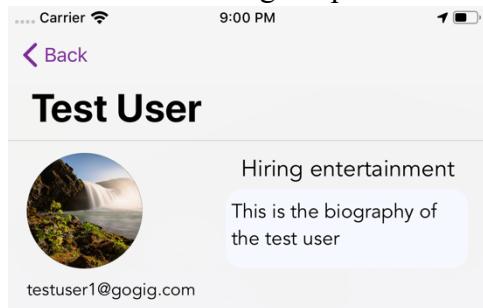
## Test 1 – Navigation

My main bug with navigation was that observing another portfolio did not provide the option to go back to the previous view in the navigation bar, despite providing a back button in EventDescriptionVC class.

```
71         let backItem = UIBarButtonItem()
72         backItem.tintColor = █
73         backItem.title = "Back"
74         navigationItem.backBarButtonItem = backItem
```

view over any programmatic buttons I added. I had to programmatically remove the navigation buttons that were added in the interface builder when refreshing the portfolio.

```
57     @objc func refreshPortfolio(){
58     |
59     print("portfolio refreshed")
60
61     //User is looking at themself
62     //id is needed incase user signs in and signs out again
63     if uid == nil || accountGateOpen {
64         accountGateOpen = false
65         uid = Auth.auth().currentUser?.uid //^
66     //User is looking at another
67     }
68
69     if observingPortfolio {
70         //hide the settings
71         navigationItem.leftBarButtonItem = nil
72         //hide the add button
73         navigationItem.rightBarButtonItem = nil
74     }
}
```



I found that the problem was my UIButtons I had added in the interface builder had a higher priority to appear in the

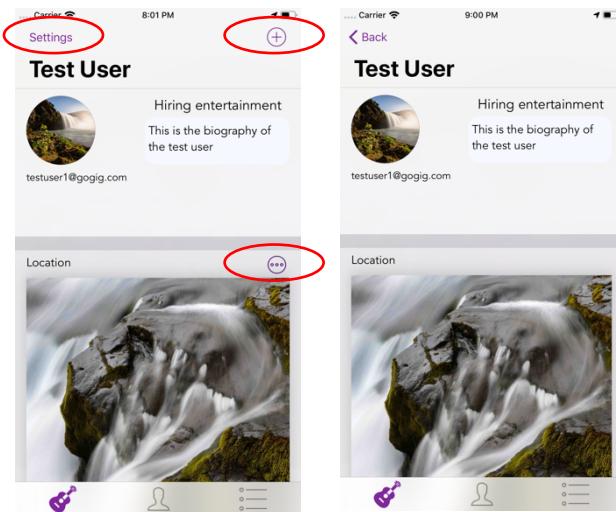
Test 2 – All event data was displayed in EventDescriptionVC as expected without error.

## Test 3 – Portfolio Refreshed to Observe

Test 1 partially fixed my issues for test 3: When observing another user's portfolio, the option to click on 'Settings' and add to the portfolio were still there. This meant that another user could, in theory, sign someone out and add a post to their portfolio without permission. Obviously, this is a horrendous bug; removing the buttons which allow it fixes my navigation bug and this one. However, they also have the ability to click on postMore button and delete someone else's post.

Therefore, I ran a check to make sure that the value of the uid is equal to the current user signed in before displaying the postMore button.

```
226     func updatepostData(cell: AccountPostCell, row: Int) {
227         if uid != Auth.auth().currentUser?.uid {
228             cell.postMoreButton.isHidden = true
229         }
}
```



## Milestone Review

Displaying the event in detail was no problem as part of the sixth milestone but I had to fix a few things to observe a portfolio. Embedding everything in a UITabBarController is beneficial here as well, because a musician can take a look at another portfolio all under the left-most (guitar) tab while still being able to view theirs with the middle tab. Clicking on various tabs resumes the state it was last in.

I believe that I solved the user access bugs effectively and met the criteria in allowing only users to access and edit their **own** information and data. Overall though, this is the first time that the purpose of the portfolio is used, and I personally think it is an effective way to assess a potential client before making any deals with them, meeting point 1 of my success criteria.

My end users reported back:

Gavin Thorrold: “I can now take a look at your test portfolio and the images and videos you have added which is really good (despite the fact they are not posts relating to music).

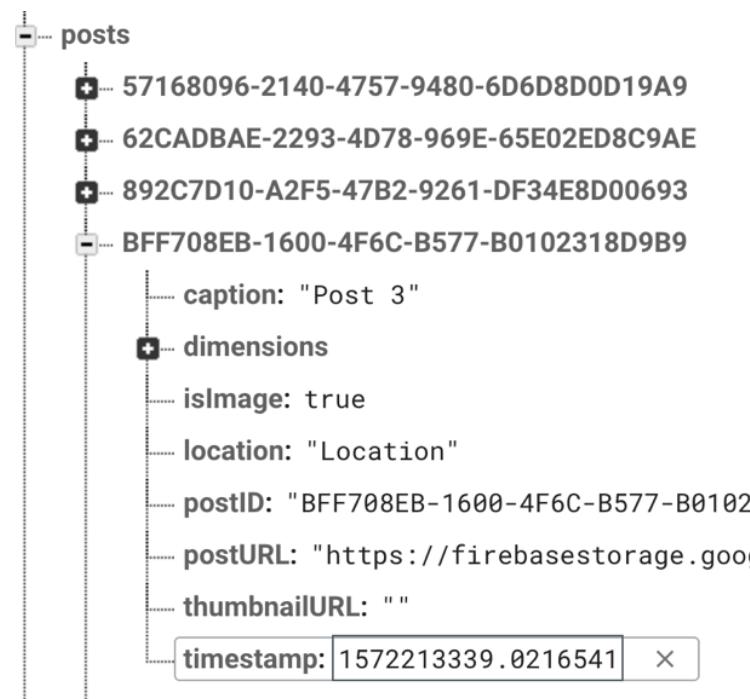
Although I have actually noticed, Lee, that the images I post (and therefore assuming the same for you) do not appear in chronological order that I posted them. I would expect the most recent media to be up the top and scroll down back in time. I did not know if you were aware of this.”

Jude Mills: “Aha, that button finally works! All the information is displayed clearly and looks like an expanded version of the card which I could drag and looks nice. Looking at your portfolio is great as well and will be really useful when meeting with real clients.”

## Adjustments from Feedback

Gavin rightly pointed out that posts do not go in chronological order but are ordered currently by their unique identifier generated by NSUUID().uuidString .

Instead, when uploading all the post data to the database, I also added a **timestamp** to the dictionary generated by NSDate().timeIntervalSince1970 which returns the amount of seconds since Jan 1<sup>st</sup> 1970.



This allows me to Quicksort the PortfolioPost objects by the timestamp property:

```

43     static func < (lhs: PortfolioPost, rhs: PortfolioPost) -> Bool {
44         //inverse so that quick sort of feed shows most recent first
45         return rhs.getTime().compare(lhs.getTime() as Date) == .orderedAscending
46     }
47
48     static func == (lhs: PortfolioPost, rhs: PortfolioPost) -> Bool {
49         return lhs.getTime() == rhs.getTime() || lhs.getTime().compare(rhs.getTime() as Date) == .orderedSame
50     }
51 }
```

Quicksort and then reload the feed to reflect the change:

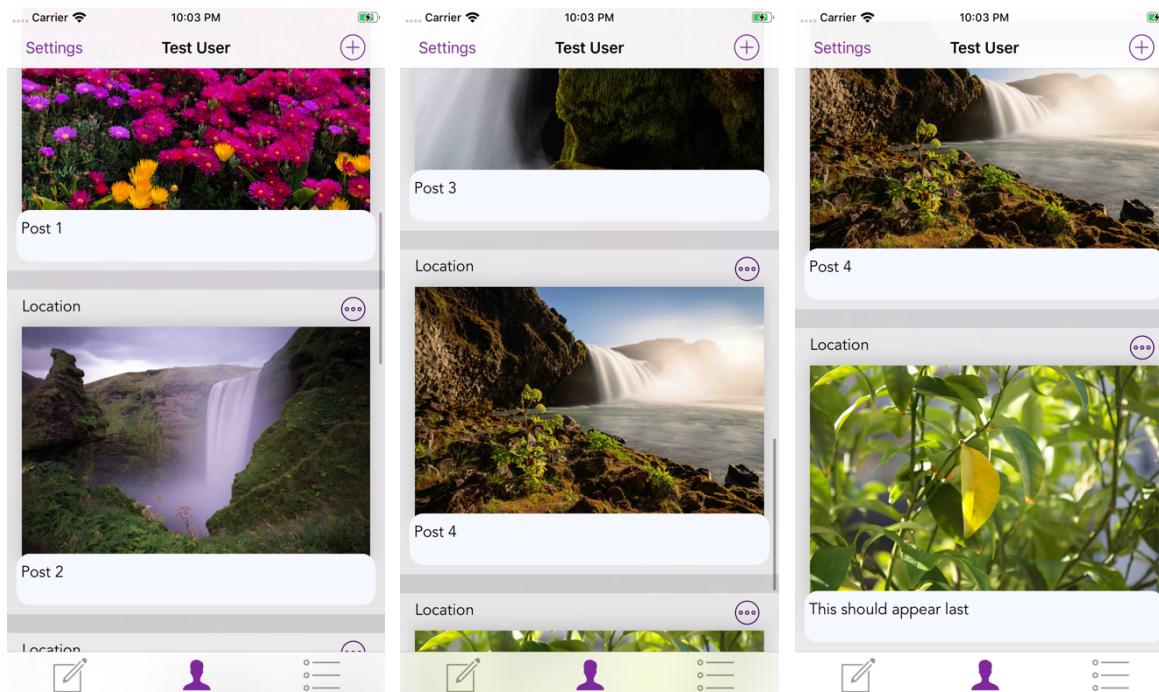
```

77     //get the profile data
78     DataService.instance.getDBUserProfile(uid: uid!) { (returnedUser) in
79         //set the user who owns the portfolio
80         self.user = returnedUser
81         //load profile picture (from cache or download)
82         self.loadImageCache(url: returnedUser.picURL, isImage: true) { (returnedProfileImage) in
83             self.profilePic = returnedProfileImage
84             DataService.instance.getDBPortfolioPosts(uid: self.uid!) { (returnedPosts) in
85                 //quick sort the posts by reverse chronological
86                 self.portfolioPosts = self.quickSort(array:returnedPosts)
87                 //show profile
88                 self.hideForLoad = false
89                 //show all the data in table view
90                 self.tableView.reloadData()
91             }
92         }
93     }

```

I ran a quick test with 5 posts captioned: [“This should appear last”, “Post 4”, “Post 3”, “Post 2”, “Post 1”] uploaded in that order.

The Quicksort worked, displaying them in the feed in reverse chronological order with most recent first:



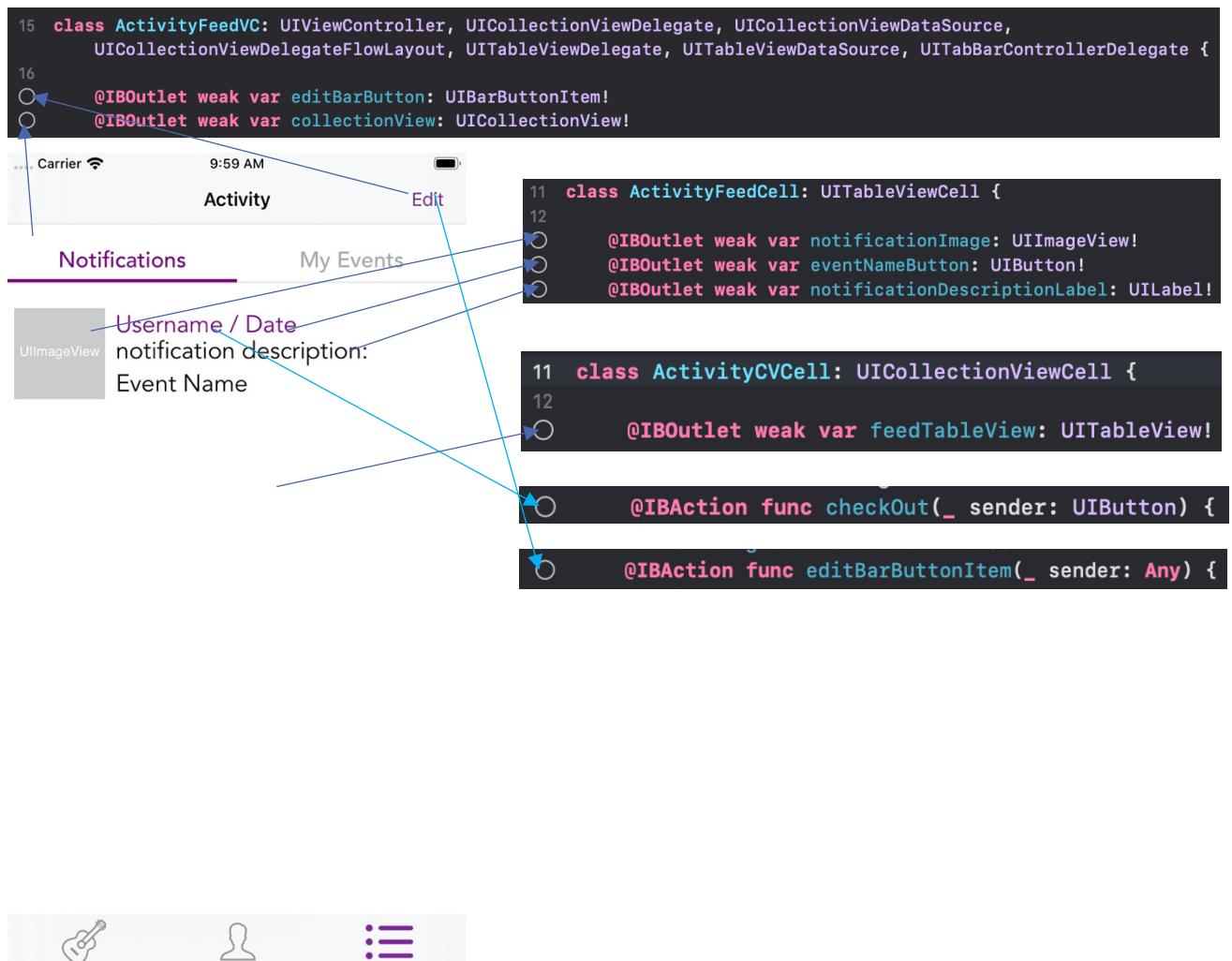
At the end of milestone 6, my end users agreed I met objectives 1, 2, 4 from the success criteria.

## Milestone 7 – Account Notifications View and Send Notifications

# Development Log

So far, an organiser can create an event for a musician to apply for but there are no notifications letting them know that an action has been taken. For this milestone I developed a subclass of `UIViewController` containing a `UICollectionView` allowing me to scroll horizontally between two `UITableViews` in each `UICollectionViewCell`. One `UITableView` would hold activity notifications and updates and the other a record of what events are associated to a user. This ultimately achieves a view for users to reference events related to them and speed up the process of securing a deal as stated in my specification.

ActivityFeedVC , ActivityFeedCVCell , ActivityFeedCell with connected outlets and actions:



To display notifications in the feed, I created an ActivityNotification model class. Like most cases, the instantiated objects will be stored in an array of notifications and displayed in the table view. These objects will be sorted by a time attribute (similar to PortfolioPosts) so they appear in reverse chronological order and you scroll back in time meaning notifications are easy to navigate through with more recent first completing success criterion 2.

```

1  class ActivityNotification: Comparable {
2
3      private var id: String           //unique id of notification
4      private var relatedEventId: String //event notification concerns
5      private var type: String         //personal, applied or reply
6      private var senderUid: String    //user id of notification sender
7      private var recieverUid: String  //user id of notification receiver
8      private var senderName: String   //senders account name
9      private var picURL: URL         //notification image download URL
10     private var description: String //contents of the notification
11     private var time: NSDate        //to sort in reverse chronological order
12
13     //instantiate a ActivityNotification object
14     init(id: String, relatedEventId: String, type: String, senderUid: String, recieverUid: String, senderName:
15           String, picURL: URL, description: String, time: NSDate) {
16         self.id = id
17         self.relatedEventId = relatedEventId
18         self.type = type
19         self.senderUid = senderUid
20         self.recieverUid = recieverUid
21         self.senderName = senderName
22         self.picURL = picURL
23         self.description = description
24         self.time = time
25     }
26
27 }
```

There will be three cases when notification(s) are uploaded to the database:

1. **Organiser just created an event – Personal Notification**
2. **Musician applies to an event – Personal and Shared Notification**
3. **Musician is hired or rejected to play at an event – Personal and Shared Notification**

### Notification Case 1: Event Created

```

157 func updateActivity() {
158     let notificationID = NSUUID().uuidString
159     let senderUid = user!.uid
160     //receiver is also the sender for a personal notification
161     let recieverUid = senderUid
162     let senderName = "You"
163     //Users profile picture
164     let notificationPicURL = user!.picURL.absoluteString
165     let notificationDescription = "Created the event: \((eventData!["title"])!)"
166     //For Quicksort
167     let timestamp = NSDate().timeIntervalSince1970
168     notificationData = ["notificationID": notificationID, "relatedEventID": eventID, "type": "personal",
169     "sender": senderUid, "reciever": recieverUid, "senderName": senderName, "picURL": notificationPicURL,
170     "description": notificationDescription, "timestamp": timestamp]
171 }
```

### Notification Case 2: Musician Applies

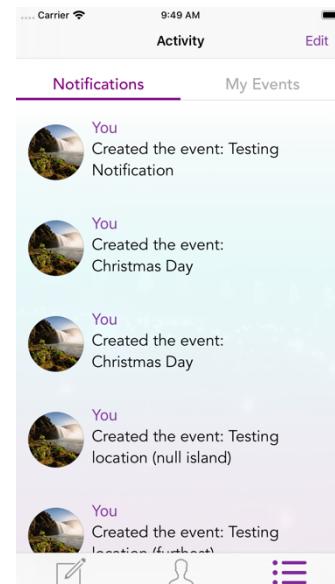
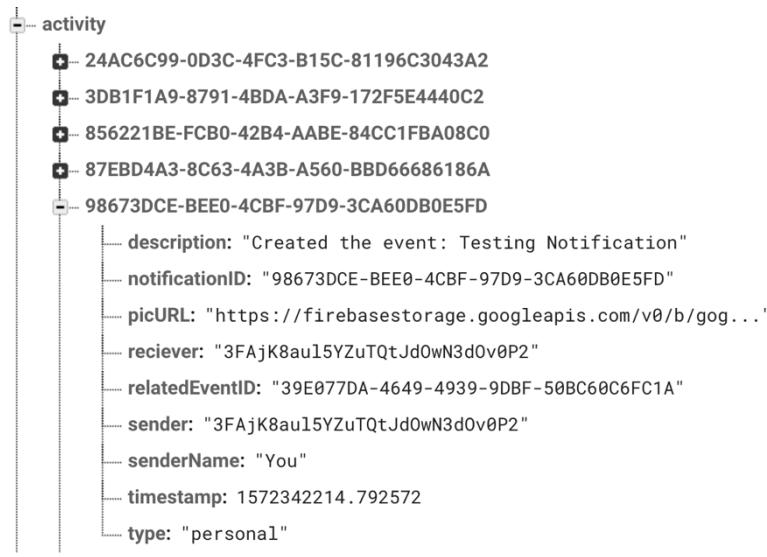
```

266 func updateActivity() {
267     let notificationID = NSUUID().uuidString
268     let relatedEventID = interactedGigEvent!.getid()
269     let senderUid = user!.uid
270     //Reciever is user that created gig
271     let recieverUid = interactedGigEvent!.getuid()
272     let senderName = user!.name
273     let notificationPicURL = user!.picURL.absoluteString
274     let notificationDescription = "applied for the event: \(interactedGigEvent!.getTitle())"
275     let timestamp = NSDate().timeIntervalSince1970
276     notificationData = ["notificationID": notificationID, "relatedEventID": relatedEventID, "type": "applied",
277                         "sender": senderUid, "reciever": recieverUid, "senderName": senderName, "picURL": notificationPicURL,
278                         "description": notificationDescription, "timestamp": timestamp]
279
280     //Notify Other User
281     DataService.instance.updateDBActivityFeed(uid: recieverUid, notificationID: notificationID,
282                                              notificationData: notificationData!) { (complete) in
283         if complete {
284             //Notify Current User about their action (sender is themself to reciever themself)
285             self.notificationData!["senderName"] = "You"
286             self.notificationData!["reciever"] = senderUid
287             self.notificationData!["type"] = "personal"
288             DataService.instance.updateDBActivityFeed(uid: senderUid, notificationID: notificationID,
289                                              notificationData: self.notificationData!) { (complete) in
290             }
291         }
292     }
293 }

```

### (Notification Case 3 takes place as part of milestone 8)

Updating the database with the notificationData dictionary:



Initially I displayed all the objects under the 'Notifications' menu bar (UICollectionViewCell) just by grabbing the activity from Database, instantiating objects and appending them in an array to display in the feedTableView (same approach as portfolio). However, there are a few things that are different when displaying notifications in this feed:

1. Notifications need to update whenever something new happens, can't wait on manual reload.
2. Downloading all notifications is unnecessary usage of mobile data downloads so we want to load more as the user scrolls (pagination).

Firebase has its own listener .observe() which will run the closure anytime there is a change to Database at the specified location. This meets a better user experience (point 1) reducing mobile data usage.

```

486     func observeDBActivityFeed(uid: String, handler: @escaping (_ events: ActivityNotification) -> ()) {
487         //Will run this closure anytime a .childAdded to Database
488         activityHandle = REF_USERS.child(uid).child("activity").observe(.childAdded, with: { (snapshot) in
489
490             //Grab an array of all notifications in the database
491             if let activityData = snapshot.value as? NSDictionary {
492
493                 if let notificationID = activityData["notificationID"] as? String {

```

I had a lot of problems trying to get pagination to work. Eventually I found that I could not do a Quicksort locally by timestamp because I had to sort the objects as I was querying the database because I was only getting 10 at a time on scroll. queryOrdered(byChild: "timestamp") puts the notifications in the correct order.

```

419     func getDBActivityFeed(uid: String, currentActivity: [ActivityNotification], handler: @escaping (_ events: [ActivityNotification]) -> ()) {
420         let lastActivity = currentActivity.last
421         var queryRef: DatabaseQuery
422
423         if lastActivity == nil || paginationGateOpen {
424             //Needed as doesn't refresh properly on sign in and sign out, fetches incorrect starting 10
425             paginationGateOpen = false
426             //Fetch first 10 if the initial query
427             queryRef = REF_USERS.child(uid).child("activity").queryOrdered(byChild:
428                 "timestamp").queryLimited(toLast: 10)
429         } else {
430             let lastTimestamp = lastActivity?.getTime().timeIntervalSince1970
431             //fetch another 10 starting at the last one in the array, progressing another 10
432             queryRef = REF_USERS.child(uid).child("activity").queryOrdered(byChild:
433                 "timestamp").queryEnding(atValue: lastTimestamp).queryLimited(toLast: 10)
434         }
435
436         //This contents of this array is appended when returned to display in table
437         var activityNotifications = [ActivityNotification]()
438
439         queryRef.observeSingleEvent(of: .value, with: { (snapshot) in
440
441             //Grab an array of all posts in the database
442             if let snapshot = snapshot.children.allObjects as? [DataSnapshot] {
443
444                 //Loop through them and grab data for instantiation
445                 for snap in snapshot {

```

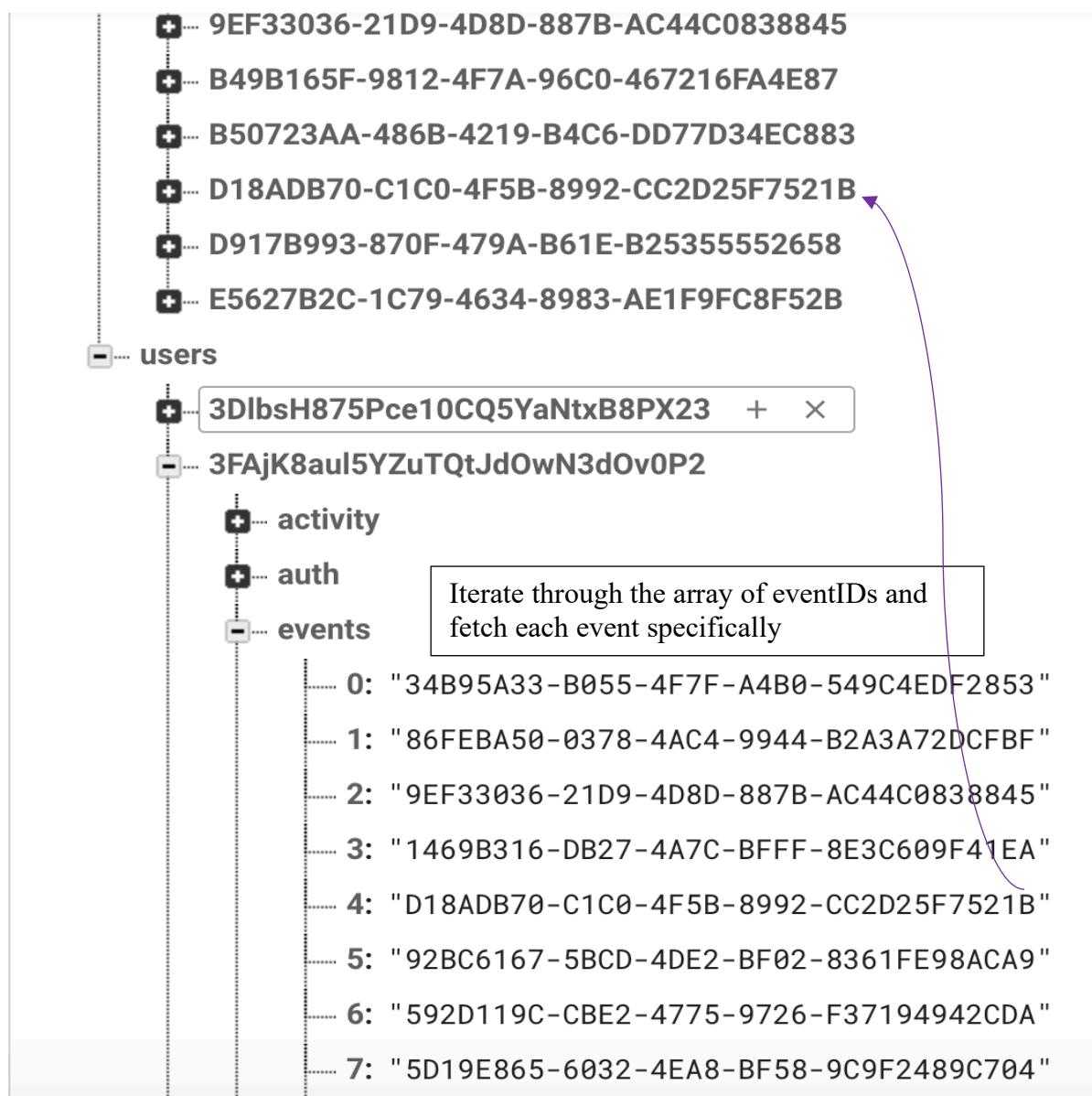
The pagination method, and image caching for the reused feedTableView cells, saves data usage for a better user experience.

For the “My Events” UICollectionViewCell I took a different approach:

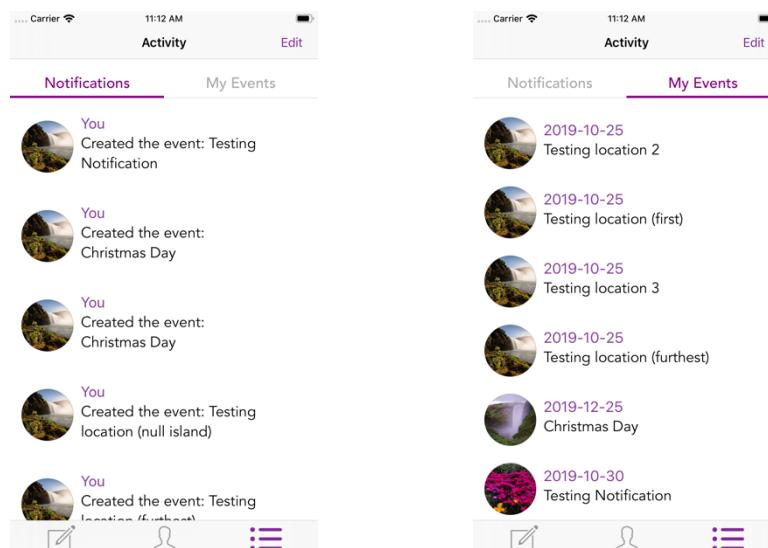
```

104         //Get and Observe the User Event Recordings
105         eventsHandle = DataService.instance.REF_USERS.child(uid).child("events").observe(.value) {
106             (snapshot) in
107             DataService.instance.getDBUserEvents(uid: uid) { (returnedEventIDs) in
108
109                 //Iterate through the list of eventIDs associated to user, get each one from public events
110                 //And insert at 0 of local array
111                 for eventID in returnedEventIDs {
112                     DataService.instance.getDBSingleEvent(uid: uid, eventID: eventID) { (returnedGigEvent,
113                         success) in
114                         if success {
115                             eventListings.insert(returnedGigEvent, at: 0)
116                             self.usersEvents = eventListings
117                             self.collectionView.reloadData()
118                             self.attemptReload()
119
120                         }
121                 }

```



Both UICollectionViewCells:



## Test Plan

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure notification object is created and added to database	5 – Notifications and activity updates		Whenever an action is taken which involves another user (musician applied or organiser responded to application), app should create a notification object associate to the user to store in the database.	Valid
2	Check notifications are added to a scrolling table view feed	1 – Good user experience and user interface.	Firebase Database	Notification objects (and their data) should be displayed in a scrolling table view feed. A new notification should be added automatically without the user having to manually refresh the feed.	Valid
3	Check events associated with that user are displayed in 'My Events' scrolling table view feed.	3 – Check validation rules to get expected results 4 – User can review anything associated to their account	Firebase Database	Events associated to a user (and their data) should be displayed in a scrolling table view feed. An event should be added automatically to this section when an organiser has created a listing and when a musician is confirmed to play for that event.	Valid

## Test 1 – Notification Object Added to Database

As shown in my development log, I had no trouble uploading data to Database...

## Test 2 – Check Notifications are Added to Feed

...but I did have issues with the Firebase `.observe` closure which is executed whenever the database changes. My first bug was when an organiser finishes creating an event as part of `PhotoCGVC` class, they are taken to `ActivityFeedVC` to see the notification of the event they have created. When testing, this notification is not added to the feed.

Before:

```
128         self.updateActivity()
129         DataService.instance.updateDBActivityFeed(uid: self.notificationData!["reciever"] as! String,
130             notificationID: self.notificationData!["notificationID"] as! String, notificationData:
131             self.notificationData!)|
132             self.tabBarController?.selectedIndex = 2
133
134         //clear the event creation and pop to root of the navigation stack
135         self.navigationController?.popToRootViewController(animated: true)
136     }
```

I had to use a completion handler because I was fetching notifications before it was uploaded.

After:

```

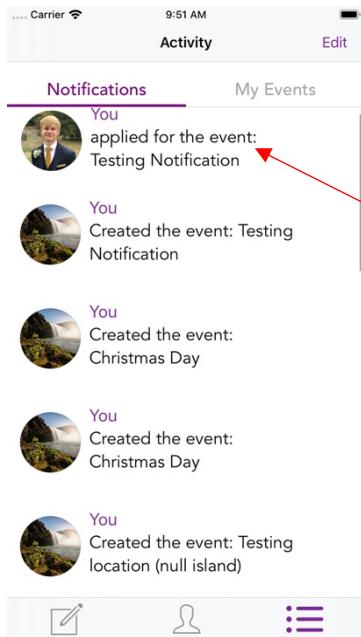
128         self.updateActivity()

129
130         //Update the activity feed in a completion handler so it updates correctly
131         DataService.instance.updateDBActivityFeed(uid: self.notificationData!["reciever"] as! String,
132             notificationID: self.notificationData!["notificationID"] as! String, notificationData:
133             self.notificationData!) { (complete) in
134
135             if complete {
136                 self.removeSpinnerView(self.loadingSpinner)
137                 //Take user to Activity tab to see their posted event
138                 //Without completion handler we were jumping to the view controller before the activity
139                 //had updated
140                 self.tabBarController?.selectedIndex = 2
141
142                 //clear the event creation and pop to root of the navigation stack
143                 self.navigationController?.popToRootViewControllerAnimated(true)
144             }
145         }

```

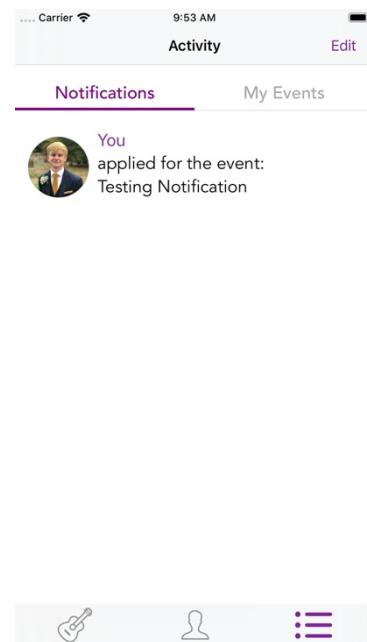
A big problem I had was to do with the authentication system. When logging out, my code still observes notifications being added to Database. This meant that when logged into another account, after signing out, users received notifications from the previous logged in account. To test this, I had to have two simulators open simultaneously.

Simulator 1: Initially logged into Lee (musician) and then signed into Test User (organiser)



After Lee applies for an event (swipes card right), a **personal** notification updates when logged in with Test User. This is because we never stopped observing Database updates from that location, so feed will continue to update with incorrect notifications which do not correspond to that account.

Simulator 2: Always logged into Lee (musician)



I discovered that this is actually also the case for the portfolio view and the event listings under 'My Events' as it will show incorrect updates when logging out and then into another account. The solution is to remove these observers when requesting to log out by using a handle global variable which holds the value of the observer closure.

```

112         alertController.addAction(UIAlertAction(title: "Log out", style: .destructive, handler: {
113             (buttonPressed) in
114             do {
115                 if let uid = Auth.auth().currentUser?.uid {
116                     //So does not update account which is not theirs
117                     DataService.instance.removeObservers(uid: uid)

```

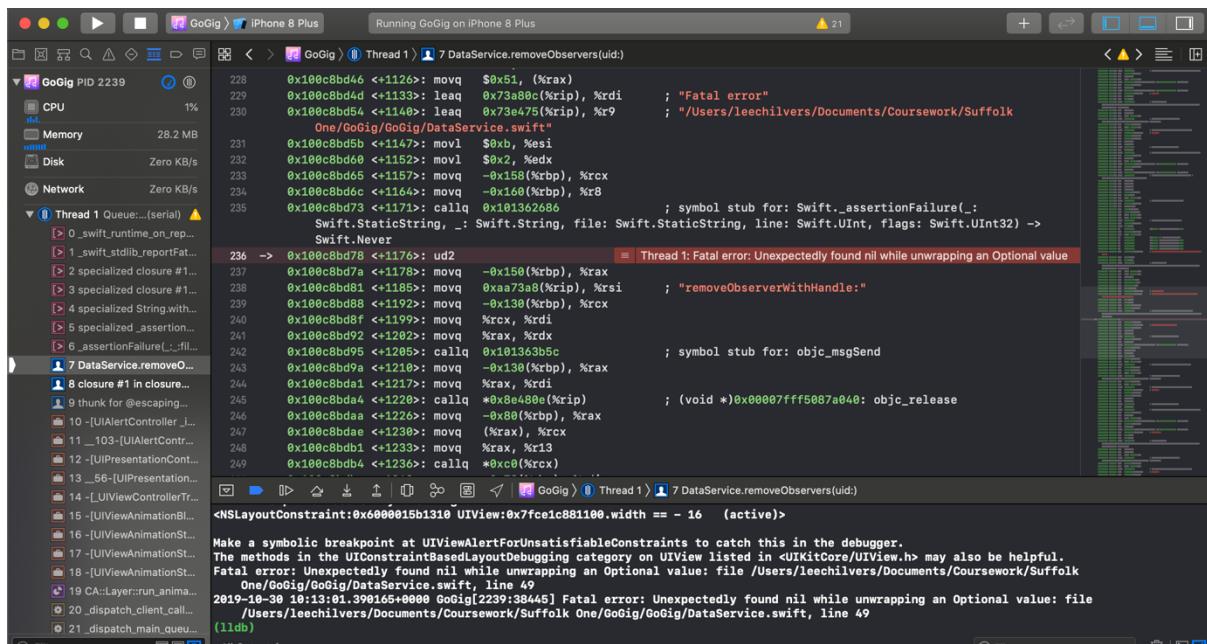
DataService:

```

47     func removeObservers(uid: String) { //so correct data is displayed for account
48         REF_USERS.child(uid).child("posts").removeObserver(withHandle: postsHandle!)
49         REF_USERS.child(uid).child("events").removeObserver(withHandle: eventsHandle!)
50         REF_USERS.child(uid).child("activity").removeObserver(withHandle: activityHandle!)
51     }

```

But this then threw a crash:



Looking at the state of variables at this crash:

```

V activityHandle = (DatabaseHandle?) 0
▶ V DB_BASE = (DatabaseReference) 0x0000600003818d80
V eventsHandle = (DatabaseHandle?) nil
▶ V instance = (GoGig.DataService) 0x00006000038186f0
V postsHandle = (DatabaseHandle?) 3

```

It was because when logging out it found nil for eventsHandle and activityHandle so it was trying to remove something that has no value. postsHandle always had a value because it is the initial view, after launch a user may not click on the ActivityFeedVC tab before logging out. After:

```

47     func removeObservers(uid: String) {
48         //Portfolio is the initial view so there will always be an observer
49         REF_USERS.child(uid).child("posts").removeObserver(withHandle: postsHandle!)
50         //May not be an observer as user may not have clicked on those tabs since launch
51         //Store the closure under a global variable (handle) and only remove it if it has a value
52         //is observing
53         if eventsHandle != nil && activityHandle != nil {
54             REF_USERS.child(uid).child("events").removeObserver(withHandle: eventsHandle!)
55             REF_USERS.child(uid).child("activity").removeObserver(withHandle: activityHandle!)
56         }
57     }

```

## Test 3 – Events Displayed in ‘My Events’

One logic error I had was that for a musician, events they had applied for were appearing in ‘My Events’. This is incorrect as it makes more sense to display listings in ‘My Events’ only when a musician has been asked to play for that particular gig. Therefore, at this stage, only organisers should have anything appear in their ‘My Events’ collection view cell (the events they have created) because I have not yet developed the view to accept applicants.

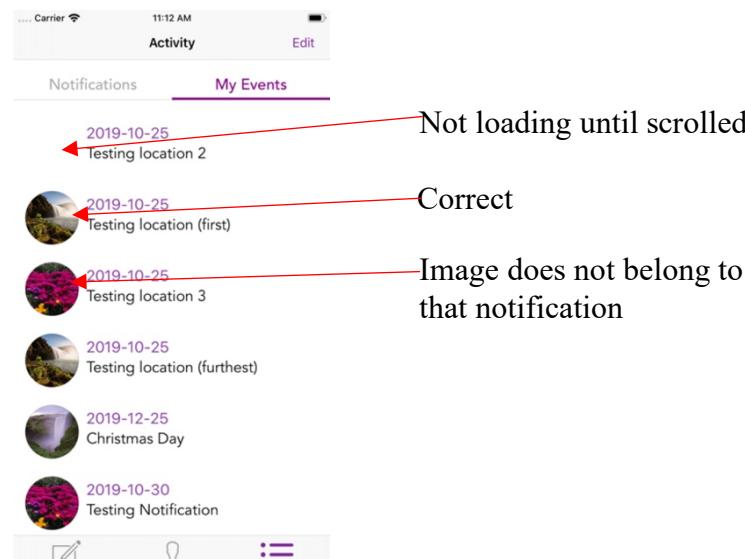
In FindGigVC before:

```
249     if applied {
250
251         DataService.instance.updateDBUserEvents(uid: user!.uid, eventID: interactedGigEvent!.getid())
252
253         updateActivity()
254     }
```

After:

```
249     if applied {
250
251         //Add the event under user to the database
252         //THIS ISNT RIGHT
253         //This should only happen when an organiser accepts a users application
254         //DataService.instance.updateDBUserEvents(uid: user!.uid, eventID: interactedGigEvent!.getid())
255
256         updateActivity()
257     }
```

Another issue I had was that images in the notification cell were appearing weirdly:



I think the problem is that because the cell gets reused, the cell flickers briefly either not displaying an image or displaying the wrong image until the feed is scrolled. The best fix I could achieve was to reload the whole table after a very short delay of 0.1 seconds:

```
136     //Needed so that the user profile pictures don't flash and display the wrong image
137     var timer: Timer?
138
139     func attemptReload() {
140         //Stop the timer
141         self.timer?.invalidate()
142
143         //Reload the collection view and all its data 0.1 seconds after timer has started
144         self.timer = Timer.scheduledTimer(timeInterval: 0.1, target: self, selector: #selector(handleReload),
145                                         userInfo: nil, repeats: false) //Doesn't repeat
146     }
147
148     @objc func handleReload(){
149         self.collectionView.reloadData()
150     }
```

What it fixed was the notificationImages refreshing correctly without scrolling, however before it refreshes there is still visible error. The feed in use works as expected. and this is

only the case when the app first launches and the view is first loaded, resuming the view does not show these image errors. I have eliminated confusion here meeting success criterion 3.

## Milestone Review

This milestone was the most challenging so far because there was a lot to consider and complete. I had to test with two simulators simultaneously to make sure that notifications were updated in real time when one simulator performed an action which involved another user. I came across the authentication bug, but solved it effectively using the debugging tools and I think I have fixed it so that notifications related to that account only will appear. This meets the success criteria that data is secure and can only be accessed under that account.

Pagination was a good technique to learn as it enhances the user experience. It was difficult to develop, and I had to make the decision on how many notification objects to initially fetch and how many to increase it each time on scroll. I found that the best combination for performance was to initially fetch 10 and increase by 10 when scrolling. Not only did I have to create this, but I pushed for a nice user experience so that it loads 1 cell in advance while scrolling. I struggled to find a solution to fixing the smoothness of this reload when in a location of weak internet connection. It does work, however with 'No Service' the scroll judders slightly while it tries to fetch more notifications.

I also had great difficulty with the images in the table view cells. Because cells are reused to enhance the performance of the user interface component, there is a noticeable flickering of the notificationImage in the cell until the feed is scrolled. I think I provided a way to improve this slightly, but it is still not perfect and not completely how I want it.

Despite this, I have met the success criteria with easy navigation (large menu bar), testing that data is only accessible under the associated account and receiving notifications helping make contact between musicians and event organisers.

My end users tested this tab:

Gavin Thorrold: "I like this view because it looks really professional due to how the bar at the top animates when I slide between the two sections. It records all the events I have created however I cannot see all the information I added about them. It would be good if I could see everything a musician sees about the event. So far, I like how it tells me when someone has applied to my event and tells me what I have done myself."

Jude Mills: "At the moment I only see updates about myself applying for a gig, but I assume that I would receive messages from other people later. I really like the layout of this view and it follows the design of other popular apps like Twitter. The edit button at the top right corner of the screen doesn't do anything yet either."

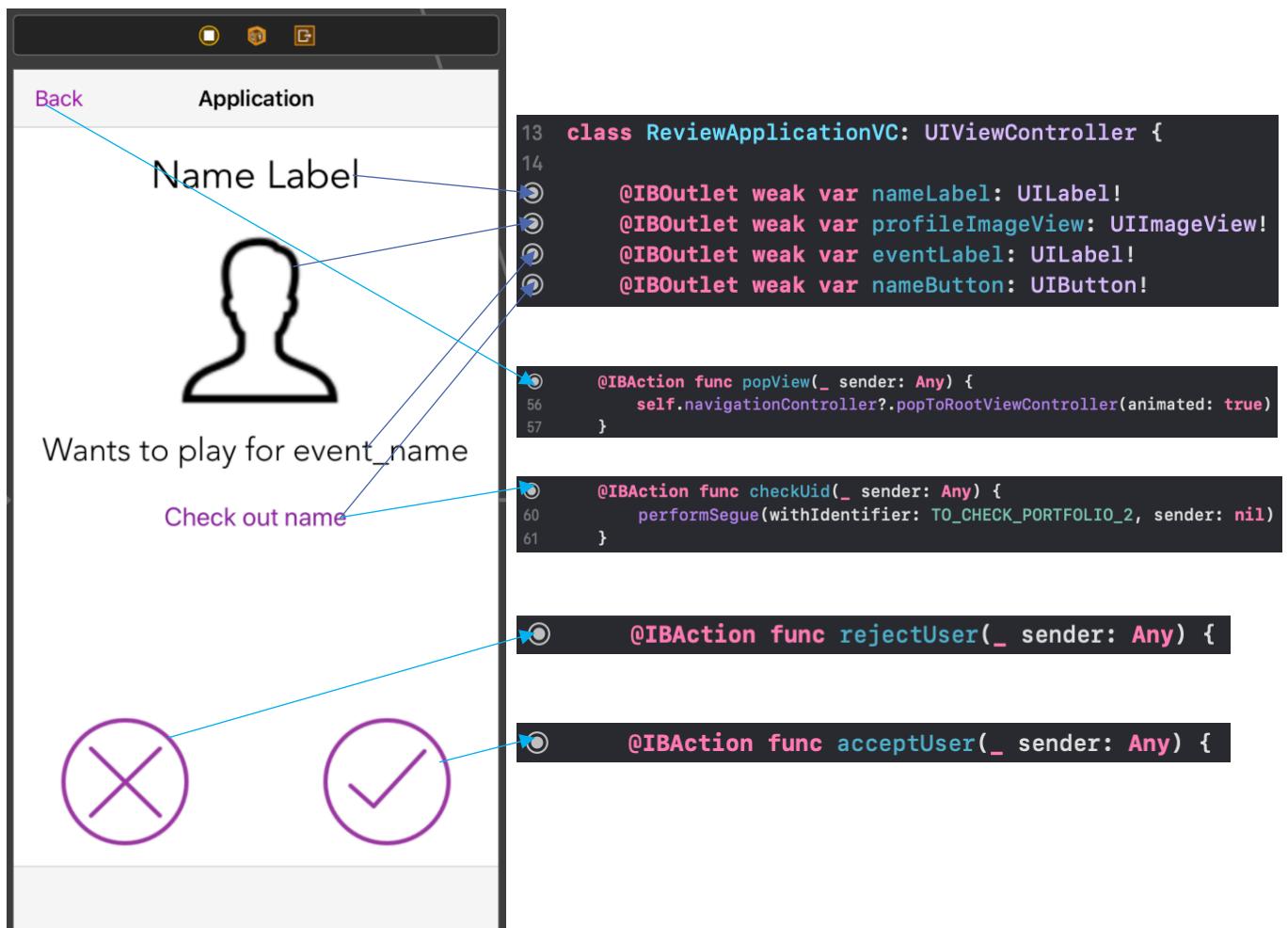
At the end of milestone 7, my end users agreed I met objectives 1, 2, 5, 6 from the success criteria.

## Milestone 8 – Review Application View (for organisers)

### Development Log

Milestone 8 is all about the event's organiser responding (and therefore sending notifications back) to musicians completing the whole purpose of the smartphone app in securing a deal (as detailed in my specification of View to Review Activity and success criterion 1).

ReviewApplicationVC with connected outlets and actions:



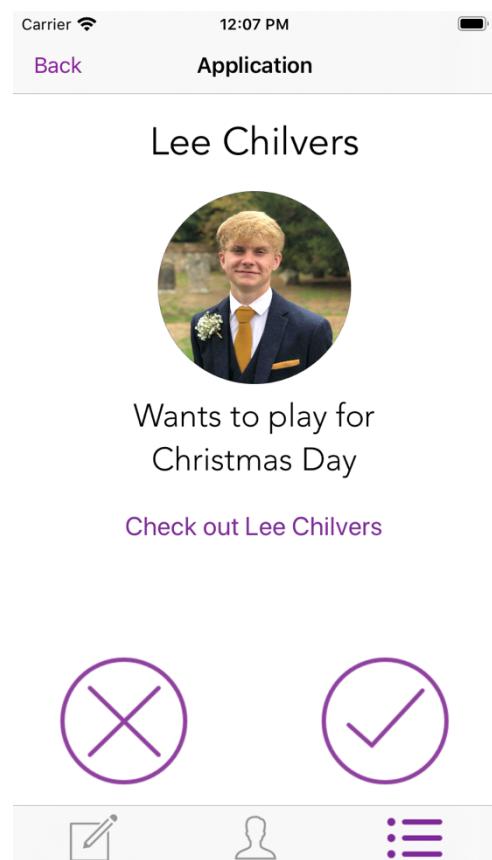
Any user can click on the eventNameButton on the ActivityFeedCell to perform a segue to observe a portfolio as shown in milestone 6 as part of ActivityFeedVC:

```

252     var checkUid: String?
253     @IBAction func checkOut(_ sender: UIButton) {
254         //Get the row of the cell with a tag
255         let row = sender.tag
256         //and get the uid of user that sent the notification
257         checkUid = activityNotifications[row].getSenderId()
258         //perform segue to observe portfolio refreshed with this uid
259         performSegue(withIdentifier: TO_CHECK_PORTFOLIO, sender: nil)

```

After clicking the cell to go to ReviewApplicationVC, the view is refreshed as so:



As you can see the correct data of the user and the event is being displayed (meeting success criterion 4) as well as providing a link to their portfolio. By presenting the portfolio to reference and the correct user information, the specification has been followed for musicians to promote their work in helping them secure a deal.

Accepting or rejecting the application will send a notification.

### Notification Case 3: Musician is hired or rejected

```

82 func updateActivity(accepted: Bool) {
83     //set all the data for the notification
84     let notificationID = NSUUID().uuidString
85     guard let senderUid = currentUser?.uid else { return }
86     guard let receiverUid = uid else { return }
87     guard let senderName = currentUser?.name else { return }
88     guard let notificationPicURL = currentUser?.picURL.absoluteString else { return }
89     guard let relatedEventTitle = relatedEvent?.getTitle() else { return }
90     guard let relatedEventID = relatedEvent?.getId() else { return }
91     //grab the event as well
92     var notificationDescription: String?
93     if accepted {
94         notificationDescription = "hired you for the event: \(relatedEventTitle)" //message if accepted
95     } else {
96         notificationDescription = "declined you for the event: \(relatedEventTitle)" //message if declined
97     }
98     let timestamp = NSDate().timeIntervalSince1970 //for quicksort in feed
99     notificationData = ["notificationID": notificationID, "relatedEventID": relatedEventID, "type": "reply", "sender": senderUid, "receiver": receiverUid, "senderName": senderName, "picURL": notificationPicURL, "description": notificationDescription!, "timestamp": timestamp]
00     //notify musician
01     DataService.instance.updateDBActivityFeed(uid: receiverUid, notificationID: notificationID, notificationData: notificationData!) { (complete) in
02         if complete && accepted {
03             //notify Current User about their action (sender is themself to receive themself)
04             self.notificationData!["senderName"] = "You"
05             self.notificationData!["receiver"] = senderUid
06             self.notificationData!["type"] = "personal"
07             self.notificationData!["description"] = "hired \(self.user!.name) for the event: \(relatedEventTitle)"
08             //update database with this notification
09             DataService.instance.updateDBActivityFeed(uid: senderUid, notificationID: notificationID, notificationData: self.notificationData!) { (complete) in
10                 if complete {
11                     }
12                 }
13             }
14         }
15         //allow interaction again
16         self.view.isUserInteractionEnabled = true

```

Clicking nameButton in this view also performs a segue to observe the portfolio of the musician.

### Test Plan

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure that only organisers can navigate to this view	2 – Easy navigation 4 – User can edit anything associated to their account only	Table view feed cell	App should check the user is an organiser and the notification clicked on is a job application to navigate to this view.	Valid

2	Portfolio view is reused and refreshed to display another user.	1 – Making contact between users easy 4 – Users can only edit things associated to their account	Firebase Database	To review the event applicant, the program should refresh the portfolio view again to display the musician's work. Options to add/delete posts should be hidden.	Valid
3	Check a notification is sent back to musician confirming that they are playing for that event.	5 – Receive notifications and activity updates	Button	When the organiser accepts the musician's application, by clicking a button, a notification object should be added to the database to inform the user they got the job. This in turn must add the event listing to the musicians 'My Events' feed.	Valid

## Test 1 – Only Organisers can Navigate to the View

At the moment any notification clicked on would take you to the review application view. I needed to make sure that the notification was of the type 'application' and the current user was an organiser.

Was an

```
104     func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
105
106         //activity notifications section
107         if tableView.tag == 0 {
108             let selectedNotification = activityNotifications[indexPath.row] ⚠️ Initial
109             //So data is transferred when segue is performed
110             checkUid = activityNotifications[indexPath.row].getSenderId()
111             selectedApplication = activityNotifications[indexPath.row]
112             performSegue(withIdentifier: TO REVIEW APPLICATION, sender: nil)
```

After:

```
104 func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
105
106     //activity notifications section
107     if tableView.tag == 0 {
108         let selectedNotification = activityNotifications[indexPath.row]
109         //Making sure only an organiser can navigate to review application
110         if user!.gigs == false && selectedNotification.getType() != "personal" && selectedNotification.getType()
111             == "applied" {
112
113             //So data is transferred when segue is performed
114             checkUid = activityNotifications[indexPath.row].getSenderId()
115             selectedApplication = activityNotifications[indexPath.row]
116             performSegue(withIdentifier: TO REVIEW APPLICATION, sender: nil)
117         }
118     }
119 }
```

## Test 2 – Refreshing Portfolio to Show Applicant

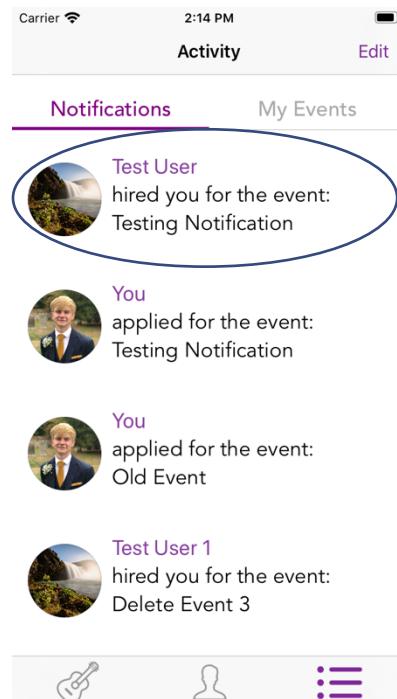
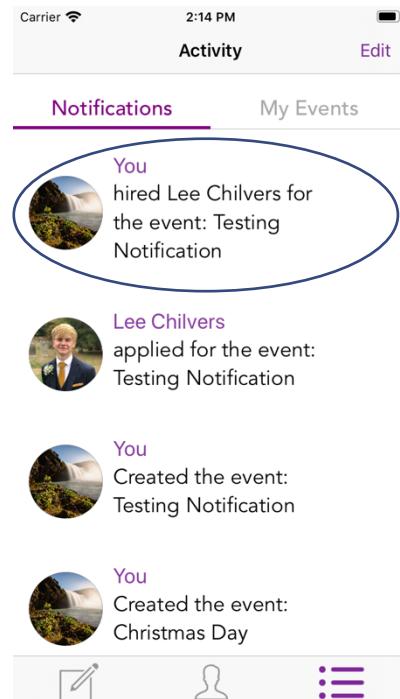
Refreshing the portfolio was the same as before and so I had no difficulty achieving this. This meant that contact was made between two users (success criterion 1).

### Test 3 – Notification Sent Back to Musician

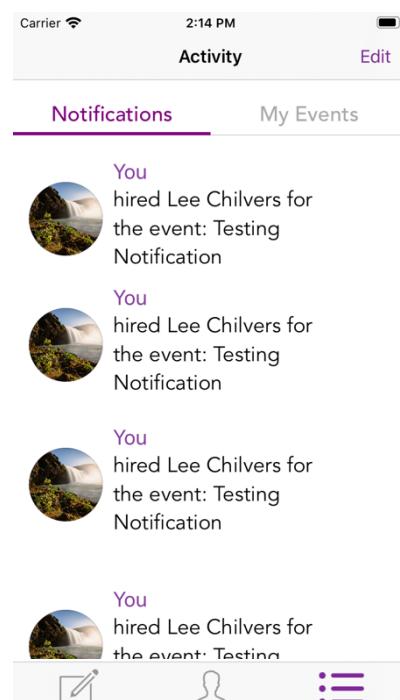
Both notifications of type 'personal' and 'reply' were sent to the correct users:

## Personal to Organiser:

## Reply to Musician:



However, if the organiser clicked the tick button multiple times before the segue back to the feed took place, then multiple notifications would occur.



I fixed this by stopping user interaction with the screen until the database was updated:

Before:

```
83 func updateActivity(accepted: Bool) {
84
85
86
87     //Set all the data for the notification
88     let notificationID = NSUUID().uuidString
89     guard let senderId = currentUser?.uid else { return }
90     guard let receiverId = uid else { return }
91     guard let senderName = currentUser?.name else { return }
92     guard let notificationPicURL = currentUser?.picURL.absoluteString else { return }
93     guard let relatedEventTitle = relatedEvent?.getTitle() else { return }
94     guard let relatedEventID = relatedEvent?.getid() else { return }
```

95

```
83 func updateActivity(accepted: Bool) {
84     //This line stops the button being pressed twice sending two activity updates
85     self.view.isUserInteractionEnabled = false
86
87     //Set all the data for the notification
88     let notificationID = NSUUID().uuidString
89     guard let senderUserID = currentUser?.uid else { return }
90     guard let receiverUserID = uid else { return }
91     guard let senderName = currentUser?.name else { return }
92     guard let notificationPicURL = currentUser?.picURL.absoluteString else { return }
93     guard let relatedEventTitle = relatedEvent?.getTitle() else { return }
94     guard let relatedEventID = relatedEvent?.getid() else { return }
```

## Milestone Review

Concluding milestone 8 means that I have come full circle completing a system where two users can put together portfolios, interact and respond to each other; everything is finally linked together.

All the tests worked pretty much first time as I was often reusing functions and snippets of code from previous milestones. I improved on a few things such as stopping user interaction to prevent a multiple notification bug. In addition, I followed validation rules to ensure that only organisers can navigate to this view.

This milestone went well and met its purpose in making contact between musicians and organisers easy with the click of a button. Necessary information is displayed as well so the hirer can make an informed decision.

My end users tested this view:

Gavin Thorrold: “This screen is good because the profile photo is big and clear and it tells me exactly what event they are applying for. I’m also glad that I can view a portfolio from here to help me make a decision”

Jude Mills: “I’m assuming that I wouldn’t normally be using this page as a musician-type user however it displays my information to the employer well and encourages them to check out the music from my profile too.”

At the end of milestone 8, my end users agreed I met objectives 1, 2, 3 from the success criteria.

## Milestone 9 – Edit Account

### Development Log

Currently, signing up to GoGig is the only opportunity to set account profile data. To fully meet success criteria 4 and 9, and allow users to review their account, I needed to reuse the current account creation view controller classes allowing them to edit their details. The only thing I did not allow them to change was their user type, this could be a future development, but for the time period of this project I would have to consider and accommodate for many more potential bugs by allowing users to do this.

This first meant navigating to the series of views. I added a button to the UIAlertController action sheet with a title of “Edit Profile” found in UserAccountVC. A global variable editingProfile keeps track of whether we are editing the account or not.

```

○  @IBAction func settingsButton(_ sender: Any) {
92    let settingsPopup = UIAlertController(title: "Settings", message: "What would you like to do?", preferredStyle: .actionSheet) //provide a popup to check
93    //Go and edit the account
94    let editProfileAction = UIAlertAction(title: "Edit profile", style: .default) { (buttonTapped) in
95        editingProfile = true
96        //perform the segue to CREATEACCOUNTVC
97        self.performSegue(withIdentifier: TO_EDIT_PROFILE, sender: nil)
98    }
99    let logoutAction = UIAlertAction(title: "Log out", style: .destructive) { (buttonTapped) in
100        let alertController = UIAlertController(title: "Log out", message: "Are you sure you want to log out?", preferredStyle: .alert)
101        alertController.addAction(editProfileAction)
102        alertController.addAction(logoutAction)
103        self.present(alertController, animated: true, completion: nil)
104    }
105    settingsPopup.addAction(editProfileAction)
106    settingsPopup.addAction(logoutAction)
107    self.present(settingsPopup, animated: true, completion: nil)
108}

```

I also auto filled all their current information in the fields for better user experience (point 1). This is done by using viewDidAppear() method, so when the account creation view appears, I get the current user profile data (only if global variable editingProfile is true) and set the input fields back in CreateProfileCAVC.

```

65 override func viewDidAppear(_ animated: Bool) {
66     //if editing account
67     if editingProfile {
68         //get the current user data...
69         if let uid = Auth.auth().currentUser?.uid {
70             DataService.instance.getDBUserProfile(uid: uid) { (returnedUser) in
71                 //..as a dictionary
72                 let returnedUserData = ["email": returnedUser.email, "bio": returnedUser.bio, "gigs": returnedUser.gigs, "name": returnedUser.name, "picURL": returnedUser.picURL, "website": returnedUser.getWebsite(), "phone": returnedUser.phone, "instagram": returnedUser.getInstagram(), "twitter": returnedUser.getTwitter(), "facebook": returnedUser.getFacebook(), "appleMusic": returnedUser.getAppleMusic(), "spotify": returnedUser.getSpotify()] as [String : Any]
73                 self.user = returnedUser
74                 //userData is the returnedUserData until changed
75                 self.userData = returnedUserData
76                 self.usernameField.text = returnedUser.name
77                 self.userBioTextView.text = returnedUser.bio
78                 self.userGigs = returnedUser.gigs
79                 self.email = returnedUser.email
80                 self.password = ""
81             }
82         }
83     }
84 }

```

Because an instantiated User object was returned, for SocialLinksCAVC and MusicLinksCAVC I could autofill the data using attributes and getters.

```

51     //View Did Appear is only called when in a navigation controller
52     override func viewDidAppear( animated: Bool) {
53         //Autofill fields when editing
54         if editingProfile == true && user != nil {
55             websiteField.text = user?.getWebsite()
56             phoneNumberField.text = user?.phone
57             instagramField.text = user?.getInstagram()
58             twitterField.text = user?.getTwitter()
59             facebookField.text = user?.getFacebook()
60         }
61     }

```

```

44     override func viewDidAppear( animated: Bool) {
45         if editingProfile == true && user != nil {
46             appleMusicField.text = user?.getAppleMusic()
47             spotifyField.text = user?.getSpotify()
48         }
49     }

```

The user should then just be able to edit their auto filled information so that the account updates the data in database. However, when they have finished, an account doesn't need to be created, just dismiss the view to the portfolio (refresh all the tabs).

```

202         if !editingProfile {
203             //creating account
204             self.performSegue(withIdentifier: TO_MAIN, sender: nil)
205         } else {
206             self.dismiss(animated: true)
207             //done with editing refresh the tab bar
208             editingProfile = false
209             NotificationCenter.default.post(name: NSNotification.Name(rawValue:
210                 "refreshTabs"), object: nil)

```

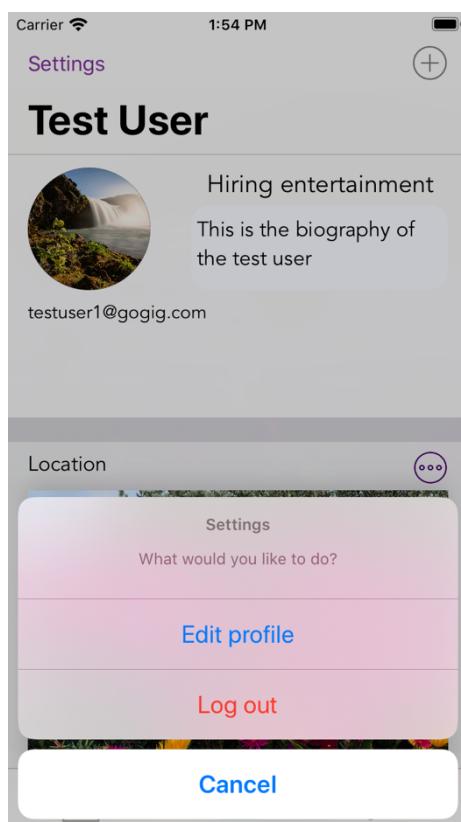
## Test Plan

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure user can navigate to the edit account views.	2 – Easy navigation	Navigation buttons	From the portfolio view, the user must be able to navigate to the same views as account creation to edit their data.	Valid
2	Text fields are automatically filled with	1 – Good user experience 1 – Simple interaction	Firebase Database	The app should display their current account data, so they <u>edit</u> their data rather than input it again entirely. Their	Valid

	their current profile data.			current data should be fetched from the database.	
3	Test valid inputs when editing.	3 – Validation rules avoid error	["bio": " Edited biography of the Test User", "instagram": "", "website": "", "appleMusic": "", "spotify": "", "phone": "", "gigs": false, "email": "testuser1@gogig.com", "name": "Test Use Editedr", "picURL": String_New_URL, "twitter": "", "facebook": ""]	Similar to account creation, the same validation rules should be applied when editing their account.	Valid and Invalid test data
4	Check account data is updated	6 – Efficient database storage	Button	The database should be updated with the new profile data and all displayed information in the portfolio should be refreshed to reflect these changes.	Valid

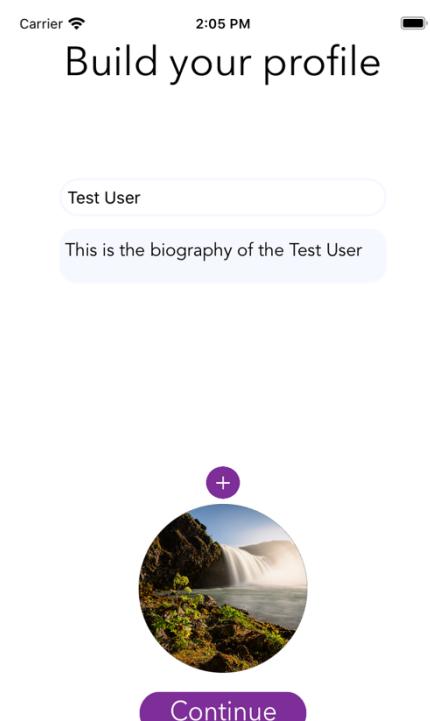
## Test 1 – Can Navigate to Edit Account

This test worked as expected and gave users options to Log Out and Edit Profile in the action sheet:



## Test 2 – Auto Fill Current User Data

The current user account data did automatically fill out however, I had a logic error in that when they progressed to the next view in the navigation stack, and then chose to go back, their current account data would automatically fill out again forgetting all of their new input. I fixed this with a variable `editingGate` which would become false when the current data fetch happened the first time.



Before:

```

65  override func viewDidAppear(_ animated: Bool) {
66      //if editing account
67      if editingProfile {
68          //get the current user data...
69          if let uid = Auth.auth().currentUser?.uid {
70              DataService.instance.getDBUserProfile(uid: uid) { (returnedUser) in
71                  //..as a dictionary
72                  let returnedUserData = ["email": returnedUser.email, "bio": returnedUser.bio, "gigs":
```

After:

```

65  override func viewDidAppear(_ animated: Bool) {
66      //if editing account
67      if editingProfile && editingGate {
68          //get the current user data...
69          if let uid = Auth.auth().currentUser?.uid {
70              DataService.instance.getDBUserProfile(uid: uid) { (returnedUser) in
71                  //..as a dictionary
72                  let returnedUserData = ["email": returnedUser.email, "bio": returnedUser.bio, "gigs":
```

This stopped the logic error.

## Test 3 – Test Valid Inputs

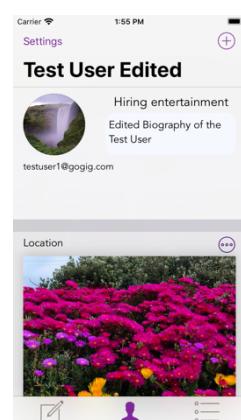
Reusing the same class meant that all the same logic for validation routines were used. The edited account data was sent to Database as expected:

```
["bio": "Edited biography of the Test User", "instagram": "", "website": "", "appleMusic": "", "spotify": "", "phone": "", "gigs": false, "email": "testuser1@gogig.com", "name": "Test User Edited", "picURL": "https://firebasestorage.googleapis.com/v0/b/gogig-db5e.appspot.com/o/3FAjk8au15YzuTQtJd0wN3d0v0P2%2FprofilePic%2F8C82BE42-DCBA-4EFD-AB23-4E6023C8F0D.jpg?alt=media&token=2ad3cd94-4351-47cd-a06e-5df9150329fd", "twitter": "", "facebook": ""]
```

```

    appleMusic: ""
    bio: "Edited biography of the Test User"
    email: "testuser1@gogig.com"
    facebook: ""
    gigs: false
    instagram: ""
    name: "Test User Edited"
    phone: ""
    picURL: "https://firebasestorage.googleapis.com/v0/b/gog... "
    spotify: ""
    twitter: ""
    website: ""
```

## Test 4 – Account Data is Updated



The account data was updated so that this change was reflected when navigating back to UserAccountVC. I did have an issue however with refreshing the tabs. The TabBarController was being navigated to and back to, which caused problems with the ViewDidAppear() functions when logging in and out and editing an account. I created a series of 'gate' variables to control resetting the state of the view (as if from first launching) whenever the user logged out or edited their account:

Before:

```

○  @IBAction func settingsButton(_ sender: Any) {
92    let settingsPopup = UIAlertController(title: "Settings", message: "What would you like to do?", preferredStyle: .actionSheet)
93    //Go and edit the account
94    let editProfileAction = UIAlertAction(title: "Edit profile", style: .default) { (buttonTapped) in
95      editingProfile = true
96
97      self.performSegue(withIdentifier: TO_EDIT_PROFILE, sender: nil)
98    }
99    let logoutAction = UIAlertAction(title: "Log out", style: .destructive) { (buttonTapped) in
100      let alertController = UIAlertController(title: "Log out", message: "Are you sure you want to log out?", preferredStyle: .alert)
101      alertController.addAction(UIAlertAction(title: "Cancel", style: .cancel))
102      alertController.addAction(UIAlertAction(title: "Log out", style: .destructive))
103      self.present(alertController, animated: true, completion: nil)
104    }
105  }
106  self.performSegue(withIdentifier: TO_EDIT_PROFILE, sender: nil)
107 }
108 }
```

After:

```

○  @IBAction func settingsButton(_ sender: Any) {
92    let settingsPopup = UIAlertController(title: "Settings", message: "What would you like to do?", preferredStyle: .actionSheet)
93    //Go and edit the account
94    let editProfileAction = UIAlertAction(title: "Edit profile", style: .default) { (buttonTapped) in
95      editingProfile = true
96      if let tabBarController = self.tabBarController {
97        tabBarController.viewControllers = tabs
98        tabGateOpen = true //Reset tabs
99        accountGateOpen = true //Reset portfolio refresh
100       cardGateOpen = true //Reset find gig card refresh
101       feedGateOpen = true //Reset activity feed refresh
102       observeGateOpen = true //Reset feed observers
103       paginationGateOpen = true //Reset activity feed pagination
104
105       //DEFAULTS.set(nil, forKey: "gigs")
106     }
107     self.performSegue(withIdentifier: TO_EDIT_PROFILE, sender: nil)
108   }
109 }
```

This required a lot of testing, checking each view would refresh after logging in and out into different accounts and editing those accounts. This solution worked so the feature functions as expected.

## Milestone Review

Finishing milestone 9 meant that users can finally edit their account information meeting success criterion 4. The development consisted of editing view controller classes so that when the user revisited the view (for the editing purpose) the view set itself up ready for changes. Testing was not too hard as they completed the same way as what they did back in milestone 1. I had an issue which was similar to logging in and out again in that all views did not refresh properly after editing an account. My fix seemed to work further meeting success criterion 4 in making sure all necessary data is presented when requested by the user. By allowing users to edit their account, I have also met the criteria by point 9 where users can decide on the amount of information they share if they want to change it later on. This will prove to be useful when I implement social media and contact information usage in milestone 11.

My end users tested this feature:

Gavin Thorrold: “I can change my account photo, name and biography at the moment which is good. I assume that this will work similarly with all the social media when you add that in.”

Jude Mills: “I change our band profile picture on social media regularly, so this was a feature we discussed and am glad you have included. As you pointed out, I can’t change my user type to ‘Hire Musicians’, maybe this could be something you choose to create later on?”

At the end of milestone 9, my end users agreed I met objectives 2, 3, 4, 5, 8, 9 from the success criteria.

## Milestone 10 – Edit/Delete Events

### Development Log

This milestone will take a similar approach to editing the account; however, I won't need to worry about resetting the state of views because the organiser will be editing their events from the TabBarController. This feature should only be accessible to the event organiser and is accessible from the EventDescriptionVC. A navigation bar is only visible if a musician is not observing it.

```
39     override func viewDidAppear(_ animated: Bool) {
40         //if musician observing event
41         if observingGigEvent {
42             //hide the edit button
43             navigationItem.rightBarButtonItem = nil
44         }
45     }
```

they only have to edit the data.

The organiser choosing to edit the event will be taken to the create event view controllers starting with TitleDateCGVC class. Similar to the previous milestone, their event data is auto filled out so that

```
68     override func viewDidAppear(_ animated: Bool) {
69         //If editing the GigEvent object
70         if editingGigEvent && editingGate {
71             //Get all the data about it...
72             if let uid = Auth.auth().currentUser?.uid {
73                 DataService.instance.getDBSingleEvent(uid: uid, eventID: editEventID) { (returnedGigEvent, success) in
74                     //...put it in a dictionary...
75                     let returnedGigEventData = ["uid": uid, "eventID": returnedGigEvent.getid(), "title": returnedGigEvent.getTitle(), "timestamp": returnedGigEvent.getTimestamp(), "latitude": returnedGigEvent.getLatitude(), "longitude": returnedGigEvent.getLongitude(), "locationName": returnedGigEvent.getLocationName(), "postcode": returnedGigEvent.getPostcode(), "payment": returnedGigEvent.getPayment(), "description": returnedGigEvent.getDescription(), "name": returnedGigEvent.getName(), "email": returnedGigEvent.getEmail(), "phone": returnedGigEvent.getPhone(), "eventPhotoURL": returnedGigEvent.getEventPhotoURL(), "appliedUsers": returnedGigEvent.getAppliedUsers()] as [String : Any]
76                     self.eventData = returnedGigEventData
77                     self.gigEvent = returnedGigEvent
78                     //...auto fill the UI inputs
79                     self.eventTitleField.text = returnedGigEvent.getTitle()
80                     let date = returnedGigEvent.getDate().addingTimeInterval(-3600)
81                     self.datePicker.setDate(date, animated: false)
82                     self.editingGate = false
83                 }
84             }
85         }
86     }
```

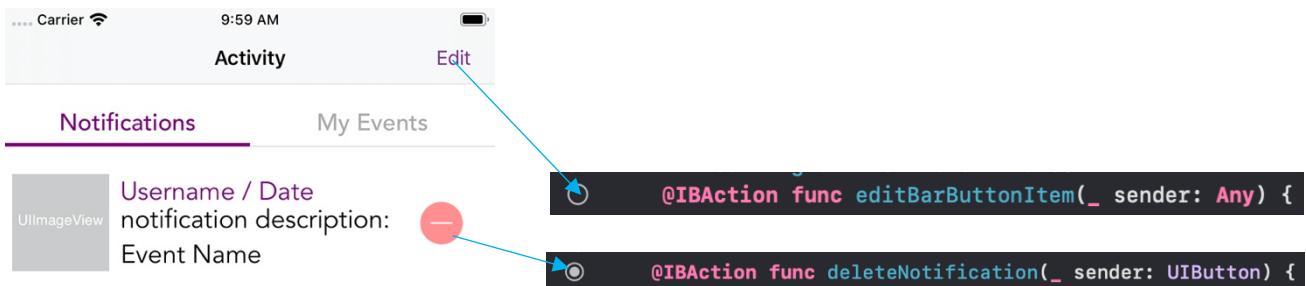
```
130
131     if !editingGigEvent {
132         self.updateActivity()
133
134         //Update the activity feed in a completion handler so it updates correctly
135         DataService.instance.updateDBActivityFeed(uid: self.notificationData!["reciever"] as! String,
136             notificationID: self.notificationData!["notificationID"] as! String, notificationData:
137             self.notificationData!) { (complete) in
138
139             if complete {
140                 self.removeSpinnerView(self.loadingSpinner)
141                 //Take user to Activity tab to see their posted event
142                 //Without completion handler we were jumping to the view controller before the activity
143                 //had updated
144                 self.tabBarController?.selectedIndex = 2
145
146                 //clear the event creation and pop to root of the navigation stack
147                 self.navigationController?.popToRootViewController(animated: true)
148             }
149         } else {
150             //If editing only
151             self.removeSpinnerView(self.loadingSpinner)
152             //clear the event creation and pop to root of the navigation stack (which is the
153             //EventDescriptionVC this time)
154             self.navigationController?.popToRootViewController(animated: true)
155             editingGigEvent = false
156             NotificationCenter.default.post(name: NSNotification.Name(rawValue: "refreshAllActivity"),
157                 object: nil)
158         }
159     }
160 }
```

And when an event is updated this time, I don't need to update the activity feed, or change the events array in the database under the user JSON object.

By allowing the organiser to edit their event means that I have met success criteria point 4. The second part of this milestone is allowing both users to delete the event. This meets success criterion 9. Musicians will delete their association with the event and organisers will delete the event object for everyone.

To clean up my design ideas, I decided against the rubbish bin symbol and followed iOS designs closer. I originally tried a swipe cell to reveal a delete button like the email app.

There was an issue with this because swiping meant that I scrolled the whole table horizontally between sections; it wasn't an option. I therefore went for a UIButton on the cell which appears when the user clicks the editBarButtonItem.



This works just as effectively as the users know what they are doing with the minus symbol as an understandable method (meeting criterion 1). Even when they try to delete the event, I chose to present a UIAlertController so that they know what they are about to do:

```

308     //Provide a warning message
309     var title = ""
310     var message = ""
311     //Warn the musician
312     if user!.gigs {
313         title = "Forget the Gig"
314         message = "You will have no association with this event"
315     //Warn the organiser
316     } else {
317         title = "Delete your Event"
318         message = "It will no longer exist to all users"
319     }

```

## Test Plan

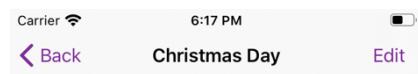
Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Make sure only organisers can navigate to views to edit their events	2 – Easy navigation 4 – User can edit anything associated to their account only	Navigation buttons	If current user is a musician, the program should block the ability to edit the event.	Invalid
2	Text fields are automatically filled with the event information	1 – Good user experience 1 – Simple interaction	Firebase Database	The app should display the current event data, so they <u>edit</u> their data rather than input it again entirely. Their current data should be fetched from the database.	Valid
3	Test valid inputs when editing an event.	3 – Validation rules avoid error	[{"eventPhotoURL": String_New_URL, "longitude": User_Lat, "uid": String_Uid, "name": "Test User", "locationName": "Suffolk One Edited", "eventId": String_Id, "title": "Test Event 1: Jazz Night Edited", "timestamp": "yyyy-MM-dd HH:mm:ss +zzzz", "description": "This is an edited event.", "latitude": 0.0, "email": "testuser1@gogig.com", "payment": 900, "phone": "01473 556601", "postcode": "IP8 3SU", "appliedUsers": ["CREATOR:String_Creator_Uid: true]}]	Similar to event creation, the same validation rules should be applied when editing the event.	Valid and Invalid test data
4	Check event data is updated	6 – Efficient database storage	Button	The database should be updated with the new event and all displayed information in the 'My Events' section should be refreshed to reflect these changes.	Valid
5	Check deleting an event takes appropriate action corresponding to type of user.	6 – Efficient database storage	Button	When deleting an event in the 'My Events' section, if the user is a musician, then it should just delete their association with that event. However, if the event's creator, it should delete the event publicly and remove it entirely from the database.	Valid

6	Check user can delete a notification from the feed and the database	4 – User can edit anything associated to their account	Button	When clicking a button, the table is refreshed removing the selected notification from the feed. In the backend, the notification object is deleted from the database. All other notifications should be unaffected.	Valid
---	---	--	--------	--	-------

## Test 1 – Only Organisers Can Navigate to Edit Event

This test worked as expected and the `editBarButton` item only appeared when the organiser was observing the event data.

Organiser:



25 December 2019  
10:00

Check out Father Christmas

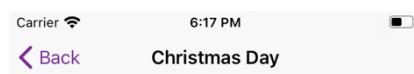
North Pole AB12 3CD

For: £25.12

'Twas the night before Christmas, when all  
through the house  
Not a creature was stirring, not even a  
mouse;  
The stockings were hung by the chimney  
with care,  
In hopes that St. Nicholas soon would be  
there;

The children were nestled all snug in their  
beds,  
While visions of sugar-plums danced in  
their heads:

Musician:



25 December 2019  
10:00

Check out Father Christmas

North Pole AB12 3CD

For: £25.12

'Twas the night before Christmas, when all  
through the house  
Not a creature was stirring, not even a  
mouse;  
The stockings were hung by the chimney  
with care,  
In hopes that St. Nicholas soon would be  
there;

The children were nestled all snug in their  
beds,  
While visions of sugar-plums danced in  
their heads:

## Test 2 – Current Event Data is Auto Filled

This was similar logic to filling out the account data, I used the same approach, and everything filled out as expected. However, there was the bug where an organiser could be editing an event, then switch to the tab to `create` an event. I want users to reach all areas of the app at all times and because the global variable `editingGigEvent` is still true when they switch tab, creating an event will auto fill as if editing. To fix this, I did a check in `ViewDidAppear()` to check what tab is selected.

```

68     override func viewDidAppear(_ animated: Bool) {
69         //We are creating not editing, this causes crash if user starts editing, then decides to create
70         if self.tabBarController?.selectedIndex == 0 {
71             editingGigEvent = false
72         }
73         //If editing the GigEvent object
74         if editingGigEvent && editingGate {
75             //Get all the data about it...

```

Also, to meet efficient database usage (point 6), I had to delete the previous image when they updated the event image in PhotoCGVC.

```

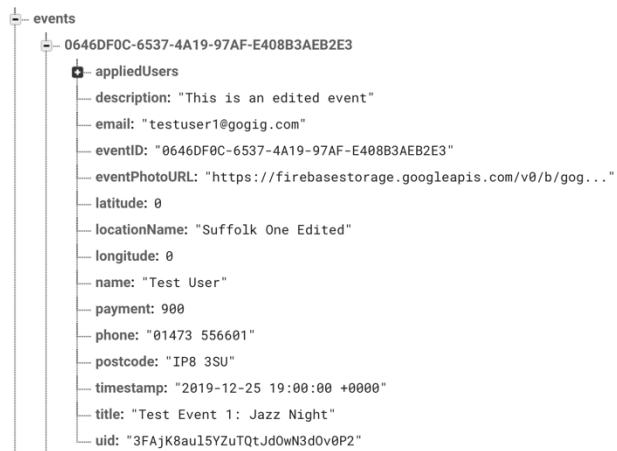
73 func picUpload(uid: String, handler: @escaping (_ url: URL) -> ()) {
74
75     //Delete the old event photo if editing
76     if editingGigEvent && gigEvent != nil {
77         DataService.instance.deleteSTFile(uid: user!.uid, directory: "events", fileID: gigEvent!.getid())
78     }
79     //Upload the photo
80     if let eventPic = eventPicView.image {
81
82         DataService.instance.updateSTPic(uid: uid, directory: "events", imageContent: eventPic, imageID: imageID,
83             uploadComplete: { (success, error) in

```

## Test 3 – Testing Valid Inputs

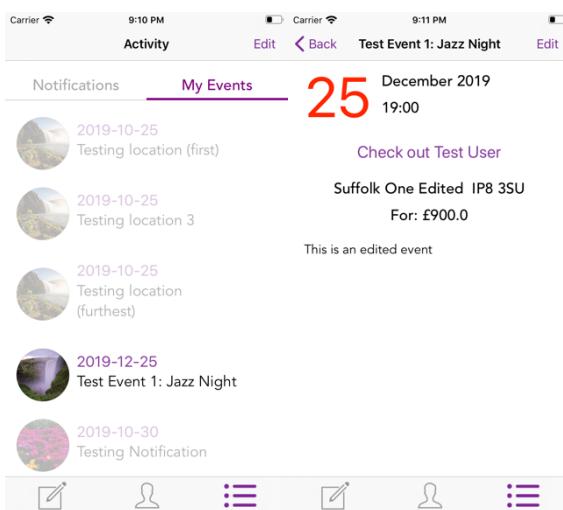
Just like the previous milestone, the validation routines are all the same. So as long as all the edited information follows the rules, the Database gets updated.

```
["eventPhotoURL": "https://firebasestorage.googleapis.com/v0/b/gogig-db55e.appspot.com/o/3FAjk8aul5YzuTQtJd0wN3d0v0P2%2Fevents%2FD18ADB70-C1C0-4F5B-8992-CC2D25F7521B?alt=media&token=c87b1d2a-f591-48b9-862b-1254045e72ab", "longitude": 0.0, "uid": Optional("3FAjk8aul5YzuTQtJd0wN3d0v0P2"), "name": "Test User", "locationName": "Suffolk One Edited", "eventID": "D18ADB70-C1C0-4F5B-8992-CC2D25F7521B", "title": "Test Event 1: Jazz Night Edited", "timestamp": "2019-12-25 19:00:04 +0000", "description": "This is an edited event", "latitude": 0.0, "email": "testuser1@gogig.com", "payment": 900.0, "phone": "01473 556601", "postcode": "IP8 3SU", "appliedUsers": ["CREATOR:3FAjk8aul5YzuTQtJd0wN3d0v0P2": true]]
```



## Test 4 – Event Data is Updated

This updated the cell information in the ‘My Events’ section and the displayed data in EventDescriptionVC.



## Test 5 – Deleting events

As you can see, I have faded out certain events in the ‘My Events’ section. These are events that are out of date. By fading events that are out of date encourages users to delete them. I still want to give users freedom to control their own data therefore they won’t be deleted automatically, this is simply an encouragement technique for efficient Database storage. This function decides whether they become faded or not:

```

242     //If old event change the UI
243     if checkOld(gigEventToCompare: usersEvents[row]) == true {
244         cell.notificationImage.alpha = 0.3
245         cell.eventNameButton.isEnabled = false
246         cell.eventNameButton.alpha = 0.3
247         cell.notificationDescriptionLabel.alpha = 0.3
248     }

```

```

273 func checkOld(gigEventToCompare: GigEvent) -> Bool {
274     //Get current date...
275     let dateObject = Date()
276     let currentDate = dateObject.addingTimeInterval(3600) //hour behind
277
278     //if the date of the GigEvent is old (less than the current date)
279     if gigEventToCompare.getDate() < currentDate {
280         return true
281     }
282     return false
283 }

```

Deleting the events themselves went wrong because when fetching the event ids they were appended to an array eventIDs, then fetching the GigEvent objects from these ids were inserted into an array eventListings at index 0. Therefore, when trying to delete an event, completely the wrong event was being deleted (as the row corresponded to the wrong index). I fixed this by reversing the eventIDs swift array before fetching the GigEvent objects.

Before:

```

104     //Get and Observe the User Event Recordings
105     eventsHandle = DataService.instance.REF_USERS.child(uid).child("events").observe(.value) { (snapshot) in
106         DataService.instance.getDBUserEvents(uid: uid) { (returnedEventIDs) in
107             var eventListings = [GigEvent]()
108             //Iterate through the list of eventIDs associated to user, get each one from public events
109             //And insert at 0 of local array
110             for eventID in returnedEventIDs {
111                 DataService.instance.getDBSingleEvent(uid: uid, eventID: eventID) { (returnedGigEvent, success) in
112                     if success {
113                         eventListings.insert(returnedGigEvent, at: 0)
114                         self.usersEvents = eventListings
115                         self.collectionView.reloadData()
116                         self.attemptReload()
117
118                     }
119                 }
120             }
121         }
122     }

```

After:

```

104     //Get and Observe the User Event Recordings
105     eventsHandle = DataService.instance.REF_USERS.child(uid).child("events").observe(.value) { (snapshot) in
106         DataService.instance.getDBUserEvents(uid: uid) { (returnedEventIDs) in
107             var eventListings = [GigEvent]()
108             //One was getting appended, the other was getting inserted at 0. Deleting didn't delete the correct
109             //gigEvent
110             self.eventIDs = returnedEventIDs.reversed()
111             //Iterate through the list of eventIDs associated to user, get each one from public events
112             //And insert at 0 of local array
113             for eventID in returnedEventIDs {
114                 DataService.instance.getDBSingleEvent(uid: uid, eventID: eventID) { (returnedGigEvent, success) in
115                     if success {
116                         eventListings.insert(returnedGigEvent, at: 0)
117                         self.usersEvents = eventListings
118                         self.collectionView.reloadData()
119                         self.attemptReload()
120
121                     }
122                 }
123             }
124         }
125     }

```

In addition to this, an event's organiser deleting the object first removed it as it is supposed to and it didn't appear to any user. However, it is still associated to the musicians in the event ids array. Therefore, when deleting anymore events, it deletes the wrong object because the eventIDs length is different to eventListings array. Therefore, I return success if the musician could fetch the GigEvent object from the database and if it failed, then remove the event id from the local and Database array:

Final:

```

104     //Get and Observe the User Event Recordings
105     eventsHandle = DataService.instance.REF_USERS.child(uid).child("events").observe(.value) { (snapshot) in
106         DataService.instance.getDBUserEvents(uid: uid) { (returnedEventIDs) in
107             var eventListings = [GigEvent]()
108             //One was getting appended, the other was getting inserted at 0. Deleting didn't delete the correct
109             //gigEvent
110             self.eventIDs = returnedEventIDs.reversed()
111             //Iterate through the list of eventIDs associated to user, get each one from public events
112             //And insert at 0 of local array
113             for eventID in returnedEventIDs {
114                 DataService.instance.getDBSingleEvent(uid: uid, eventID: eventID) { (returnedGigEvent, success) in
115                     if success {
116                         eventListings.insert(returnedGigEvent, at: 0)
117                         self.usersEvents = eventListings
118                         self.collectionView.reloadData()
119                         self.attemptReload()
120
121                         //Couldn't get GigEvent
122                     } else {
123                         //User (should be musician) has an eventID listed which the organiser has already deleted
124                         //the event, clean up the DB
125                         if let index = self.eventIDs.firstIndex(of: eventID) {
126                             self.eventIDs.remove(at: index)
127                             DataService.instance.deleteDBUserEvents(uid: uid, eventIDs: self.eventIDs)
128                         }
129                         self.attemptReload()
130                     }
131                 }
132             }
133         }
134     }

```

Deleting an event then always worked so Database data usage is efficient.

## Test 6 – Deleting Notifications

Deleting notifications was fine because it was just a case of removing that value from the database and removing it locally and refreshing the feed. I did not need to consider how two user's actions would affect each other.

## Milestone Review

Editing an event was a similar process to editing a profile and so I knew what approach to take and where potential errors would occur. Deleting events were more complicated because I actually didn't consider, at first, how it would affect other users that reference the same Database object. I put together a solution however that meets efficient Database usage (success criterion 6) and a feature which allows users to edit and review data associated to their account (point 4).

My end users' responses:

Gavin Thorrold: “I like this feature because I can initially search for musician applicants and then later confirm the date and details of our event.”

Jude Mills: “It works really well! My only point is that I would want the hirer to know that I have backed out of playing at the gig. I would probably email or ring them anyway, but perhaps it might be good to send a push notification too if you get round to it?”

At the end of milestone 10, my end users agreed I met objectives 1, 2, 3, 4, 6 from the success criteria.

# Milestone 11 – Device Push Notifications and Social Media Links

## Development Log

For the social media links, there are several validation rules to ensure that the links work correctly when clicked on. Website URLs will have to be copy-and-pasted in full but social media usernames are entered and concatenated into a deep link to the social app. If any links do not work when clicked on, a UIAlertController will notify that the link does not work, reporting back error and hence meeting criteria 2 and 3. The purpose of this milestone is to match the specification of User Information so that work can be promoted in the Portfolio with social media and music streaming URLs.

## Validation Routines

IBoutlet	Rules	Recovery
phoneNumberField	<ul style="list-style-type: none"> <li>• Is optional input</li> <li>• Length is more than 10</li> <li>• Character limit is 16</li> </ul>	<u>Display Error</u> Title: “Oops” Message: “Please enter a valid website (optional)”
websiteField	<ul style="list-style-type: none"> <li>• Optional input</li> <li>• Input contains “.”</li> <li>• Length is more than 5</li> <li>• Character limit is 100</li> </ul>	<u>Display Error</u> Title: “Oops” Message: “Please enter a valid phone number (optional)”
instagramField	<ul style="list-style-type: none"> <li>• Optional input</li> <li>• Must not contain “@”</li> <li>• Must not contain space</li> <li>• Character limit is 30</li> </ul>	<u>Display Error</u> Title: “Oops” Message: “Please enter a valid Instagram username (optional)”
twitterField	<ul style="list-style-type: none"> <li>• Optional input</li> <li>• Must not contain “@”</li> <li>• Must not contain space</li> <li>• Character limit is 30</li> </ul>	<u>Display Error</u> Title: “Oops” Message: “Please enter a valid Twitter username (optional)”
facebookField	<ul style="list-style-type: none"> <li>• Optional input</li> <li>• Length is more than 10</li> <li>• Must not contain space</li> </ul>	<u>Display Error</u> Title: “Oops” Message: “Please enter a valid Facebook Page ID (optional)”
appleMusicField	<ul style="list-style-type: none"> <li>• Optional input</li> <li>• Input contains “.”</li> <li>• Length is more than 5</li> </ul>	<u>Display Error</u> Title: “Oops” Message: “Please enter a valid Apple Music URL (optional)”
spotifyField	<ul style="list-style-type: none"> <li>• Optional input</li> <li>• Input contains “.”</li> </ul>	<u>Display Error</u> Title: “Oops”

	<ul style="list-style-type: none"> <li>Length is more than 5</li> </ul>	Message: “Please enter a valid Spotify URL (optional)”
--	---	--

Displaying the link UIButtons in the AccountHeaderCell as part of UserAccountVC consists of a series of IF statements which checks if the user inputted anything:

```

175     //If user hasn't got Facebook
176     if user?.getFacebook() == "" {
177         //Put it at the right of the horizontal stack
178         cell.socialLinkStackView.insertArrangedSubview(cell.facebookLinkButton, at: 5)
179         //Disable the button
180         cell.facebookLinkButton.isEnabled = false
181         //And hide it
182         cell.facebookLinkButton.alpha = 0.0
183         //Has Facebook
184     } else {
185         //Enable it
186         cell.facebookLinkButton.isEnabled = true
187         //Show it
188         cell.facebookLinkButton.alpha = 1.0
189     }
190     //Same for Twitter etc
191     if user?.getTwitter() == "" {
192         cell.socialLinkStackView.insertArrangedSubview(cell.twitterLinkButton, at: 5)

```

Facebook doesn't follow the same structure as Instagram and Twitter. Facebook you have to provide a 'Page ID' to embed in the link for it to work. This is a series of numbers which can be found on a business page in-app and found by looking closely at the URL with a browser. There is no easy way to attach a Facebook account therefore, unless I use the Facebook Login API, this is therefore an essential for future development but to save time in this project I will look over this issue and explain it to my end users.

For the push notifications, I used Google Firebase's Cloud Messaging and installed the CocoaPod **pod 'Firebase/Messaging'** to access the library. A message is sent to a specific deviceFCMToken (Firebase Cloud Messaging Token). When a user logs in their token is updated in Database and when they log out it should be set to nil (or null) so notifications are not received when they are not logged in. Therefore, whenever a user logs in, creates or edits an account, this logic occurs:

```

145             InstanceID.instanceID().instanceID { (result, error) in
146                 if let error = error {
147                     print("Error fetching remote instance ID: \(error)")
148                 } else if let result = result {
149                     print("Remote instance ID token: \(result.token)")
150                     deviceFCMToken = result.token
151                 }
152             }

```

And then update the token in Database when portfolio view appears:

```

311 //Refresh the FCM Token for push notifications
312 func refreshFCMToken() {
313     if deviceFCMToken != nil && pushNotificationGateOpen == true {
314         if let uid = Auth.auth().currentUser?.uid { //update token in database
315             DataService.instance.updateDBUserFCMToken(uid: uid, token: deviceFCMToken!)
316             pushNotificationGateOpen = false
317         }
318     }
319 }
```

## Test Plan

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Send a test notification from Firebase Website to specific devices	5 – Receive notifications when app is not in use	Firebase Cloud Messaging  <b>Title: Test Notification</b> <b>Message: This is a test notification</b>	This is to check that a device can receive a push notification when one is sent. The push notification should come through with custom text, appear on the lock screen and when the app is not in use. The push notification should not be received when the app is in use.	Valid
2	Send test notification from one device to another.	5 – Receive notifications when app is not in use	#1 <b>Title: Application Pending</b> <b>Message: Sender has applied for your event</b>  #2 <b>Title: You got the gig!</b> <b>Message: Sender has hired you for the event: <i>eventName</i></b>	This is to check that a device can send and receive a push notification. The push notification should come through with custom text, appear on the lock screen and when the app is not in use. The push notification should not be received when the app is in use.	Valid
3	Check push notification is sent whenever necessary.	5 – Activity updates	Device	The program should send a push notification to the correct device whenever the database is updated with a new notification object.	Valid
4	Make sure the app can be opened from the push notification.	1 – Good user experience	Push notification	The app should launch after opening from the push notification on the lock screen or the device home page.	Valid

5	User can add social media links and check against validation rules.	1 – Make contact between users 3 – Check against rules to avoid error 9 – Social and contact links	Phone: +44 7496243062 Website: chilly-designs.com Instagram: l_chilvers Twitter: chilvers_lee Facebook: 1837812439827573 Apple Music: <a href="https://music.apple.com/gb/artis/cody-fry/289832557">https://music.apple.com/gb/artis/cody-fry/289832557</a> Spotify: <a href="https://open.spotify.com/artist/7dOCnyDR2oEa1hQlvTXvdT">https://open.spotify.com/artist/7dOCnyDR2oEa1hQlvTXvdT</a>	The app should compare entered information against validation rules to ensure that the social media link will likely work when displayed on the portfolio page.	Valid and Invalid test data
6	Check we can open social media apps and web URLs from GoGig.	2 – Easy navigation	Button	Social apps corresponding to the button pressed should be opened and navigate to the correct account. If apps are not installed, then it should open a website in Safari web browser. If entered information is invalid, the app should notify the user that this link does not exist rather than throw error.	Valid and Invalid

## Test 1 – Test Notification from Firebase Website

Before I could send anything, I needed to generate and export an APNS client TLS certificate from developer.apple.com. This will basically configure the app for push notifications:

### Certificates, Identifiers & Profiles

All Keys

Register a New Key

Key Name: GoGig Push Notification Key

ENABLE NAME SERVICE

<input checked="" type="checkbox"/>	Apple Push Notifications service (APNs)	Establish connectivity between your notification server and the Apple Push Notification service. One key is used for all of your apps. <a href="#">Learn more</a>
<input type="checkbox"/>	DeviceCheck	Access per-device, per-developer data that your associated server can use in its business logic. One key is used for all of your apps. <a href="#">Learn more</a>
<input type="checkbox"/>	MapKit JS	Use Apple Maps on your websites. Show a map, display search results, provide directions, and more. <a href="#">Learn more</a> ⚠ There are no identifiers available that can be associated with the key
<input type="checkbox"/>	MusicKit	Access the Apple Music catalog and make personalized requests for authorized users. <a href="#">Learn more</a> ⚠ There are no identifiers available that can be associated with the key
<input type="checkbox"/>	Sign in with Apple	Enable your apps to allow users to authenticate in your application with their Apple ID. Configuration is required to enable this feature. <a href="#">Learn more</a> ⚠ There are no identifiers available that can be associated with the key

Continue

Copyright © 2019 Apple Inc. All rights reserved. | [Terms of Use](#) | [Privacy Policy](#)

I can then target a specific device with the FCM token from the Firebase console:

Notification title: Test Notification

Notification text: This is a test notification

Notification image (optional): Example: <https://yourapp.com/image.png>

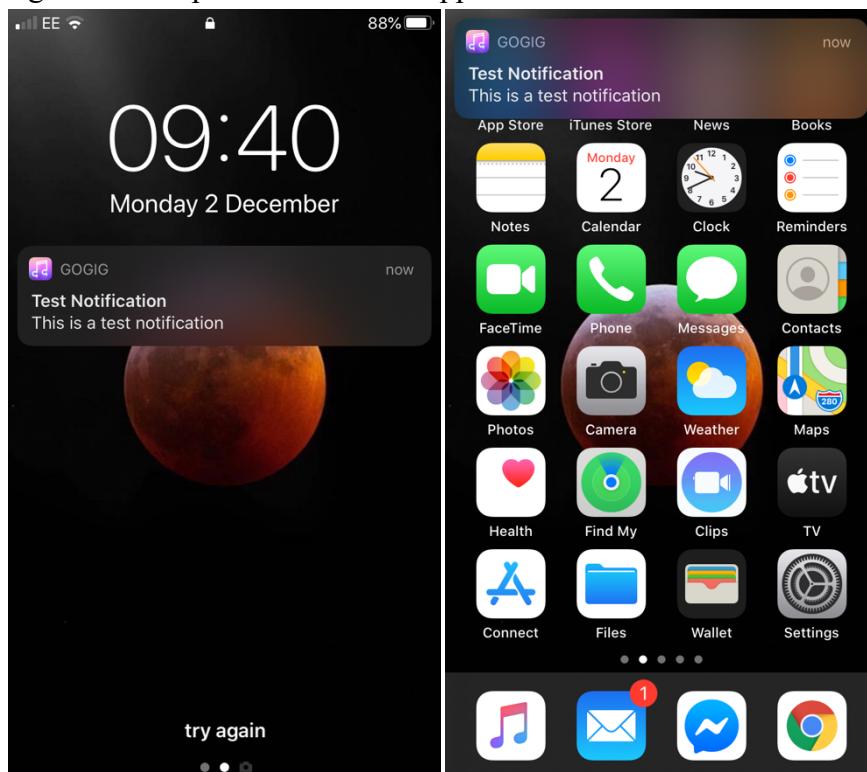
Notification name (optional): Enter optional name

Device preview: Shows a smartphone screen with the notification 'Test Notification' and 'This is a test notification'. Buttons for 'Initial state' and 'Expanded view' are shown. A 'Send test message' button is also present.

Send to this FCM

Test

This sent a notification to the lock screen and home screen. I also tested to make sure that the app could be opened from this notification which it could be, meeting success criterion 1 for a good user experience from the app.



## Test 2 – Send Notification from Another Device

This was a difficult task to complete because I had to create a function to make HTTP POST network requests:

```

658 //Send a notification from a device to another device
659 func sendPushNotification(to token: String, title: String, body: String) {
660     //URL to send notification
661     let urlString = "https://fcm.googleapis.com/fcm/send"
662     let url = NSURL(string: urlString)!
663     //Parameters of the notification
664     let paramString: [String : Any] = ["to" : token,
665                                         "notification" : ["title" : title, "body" : body],
666                                         "data" : ["user" : "test_id"]]
667     //Create request object using url
668     let request = NSMutableURLRequest(url: url as URL)
669     request.httpMethod = "POST" //Http method is POST
670     //Convert paramString to JSON and set it as request body
671     request.httpBody = try? JSONSerialization.data(withJSONObject:paramString, options: [.prettyPrinted])
672     request.setValue("application/json", forHTTPHeaderField: "Content-Type")
673     //Server key
674     request.setValue("key=AAAAjb8BHzs:APA91bEBkZ3IfE6dU4xclX1P4qGVqyFhMLQEuCTA8NtFjKC7WGN_L8LeuaH_t7142RWGLbuYqjSHozuiz7HtmAADhGEz67yMojN416Z-EdbIE9FXJ-0uyI37mcQ6bcMzbhoxSz"
675         forHTTPHeaderField: "Authorization")
676     //Data task, send the request to the server in a completion handler (executed when the request's response is returned to the app
677     let task = URLSession.shared.dataTask(with: request as URLRequest) { (data, response, error) in
678         do {
679             //If there is JSON data
680             if let jsonData = data {
681                 //Try and convert to a dictionary
682                 if let jsonDataDict = try JSONSerialization.jsonObject(with: jsonData, options: JSONSerialization.ReadingOptions.allowFragments) as? [String: AnyObject] {
683                     NSLog("Received data:\n\(jsonDataDict)")
684                 }
685                 //If failed catch it
686             } catch let err as NSError {
687                 //Print the error
688                 print(err.debugDescription)
689             }
690         }
691         //Resume the task
692         task.resume()
693     }

```

This is a DataService function and so can be called anywhere. These push notifications are sent whenever an ActivityNotification is uploaded to the Database:

FindGigVC when musician swipes card right:

```

278 //Send a push notification to other user
279 DataService.instance.getDBUserProfile(uid: recieverUid) { (returnedUser) in
280     DataService.instance.sendPushNotification(to: returnedUser.getFCMToken(), title: "Application pending", body:
281         "\u{senderName} has applied for your event")
282 }

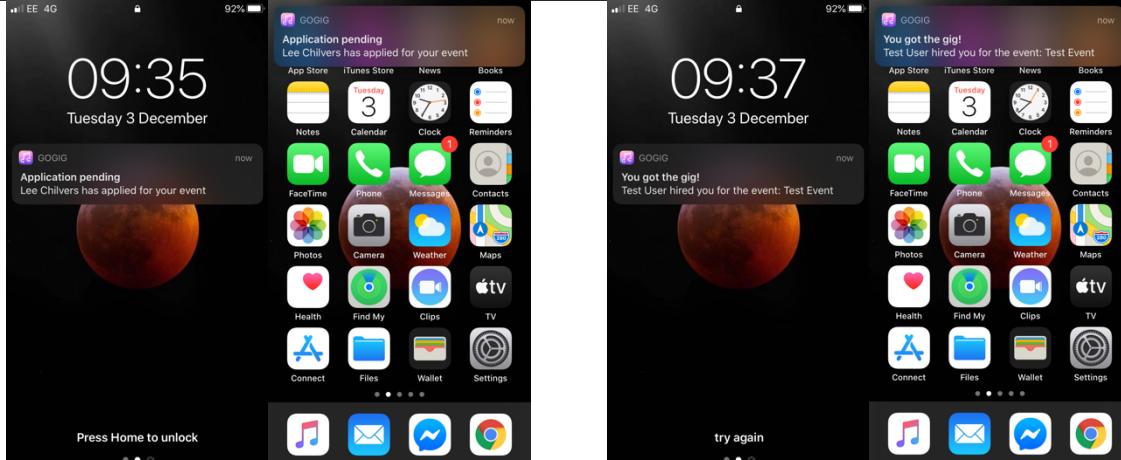
```

ReviewApplicationVC when organiser accepts application:

```

101 //Send a push notification to other user
102 DataService.instance.getDBUserProfile(uid: recieverUid) { (returnedUser) in
103     DataService.instance.sendPushNotification(to: returnedUser.getFCMToken(), title: "You got the gig!", body:
104         "\u{senderName} hired you for the event: \u{relatedEventTitle}")
105 }

```

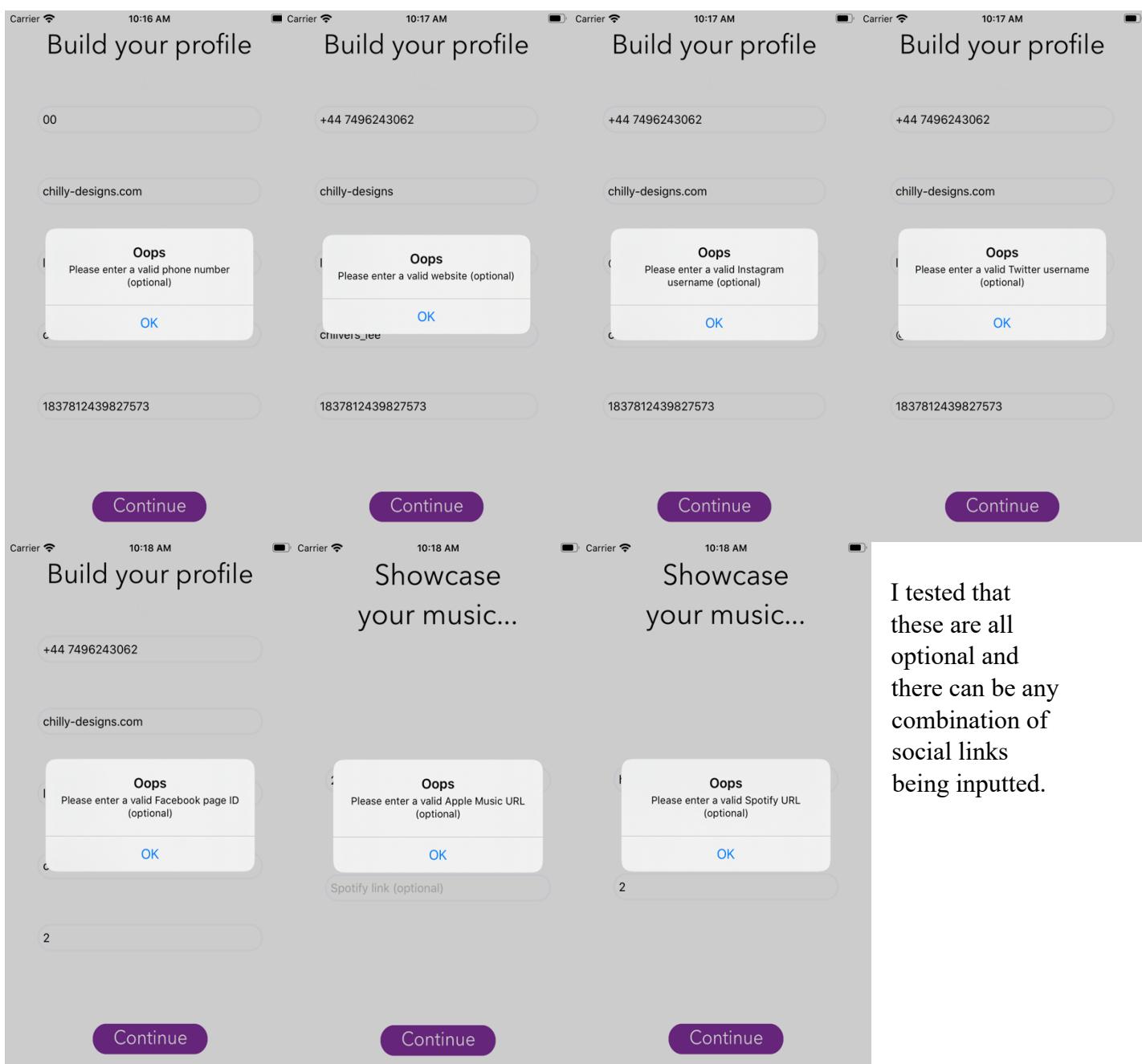


## Test 3 & 4 – Push Notification Sent to Device

As shown in Test 2, the notification gets sent to the physical device and the appropriate message is sent whenever I log into the different accounts. I only have one physical iPhone, and notifications won't send to the iOS simulator so therefore I will ask my end users to check this feature works with their iPhones. These notifications open the app to meet a good user experience. I have met success criterion 5 here in receiving notifications corresponding to what account is logged in with the authentication system.

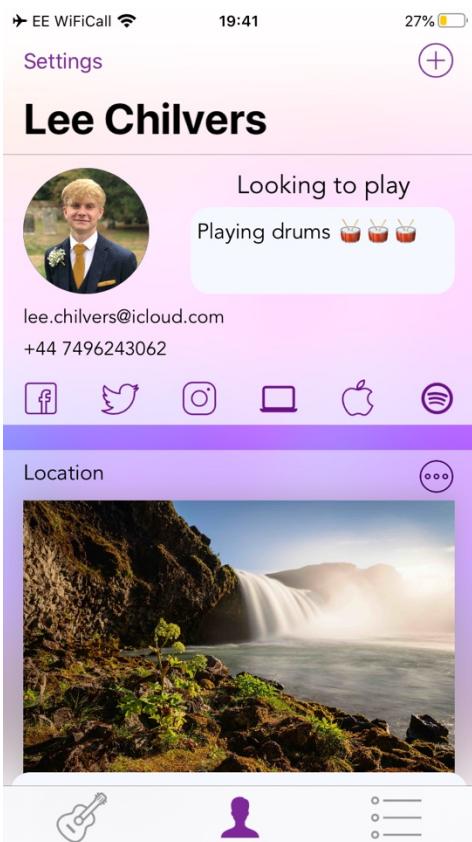
## Test 5 – Can Add Social Links and Checks Against Validation Rules

All of the correct notification errors appeared:



## Test 6 – Can Open Social Links

Displaying the social link `UIButtons` looked like this:

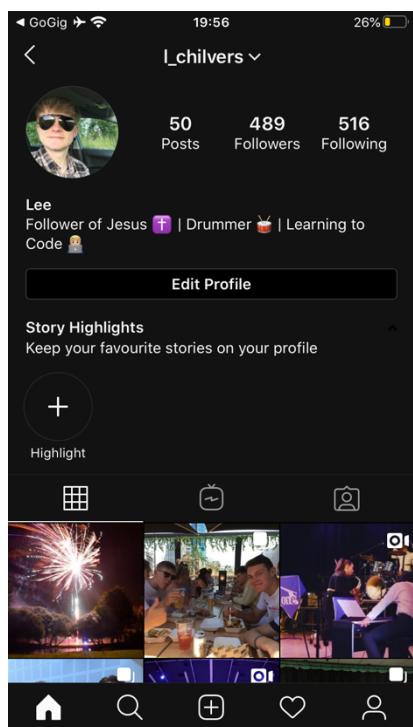


And if any of the inputs were left blank, the button is pushed to the far right, disabled and hidden. When clicking on a social media button (Facebook, Twitter, Instagram) it follows this procedure:

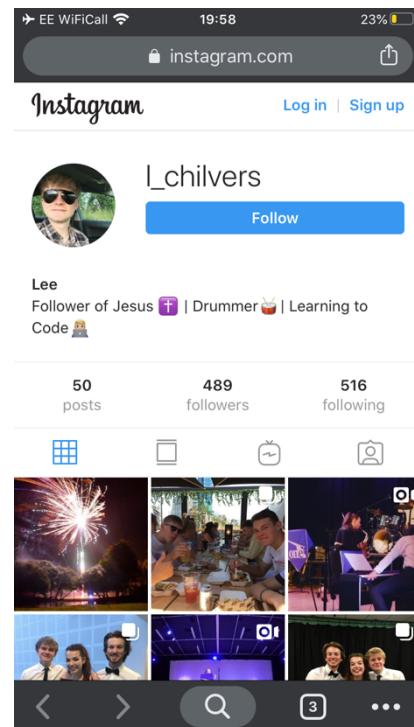
```
○ @IBAction func twitterLink(_ sender: Any) {
342    //get the username
343    let username = user?.getTwitter()
344    //make a url from username
345    if let appURL = URL(string: "twitter://user?screen_name=(username!)") {
346        let application = UIApplication.shared
347        //if app installed...
348        if application.canOpenURL(appURL) {
349            //...open in app
350            application.open(appURL)
351        } else {
352            //If not installed, open in Safari browser
353            let webURL = URL(string: "https://twitter.com/(username!)!")
354            if application.canOpenURL(webURL){
355                application.open(webURL)
356            //URL doesn't work, display error
357            } else {
358                displayError(title: "", message: "Couldn't open Twitter account")
359            }
360        }
361    }
362 }
```

Therefore, the URL opened in app (when installed) and in safari (when not installed) and it flagged up when the URL is not followable if the app is uninstalled. Example with Instagram:

App:



## Browser:



However, it did not work exactly as expected for websites and music streaming links.

I originally had this procedure:

Before:

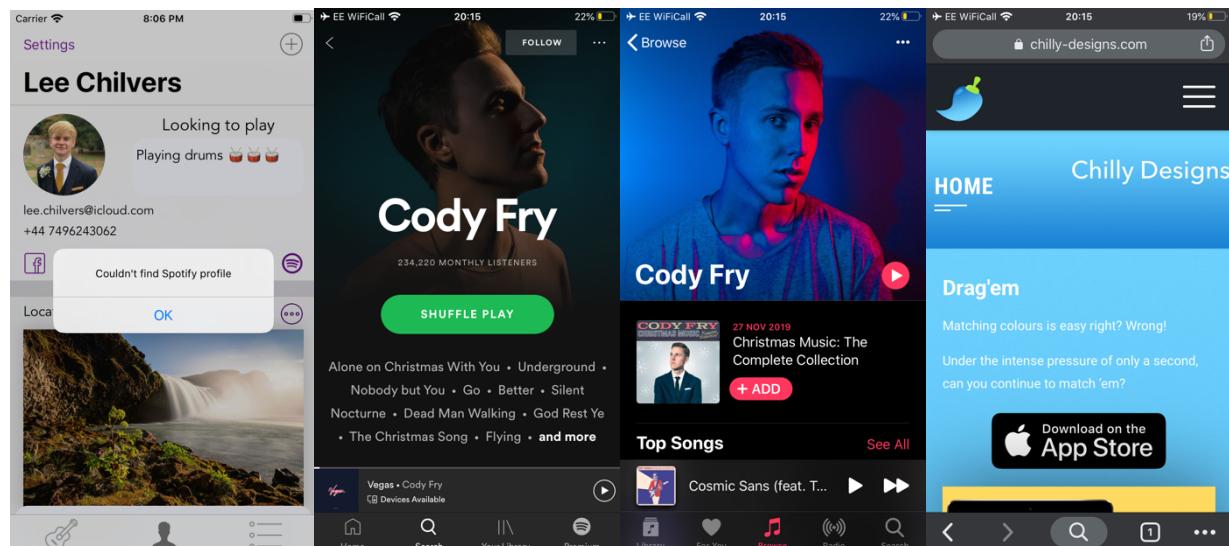
```
○ @IBAction func websiteLink(_ sender: Any) {
382     let userWebsite = user?.getWebsite()
383     if let webURL = URL(string: userWebsite!) {
384         let application = UIApplication.shared
385         if application.canOpenURL(webURL) {
386             application.open(webURL)
387         }
388     } else {
389         displayError(title: "", message: "Couldn't find website")
390     }
391 }
```

But when the input is not entered to give an actual URL, the error is not displayed. This was because I was displaying the error only if a URL object could not be instantiated, however, one is instantiated, but it simply doesn't lead anywhere. I changed it so that error is displayed when application can't open URL.

After:

```
○ @IBAction func spotifyLink(_ sender: Any) {
408     //get spotify string url
409     let userStreaming = user?.getSpotify()
410     //instantiate a url object
411     if let webURL = URL(string: userStreaming!) {
412         let application = UIApplication.shared
413         //if can open in browser (or app)
414         if application.canOpenURL(webURL) {
415             application.open(webURL)
416             //if can't open
417         } else {
418             displayError(title: "", message: "Couldn't find Spotify profile")
419         }
420         //if instantiation failed
421     } else {
422         displayError(title: "", message: "Couldn't find Spotify profile")
423     }
424 }
```

And the test then worked for all Spotify, Apple Music and websites:



## Milestone Review

Both the push notifications and social links work to ultimately make contact between two users securing a deal easier and quicker. I believe I have put together a good user experience because users can reference social media accounts or websites without having dedicated apps installed as they should be taken to the browser; everyone can access them on iPhone. This is also an optional feature, which is important because some people may want to keep accounts private. Feedback is also given if the link does not work.

If I were to do it again, though, I would definitely do a check to make sure the URL works when adding them or editing an account because, obviously at the moment, it is possible with my validation rules to create URL that leads nowhere. This could be improved upon by checking `application.canOpen(url)` when inputting the link.

My end users tested these features and I checked push notifications with them on their physical devices:

Gavin Thorrold: “I get a push notification sent to my device and I’m happy with the buttons that take me to Instagram, Twitter etc. It matches the design well and is really clear on what the buttons are for with your chosen symbols. My one criticism is that I had no clue what a Facebook Page ID was and wouldn’t have known how to find mine without your guidance. It’s not particularly clear for other users and will likely reduce the amount of people that link up a Facebook account.”

Jude Mills: “The messages come through on my lock screen telling me that I ‘Got the gig’ which looks really cool and professional. I initially tried to input my Twitter name with a @ at the front, but it didn’t like that. Maybe you could make a way so it works for both with and without that character?”.

## Adjustments from Feedback

Jude pointed out that input did not allow “@” for Twitter and Instagram. Most accounts are advertised in the format @username so it is likely most people will try and input this. I adjusted the logic to remove the first character of the input if it is a “@” character. I also added a breakpoint to examine the string too.

```

114     if let userTwitterStr = twitterField.text {
115         //Make variable from constant
116         var userTwitter = userTwitterStr
117         //Check first char
118         if userTwitter[userTwitter.startIndex] == "@" {
119             //Make a substring (remove the @)
120             userTwitter = userTwitter.substring(start: 1, end: userTwitter.count)
121         }
122         if (!userTwitter.contains("@") && !userTwitter.contains(" ")) || userTwitter == "" {
123             userData!["twitter"] = userTwitter
124         } else {
125             displayError(title: "Oops", message: "Please enter a valid Twitter username (optional)")
126             continueFine = false
127         }
128     }

```

▶ L userTwitterStr = (String) "@chilvers\_lee"  
 ▶ L userTwitter = (String) "chilvers\_lee"

I then had a crash where if the user left the field empty, the index was out of range as there is no start index to check for “@”. I therefore changed the Boolean logic to this:

```
//Check first char
if userInstagram != "" && userInstagram.count > 1 && userInstagram[userInstagram.startIndex] == "@" {
    //Make a substring (remove the @)
    userInstagram = userInstagram.substring(start: 1, end: userInstagram.count)
}
► L userInstagramStr = (String) "@l_chilvers"
► L userInstagram = (String) "l_chilvers"
```

And it continued to work for Twitter and Instagram inputs.

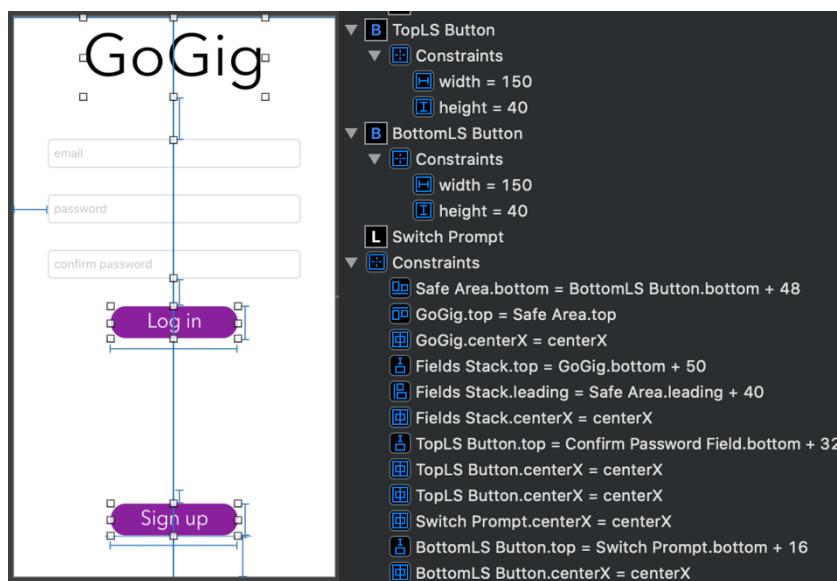
At the end of milestone 11, my end users agreed I met objectives 1, 2, 3, 4, 5, 9 from the success criteria.

## Milestone 12 – Constraints for all iPhone Devices and User Interface Finishing Touches

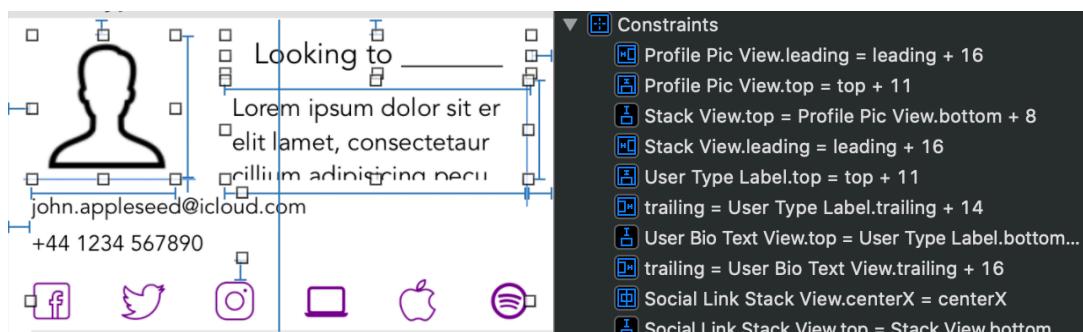
### Development Log

Quite an important specification point is making sure that the app can be used on most iPhone devices. Before I match the design of the user interface, I need to implement constraints on the UI elements so as the screen dimensions change, the width, height and positioning of these elements automatically change. Each element in the view needs to know four things: a width, height, x and y coordinates. This is the general rule when putting together the auto layout. By setting constraints, it ensures no buttons are obstructed on any device for freedom in navigation which completes success criterion 2.

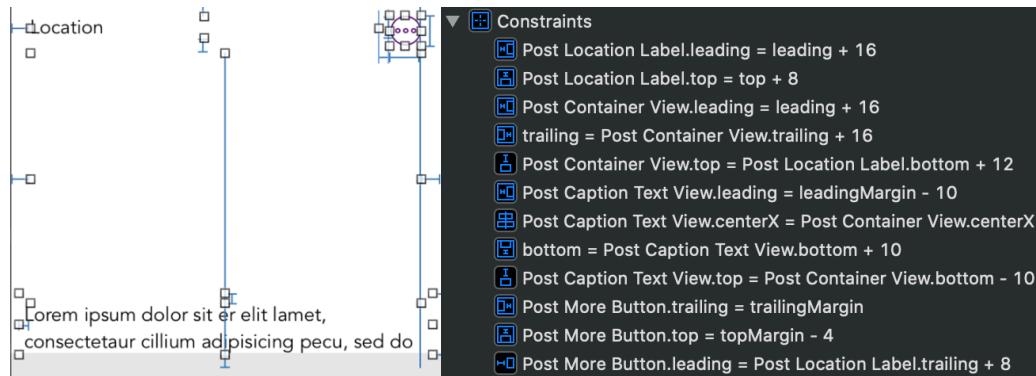
A typical form would look like this in terms of constraints. Where I start from the top of the safe area and give an element a top constraint to give spacing between. Items are centred on the x-axis. A continue button (or in this case swap mode button) is always given a bottom constraint to bottom safe area to fill the view rather than have a large gap. Originally, I linked the continue button to the element above, and when the height of the device changed, they are therefore pushed down ‘below’ the screen. Here is an example:



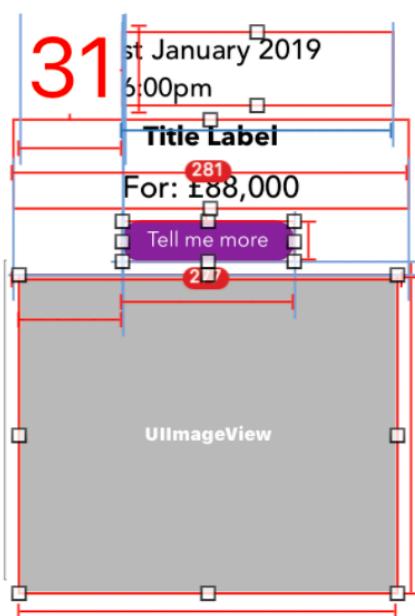
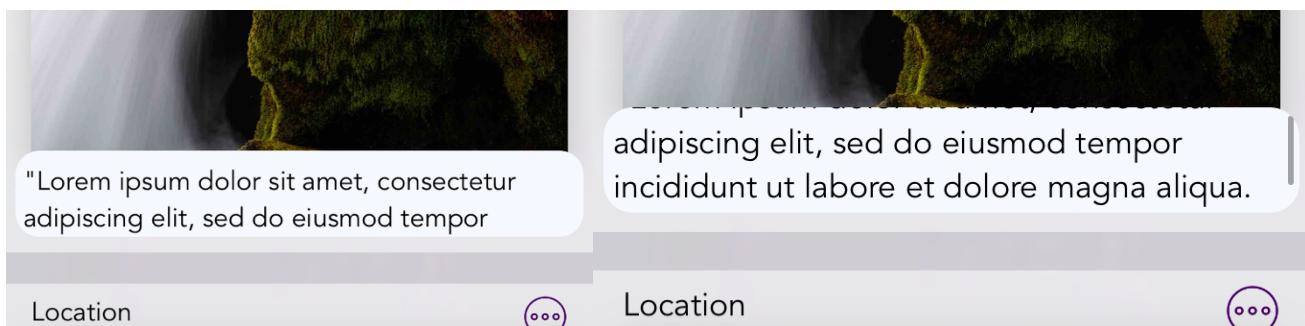
For the AccountHeaderCell in the portfolio, each element is pinned to the sides of the cell with a margin of 16. The social media links are all contained into a horizontal stack UIView which evenly spaces out the buttons. This is a similar setup to the ActivityFeedCell too.



For the AccountPostCell in the same table view, the cell auto resizes depending on the content of the cell at runtime. The dimensions of posts change depending on what image the user had uploaded to Database, therefore, PostContainerView changes height (while maintaining a constant width controlled by the leading and trailing constraints). For this to work I need to link up constraints from the top of the cell to the bottom, so that it knows how to resize the elements contained in the view.



This was put together in milestone 2 in order to progress with development, and at this point, I still struggled to get the caption UITextView to resize depending on the length of the text. The work around I had, although not ideal, was making the UITextView scrollable, so users can scroll through its contents to read the full caption:



The gig cards in FindGigVC were slightly more challenging to add constraints to. Because I use a gesture recogniser to drag the card around, I don't want it fixed in position by constraints. I therefore only use constraints within the UIView itself so that each element remains in the right position relative to it. Because the card is centred programmatically at runtime, and the IDE doesn't know the exact x and y coordinates, these constraints flag red as an error. However, I should be able to ignore them.

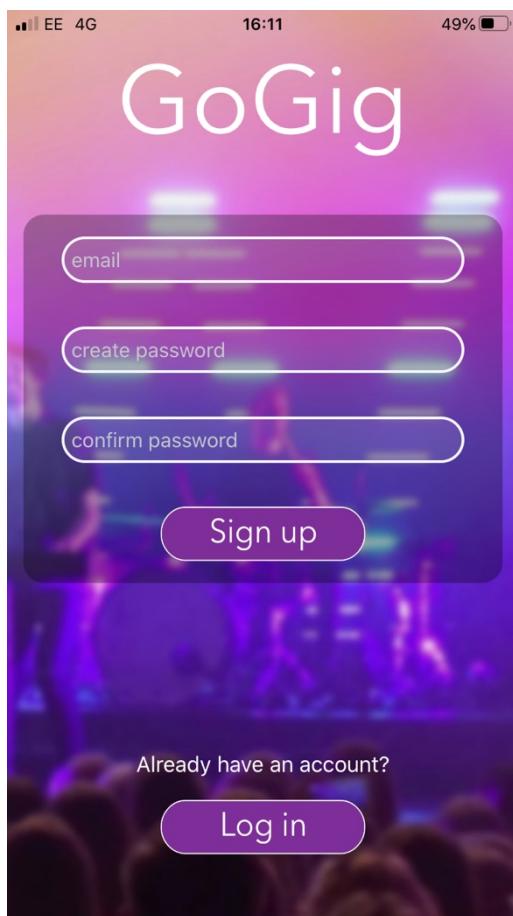
To match my designs, I wrote a series of extension functions, `setupView()`, for each `UIViewController` class. These, basically, add a background to the view and adjust the aesthetics of any UI elements programmatically (if they haven't been adjusted within the storyboard file). For some views, I have added a transparent `UIView` so there is a high enough contrast between white text and the background.

Code example for `LoginSignupVC`:

```

11 extension LoginSignupVC {
12     func setupView() {
13         //Change the colour of the textfield placeholders so it can be seen on background
14         emailField.attributedPlaceholder = NSAttributedString(string: "email",
15             attributes: [NSAttributedString.Key.foregroundColor: UIColor.systemGray3])
16         passwordField.attributedPlaceholder = NSAttributedString(string: "password",
17             attributes: [NSAttributedString.Key.foregroundColor: UIColor.systemGray3])
18         confirmPasswordField.attributedPlaceholder = NSAttributedString(string: "confirm password",
19             attributes: [NSAttributedString.Key.foregroundColor: UIColor.systemGray3])
20         //Closure to instantiate transparent uiview object
21         let transparentView: UIView = {
22             let tv = UIView()
23             tv.backgroundColor = black
24             //Transparent
25             tv.alpha = 0.3
26             //Rounded corners
27             tv.layer.cornerRadius = 15
28             //Can be autoresized by programmatic constraints
29             tv.translatesAutoresizingMaskIntoConstraints = false
30             return tv
31         }()
32         //Set up a background image and stretch it to all edges of the screens
33         let background = UIImage(named: "Background")
34         var imageView : UIImageView!
35         imageView = UIImageView(frame: view.bounds)
36         imageView.contentMode = UIView.ContentMode.scaleAspectFill
37         imageView.clipsToBounds = true
38         imageView.image = background
39         imageView.center = view.center
40
41         //Add the transparent view to view
42         view.addSubview(transparentView)
43         self.view.sendSubviewToBack(transparentView)
44
45         //Set transparent view constraints so it sits behind the fields stack
46         NSLayoutConstraint.activate([
47             transparentView.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor, constant: 130),
48             transparentView.centerXAnchor.constraint(equalTo: view.centerXAnchor),
49             transparentView.widthAnchor.constraint(equalToConstant: fieldsStack.frame.width + 56),
50             //transparentView.heightAnchor.constraint(equalToConstant: fieldsStack.frame.height + 112)
51             transparentView.bottomAnchor.constraint(equalTo: topLButton.bottomAnchor, constant: 16)
52         ])
53         //Add the background to view
54         view.addSubview(imageView)
55         //Send it to the back of all the subviews
56         self.view.sendSubviewToBack(imageView)
57     }
58 }
```

Therefore when setupView() is called from ViewDidLoad():



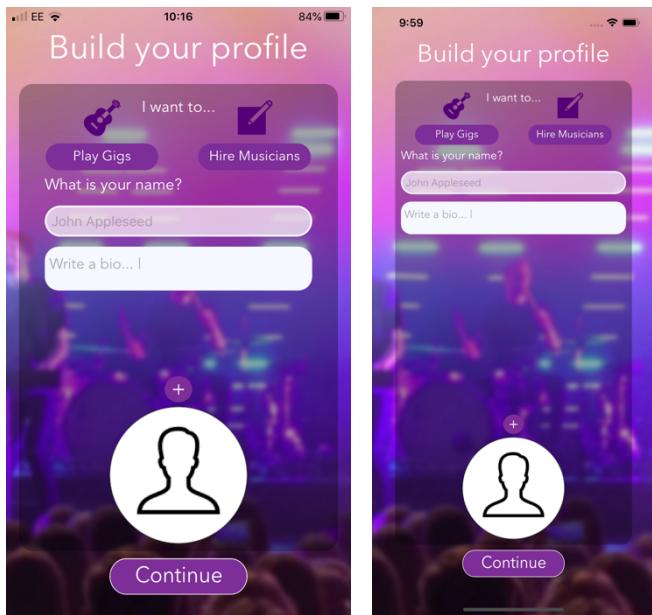
## Test Plan

Test No	Test Explanation	Success Criteria Point/s	Input Dataset/s	Expected Outcome	Valid/Invalid
1	Compare features to the app design	1 – Aesthetically pleasing UI		The look of the program should match the layout and design of the solution, unless improvements are decided in development and testing.	Valid
2	Check auto-layout constraints work as expected	1 – Aesthetically pleasing UI	Auto-layout constraints	Running the app on the minimum iPhone specified and upwards should show a similar layout. No features should be inaccessible on a particular device.	Valid

## Test 1 & 2 – Check Designs and Constraints

I tested on my physical device (iPhone 6s) and used the iPhone simulator to test the designs and constraints from iPhone 5C to iPhone XS Max. The screenshots therefore represent the two appearances of iPhones with and without a notch (cut-out at the top of display for camera).

### CreateProfileCAVC:



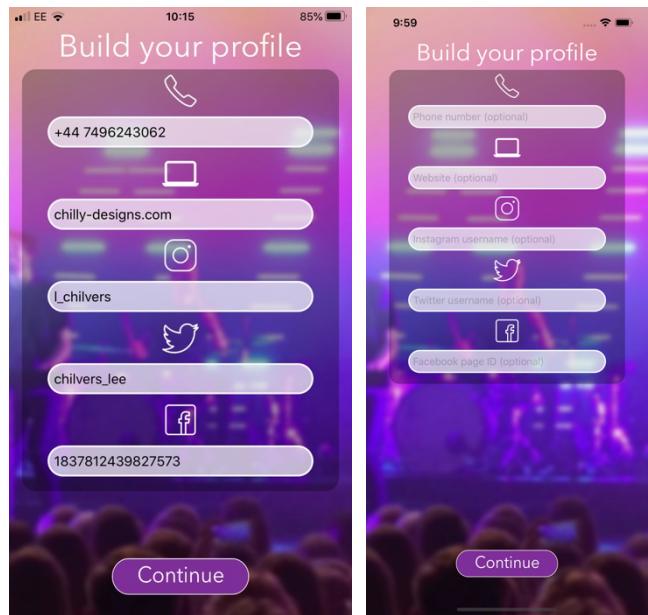
From my designs, I changed the top button designs and used symbols to reduce text. I have kept the general look of the view.

I gave the table view a nice purple gradient background. The simulator didn't show it, but it does work on physical devices. I changed how the links are arranged here, it looks neater if they are all in the same horizontal stack.

### UserAccountVC:



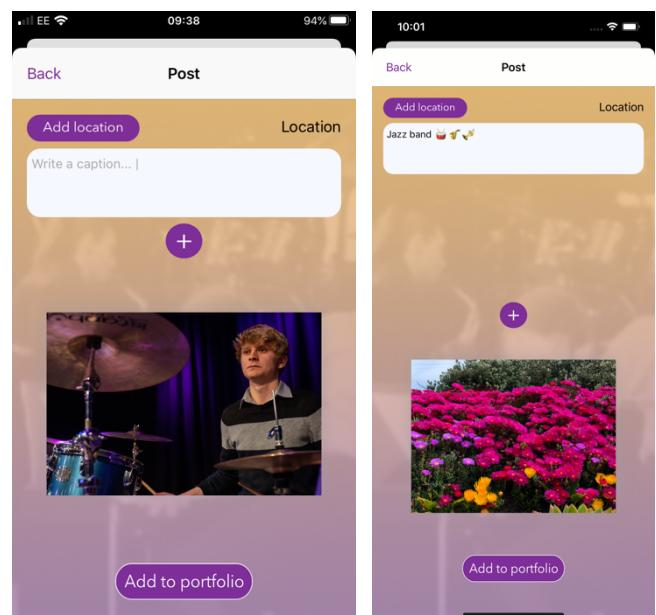
### SocialLinksCAVC:

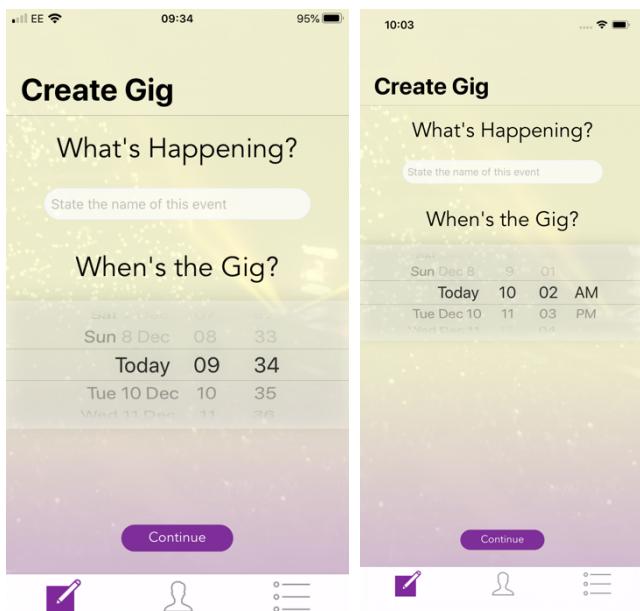


This is slightly different to my designs in that all input fields have symbols and is spaced evenly in a stack. Transparent views provide larger contrast.

I provided a contrast here using an orange to purple gradient and kept all buttons sticking to the purple colour scheme. *(The popover UI is covered in the next stage of development)*

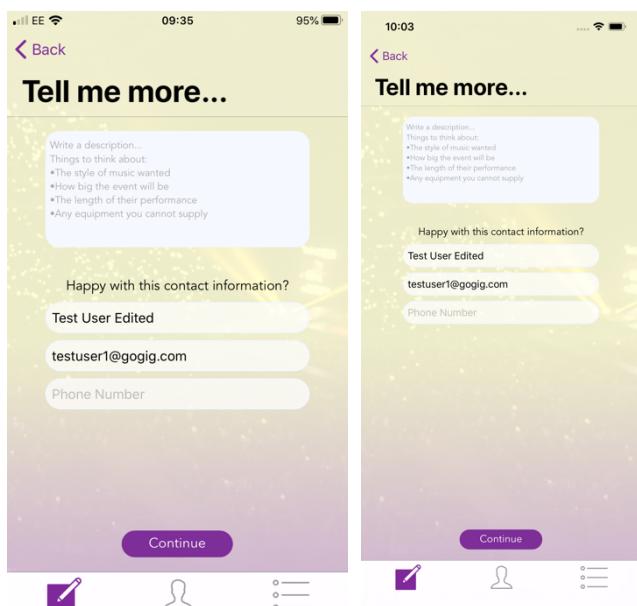
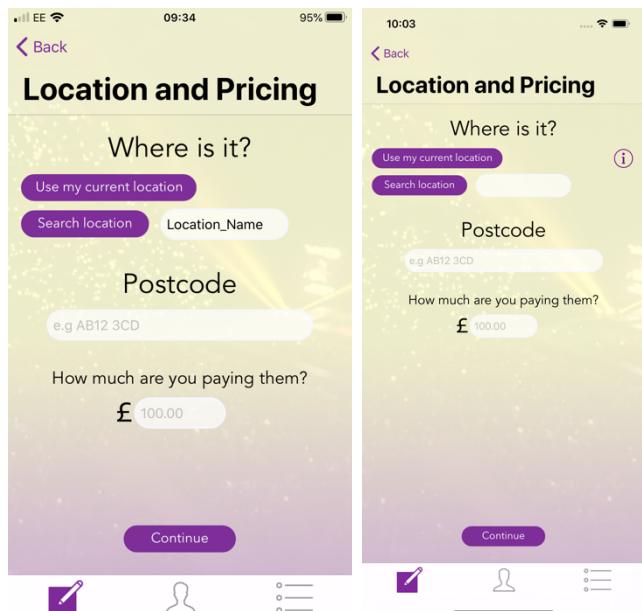
### PortfolioPostVC:



TitleDateCGVC:

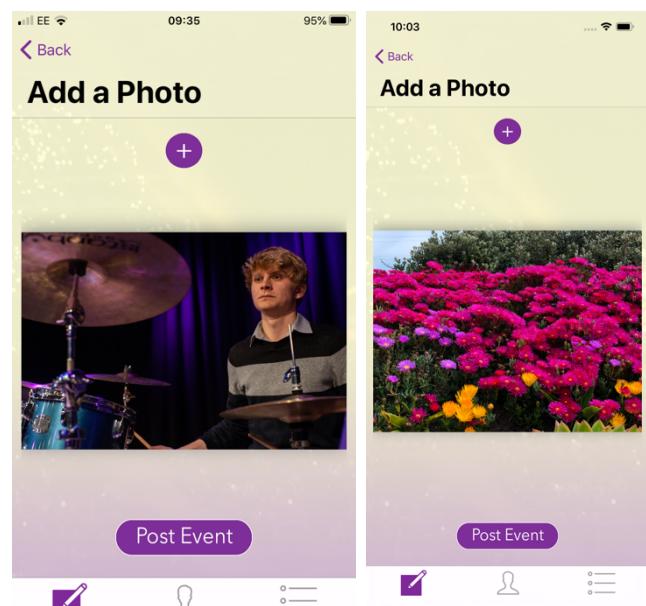
I used a photo of a concert with a yellow to purple gradient instead of just using a yellow gradient. I also made the UI elements transparent and gave the views large titles to fit with iOS 12 common designs.

Each gradient used has purple as the bottom stop to simulate the continue or tab bar 'glowing'. I mainly stuck to my design but was efficient with space and put instructions for the description as a placeholder inside the text view.

InfoContactCGVC:LocationPriceCGVC:

The layout of this view has slightly changed, I provided a text box to manually add location string. I also said I would use buttons with blue text, however, it looked ugly and inconsistent with the rest of the app (Nielsen's Heuristics) so the user could get confused.

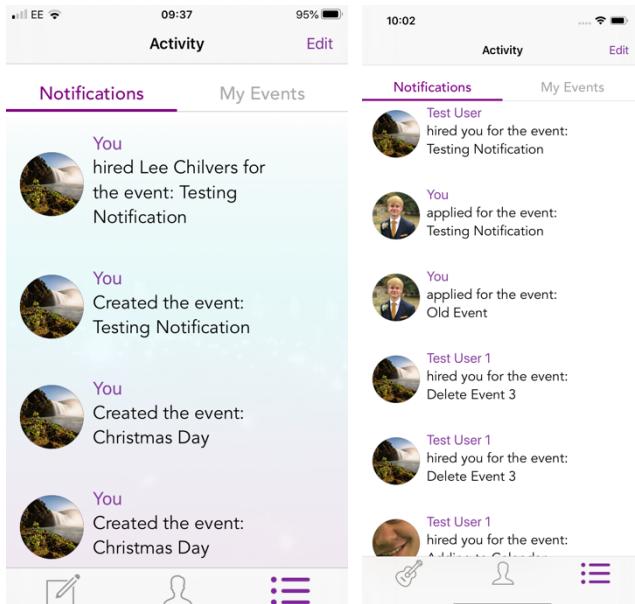
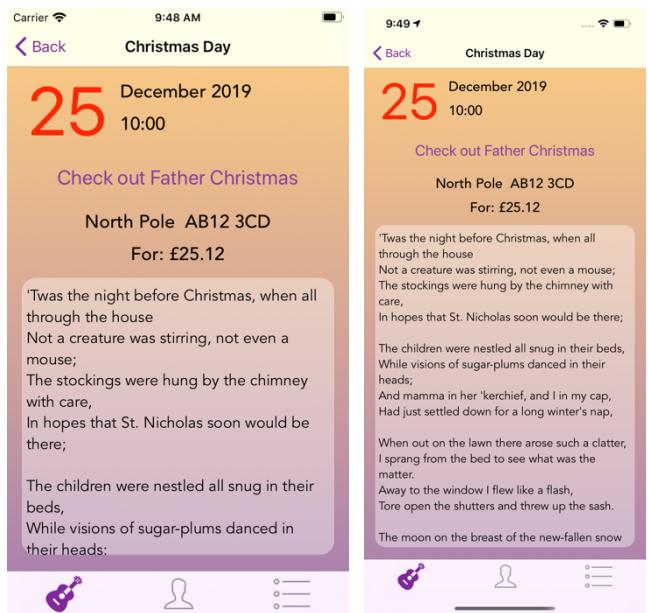
This is similar to my designs and I have stuck with purple buttons. The images have been given a small shadow to provide depth to the view. 'Back' buttons are all purple and clearly labelled.

PhotoCGVC:

FindGigVC:

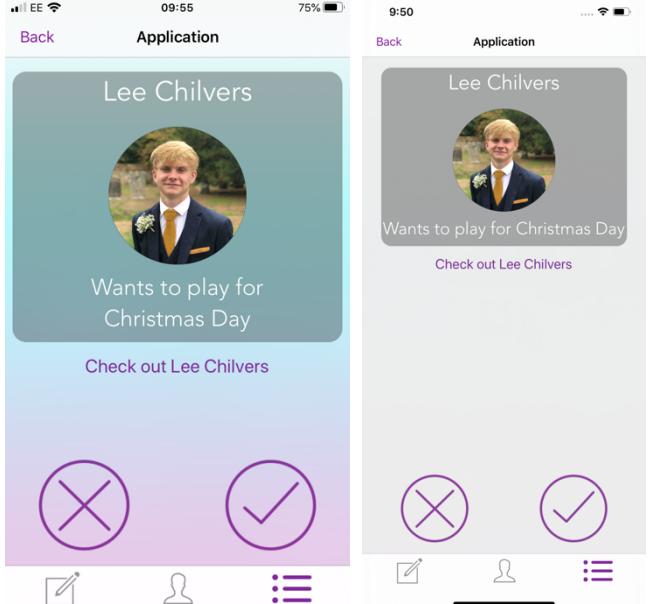
This view has been given the same background for creating an event. A transparent view has been put behind the contact information for contrast and readability. The cards have been given a shadow too, which gives the stack of cards a better visual effect. All the constraints worked for these cards on all iPhones too.

This tab was given a blue and purple colour gradient (again background didn't appear on simulator). I removed the rubbish bins and swapped to a minus symbol to delete, this is still a meaningful picture which explains its function.

ActivityFeedVC:EventDescriptionVC:

This view has been given the same gradient as the gig cards, so it appears as though the user has zoomed in on the card to look at it in more detail. Constraints adjusted the UITextField so more information is shown on a larger screen, so less has to be scrolled through.

All buttons I swapped to purple for this view to keep consistent so that the user knows it performs a subroutine. I also stuck a transparent view behind the important information to provide contrast and highlight the important information.

ReviewApplicationVC:

## Milestone Review

The constraints took a long time to get right but after doing a few views, I got the hang of what was required so that the same user interface remained consistent on all iPhones. I could only test on my own physical device (iPhone 6s) and then all other iPhones on the simulator. I can therefore only assume that they work until I meet with my end users to check all the constraints on their devices (iPhone X).

I also feel any changes I made to my designs in the front-end development are justified with a better user experience meeting Neilsen's Heuristics of good user interaction with the application. The way I programmed these visual effects is by creating an extension function for each UIViewController in a separate file so that they can be called when the view loads. This is a modular, clean way of programming keeping it readable and the front-end and back-end code separate. My app is now complete in meeting its purpose with all user interface finishing touches.

I checked with my end users at the end of this milestone:

Gavin Thorold: "You have improved on the designs you sent me originally and have really brought the idea to life! I particularly love your use of colour and you have used space wisely. It looks really professional."

Jude Mills: "These designs look really cool on my iPhone and looks like a proper app ready for the App Store. I really like how everything I have tested until this point has finally come together so that I can appreciate the app as a whole."

The constraints worked on their iPhone X (and background images appeared on their device too!). Therefore, I assume that the simulator reflects everything working on all physical devices.

At the end of milestone 12, my end users agreed I met objectives 1 and 2 from the success criteria.

# Adding Events to the Calendar and iOS 13 Maintenance

## Development Log

I wanted to push myself to meet one non-essential point of the success criteria and the most important (which Jude requested in the design section) was to add events to the device calendar so that users could receive reminder notifications about it closer to the time.

I had to **import EventKit** and created an extension function of UIViewController:

```

231     func addEventToCalendar(title: String, description: String?, startDate: Date, endDate: Date, completion: ((_ success: Bool,
232                               _ error: NSError?) -> Void)? = nil) {
233         let eventStore = EKEventStore()
234
235         //Add to event to calendar
236         //Request calendar access
237         eventStore.requestAccess(to: .event, completion: { (granted, error) in
238             //If given access
239             if (granted) && (error == nil) {
240                 //Create an event
241                 let event = EKEvent(eventStore: eventStore)
242                 //Set title, start and end, and any notes
243                 event.title = title
244                 event.startDate = startDate
245                 event.endDate = endDate
246                 event.notes = description //Notes is the description of the event
247
248                 //Set calendar event will be saved to
249                 event.calendar = eventStore.defaultCalendarForNewEvents
250                 do {
251                     //Save the event with completion
252                     try eventStore.save(event, span: .thisEvent)
253                     //Process any errors
254                     } catch let e as NSError {
255                         completion?(false, e)
256                         return
257                     }
258                     completion?(true, nil)
259                 } else {
260                     completion?(false, error as NSError?)
261                 }
262             }
263         }

```

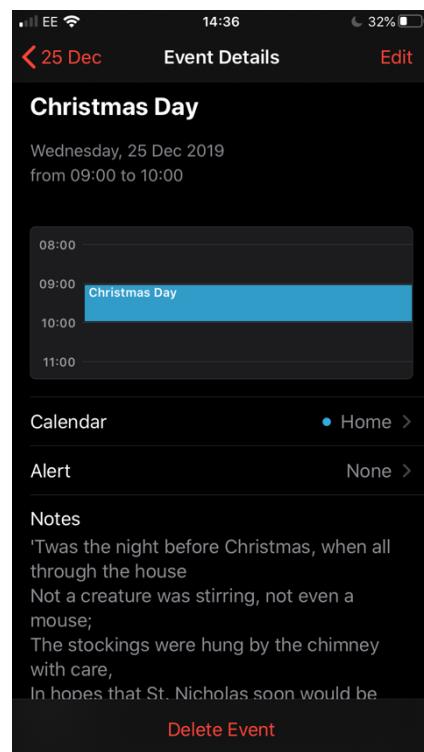
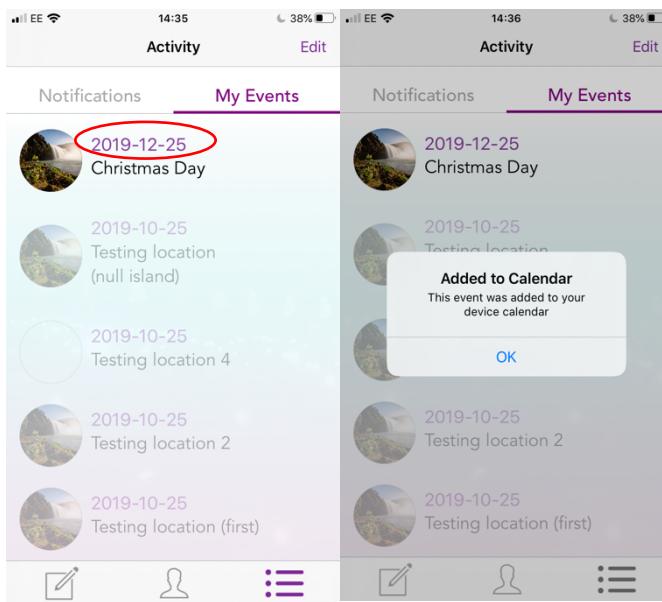
And call the function when the user clicks on the date in the 'My Events' section of the ActivityFeedVC:

```

264         //My Events Section
265     } else {
266         //Get the GigEvent object
267         let calendarEvent = usersEvents[row]
268         //Use it to add an event to the calendar
269         addEventToCalendar(title: calendarEvent.getTitle(), description: calendarEvent.getDescription(), startDate:
270                           calendarEvent.getDate().addingTimeInterval(-3600), endDate: calendarEvent.getDate())
271         //Notify the user that it has been added
272         displayError(title: "Added to Calendar", message: "This event was added to your device calendar")
273     }

```

Therefore, when the user clicks the date it gets added to the calendar:



I could improve on this slightly and create a dedicated calendar purely for these events (within the same function).

```

247
248 //Check to see if "GoGig" calendar has been created - First time only
249 if DEFAULTS.object(forKey: "GoGigCalendar") == nil {
250     //If new create the calendar
251     let newCalendar = EKCalendar(for: .event, eventStore: eventStore)
252     //Calendar title
253     newCalendar.title = "GoGig - My Gigs"
254     //Colour
255     newCalendar.cgColor = UIColor.purple.cgColor
256     let sourcesInEventStore = eventStore.sources
257     newCalendar.source = sourcesInEventStore.filter {
258         (source: EKSource) -> Bool in
259         source.sourceType.rawValue == EKSourceType.local.rawValue
260     }.first!
261     do {
262         //Create new calendar (first time only, then set identifier with UserDefaults)
263         try eventStore.saveCalendar(newCalendar, commit: true)
264         DEFAULTS.set(newCalendar.calendarIdentifier, forKey: "GoGigCalendar")
265         print("new calendar created")
266     } catch {
267         self.displayError(title: "Oops", message: "Something went wrong")
268     }
269 }
270
271 //Set calendar event will be saved to
272 event.calendar = eventStore.calendar(withIdentifier: DEFAULTS.object(forKey: "GoGigCalendar") as! String)

```

So now when the user adds the event, it is under its own 'GoGig – My Gigs' calendar which is purple:



As you can see, there's not a high enough contrast between the purple and the dark tab bar. In addition, all black text will automatically appear white in dark mode, so therefore most of the text in my views were unreadable.

To solve this, I had to change a property in the info.plist file:

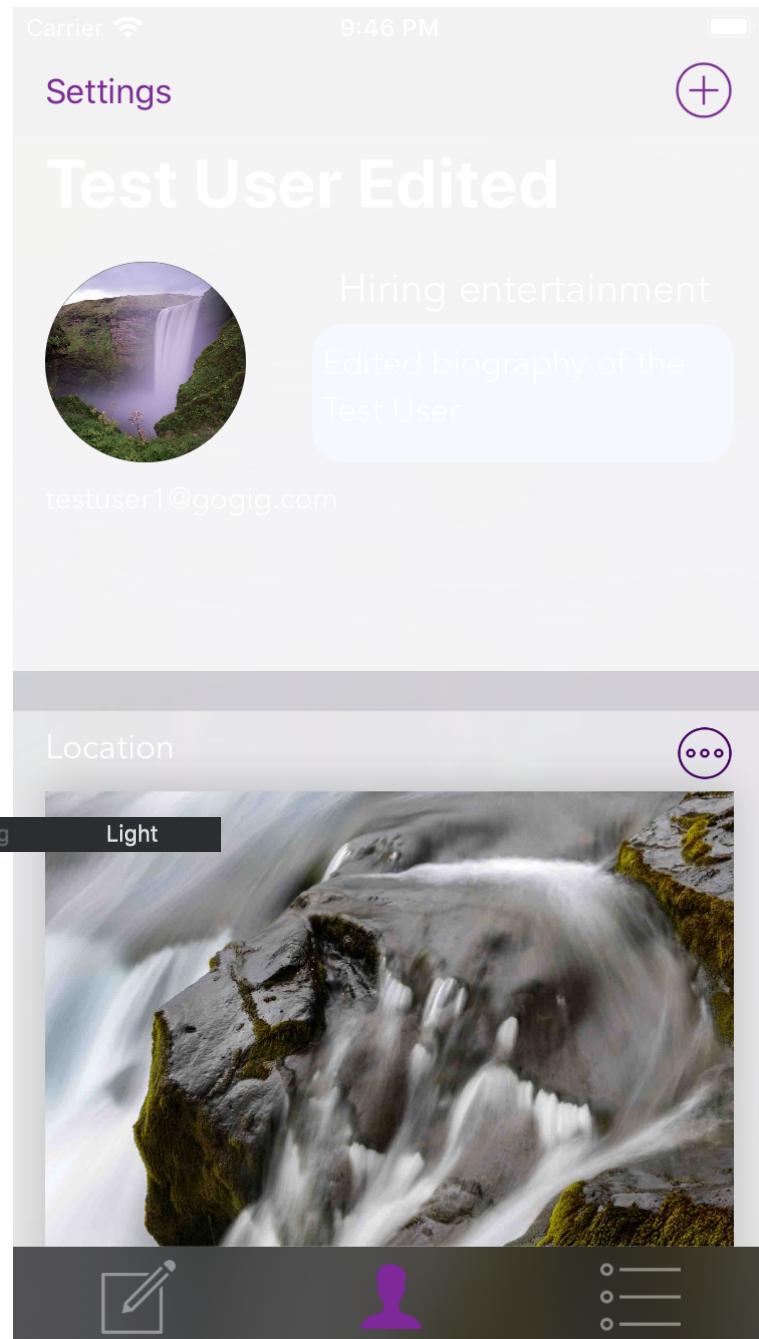
User Interface Style String

This means that even if the user has enabled dark mode for iOS 13, my app's elements will always appear light. For future development, I could allow dark mode, I would just have to adjust the contrast between colours of icons and text.

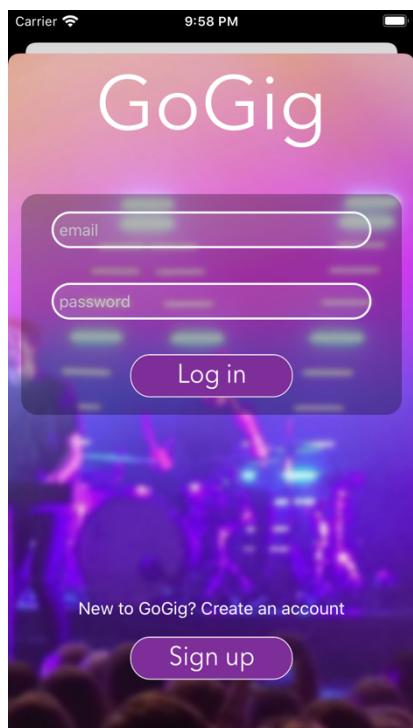
## iOS 13 Maintenance

At the end of September 2019, Apple released iOS 13 and the Xcode 11 IDE. The new IDE and operating system made changes to my project in various places.

The first example is that iOS 13 comes with dark mode. Therefore, whatever the user has chosen in their system preferences, UIViewController elements will appear in that mode. For example, if dark mode is enabled the portfolio will appear like this:



The second change that occurred was that whenever a view is presented modally, there is a new card-like UI design. My main problem was with the LoginSignupVC when it is presented:

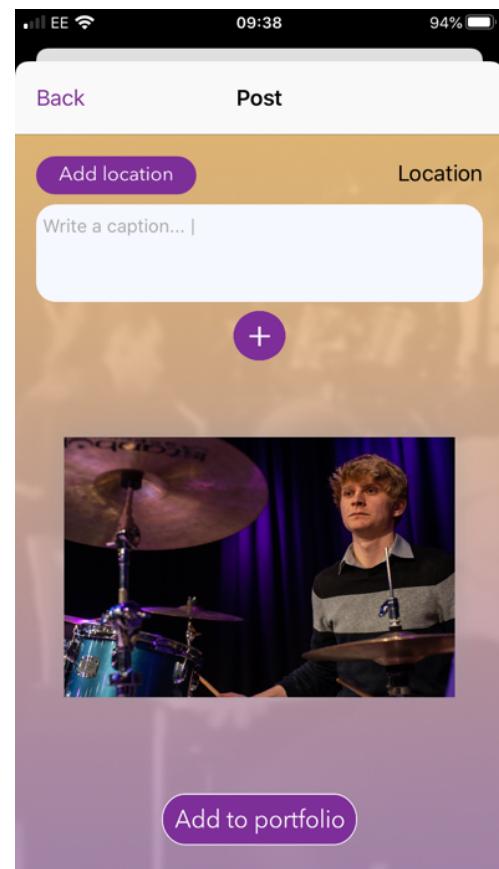


While it looks visually effective, this view can be dismissed by dragging downwards from the top. For the login page, it means that a user can access the rest of the app without being logged into an account at all which means we can never fetch data from the database as there is no current user to make reference to. The fix for this was to provide every view that gets presented modally with this line:

```
self.modalPresentationStyle = .overFullScreen
```

Ensuring the view will be full screen as before.

I did decide to keep this style for the PortfolioPostVC though, as dismissing the view with a swipe is a nice feature and the portfolio can be seen behind the view. As shown in milestone 12, it looks like this:



My final bug came from uploading a video to Firebase Storage with my current code, this error is printed to the console:

```
Error Domain=FIRStorageErrorDomain Code=-13000 "An unknown error occurred, please check the server response."
```

I found that Google Firebase had not updated their API for iOS 13 and the file paths when trying to upload were different. The issue was with the .putFile() method. I therefore tried to take the same approach as uploading an image and convert it to data rather than uploading as a file, (since there were no upload bugs with images).

Before:

```

554 func updateSTVid(uid: String, directory: String, vidContent: URL, imageID: String, uploadComplete: @escaping (_ status:
555     Bool, _ error: Error?) -> ()) {
556     //Uploading the content with unique string ID
557
558     //This time we use .putFile to upload the URL and not imageData
559     REF_ST.child(uid).child(directory).child(imageID).putFile(from: vidContent, metadata: nil, completion: { (metadata,
560         error) in
561         if error != nil {
562             uploadComplete(false, error)
563             return
564         }
565         uploadComplete(true, nil)
566     })

```

After:

```

554 func updateSTVid(uid: String, directory: String, vidContent: URL, imageID: String, uploadComplete: @escaping (_ status:
555     Bool, _ error: Error?) -> ()) {
556     //Convert to data and upload the data to Storage
557     if let videoData = NSData(contentsOf: vidContent) as Data? {
558         REF_ST.child(uid).child(directory).child(imageID).putData(videoData, metadata: nil, completion: { (metadata, error)
559             in
560             if error != nil {
561                 uploadComplete(false, error)
562                 return
563             }
564             uploadComplete(true, nil)
565         })
566     }

```

After uploading to Storage however, Firebase didn't know how to make sense of the video data. It was just stored as a text file with a long string of characters. The knock-on effect was trying to add the video URL to the PostContainerView to play the video didn't work because it didn't know what to do with the data. I therefore had to specify the MIME type at upload:

```

569 let metadata = StorageMetadata()
570 metadata.contentType = "video/quicktime"
571 //Convert to data and upload the data to Storage
572 if let videoData = NSData(contentsOf: vidContent) as Data? {
573     REF_ST.child(uid).child(directory).child(imageID).putData(videoData, metadata: metadata, completion: { (metadata,
574         error) in
575         if error != nil {
576             uploadComplete(false, error)
577             return
578         }
579         uploadComplete(true, nil)
580     })
581 }
582 }

```

So that in Storage:

	9BF5D59A-8E39-4F76-92F1-9D9677AB90A0.mov	2.2 MB	application/octet-stream
Became:			
	0902052C-2EC6-488E-B17B-C5CBED38E3F6.mov	2.2 MB	video/quicktime

# Section 4 – Evaluation

## Stakeholder Test Plan

I got both my stakeholders together to complete these series of tasks at the same time. Due to different actions required by both people, these tests will be split by each user when required:

Test No	Task for End User	Success Criteria Point/s	Justification	Valid/Invalid	End User Comments
1	Open the app from the iPhone home screen	2 – Easy navigation	Users should be able to find the app and search for it from the phone home screen.	Valid	Gavin Thorrold – Organiser App icon is bright and can be searched for Jude Mills – Musician I like the icon design
2	Login view swaps between Sign up and Log in	1 – Simple and understandable interaction 2 – Easy navigation	So that a user can choose to login or create a new account at any time.	Valid	The view swaps between the two and is easy to understand. I particularly like the placeholders on the email and password fields telling me what to enter.
3	Can navigate to account creation process and you can enter your information with ease	1 – Simple and understandable interaction 2 – Easy navigation 8 – Personalise their experience	So that users can start creating their account	Valid	Takes me to create my account. Works as expected and tells me what I'm about to do.
4	Try entering invalid data (leaving a field blank or adding a name more than 50 characters)	2 – User errors are reported back and easily recoverable	Check to see if the app prevents invalid data entry and prevent progression if anything wrong is entered.	Invalid	The app told me I had to choose a type of user and input a name before continuing. Tried entering my name in the email field and it reported the unexpected input. Trying to write a name more than 50 length does not let me write any more.
5	Is the action sheet shown when you add a profile picture? And is an image returned when you use the	8 – Able to access the iPhone camera and library	Allows the user to add a picture for their profile to personalise the experience.	Valid	I can choose the pictures from my photo library and camera. I've entered a picture of my band performing as my profile picture.

	camera or pick from the image library?				
6	Add social links when creating an account, does the app report back if you have entered something wrong?	2 – Report back error 9 – Social media links	This test ensures that the system reports back anything wrong with the social links that they provided.	Valid and Invalid test data	I entered jibberish into the website field and it doesn't let me continue. I added my social media accounts and it seemed to like them. Adding a Facebook Page ID is not easy to understand though. I've input all of the social media my band is on. I like how it's all optional as my band doesn't have a website.
7	Progress with 'Continue' without adding any social information. This feature should be optional.	2 – Navigation 9 – User decides on information associated to account	This test ensures that the social links are optional and an account can be created without them.	Valid	Works. Works.
8	Only a musician can navigate to the music links view	2 - Navigation	This test ensures that only musicians have the option to add a music links as only they can navigate to it.	Valid	I reached this screen and inputted both my Apple Music and Spotify links.
9	App reports back when music links are added incorrectly and can even progress without adding any at all as this is an optional feature.	2 – Report back error 9 – User decides on information associated to account	The test makes sure that music links are optional and a musician account can be created without them.	Invalid	Does the same thing if I miss out one or both of them.
10	Do you get taken to the portfolio view where you can see your account data just entered?	2 – Navigation 5 – Data is held under account	Test ensures the app takes them to the portfolio view automatically.	Valid	Works. Works.
11	Press 'Settings' and choose to log out. Is the login page presented	5 – Working authentication system	Test ensures a user can log in and out of their account and	Valid	Works as I expect a login system to work. Yeah the login works as expected.

	and can you log into your account again. Your data should be seen same as before. Login to each other's accounts to make sure Authentication works.		even access their account on a different device with Authentication.		
12	Double tap the iPhone home button and flick the app up. This closes the app. Now press the app icon again to launch it. You should be taken straight to your portfolio.	5 – User is remained logged in until they choose to log in	Test ensures that they remain logged in ready to receive push notifications later on.	Valid	Keeps me logged in which is good. I'm taken straight to my 'portfolio' as expected.
13	Navigate to 'Add a Post' page, and choose a post from your camera, image library and video library.	2 – Navigation 8 – Able to access camera and library	Test ensures that the user can post whatever they like to be shown in their portfolio which potential employers will see.	Valid	I really like the popup design of this page. I can choose both videos and photos and preview them before posting them.
14	If you click 'Add Location' is a search view presented (however searching will not work)	7 – Location features	This test ensures that the Google API view gets presented to the user despite the fact they can't use it due to the fact it needs to be paid for.	Valid	It appears but I can't use it. Just says "Can't load search results".
15	Try posting without choosing an image (or video)	2 – Report back error	The app should stop the user making a post without actually adding an image or video.	Invalid	Popup tells me there is nothing to post. Tells me I can't make a post.

16	After making a post, you should be taken back to your portfolio and your posts should be in the feed in reverse chronological order.	5 – Post Data held under account 6 – Post data stored efficiently	Test ensures that a portfolio is built with all the posts' data shown in the table view cells. Other users will see this later.	Valid	<p><b>It has been added for me to view and is in the correct order.</b></p> <p>I can watch the video I just posted. I really like the play button design too.</p>
17	Clicking the three dots by the post you can delete one of your posts.	4 – Users can edit anything associated to their account 6 – Efficient database storage 9 – Users decide on amount of information they share	This test will make sure that a post is removed from the database and from the portfolio view so it cannot be seen by any user.	Valid	<p><b>Gets rid of it as expected.</b></p> <p>It works and deletes the post.</p>
18	Make sure only three tabs are seen at the bottom of the screen. Musician should see a guitar and an organiser should see a notepad. Is this consistent when you close the app and log in and out?	2 – Easy to navigate 3 – Avoid error	This test will make sure that users can only navigate to the areas of the app which belong to that type of user.	Valid	<p><b>My tabs are seen. I have a little Notepad at the bottom left, I get the guitar when I log into Jude's account.</b></p> <p>The guitar symbol is there along with the central and right-hand tab. Stays like this when logging out and in.</p>
19	Click on every social link in your portfolio.	9 – Social Media links	This test ensures that the links entered do in fact open up the correct apps and show the user's profile.	Valid	<p><b>Takes me to all of CEG's profiles on social media apps and takes me to Safari to our website.</b></p> <p>Yep this takes me everywhere I expect to go.</p>

**Gavin Thorrold (Event Organiser)**

20	Click on the left-most (notepad) tab.	2 – Easy navigation	This checks that the organiser can navigate to create a public event.	Valid	Works.
----	---------------------------------------	---------------------	---	-------	--------

21	Try and enter some invalid data (miss out a field and try to select a date back in time)	3 – Avoid error	Making sure that only valid information about the event being created can be entered.	Invalid	Entering an old date corrects me and takes me to the current date. It won't let me continue without adding a title.
22	Press 'Use current location' does the compass icon come up next to your battery percentage?	7 – Location services are accessed	This makes sure that the device location can be accessed and used when creating an event to sort by distance later.	Valid	I had to request permission for the app to use my location. Compass appears so I assume my location is being used.
23	Choose an image for the event.	8 – Camera and photo library access	This test ensures that an image URL is attached to the event before posting so it will be shown to the musician later.	Valid	Works.
24	Click 'Post Event'. Are you taken to your activity feed with the most recent notification being "You created the event: eventName"?	2 – Navigation 3 – Check all input against validation 4 – User can review anything associated to their account	This test will tell the organiser their event has been created.	Valid	I'm taken to the right-hand tab and can see the notification specified.
25	Click 'My Events' in the menu bar, the event you just created should be seen there.	4 – User can review anything associated to their account	Test ensures that the event can be viewed and edited by the event's organiser.	Valid	The event I just made is there.
26	Click on the notification in the 'My Events' section	2 – Navigation 4 – User can review anything associated to their account	Test ensures that both users can see all the information about the event in detail.	Valid	Takes me to view all the information I entered about the event.
27	Click 'Edit'. Are you taken to the same page as when you created the event? Try	4 – User can edit anything associated to their account at any time.	Test makes sure that the organiser is able to change features of	Valid and Invalid test data	Navigation works and the event is updated in the activity feed when I changed the picture and description about it.

	changing some of the values you entered and edit the event.		their event. It also checks that information is auto filled out so that if anything is not changed, it keeps the same value.		
28	Close the app	N/A	Gavin is ready to receive push notifications from Jude Mills	N/A	
<b>Jude Mills (musician)</b>					
29	Gig opportunities appear created by Gavin Thorrold, click "Tell me more". Click "Check out Gavin Thorrold" to view his portfolio.	1 – Contact made between musicians and organisers 2 – Easy navigation	This test makes sure that the musician can navigate to read the event in full detail. It also tests that they can go and view the other user's portfolio before making a decision. Here they can click on social links and view the portfolio like they did their own before.	Valid	Card appears and I can click the button which tells me about the event in detail. I can view all of Gavin's posts in his portfolio as well.
30	Navigate back to the previous view and drag the card right. There will either be more gig opportunities created by Gavin or you will be notified that there are no more to apply to. When there are no more,	1 – Simple and understandable interaction. 3 – Avoid error to avoid confusion	This test makes sure that musicians can apply to events and that they do not appear again when the view is refreshed as they have already been interacted with.	Valid	I really do like the design of this view; it has nice colour and the animation for the card is nice. Refreshing doesn't do anything but I assume that is because I have seen all of the available gigs already.

	press the refresh button.				
31	Close the app.	N/A	Jude Mills is ready to receive push notifications from Gavin	N/A	
<b>Gavin Thorrold (organiser)</b>					
32	You should have a push notification come through from Jude Mills applying to your event. Go to notification centre and open the app through the notification.	1 – Good user experience 1 – Contact made 2 – Easy navigation 5 – Receive notifications and activity updates	This test is to make sure that an organiser receives a push notification when a musician applies to their event.	Valid	The notification is informative and comes through. A little red badge does not appear on the front of the app icon like a typical text message would though.
33	Click on the notification in the activity feed telling you that Jude has applied. Click “Check out Jude Mills” to view his portfolio.	1 – Contact made between musicians and organiser 2 – Easy navigation 4 – Review anything associated to their account. 5 – Activity updates	This test will make sure that the activity feed updates that Jude has applied and will also test if the organiser can view a musician’s portfolio and click on all his social links.	Valid	I can click on the notification and it takes me to my notifications. I’m taken to Jude’s portfolio as well so I can see his band’s work.
34	Navigating back, accept Jude’s application by clicking the big tick.	1 – Contact made 1 – Good user experience	Test ensures that organiser is sent back to the activity feed after replying to an application.	Valid	I’m taken back to my notifications and it is really clear how to accept Jude’s application.
<b>Jude Mills (musician)</b>					
35	You should receive a push notification telling you that Gavin accepted you to play. Open the app from the notification.	1 – Good user experience 1 – Contact made 5 – Activity updates	This test ensures that the musician receives push notifications too letting them know if they were rejected or accepted.	Valid	I got the notification and I can click on it to open the app.

36	You should see a notification in the feed telling you that you that you are hired.	1 – Contact made 5 – Activity updates	Test ensures that Jude gets a notification telling him he is booked to play for Gavin's event.	Valid	Works.
37	This means that if you click "My Events", you can view all the events you are booked to play for here and view them in detail.	2 – Easy navigation 4 – Can review anything associated to their account at any time.	Test ensures that the gig object has been associated to the musician for them to refer back to.	Valid	The event Gavin has put together is in the 'My Events' section, I can click on this in detail and view Gavin's portfolio whenever which I like.

**Both Users**

38	Click the date of the event in "My Events" section and navigate to the calendar app on the iPhone.	Non-essential point covered: 10 – Smartphone calendar integration.	This test ensures that the event and its description is added to the device calendar so that the user can receive alerts when it's closer to the time.	Valid	I'm told the event was added to my calendar through a popup. This is a really handy feature! I'm constantly forgetting I have a gig coming up.
39	Navigate back to your portfolio, click "settings" and "edit profile". You are taken to the same screen as when you first created your account. Change a few items such as your profile picture.	2 – Easy navigation 2 – Report back error. 3 – Check all input against validation rules to avoid error and confusion 4 – User can edit anything associated to their account 6 – Efficient data storage in Firebase 8 – Access camera and photo library 9 – Decide on amount of	This test will check to see if the user can edit their account information and this change will be reflected everywhere necessary in the app. Allowing the user to edit their information gives them freedom in the app's usage.	Valid and Invalid test data	This time I took a profile picture with the camera and changed my bio. My accounts has been updated. I added my band's Youtube under the website bit. It works because I'm taken to Safari to watch my video.

	information that they share. Social media is optional.			
--	--	--	--	--

## Analysis

The 39 tests show that my app can be used without any difficulty and all the features I developed worked. I believe have met the standard of quality my end users wished for. The only criticism in the testing session was that both users had to ask what a Facebook Page ID was and how to add one. I am in agreement that adding this is not particularly the most obvious thing.

## Questionnaire

12. On a scale of 1 – 10, how much did you like the design of the app and its user interface?
13. On a scale of 1 – 10, how satisfied were you with the user experience?
14. Were there any stages when you didn't know what to do as part of the testing session?
15. Did you spot any errors or bugs when you used the app? If so, what were they?
16. On a scale of 1 – 10, how satisfied were you with the error messages for your invalid data input?
17. On a scale of 1 – 10, how much user freedom in the app's navigation did you feel you had?
18. On a scale of 1 – 10, how much did you feel that you decided how much information you share?
19. On a scale of 1 – 10, how satisfied were you with the personalisation options of your account and user experience?
20. Are you comfortable with the app using your current location to make results more relevant? If not, why?
21. Are you happy with how I have tackled the problem and satisfied with my solution? If not, how could it be improved?
22. On a scale of 1 – 10, how much do you feel this app is a good solution to helping musicians and organisers get in contact and organise music events quickly?

## Analysis

Question 1 both users gave an 8. They liked the colour scheme and how everything was structured with tabs at the bottom. The said buttons were well labelled, and the UI was consistent everywhere in the application.

Question 2 Jude gave an 8 and Gavin gave a 7. Jude liked the draggable card feature most and Gavin really liked how the portfolio system works. Gavin noticed however that videos take noticeably longer to load and play than an app such as Instagram. Gavin also wanted little red numbers in the app and on the app icon whenever you receive a push notification letting you know that there is something to look at.

Question 3 both users referred to the Facebook Page ID problem when inputting social media links. Jude mentioned that on initial usage, musicians wouldn't know they had to drag the gig card left and right but would be easy to figure out. He only knew due to our correspondence. Other than this they agreed that buttons and icons were appropriately named and designed.

Question 4 both users responded that they didn't notice any bugs and everything ran as they were told it would do

Question 5 both gave a 9 and said that the error popups were informative and helpful if they missed anything.

Question 6 users gave a 9 and said they could go to any tab whenever they wanted and do anything at any time.

Question 7 Gavin gave a 5 and Jude gave a 6, the main reason why this suffered, they reported, was that users had to select a profile photo when signing up. Other social media apps let you create an account without choosing a photo. But they liked how social media and location features were all optional.

Question 8 both users gave a 7. They liked how you can edit your account and choose what images to post to make the portfolio your own. An idea they collectively put together was perhaps I could give the user choice over the colour scheme of the portfolio to make the space their own.

Question 9 both users said they didn't mind the app using their location because it enhances their experience. I confirmed that it can be turned off in the iPhone settings at any time which they both liked if they were to change their mind.

Question 10 they both said that my solution was great and liked seeing my ideas come to the screen. Jude would like to perhaps post adverts of his own upcoming performances which he has put together, which I think may be an essential point if I were to develop the application further.

Question 11 Gavin gave a 7 and Jude gave an 8. They both agreed it's a good app which achieves what we planned together and would get things done quickly. Gavin said that he would use the app as an additional option but would most likely still do most of his correspondence through phone calls and emails. Building on this, Jude made the point that the app only sparks the initial contact but does not provide any way to privately message other people and so would most likely continue to use Facebook features for this purpose.

## Evaluation Against Success Criteria

**Success Criterion 1:** A good user experience is provided with an aesthetically pleasing user interface. This makes user interaction simple and understandable so that the solution meets its purpose of making contact between musicians and organisers quick and easy.

This criterion has been met as part of tests in milestones 1, 2, 6, 7, 8, 9, 10, 11 and 12. One of the main features that helps meet this criterion is the portfolio, I had a few difficulties getting the portfolio how I want it and I still think it could be improved in future development. An example being how images and videos load before they are visible when scrolling. The notification system worked really well and was an essential feature to make contact quick and easy.

**Success Criterion 2:** The system is easy to navigate, and any user errors are reported back and easily recoverable. All parts of the app are reachable at any time.

This criterion has been fully met as part of tests in milestones 2, 3, 6, 8, 9, 10 and 11. My app provides an experience where all views can be navigated to at any time. This is achieved well using the tabs at the bottom of the screen as one view can be paused and resumed while navigating to other parts of the app. My end users agreed that when invalid data was inputted, feedback was informative and presented in a nice way. I personally haven't come across any unrecoverable problems since the app has been finished and therefore, I feel this successful.

**Success Criterion 3:** The system should check all input data against validation rules to avoid error, unexpected results and user confusion.

This criterion has been fully met as part of tests in milestones 1, 2, 3, 4, 6, 7, 9, 10 and 11. My end users agreed that error was reported back and prevented with various methods in the stakeholder testing. By limiting what the user can input with various types of keyboards and UI elements, this helped make sure that the correct data was added to move forward. One thing I would try if I were to do this again is that when the user creates an account, rather than presenting a popup when they try to progress with invalid data, I would visibly disable the button until the information is entered. This method removes the usage of UIAlerts popping up all the time and may actually improve the user experience as part of success criteria 1. Jude mentioned in the questionnaire that there may be confusion about what the musician has to do to apply with the gig card. To improve, I could have included a little instruction label which wouldn't be too complicated to include.

**Success Criterion 4:** The user can edit and review anything associated to their account at any time. Important data is displayed.

This criterion has been fully met as part of tests in milestones 2, 6, 7, 8, and 10. I made sure I fulfilled this whenever a user entered anything about their account by allowing them to edit it later. Main examples would be being able to delete posts, edit event details and edit their account profile. Generally, anything which could be seen by another user needed to have an edit or delete feature in case a user realised they didn't want that data to be displayed. My end users agreed that I gave them good control in being able to review and edit their data in Stakeholder tests 17, 25, 37 and 39. The only thing that I didn't allow users to change was their type of user when they edited their account as this opened up a lot of potential bugs I did not have time to think about as part of this project. Bugs may include being able to apply to your own event or a clash in the notification types received. I am confident that I have given users power over their own data to meet this success criterion.

**Success Criterion 5:** A login authentication system holds data under an account. The app should remain logged in even when the app is not in use to receive notifications and activity updates.

This criterion has been fully met as part of tests in milestones 1, 2, 7, 8, and 11. My end users tested this as part of Stakeholder tests 11 and 12. My main problems associated with this success criterion came from logging out and back in again to another account once the app had first launched. I had many problems refreshing the tabs and to do with notification updates. When logging in and out to a different type of user, they could access different parts

of the app they weren't supposed to, however I fixed it by using User Defaults which saves data to the device to decide the tabs rather than pulling the user type from the database. Also, when logging into a different account I received notifications in the feed which were not meant for that account, this was fixed by removing Database observers. Overall, I think I have built a good working Authentication system which holds data under an account. This authentication system allows notifications to be sent to a specific device which reinforces success criterion 1 by helping make contact quickly and efficiently.

### Success Criterion 6: All data is stored efficiently with Firebase Database and Storage API.

My end users cannot measure how well I met this criterion. However, whenever an object was created, there is always a way to delete it from their account and therefore from the database at the same time which I believe to be efficient usage. Users are even encouraged to delete events associated to them in 'My Events' section as part of ActivityFeedVC as events which have passed their date are transparent to show they are no longer relevant. Meeting success criterion 4, I made sure that they get the choice to delete it. Changes in photographs mean that they are replaced in Firebase Storage and not simply added to it. I feel this success criterion was fully met as part of my personal tests in milestones 1, 3, 4, 9 and 10.

### Success Criterion 7: Location services are accessed to aid with sorting data and meet stakeholders' needs. This is optional in case it goes against users' preferences.

This criterion has been fully met as part of tests in milestones 2, 3, 4 and 5. Sorting gig cards using the distances calculated from user location works successfully and therefore the main purpose of location usage has been met fully met. The Quicksort algorithm used as part of this milestone also came in handy when sorting portfolio posts. This feature is also not required for the app's usage and still works when location is off which is very important. In terms of using the Google Places API (although it does not use location services but it is used to search location names), it is a shame that the service requires payment to use and so a couple features were not as good as they could have been such as posts and creating events. I could build my own search view controller as part of future developments using another API however to save time in development, I used this one. The main point is that the app's usage did not suffer as they can manually input a location string, this was only a fancy search tool.

### Success Criterion 8: Able to access the iPhone camera and photo library to personalise the experience and share a portfolio with other users.

This criterion has been fully met as part of tests in milestones 1, 2, 3. Meeting this criterion was as simple as developing a few helper extension functions for UIViewController which could be inherited by every class and used whenever I needed to access image usage. I had no problems accessing them and so this was met really well. My end users agreed that this was an essential and done really well in helping build a portfolio which ultimately supports making informed decisions. Therefore, by allowing users to add their own images to events and portfolio, it reinforces success criterion 1 helping each user get in touch with the people they want to.

## Success Criterion 9: Users can decide on the amount of information they share. Social media and contact sources are optional.

This criterion has been met as part of tests in milestones 5 and 11. Social contact and music links are all optional when creating or editing an account and when they are inputted, they do what they are supposed to. This gives the user complete freedom of what they want to add. The only cases when users do not get complete freedom is when they provide an email to use the service. An email is required to subscribe to Authentication and also, so users have at least one form of contact if nothing else is provided. My stakeholders mentioned as part of the questionnaire that a user has to provide a profile picture which they did not like. I would edit this in a future update as it is a really important point which most other social apps take note of. Musicians do not even have to share their social profiles if they do not want to, they can privately email an organiser after seeing the gig advert rather than applying to it. I believe I have provided a range of options but would change what was requested by my stakeholders to meet the success criterion even better.

## Non-Essential Success Criterion 10: Smartphone calendar integration.

I stretched myself to meet, what I listed as, a non-essential criterion for the project. Jude Mills wanted to add events he was set to play for to his iPhone calendar so that he can plan all his dates. I think I have done it well because initially I just added the event to the default calendar when requested, however I then created my own separate calendar under 'GoGig – My Gigs' so that they can be searched for easier in the calendar app. Ultimately this just improves the user experience which strengthens meeting success criterion 1. In addition, a user can decide in iPhone preferences whether they want to allow the app to access their calendar and therefore this is an optional feature too.

## Evaluation of Usability Features

Basing my designs off Neilson's 10 Heuristics, my stakeholders told me I designed my UI well and what I could have done better.

My design of the UI is consistent throughout all of the different views of the app. Whenever there was an interactive button, this was always presented with a purple tint so that a user knew it performed an action. The app and its input UI elements had a very simplistic design to avoid any confusion and so things could be found easily. In question 1 of the questionnaire they said they liked the colour scheme, but in question 8 said that as a personalisation option, they could choose their own colour scheme. I personally tried to stick to similar designs of built-in iPhone apps so that a user will be familiar with navigation techniques and animations. For example, using a navigation bar at the top with a back button is used all across iOS. Another example would be displaying an error as a UIAlert because the user will know that this is something of importance and is giving instruction.

The buttons had concise, large and clear labels on them so that it was obvious what action a user would take when they clicked on them. This was also confirmed by my end users in question 1 of the questionnaire.

Any symbols I used (tabs on the tab bar and compass to represent location) followed Neilson's Heuristic of a match between the app and the real world. The symbols added to the

effect to the minimalist design as it removed the need for words and meant the app was still easy to navigate. My stakeholders did not mention this as part of the questionnaire but considering they could figure out where everything was with little to no help proves that they were effective and served their purpose. Symbols such as the minus sign when deleting events is also a consistent design with other built-in iPhone apps (such as Reminders) and so a user would recognise what the button does.

As Jude stated in question 3 of the questionnaire, perhaps it wasn't obvious what a musician should do to apply to the gig presented to them. Rather than supplying a full help menu, I could add little labels in places such as this to guide them.

Neilson's Heuristics also talks about preventing error and helping users recovering from them. My end users agreed that my popups when invalid data was entered were very useful and helped them fix what they inputted. I was very careful that input from the user did not mean errors appeared to other users and I achieved this with all my validation routines and various methods such as character limits on text fields and type of keyboards used. Recognition rather than recall was another point, I made sure I included a loading spinner when objects were being uploaded to prevent user confusion so that they knew what the system was doing.

## Maintenance

My app is modular and has been developed using the Model-View-Controller design pattern. This means that if my stakeholders wanted any feature updates or discovered a bug it would be easy for me or any developer to track where the maintenance had to be made. Using this design pattern also means that I can change the design of the UI completely without ultimately affecting any of the backend algorithms because all my view code is organised in their own custom classes and extension functions of their corresponding controllers. In addition to this, I have good iOS development practices and placed all my delegate methods associated with UITableViews and UICollectionViews in extension files. Following this and the MVC pattern has helped me keep files concise and easy to maintain.

My code is also well commented. Using //MARK: where there is an important section of the code file means that this location can be searched for within the Xcode IDE. Comments explain my intentions, thoughts and mistakes so another developer can pick up where I left off when maintaining the app. This would also be essential if working as part of a team. I also used source control with git committing important changes made to my code, therefore if I need to revert back to a previous version due to a bug, I can easily do so. Finally, my variables, classes and functions have been appropriately named following good coding practices to help programmers read my code.

## Future Developments

I would like to continue work with my stakeholders to try and submit this project to the App Store. The future developments and versions to achieve this include:

### Google and Facebook Log In

People do not like to create multiple accounts and try to sign up to things all under one account if they can. Login With Facebook and Google Sign-In API would allow me to

achieve this. This would fix the Facebook Page ID issue as we can grab data straight from the user's Facebook profile given their permission and overall create a faster account creation process.

### Storing Gigs by Location

At the moment, my algorithm of fetching gig opportunities is fetching all opportunities everywhere in the world at one time. If the app were to receive many new users and became popular, downloading every event created would not be efficient and would use mobile data unnecessarily. I could either store every gig in a hash table by hash function related to their location so closest still appears first, take a similar approach to the notifications and only fetch more when the user has interacted with a few or take an AI approach and only recommend events which the user is interested in.

### Improving Contact

One way I can make sure people spend more time in the app is by providing direct messaging inside. Therefore, they do not have to resort to Facebook or texts from the portfolio because they can discuss requirements from GoGig. In addition to this, it would be great to include a 'like' and comment system in the portfolio so that a musician has feedback on their performances. These improvements would help meet success criterion 1 in bridging contact between two users.

### Extra Features

In future versions of the software it would be nice to allow musicians to add adverts about their upcoming performances which all users could see. I would also like to include Apple Pay so that organisers could pay musicians directly in the app rather than payment being considered just a label. At the moment I have not developed things such as password recovery, but Authentication provides a way to send a personalised email to the account and reset it all privately without my knowledge of the password.

### Conclusion

To conclude, the app I have developed works well and meets what my end users have asked for. I have learnt many new development techniques throughout this project, and I would like to stay in touch with my stakeholders to plan on submitting it to the App Store for release near in the future.

# Final Source Code

## AppDelegate.swift

```

1. import UIKit
2. import Firebase
3. import FirebaseAuth
4. import GooglePlaces
5. import FirebaseMessaging
6. import UserNotifications
7.
8.
9. @UIApplicationMain
10. class AppDelegate: UIResponder, UIApplicationDelegate,
    MessagingDelegate {
11.
12.     var window: UIWindow?
13.
14.
15.     func application(_ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions:
        [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
16.
17.         //Configure app with Firebase
18.         FirebaseApp.configure()
19.         //Googe Places API Key
20.         GMSPlacesClient.provideAPIKey("AIzaSyDtabGvbeAPg-
    FjUWaRQFEg2B86a3dr_gg")
21.
22.         //If we are logged out, then present the LoginVC
23.         if Auth.auth().currentUser == nil {
24.             print("No current user")
25.             let storyboard = UIStoryboard(name: "Main", bundle:
    Bundle.main)
26.             let loginVC =
    storyboard.instantiateViewController(withIdentifier:
    "LoginSignupVC")
27.             //Full screen for iOS 13
28.             loginVC.modalPresentationStyle = .overFullScreen
29.             window?.makeKeyAndVisible()
30.             window?.rootViewController?.present(loginVC,
    animated: true)
31.         }
32.
33.         //Push Notifications
34.         //Get a new FCM Token everytime the user signs out
35.         registerForPushNotifications()
36.         Messaging.messaging().delegate = self
37.
38.         return true
39.     }
40.
41.     //MARK: PUSH NOTIFICATIONS
42.

```

```
43.          //Request push notifications of type alert, with sounds and
44.          //a badge
45.          func registerForPushNotifications() {
46.              UNUserNotificationCenter.current()
47.                  .requestAuthorization(options: [.alert, .sound,
48.                      .badge]) {
49.                      [weak self] granted, error in
50.                          print("Permission granted: \(granted)")
51.                      //guard is for when permission was not granted
52.                      //it returns so user does not get notifications
53.                      guard granted else { return }
54.                      //if granted, call to get the notification
55.                      settings
56.                          self?.getNotificationSettings()
57.                      }
58.
59.          func getNotificationSettings() {
60.
61.              UNUserNotificationCenter.current().getNotificationSettings {
62.                  settings in
63.                      print("Notification settings: \(settings)")
64.                      //Checks authorised
65.                      guard settings.authorizationStatus == .authorized
66.                      else { return }
67.                      DispatchQueue.main.async {
68.                          //Triggers the registration to get device token
69.                      }
70.
71.                      //Sets the new device FCM token
72.                      func application(
73.                          _ application: UIApplication,
74.                          didRegisterForRemoteNotificationsWithDeviceToken
75.                          deviceToken: Data
76.                          ) {
77.                              Messaging.messaging().apnsToken = deviceToken
78.                          }
79.                          //Failed to register device for notifications
80.                          func application(
81.                              _ application: UIApplication,
82.                              didFailToRegisterForRemoteNotificationsWithError error:
83.                                  Error) {
84.                                  print("Failed to register: \(error)")
85.                              }
86.                              //Team ID: SCZ5X2T8PZ
87.
88.                              //Recieves the new device FCM token, whenever it is
     refreshed
```

```

87.         func messaging(_ messaging: Messaging,
88.             didReceiveRegistrationToken fcmToken: String) {
89.                 print("Firebase registration token: \(fcmToken)")
90.             let dataDict:[String: String] = ["token": fcmToken]
91.             NotificationCenter.default.post(name:
92.                 Notification.Name("FCMToken"), object: nil, userInfo: dataDict)
93.             deviceFCMToken = fcmToken
94.             // TODO: If necessary send token to application server.
95.             // Note: This callback is fired at each app startup and
96.             // whenever a new token is generated.
96.         }
97.     }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
window	UIWindow	Public	The backdrop for the app's user interface.

## Constants.swift

```

1. import Foundation
2.
3. //MARK: SEGUES
4.
5. let TO_CREATE_PROFILE = "toCreateProfile"
6. let TO_EDIT_PROFILE = "toEditProfile"
7. let TO_SOCIAL_LINKS = "toSocialLinks"
8. let TO_MUSIC_LINKS = "toMusicLinks"
9. let TO_MAIN = "toMain"
10.    let TO_MAIN_2 = "toMain2"
11.
12.    let TO_CREATE_GIG = "toCreateGig"
13.    let TO_EDIT_GIG_EVENT = "toEditGigEvent"
14.    let TO_TITLE_DATE = "toTitleDate"
15.    let TO_LOCATION_PRICING = "toLocationPricing"
16.    let TO_INFO_CONTACT = "toInfoContact"
17.    let TO_ADD_PHOTO = "toAddPhoto"
18.
19.    let TO_FIND_GIG = "toFindGig"
20.    let TO_EVENT_DESCRIPTION = "toEventDescription"
21.    let TO_EVENT_DESCRIPTION_2 = "toEventDescription2"
22.
23.    let TO_CHECK_PORTFOLIO = "toCheckPortfolio"
24.    let TO_CHECK_PORTFOLIO_2 = "toCheckPortfolio2"
25.    let TO_CHECK_PORTFOLIO_3 = "toCheckPortfolio3"
26.    let TO_REVIEW_APPLICATION = "toReviewApplication"
27.

```

```

28.
29.    //MARK: USER DEFAULTS
30.
31.    let DEFAULTS = UserDefaults.standard
32.
33.    let LOGGED_IN_KEY = "loggedIn"
34.    let USER_EMAIL = "userEmail"
35.
36.
37.    //MARK: GLOBAL VARIABLES
38.
39.    //Sign In/Out Gates
40.    var tabGateOpen = true
41.    var accountGateOpen = true
42.    var cardGateOpen = true
43.    var feedGateOpen = true
44.    var observeGateOpen = true
45.    var paginationGateOpen = true
46.    var pushNotificationGateOpen = true
47.
48.
49.    var launchedFromNotification = false
50.    var editingProfile = false
51.    var editingGigEvent = false

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
TO_CREATE_PROFILE TO_EDIT_PROFILE TO_SOCIAL_LINKS ... TO REVIEW APPLICATION	String	Global	So can perform segue between View Controllers. Use constant so I do not make mistake in development.
DEFAULTS	NSObject	Global	Instance of UserDefaults to save values under a key even when app closes
LOGGED_IN_KEY	String	Global	Key to save log in state
USER_EMAIL	String	Global	Key to save the user's email
tabGateOpen	Boolean	Global	Controls tab refresh, they only refresh once and not every time view appears.
accountGateOpen	Boolean	Global	Controls account data refresh in portfolio, refreshes once and not every time view appears.
cardGateOpen	Boolean	Global	Controls gig cards refresh, they refresh once and not every time the view appears.
feedGateOpen	Boolean	Global	Controls activity feed refresh, it refreshes once and not every time the view appears.

observeGateOpen	Boolean	Global	Controls activity feed refresh, it refreshes once and not every time the view appears.
paginationGateOpen	Boolean	Global	Controls pagination and the amount of notifications user should be returned.
pushNotificationGateOpen	Boolean	Global	Controls FCM device token refresh, refreshes at log in only and not every time view appears.
launchedFromNotification	Boolean	Global	To keep track of whether the app had been launched from tapping a push notification
editingProfile	Boolean	Global	To keep track of whether the user is editing or creating a profile
editingGigEvent	Boolean	Global	To keep track of whether the organiser is editing or creating an event

## Extensions

### ApplicationExtensions.swift

```

1. import UIKit
2.
3. extension UIApplication {
4.     class func tryURL(urls: [String]) {
5.         let application = UIApplication.shared
6.         for url in urls {
7.             if application.canOpenURL(URL(string: url)!) {
8.                 if #available(iOS 10.0, *) {
9.                     application.open(URL(string: url)!, options:
10.                         [:], completionHandler: nil)
11.                 }
12.                 else {
13.                     application.openURL(URL(string: url)!)
14.                 }
15.             }
16.         }
17.     }
18. }
```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
application	UIApplication	Local	The singleton app instance.

### UserAccountUI.swift

```

1. import UIKit
2.
3. extension LoginSignupVC {
```

```
4.     func setupView() {
5.         //Change the colour of the textfield placeholders so it can
6.         //be seen on background
7.         emailField.attributedPlaceholder =
8.             NSAttributedString(string: "email",
9.                 attributes: [NSAttributedString.Key.foregroundColor:
10.                     UIColor.systemGray3])
11.         passwordField.attributedPlaceholder =
12.             NSAttributedString(string: "password",
13.                 attributes: [NSAttributedString.Key.foregroundColor:
14.                     UIColor.systemGray3])
15.         confirmPasswordField.attributedPlaceholder =
16.             NSAttributedString(string: "confirm password",
17.                 attributes: [NSAttributedString.Key.foregroundColor:
18.                     UIColor.systemGray3])
19.         //Closure to instantiate transparent uiview object
20.         let transparentView: UIView = {
21.             let tv = UIView()
22.             tv.backgroundColor = colorLiteral(red: 0, green:
23.                 0, blue: 0, alpha: 1)
24.             //Transparent
25.             tv.alpha = 0.3
26.             //Rounded corners
27.             tv.layer.cornerRadius = 15
28.             //can be autoresized by programmatic constraints
29.             tv.translatesAutoresizingMaskIntoConstraints =
30.                 false
31.             return tv
32.         }()
33.         //Set up a background image and stretch it to all edges
34.         //of the screens
35.         let background = UIImage(named: "Background")
36.         var imageView : UIImageView!
37.         imageView = UIImageView(frame: view.bounds)
38.         //scale the image so it fills the image view
39.         imageView.contentMode =
40.             UIView.ContentMode.scaleAspectFill
41.         imageView.clipsToBounds = true
42.         imageView.image = background
43.         //image view is in the center of the view and clip it
44.         //to the edges of screen
45.         imageView.center = view.center
46.         //Add the transparent view to view
47.         view.addSubview(transparentView)
48.         //send it to the back of all UI elements
49.         self.view.sendSubviewToBack(transparentView)
50.
51.         //Set transparent view constraints so it sits behind
52.         //the fields stack with right dimensions
53.         NSLayoutConstraint.activate([
54.             transparentView.topAnchor.constraint(equalTo:
55.                 view.safeAreaLayoutGuide.topAnchor, constant: 130),
56.             transparentView.centerXAnchor.constraint(equalTo:
57.                 view.centerXAnchor),
```

```

44.    transparentView.widthAnchor.constraint(equalToConstant:
45.                                              fieldsStack.frame.width + 56),
46.                                              transparentView.bottomAnchor.constraint(equalTo:
47.                                              topLSButton.bottomAnchor, constant: 16)
48.                                              ])
49.                                              //add the background to view
50.                                              view.addSubview(imageView)
51.                                              //send it to the back of all the subviews
52.                                              self.view.sendSubviewToBack(imageView)
53.
54.    extension CreateProfileCAVC {
55.        func setupView() {
56.            //Closure to instantiate transparent uiview object
57.            let transparentView: UIView = {
58.                let tv = UIViewcolorLiteral(red: 0, green:
60.                                         0, blue: 0, alpha: 1)
61.                tv.alpha = 0.3
62.                tv.layer.cornerRadius = 15
63.                tv.translatesAutoresizingMaskIntoConstraints =
64.                    false
65.                return tv
66.            }()
67.            //Set up a background image and stretch it to all edges
68.            // of the screens
69.            let background = UIImage(named: "Background")
70.            var imageView : UIImageView!
71.            imageView = UIImageView(frame: view.bounds)
72.            imageView.contentMode =
73.                UIView.ContentMode.scaleAspectFill
74.                imageView.clipsToBounds = true
75.                imageView.image = background
76.                imageView.center = view.center
77.                //Add the transparent view to view
78.                view.addSubview(transparentView)
79.                //send it to the back of all UI elements
80.                self.view.sendSubviewToBack(transparentView)
81.                //Set transparent view constraints so it sits behind
82.                // the fields stack with right dimensions
83.                NSLayoutConstraint.activate([
84.                    transparentView.topAnchor.constraint(equalTo:
85.                                              view.safeAreaLayoutGuide.topAnchor, constant: 64),
86.                    transparentView.centerXAnchor.constraint(equalTo:
87.                                              view.centerXAnchor),
88.                    transparentView.widthAnchor.constraint(equalToConstant:
89.                                              nameBioStack.frame.width + 56),
90.                    transparentView.bottomAnchor.constraint(equalTo:
91.                                              profileImageView.bottomAnchor, constant: 8)
92.                ])
93.                //add the background to view
94.                view.addSubview(imageView)

```

```

86.          //send it to the back of all the subviews
87.          self.view.sendSubviewToBack(imageView)
88.      }
89.  }
90.
91.  extension SocialLinksCAVC {
92.      func setupView() {
93.          //Closure to instantiate transparent uiview object
94.          let transparentView: UIView = {
95.              let tv = UIView()
96.              tv.backgroundColor = colorLiteral(red: 0, green:
97.                  0, blue: 0, alpha: 1)
98.              tv.alpha = 0.3
99.              tv.layer.cornerRadius = 15
100.             tv.translatesAutoresizingMaskIntoConstraints =
101.                 false
102.             return tv
103.         }()
104.         //Set up a background image and stretch it to all edges
105.         // of the screens
106.         let background = UIImage(named: "Background")
107.         var imageView : UIImageView!
108.         imageView = UIImageView(frame: view.bounds)
109.         imageView.contentMode =
110.             UIView.ContentMode.scaleAspectFill
111.         imageView.clipsToBounds = true
112.         imageView.image = background
113.         imageView.center = view.center
114.         //Add the transparent view to view
115.         view.addSubview(transparentView)
116.         //send it to the back of all UI elements
117.         self.view.sendSubviewToBack(transparentView)
118.         //Set transparent view constraints so it sits behind
119.         // with right dimensions
120.         NSLayoutConstraint.activate([
121.             transparentView.topAnchor.constraint(equalTo:
122.                 view.safeAreaLayoutGuide.topAnchor, constant: 48),
123.             transparentView.centerXAnchor.constraint(equalTo:
124.                 view.centerXAnchor),
125.             transparentView.widthAnchor.constraint(equalToConstant:
126.                 fieldsStack.frame.width + 56),
127.             transparentView.bottomAnchor.constraint(equalTo:
128.                 fieldsStack.bottomAnchor, constant: 16)
129.         ])
130.         //add the background to view
131.         view.addSubview(imageView)
132.         //send it to the back of all the subviews
133.         self.view.sendSubviewToBack(imageView)
134.     }
135. }
136.
137. extension MusicLinksCAVC {
138.     func setupView() {
139.         //Closure to instantiate transparent uiview object

```

```
131.         let transparentView: UIView = {
132.             let tv = UIView()
133.             tv.backgroundColor = colorLiteral(red: 0, green:
134.                 0, blue: 0, alpha: 1)
135.             tv.alpha = 0.3
136.             tv.layer.cornerRadius = 15
137.             tv.translatesAutoresizingMaskIntoConstraints =
138.                 false
139.         }
140.         return tv
141.     }()
142.     //Set up a background image and stretch it to all edges
143.     // of the screens
144.     let background = UIImage(named: "Background")
145.     var imageView : UIImageView!
146.     imageView = UIImageView(frame: view.bounds)
147.     imageView.contentMode =
148.         UIView.ContentMode.scaleAspectFill
149.     imageView.clipsToBounds = true
150.     imageView.image = background
151.     imageView.center = view.center
152.     //Add the transparent view to view
153.     view.addSubview(transparentView)
154.     //send it to the back of all UI elements
155.     self.view.sendSubviewToBack(transparentView)
156.     //Set transparent view constraints so it sits behind
157.     // with right dimensions
158.     NSLayoutConstraint.activate([
159.         transparentView.centerXAnchor.constraint(equalTo:
160.             view.centerXAnchor),
161.         transparentView.centerYAnchor.constraint(equalTo:
162.             view.centerYAnchor),
163.         transparentView.widthAnchor.constraint(equalToConstant:
164.             fieldsStack.frame.width + 56),
165.         transparentView.heightAnchor.constraint(equalToConstant:
166.             fieldsStack.frame.height + 24)
167.     ])
168.     //add the background to view
169.     view.addSubview(imageView)
170.     //send it to the back of all the subviews
171.     self.view.sendSubviewToBack(imageView)
172. }
173. }

174. extension UserAccountVC {
175.     func setupView() {
176.         //use a large title as the name at top of portfolio
177.         self.navigationController?.navigationBar.prefersLargeTitles = true
178.         //if iOS 13
179.         if #available(iOS 13.0, *) {
180.             let navBarAppearance = UINavigationBarAppearance()
181.             //set the navigation bar as opaque...
182.             navBarAppearance.configureWithOpaqueBackground()
```

```
174.          //...with a white colour
175.          navBarAppearance.backgroundColor =
176.              UIColor.white.withAlphaComponent(0.75)
177.          //set this appearance when table view is still and
178.          //scrolling
179.          self.navigationController?.navigationBar.standardAppearance =
180.              navBarAppearance
181.          }
182.          //background image will be the background of the table
183.          view
184.          let backgroundImage = UIImage(named: "Background")
185.          let imageView = UIImageView(image: backgroundImage)
186.          self.tableView.backgroundView = imageView
187.          self.tableView.separatorStyle = .none
188.          imageView.contentMode = .scaleAspectFit
189.          }
190.          //provide a blur effect over the feed background, light
191.          style
192.          let blurEffect = UIBlurEffect(style:
193.              UIBlurEffect.Style.light)
194.          let blurView = UIVisualEffectView(effect: blurEffect)
195.          blurView.frame = imageView.bounds
196.          //add the image as background to table view
197.          imageView.addSubview(blurView)
198.          //automatically resize the cells depending on its
199.          content dimensions
200.          tableView.rowHeight = UITableView.automaticDimension
201.          //if fails (or loading) provide estimate height
202.          tableView.estimatedRowHeight = 350
203.          }
204.          }
205.          extension PortfolioPostVC {
206.          func setupView() {
207.              //make the navigation bar white with set opacity and
208.              //white colour
209.              navigationController?.navigationBar.barTintColor =
210.                  UIColor.white.withAlphaComponent(0.90)
211.              //Set up a background image and stretch it to all edges
212.              //of the screens
213.              let background = UIImage(named: "Background5")
214.              var imageView : UIImageView!
215.              imageView = UIImageView(frame: view.bounds)
216.              imageView.contentMode =
217.                  UIView.ContentMode.scaleAspectFill
218.              imageView.clipsToBounds = true
219.              imageView.image = background
220.              imageView.center = view.center
221.              imageView.alpha = 0.7
222.              //set background image and pin to edges of view
223.              view.addSubview(imageView)
224.          }
```

```

215.          //send it behind all UI elements
216.          self.view.sendSubviewToBack(imageView)
217.      }
218.  }
219.
220. extension CreateGigVC {
221.     func setupView() {
222.         //Closure to instantiate transparent uiview object
223.         let transparentView: UIView = {
224.             let tv = UIView()
225.             tv.backgroundColor = colorLiteral(red: 0, green:
226.                 0, blue: 0, alpha: 1)
227.             tv.alpha = 0.4
228.             tv.layer.cornerRadius = 15
229.             //use frame for dimensions rather than constraints
230.             tv.frame = CGRect.init(x: 0, y: 0, width:
231.                 descriptionStack.frame.width + 20, height:
232.                 descriptionStack.frame.height + 20)
233.             tv.center = CGPoint(x: self.view.frame.width / 2,
234.                 y: self.view.frame.height / 2)
235.             return tv
236.         }()
237.
238.         //Set up a background image and stretch it to all edges
239.         //of the screens
240.         let background = UIImage(named: "Background2")
241.         var imageView : UIImageView!
242.         imageView = UIImageView(frame: view.bounds)
243.         imageView.contentMode =
244.             UIView.ContentMode.scaleAspectFill
245.         imageView.clipsToBounds = true
246.         imageView.image = background
247.         imageView.center = view.center
248.         imageView.alpha = 0.7
249.         //Add the transparent view to view
250.         view.addSubview(transparentView)
251.         //send it to the back of all UI elements
252.         self.view.sendSubviewToBack(transparentView)
253.         //add the background to view
254.         view.addSubview(imageView)
255.         //send it to the back of all the subviews
256.         self.view.sendSubviewToBack(imageView)
257.     }
258. }
259.
260. extension TitleDateCGVC {
261.     func setupView() {
262.         //large title
263.         self.navigationController?.navigationBar.prefersLargeTitles = true
264.         //if iOS 13
265.         if #available(iOS 13.0, *) {
266.             //set the navigation bar with the large title as
267.             //basically clear
268.             let navBarAppearance = UINavigationBarAppearance()

```

```
262.             navBarAppearance.configureWithOpaqueBackground()
263.             navBarAppearance.backgroundColor =
264.                 UIColor.white.withAlphaComponent(0.1)
264.
265.             self.navigationController?.navigationBar.standardAppearance =
265.                 navBarAppearance
265.
266.             }
267.             //hide the default back button
268.             self.navigationItem.hidesBackButton = true
269.             //make our own back button
270.             let backItem = UIBarButtonItem()
271.             backItem.tintColor = colorLiteral(red: 0.4942619801,
271.                 green: 0.1805444658, blue: 0.5961503386, alpha: 1)
272.             backItem.title = "Back"
273.             navigationItem.backBarButtonItem = backItem
274.
275.             //set background
276.             let background = UIImage(named: "Background2")
277.             var imageView : UIImageView!
278.             imageView = UIImageView(frame: view.bounds)
279.             imageView.contentMode =
279.                 UIView.ContentMode.scaleAspectFill
280.             imageView.clipsToBounds = true
281.             imageView.image = background
282.             imageView.center = view.center
283.             imageView.alpha = 0.4
284.
285.             //set opacity of datepicker
286.             datePicker.backgroundColor =
286.                 UIColor.white.withAlphaComponent(0.3)
287.             //give it a shadow
288.             datePicker.layer.shadowColor = colorLiteral(red:
288.                 0.2549019754, green: 0.2745098174, blue: 0.3019607961, alpha: 1)
289.             //set size and strength and shape of shadow
290.             datePicker.layer.shadowRadius = 10.0
291.             datePicker.layer.shadowOpacity = 0.5
292.             datePicker.layer.cornerRadius = 10.0
293.             //add the background to view
294.             view.addSubview(imageView)
295.             //send it to the back of all the subviews
296.             self.view.sendSubviewToBack(imageView)
297.         }
298.     }
299.
300.     extension LocationPriceCGVC {
301.         func setupView() {
302.
303.             self.navigationController?.navigationBar.prefersLargeTitles = true
303.             if #available(iOS 13.0, *) {
304.                 let navBarAppearance = UINavigationBarAppearance()
305.                 navBarAppearance.configureWithOpaqueBackground()
```

```
306.             navBarAppearance.backgroundColor =
    UIColor.white.withAlphaComponent(0.1)
307.             self.navigationController?.navigationBar.standardAppearance =
    navBarAppearance
308.             self.navigationController?.navigationBar.scrollEdgeAppearance =
    navBarAppearance
309.         }
310.
311.             let backItem = UIBarButtonItem()
312.             backItem.tintColor = colorLiteral(red: 0.4942619801,
    green: 0.1805444658, blue: 0.5961503386, alpha: 1)
313.             backItem.title = "Back"
314.             navigationItem.backBarButtonItem = backItem
315.             //Set up a background image and stretch it to all edges
    of the screens
316.             let background = UIImage(named: "Background2")
317.             var imageView : UIImageView!
318.             imageView = UIImageView(frame: view.bounds)
319.             imageView.contentMode =
    UIView.ContentMode.scaleAspectFill
320.             imageView.clipsToBounds = true
321.             imageView.image = background
322.             imageView.center = view.center
323.             imageView.alpha = 0.4
324.             //add the background to view
325.             view.addSubview(imageView)
326.             //send it to the back of all the subviews
327.             self.view.sendSubviewToBack(imageView)
328.         }
329.     }
330.
331.     extension InfoContactCGVC {
332.         func setupView() {
333.
    self.navigationController?.navigationBar.prefersLargeTitles = true
334.         if #available(iOS 13.0, *) {
335.             let navBarAppearance = UINavigationBarAppearance()
336.             navBarAppearance.configureWithOpaqueBackground()
337.             navBarAppearance.backgroundColor =
    UIColor.white.withAlphaComponent(0.1)
338.
    self.navigationController?.navigationBar.standardAppearance =
    navBarAppearance
339.             self.navigationController?.navigationBar.scrollEdgeAppearance =
    navBarAppearance
340.         }
341.             overrideUserInterfaceStyle = .light
342.
    let backItem = UIBarButtonItem()
343.             backItem.tintColor = colorLiteral(red: 0.4942619801,
    green: 0.1805444658, blue: 0.5961503386, alpha: 1)
344.             backItem.title = "Back"
```

```
346.             navigationItem.backBarButtonItem = backItem
347.             //Set up a background image and stretch it to all edges
348.             of the screens
349.             let background = UIImage(named: "Background2")
350.             var imageView : UIImageView!
351.             imageView = UIImageView(frame: view.bounds)
352.             imageView.contentMode =
353.             UIView.ContentMode.scaleAspectFill
354.             imageView.clipsToBounds = true
355.             imageView.image = background
356.             imageView.center = view.center
357.             imageView.alpha = 0.4
358.             //add the background to view
359.             view.addSubview(imageView)
360.             //send it to the back of all the subviews
361.             self.view.sendSubviewToBack(imageView)
362.
363.         extension PhotoCGVC {
364.             func setupView() {
365.
366.                 self.navigationController?.navigationBar.prefersLargeTitles = true
367.                 if #available(iOS 13.0, *) {
368.                     let navBarAppearance = UINavigationBarAppearance()
369.                     navBarAppearance.configureWithOpaqueBackground()
370.                     navBarAppearance.backgroundColor =
371.                     UIColor.white.withAlphaComponent(0.1)
372.                 }
373.                 //give the eventPicView a shadow and set its size,
374.                 shape and strength
375.                 eventPicView.layer.shadowColor = colorLiteral(red:
376.                     0.2549019754, green: 0.2745098174, blue: 0.3019607961, alpha: 1)
377.                 eventPicView.layer.shadowRadius = 10.0
378.                 eventPicView.layer.shadowOpacity = 0.5
379.                 eventPicView.layer.cornerRadius = 20.0
380.                 //Set up a background image and stretch it to all edges
381.                 of the screens
382.                 let background = UIImage(named: "Background2")
383.                 var imageView : UIImageView!
384.                 imageView = UIImageView(frame: view.bounds)
385.                 imageView.contentMode =
386.                 UIView.ContentMode.scaleAspectFill
387.                 imageView.clipsToBounds = true
388.                 imageView.image = background
389.                 imageView.center = view.center
390.                 imageView.alpha = 0.4
391.                 //add the background to view
392.                 view.addSubview(imageView)
```

```

389.          //send it to the back of all the subviews
390.          self.view.sendSubviewToBack(imageView)
391.      }
392.  }
393.
394.  extension ActivityFeedVC {
395.      func setupView(tableView: UITableView){
396.          let backgroundImage = UIImage(named: "Background3")
397.          let imageView = UIImageView(image: backgroundImage)
398.          imageView.clipsToBounds = true
399.          imageView.contentMode = .scaleAspectFill
400.          //tableView.backgroundView = imageView
401.          tableView.separatorStyle = .none
402.          imageView.alpha = 0.5
403.      }
404.  }
405.
406.  extension FindGigVC {
407.      func setupView() {
408.          //Closure to instantiate transparent uiview object
409.          let transparentView: UIView = {
410.              let tv = UIView()
411.              tv.backgroundColor = colorLiteral(red: 0, green:
412. 0, blue: 0, alpha: 1)
413.              tv.alpha = 0.4
414.              tv.layer.cornerRadius = 15
415.              tv.translatesAutoresizingMaskIntoConstraints = false
416.          }
417.          //Set up a background image and stretch it to all edges
418.          // of the screens
419.          let background = UIImage(named: "Background2")
420.          var imageView : UIImageView!
421.          imageView = UIImageView(frame: view.bounds)
422.          imageView.contentMode =
423.              UIView.ContentMode.scaleAspectFill
424.          imageView.clipsToBounds = true
425.          imageView.image = background
426.          imageView.center = view.center
427.          imageView.alpha = 0.5
428.          //add the background to view
429.          view.addSubview(imageView)
430.          //send it to the back of all the subviews
431.          self.view.sendSubviewToBack(imageView)
432.
433.          //add the transparent view
434.          view.addSubview(transparentView)
435.          //set it at z index 4
436.          self.view.insertSubview(transparentView, at: 4)
437.          //ensure the contact details are z index 5 (sits ontop)
438.          self.view.insertSubview(contactStack, at: 5)
439.          //Set transparent view constraints so it sits behind
440.          // with right dimensions
441.          NSLayoutConstraint.activate([

```

```

439.                 transparentView.topAnchor.constraint(equalTo:
440.                               view.safeAreaLayoutGuide.topAnchor, constant: 4),
441.                 transparentView.centerXAnchor.constraint(equalTo:
442.                               view.centerXAnchor),
443.                 transparentView.widthAnchor.constraint(equalToConstant:
444.                               contactStack.frame.width + 20),
445.             ])
446.
447.     extension ReviewApplicationVC {
448.         func setupView() {
449.             //Closure to instantiate transparent uiview object
450.             let transparentView: UIView = {
451.                 let tv = UIView()
452.                 tv.backgroundColor = colorLiteral(red: 0, green:
453.                               0, blue: 0, alpha: 1)
454.                 tv.alpha = 0.3
455.                 tv.layer.cornerRadius = 15
456.                 tv.translatesAutoresizingMaskIntoConstraints = false
457.             }
458.             //Set up a background image and stretch it to all edges
of the screens
459.             let background = UIImage(named: "Background3")
460.             var imageView : UIImageView!
461.             imageView = UIImageView(frame: view.bounds)
462.             imageView.contentMode =
463.                 UIView.ContentMode.scaleAspectFill
464.             imageView.clipsToBounds = true
465.             imageView.image = background
466.             imageView.center = view.center
467.             imageView.alpha = 0.4
468.             //give the background a blur effect
469.             let blurEffect = UIBlurEffect(style:
470.                               UIBlurEffect.Style.light)
471.             let blurView = UIVisualEffectView(effect: blurEffect)
472.             blurView.frame = imageView.bounds
473.             imageView.addSubview(blurView)
474.             //Add the transparent view to view
475.             view.addSubview(transparentView)
476.             //send it to the back of all UI elements
477.             self.view.sendSubviewToBack(transparentView)
478.             //Set transparent view constraints so it sits behind
with right dimensions
479.             NSLayoutConstraint.activate([
480.                 transparentView.topAnchor.constraint(equalTo:
481.                               nameLabel.topAnchor, constant: 8),
482.                 transparentView.centerXAnchor.constraint(equalTo:
483.                               view.centerXAnchor),

```

```

480.
    transparentView.widthAnchor.constraint(equalToConstant:
        nameLabel.frame.width - 16),
481.            transparentView.bottomAnchor.constraint(equalTo:
        eventLabel.bottomAnchor, constant: 8)
482.        ])
483.            //add the background to view
484.            view.addSubview(imageView)
485.            //send it to the back of all the subviews
486.            self.view.sendSubviewToBack(imageView)
487.            //make the profile image view a circle and not a square
488.            profileImageView.layer.borderWidth = 0.1
489.            profileImageView.layer.masksToBounds = false
490.            profileImageView.layer.cornerRadius =
    profileImageView.frame.height/2
491.            profileImageView.clipsToBounds = true
492.        }
493.    }
494.
495.
496.
497.    extension EventDescriptionVC {
498.        func setupView() {
499.            //make the background of text view white but see-
through
500.            descriptionTextView.backgroundColor =
    UIColor.white.withAlphaComponent(0.4)
501.
502.            //create a gradient (rather than image) to use as
background
503.            let gradient: CAGradientLayer = CAGradientLayer()
504.            //two stops, orange to purple
505.            gradient.colors = [UIColor(red: 255.0/255.0, green:
    159.0/255.0, blue: 2.0/255.0, alpha: 0.5).cgColor, UIColor(red:
    104.0/255.0, green: 35.0/255.0, blue: 128.0/255.0, alpha:
    0.6).cgColor]
506.            //from top to bottom
507.            gradient.locations = [0.0, 1.0]
508.            //full screen
509.            gradient.frame = CGRect(x: 0.0, y: 0.0, width:
    self.view.bounds.size.width, height: self.view.bounds.size.height)
510.            gradient.cornerRadius = 10.0
511.            view.layer.insertSublayer(gradient, at: 0)
512.        }
513.    }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
transparentView	UIView	Local	A transparent black view that creates a larger contrast between background and white text in the UI.
backgroundImageView	UIImage UIImageView	Local	The background image that is displayed in an imageView

			and stretched to the corners of the screen.
navBarAppearance	UINavigationBarAppearance	Local	To create a custom appearance of navigation bars.
blurEffect blurView	UIBlurEffect UIVisualEffectView	Local	To give the background of the portfolio table view a blurred effect.
gradient	CAGradientLayer	Local	To provide a gradient background rather than a UIImage in EventDescriptionVC.

## Extensions.swift

```

1. import Foundation
2. import UIKit
3. import AVKit
4. import AVFoundation
5. import GoogleMaps
6. import GooglePlaces
7. import EventKit
8.
9. let imageCache = NSCache<NSString, UIImage>()
10.
11.     extension UIViewController: UIImagePickerControllerDelegate,
12.         UINavigationControllerDelegate {
13.
14.         //MARK: HIDE KEYBOARD
15.         //To hide keyboard
16.         func hideKeyboard() {
17.             //Gesture recogniser so we call method when the screen
18.             //is tapped
19.             let tap: UITapGestureRecognizer =
20.                 UITapGestureRecognizer(target: self, action:
21. #selector(UIViewController.dismissKeyboard))
22.             //if the user taps inside the keyboard, ignore
23.             tap.cancelsTouchesInView = false
24.             view.addGestureRecognizer(tap)
25.         }
26.
27.         //MARK: ERROR NOTIFICATION
28.         func displayError(title: String, message: String) {
29.             //alert controller with a title and message and 'OK'
30.             //dismiss button
31.             let alertController = UIAlertController(title: title,
32.             message: message, preferredStyle: .alert)
33.             alertController.addAction(UIAlertAction(title: "OK",
34.             style: .default, handler: nil))
35.             //present the alert

```

```

33.          self.present(alertController, animated: true,
34.            completion: nil)
35.
36.          //MARK: LOADING INDICATOR
37.          func createSpinnerView( _ child: SpinnerViewController) {
38.              // add the spinner view controller
39.              //user cannot interact with the view
40.              self.view.isUserInteractionEnabled = false
41.              addChild(child)
42.              //fill screen and add to view
43.              child.view.frame = view.frame
44.              view.addSubview(child.view)
45.              child.didMove(toParent: self)
46.          }
47.          func removeSpinnerView( _ child: SpinnerViewController) {
48.              //give user control again
49.              self.view.isUserInteractionEnabled = true
50.              //and remove from view
51.              child.willMove(toParent: nil)
52.              child.view.removeFromSuperview()
53.              child.removeFromParent()
54.          }
55.
56.          //MARK: GENERIC QUICK-SORT
57.          //Quick Sort an array of type generic
58.          func quickSort<T: Comparable>(array:[T]) -> [T] {
59.              //Base case for recursion (escape)
60.              if array.isEmpty { return [] }
61.              //Store the first element of the array to compare it
62.              //with smaller or larger number
63.              let first = array.first!
64.              //first half = all values smaller or equal to first
65.              let smallerOrEqual = array.dropFirst().filter { $0 <=
66.                  first }
67.              //second half = values larger than first
68.              let larger = array.dropFirst().filter { $0 >
69.                  first }
70.              //First and secoond half are recursed and inserted
71.              //either side of the first value
72.              return quickSort(array: smallerOrEqual) + [first] +
73.                  quickSort(array: larger)
74.          }
75.
76.          //MARK: IMAGE PICKER ACTION SHEET
77.          func openPhotoPopup(video: Bool, imagePicker:
78.            UIImagePickerController, title: String, message: String){
79.              //The UIAlertController with title and message
80.              let photoPopup = UIAlertController(title: title,
81.                message: message, preferredStyle: .actionSheet)
82.                //First Choice - Camera
83.                let cameraAction = UIAlertAction(title: "Camera",
84.                  style: .default) { (buttonTapped) in
85.                    do {

```

```
79.                                //open the camera
80.                                self.openImagePicker(imagePicker: imagePicker,
81.                                source: .camera)
82.                                }
83.                                //Second Choice - Photo Library
84.                                let photoAction = UIAlertAction(title: "Photo Library",
85.                                style: .default) {
86.                                    (buttonTapped) in
87.                                    do {
88.                                        //open the library of photos
89.                                        self.openImagePicker(imagePicker: imagePicker,
90.                                source: .photoLibrary)
91.                                }
92.
93.                                //So choosing videos is only sometimes an option
94.                                if video {
95.
96.                                    //Third Choice - Video Library
97.                                    let videoAction = UIAlertAction(title: "Video
98.                                Library", style: .default) {
99.                                    (buttonTapped) in
100.                                    do {
101.                                        //open the library of videos
102.                                        self.openImagePicker(imagePicker:
103.                                imagePicker, source: .savedPhotosAlbum)
104.                                }
105.                                }
106.
107.                                photoPopup.addAction(videoAction)
108.                                photoPopup.addAction(photoAction)
109.                                //present the action sheet
110.                                present(photoPopup, animated: true, completion: nil)
111.                                }
112.
113.                                //MARK: IMAGE PICKER
114.                                func openImagePicker(imagePicker: UIImagePickerController,
115.                                source: UIImagePickerController.SourceType) {
116.
117.                                    //imagepicker is the user Photo Library/Camera/Video
118.                                    //Library
119.                                    imagePicker.sourceType = source
120.
121.                                    if source == .savedPhotosAlbum {
122.                                        //Limit library to videos
123.                                        imagePicker.mediaTypes = ["public.movie"]
124.                                    } else {
125.                                        //Limit library to photos
126.                                        imagePicker.mediaTypes = ["public.image"]
127.                                    }
128.
129.                                }
130.
131.                                }
132.
133.                                }
134.
135.                                }
136.
137.                                }
138.
139.                                }
140.
141.                                }
142.
143.                                }
144.
145.                                }
146.
147.                                }
148.
149.                                }
150.
151.                                }
152.
153.                                }
154.
155.                                }
156.
157.                                }
158.
159.                                }
160.
161.                                }
162.
163.                                }
164.
165.                                }
166.
167.                                }
168.
169.                                }
170.
171.                                }
172.
173.                                }
174.
175.                                }
176.
177.                                }
178.
179.                                }
180.
181.                                }
182.
183.                                }
184.
185.                                }
186.
187.                                }
188.
189.                                }
190.
191.                                }
192.
193.                                }
194.
195.                                }
196.
197.                                }
198.
199.                                }
200.
201.                                }
202.
203.                                }
204.
205.                                }
206.
207.                                }
208.
209.                                }
210.
211.                                }
212.
213.                                }
214.
215.                                }
216.
217.                                }
218.
219.                                }
220.
221.                                }
222.
223.                                }
224.
225.                                }
226.
227.                                }
228.
229.                                }
230.
231.                                }
232.
233.                                }
234.
235.                                }
236.
237.                                }
238.
239.                                }
240.
241.                                }
242.
243.                                }
244.
245.                                }
246.
247.                                }
248.
249.                                }
250.
251.                                }
252.
253.                                }
254.
255.                                }
256.
257.                                }
258.
259.                                }
260.
261.                                }
262.
263.                                }
264.
265.                                }
266.
267.                                }
268.
269.                                }
270.
271.                                }
272.
273.                                }
274.
275.                                }
276.
277.                                }
278.
279.                                }
280.
281.                                }
282.
283.                                }
284.
285.                                }
286.
287.                                }
288.
289.                                }
290.
291.                                }
292.
293.                                }
294.
295.                                }
296.
297.                                }
298.
299.                                }
300.
301.                                }
302.
303.                                }
304.
305.                                }
306.
307.                                }
308.
309.                                }
310.
311.                                }
312.
313.                                }
314.
315.                                }
316.
317.                                }
318.
319.                                }
320.
321.                                }
322.
323.                                }
324.
325.                                }
326.
327.                                }
328.
329.                                }
330.
331.                                }
332.
333.                                }
334.
335.                                }
336.
337.                                }
338.
339.                                }
340.
341.                                }
342.
343.                                }
344.
345.                                }
346.
347.                                }
348.
349.                                }
350.
351.                                }
352.
353.                                }
354.
355.                                }
356.
357.                                }
358.
359.                                }
360.
361.                                }
362.
363.                                }
364.
365.                                }
366.
367.                                }
368.
369.                                }
370.
371.                                }
372.
373.                                }
374.
375.                                }
376.
377.                                }
378.
379.                                }
380.
381.                                }
382.
383.                                }
384.
385.                                }
386.
387.                                }
388.
389.                                }
390.
391.                                }
392.
393.                                }
394.
395.                                }
396.
397.                                }
398.
399.                                }
400.
401.                                }
402.
403.                                }
404.
405.                                }
406.
407.                                }
408.
409.                                }
410.
411.                                }
412.
413.                                }
414.
415.                                }
416.
417.                                }
418.
419.                                }
420.
421.                                }
422.
423.                                }
424.
425.                                }
426.
427.                                }
428.
429.                                }
430.
431.                                }
432.
433.                                }
434.
435.                                }
436.
437.                                }
438.
439.                                }
440.
441.                                }
442.
443.                                }
444.
445.                                }
446.
447.                                }
448.
449.                                }
450.
451.                                }
452.
453.                                }
454.
455.                                }
456.
457.                                }
458.
459.                                }
460.
461.                                }
462.
463.                                }
464.
465.                                }
466.
467.                                }
468.
469.                                }
470.
471.                                }
472.
473.                                }
474.
475.                                }
476.
477.                                }
478.
479.                                }
480.
481.                                }
482.
483.                                }
484.
485.                                }
486.
487.                                }
488.
489.                                }
490.
491.                                }
492.
493.                                }
494.
495.                                }
496.
497.                                }
498.
499.                                }
500.
501.                                }
502.
503.                                }
504.
505.                                }
506.
507.                                }
508.
509.                                }
510.
511.                                }
512.
513.                                }
514.
515.                                }
516.
517.                                }
518.
519.                                }
520.
521.                                }
522.
523.                                }
524.
525.                                }
526.
527.                                }
528.
529.                                }
530.
531.                                }
532.
533.                                }
534.
535.                                }
536.
537.                                }
538.
539.                                }
540.
541.                                }
542.
543.                                }
544.
545.                                }
546.
547.                                }
548.
549.                                }
550.
551.                                }
552.
553.                                }
554.
555.                                }
556.
557.                                }
558.
559.                                }
560.
561.                                }
562.
563.                                }
564.
565.                                }
566.
567.                                }
568.
569.                                }
570.
571.                                }
572.
573.                                }
574.
575.                                }
576.
577.                                }
578.
579.                                }
580.
581.                                }
582.
583.                                }
584.
585.                                }
586.
587.                                }
588.
589.                                }
590.
591.                                }
592.
593.                                }
594.
595.                                }
596.
597.                                }
598.
599.                                }
599.
600.                                }
601.
602.                                }
603.
604.                                }
605.
606.                                }
607.
608.                                }
609.
609.
610.                                }
611.
612.                                }
613.
614.                                }
615.
616.                                }
617.
617.
618.                                }
619.
620.                                }
621.
622.                                }
623.
624.                                }
625.
626.                                }
627.
627.
628.                                }
629.
630.                                }
631.
632.                                }
633.
634.                                }
635.
636.                                }
637.
637.
638.                                }
639.
640.                                }
641.
642.                                }
643.
644.                                }
645.
646.                                }
647.
647.
648.                                }
649.
650.                                }
651.
652.                                }
653.
654.                                }
655.
656.                                }
657.
657.
658.                                }
659.
660.                                }
661.
662.                                }
663.
664.                                }
665.
666.                                }
667.
667.
668.                                }
669.
670.                                }
671.
672.                                }
673.
674.                                }
675.
676.                                }
677.
677.
678.                                }
679.
680.                                }
681.
682.                                }
683.
684.                                }
685.
686.                                }
687.
687.
688.                                }
689.
690.                                }
691.
692.                                }
693.
694.                                }
695.
696.                                }
697.
697.
698.                                }
699.
700.                                }
701.
702.                                }
703.
704.                                }
705.
706.                                }
707.
707.
708.                                }
709.
710.                                }
711.
712.                                }
713.
714.                                }
715.
715.
716.                                }
717.
718.                                }
719.
720.                                }
721.
722.                                }
723.
723.
724.                                }
725.
726.                                }
727.
728.                                }
729.
729.
730.                                }
731.
732.                                }
733.
734.                                }
735.
736.                                }
737.
737.
738.                                }
739.
740.                                }
741.
742.                                }
743.
744.                                }
745.
745.
746.                                }
747.
748.                                }
749.
750.                                }
751.
752.                                }
753.
753.
754.                                }
755.
756.                                }
757.
758.                                }
759.
759.
760.                                }
761.
762.                                }
763.
764.                                }
765.
765.
766.                                }
767.
768.                                }
769.
770.                                }
771.
771.
772.                                }
773.
774.                                }
775.
776.                                }
777.
777.
778.                                }
779.
780.                                }
781.
782.                                }
783.
784.                                }
785.
785.
786.                                }
787.
788.                                }
789.
789.
790.                                }
791.
792.                                }
793.
794.                                }
795.
795.
796.                                }
797.
798.                                }
799.
799.
800.                                }
801.
802.                                }
803.
804.                                }
805.
805.
806.                                }
807.
808.                                }
809.
809.
810.                                }
811.
812.                                }
813.
814.                                }
815.
815.
816.                                }
817.
818.                                }
819.
819.
820.                                }
821.
822.                                }
823.
823.
824.                                }
825.
826.                                }
827.
827.
828.                                }
829.
830.                                }
831.
831.
832.                                }
833.
834.                                }
835.
835.
836.                                }
837.
838.                                }
839.
839.
840.                                }
841.
842.                                }
843.
843.
844.                                }
845.
846.                                }
847.
847.
848.                                }
849.
850.                                }
851.
851.
852.                                }
853.
854.                                }
855.
855.
856.                                }
857.
858.                                }
859.
859.
860.                                }
861.
862.                                }
863.
863.
864.                                }
865.
866.                                }
867.
867.
868.                                }
869.
870.                                }
871.
871.
872.                                }
873.
874.                                }
875.
875.
876.                                }
877.
878.                                }
879.
879.
880.                                }
881.
882.                                }
883.
883.
884.                                }
885.
886.                                }
887.
887.
888.                                }
889.
889.
890.                                }
891.
892.                                }
893.
893.
894.                                }
895.
896.                                }
897.
897.
898.                                }
899.
900.                                }
901.
902.                                }
903.
903.
904.                                }
905.
906.                                }
907.
907.
908.                                }
909.
910.                                }
911.
911.
912.                                }
913.
914.                                }
915.
915.
916.                                }
917.
918.                                }
919.
919.
920.                                }
921.
922.                                }
923.
923.
924.                                }
925.
926.                                }
927.
927.
928.                                }
929.
930.                                }
931.
931.
932.                                }
933.
934.                                }
935.
935.
936.                                }
937.
938.                                }
939.
939.
940.                                }
941.
942.                                }
943.
943.
944.                                }
945.
946.                                }
947.
947.
948.                                }
949.
950.                                }
951.
951.
952.                                }
953.
954.                                }
955.
955.
956.                                }
957.
958.                                }
959.
959.
960.                                }
961.
962.                                }
963.
963.
964.                                }
965.
966.                                }
967.
967.
968.                                }
969.
970.                                }
971.
971.
972.                                }
973.
974.                                }
975.
975.
976.                                }
977.
978.                                }
979.
979.
980.                                }
981.
982.                                }
983.
983.
984.                                }
985.
986.                                }
987.
987.
988.                                }
989.
990.                                }
991.
991.
992.                                }
993.
994.                                }
995.
995.
996.                                }
997.
998.                                }
999.
999.
1000.                                }
```

```
127.         //Present View Controller
128.         present(imagePicker, animated: true, completion: nil)
129.     }
130.
131.     //Dismiss the imagePicker if the OS Cancels
132.     public func imagePickerControllerDidCancel(_ picker:
133.         UIImagePickerController) {
134.         picker.dismiss(animated: true, completion: nil)
135.     }
136.     //MARK: DOWNLOAD IMAGE
137.     func downloadImage(url: URL, handler: @escaping (_
138.         returnedImage: UIImage) -> ()) {
139.         let task = URLSession.shared.dataTask(with: url) {
140.             data, response, error in
141.             if let error = error{
142.                 //print the error if one
143.                 print(error.localizedDescription)
144.             } else {
145.                 //So picture if first when UI loads
146.                 DispatchQueue.main.async() {
147.                     //Converts the image data to a UIImage
148.                     if let downloadedImage = UIImage(data:
149.                         data!) {
150.                         handler(downloadedImage)
151.                     }
152.                 }
153.             }
154.         }
155.     }
156. }
157. //Resumes the task after image is set
158. task.resume()
159.
160. }
161.
162. //MARK: IMAGE CACHE
163. //Storing images as cache reduces the network usage of the
164. //app
165. func loadImageCache(url: URL, isImage: Bool, handler:
166.     @escaping (_ returnedImage: UIImage) -> ()) {
167.     let urlString = url.absoluteString as NSString
168.     //Check for a cached image under that URL
169.     if let cachedImage = imageCache.object(forKey:
170.         urlString) {
171.         handler(cachedImage)
172.     } else {
173.         //Get's the data of the URL
```

```
174.             let task = URLSession.shared.dataTask(with: url) {
175.                 data, response, error in
176.                     if let error = error{
177.                         print(error.localizedDescription)
178.                     } else {
179.                         //So picture if first when UI loads
180.                         DispatchQueue.main.async() {
181.                             //Converts the image data to a UIImage
182.                             if let downloadedImage = UIImage(data:
183.                                 data!) {
184.                                     //set the cache for reused image
185.                                     imageCache.setObject(downloadedImage, forKey: urlString)
186.                                     handler(downloadedImage)
187.                                 }
188.                             }
189.                         }
190.                         }
191.                         }
192.                         }
193.                         }
194.                         }
195.                         //Resumes the task after image is set
196.                         task.resume()
197.                         }
198.                     }
199.                 }
200.             }
201.             //MARK: VIDEO THUMBNAIL
202.             func generateThumbnail(url: URL) -> UIImage {
203.                 do {
204.                     let asset = AVURLAsset(url: url)
205.                     let imageGenerator = AVAssetImageGenerator(asset:
206.                         asset)
207.                     imageGenerator.appliesPreferredTrackTransform =
208.                         true
209.                     //Grab an image right at the start of the video
210.                     let cgImage = try imageGenerator.copyCGImage(at:
211.                         .zero,
212.                         actualTime: nil)
213.                     //Return the image
214.                     return UIImage(cgImage: cgImage)
215.                 } catch {
216.                     print(error.localizedDescription)
217.                     //Return placeholder if there is an error
218.                     return UIImage(named: "second")!
219.                 }
220.             }
221.         }
```

```

222.          //MARK: ADD EVENT TO CALENDAR
223.          func addEventToCalendar(title: String, description:
224.          String?, startDate: Date, endDate: Date, completion: ((_ success:
225.          Bool, _ error: NSError?) -> Void)? = nil) {
226.          let eventStore = EKEventStore()  

227.          //Add to event to calendar
228.          //Request calendar access
229.          eventStore.requestAccess(to: .event, completion: {
230.              (granted, error) in
231.                  //If given access
232.                  if (granted) && (error == nil) {
233.                      //Create an event
234.                      let event = EKEvent(eventStore: eventStore)
235.                      //Set title, start and end, and any notes
236.                      event.title = title
237.                      event.startDate = startDate
238.                      event.endDate = endDate
239.                      event.notes = description //Notes is the
240.                          description of the event
241.                          //Check to see if "GoGig" calendar has been
242.                          //created - First time only
243.                          if DEFAULTS.object(forKey: "GoGigCalendar") ==
244.                              nil {
245.                              //If new create the calendar
246.                              let newCalendar = EKCalendar(for: .event,
247.                              eventStore: eventStore)
248.                              //Calendar title
249.                              newCalendar.title = "GoGig - My Gigs"
250.                              //Colour
251.                              newCalendar.cgColor =
252.                                  UIColor.purple.cgColor
253.                                  let sourcesInEventStore =
254.                                      eventStore.sources
255.                                      newCalendar.source =
256.                                          sourcesInEventStore.filter {
257.                                              (source: EKSource) -> Bool in
258.                                              source.sourceType.rawValue ==
259.                                              EKSourceType.local.rawValue
260.                                              }.first!
261.                                              do {
262.                                              //Create new calendar (first time only,
263.                                              //then set identifier with UserDefaults)
264.                                              try eventStore.saveCalendar(newCalendar,
265.                                              commit: true)
266.                                              DEFAULTS.set(newCalendar.calendarIdentifier, forKey:
267.                                              "GoGigCalendar")
268.                                              print("new calendar created")
269.                                              } catch {
270.                                              self.displayError(title: "Oops",
271.                                              message: "Something went wrong")
272.                                              }
273.                                              }
274.                                              }
275.                                              }
276.                                              }
277.                                              }
278.                                              }
279.                                              }
280.                                              }
281.                                              }
282.                                              }
283.                                              }
284.                                              }
285.                                              }
286.                                              }
287.                                              }
288.                                              }
289.                                              }
290.                                              }
291.                                              }
292.                                              }
293.                                              }
294.                                              }
295.                                              }
296.                                              }
297.                                              }
298.                                              }
299.                                              }
300.                                              }
301.                                              }
302.                                              }
303.                                              }
304.                                              }
305.                                              }
306.                                              }
307.                                              }
308.                                              }
309.                                              }
310.                                              }
311.                                              }
312.                                              }
313.                                              }
314.                                              }
315.                                              }
316.                                              }
317.                                              }
318.                                              }
319.                                              }
320.                                              }
321.                                              }
322.                                              }
323.                                              }
324.                                              }
325.                                              }
326.                                              }
327.                                              }
328.                                              }
329.                                              }
330.                                              }
331.                                              }
332.                                              }
333.                                              }
334.                                              }
335.                                              }
336.                                              }
337.                                              }
338.                                              }
339.                                              }
340.                                              }
341.                                              }
342.                                              }
343.                                              }
344.                                              }
345.                                              }
346.                                              }
347.                                              }
348.                                              }
349.                                              }
350.                                              }
351.                                              }
352.                                              }
353.                                              }
354.                                              }
355.                                              }
356.                                              }
357.                                              }
358.                                              }
359.                                              }
360.                                              }
361.                                              }
362.                                              }
363.                                              }
364.                                              }
365.                                              }
366.                                              }
367.                                              }
368.                                              }
369.                                              }
370.                                              }
371.                                              }
372.                                              }
373.                                              }
374.                                              }
375.                                              }
376.                                              }
377.                                              }
378.                                              }
379.                                              }
380.                                              }
381.                                              }
382.                                              }
383.                                              }
384.                                              }
385.                                              }
386.                                              }
387.                                              }
388.                                              }
389.                                              }
390.                                              }
391.                                              }
392.                                              }
393.                                              }
394.                                              }
395.                                              }
396.                                              }
397.                                              }
398.                                              }
399.                                              }
400.                                              }
401.                                              }
402.                                              }
403.                                              }
404.                                              }
405.                                              }
406.                                              }
407.                                              }
408.                                              }
409.                                              }
410.                                              }
411.                                              }
412.                                              }
413.                                              }
414.                                              }
415.                                              }
416.                                              }
417.                                              }
418.                                              }
419.                                              }
420.                                              }
421.                                              }
422.                                              }
423.                                              }
424.                                              }
425.                                              }
426.                                              }
427.                                              }
428.                                              }
429.                                              }
430.                                              }
431.                                              }
432.                                              }
433.                                              }
434.                                              }
435.                                              }
436.                                              }
437.                                              }
438.                                              }
439.                                              }
440.                                              }
441.                                              }
442.                                              }
443.                                              }
444.                                              }
445.                                              }
446.                                              }
447.                                              }
448.                                              }
449.                                              }
450.                                              }
451.                                              }
452.                                              }
453.                                              }
454.                                              }
455.                                              }
456.                                              }
457.                                              }
458.                                              }
459.                                              }
460.                                              }
461.                                              }
462.                                              }
463.                                              }
464.                                              }
465.                                              }
466.                                              }
467.                                              }
468.                                              }
469.                                              }
470.                                              }
471.                                              }
472.                                              }
473.                                              }
474.                                              }
475.                                              }
476.                                              }
477.                                              }
478.                                              }
479.                                              }
480.                                              }
481.                                              }
482.                                              }
483.                                              }
484.                                              }
485.                                              }
486.                                              }
487.                                              }
488.                                              }
489.                                              }
490.                                              }
491.                                              }
492.                                              }
493.                                              }
494.                                              }
495.                                              }
496.                                              }
497.                                              }
498.                                              }
499.                                              }
500.                                              }
501.                                              }
502.                                              }
503.                                              }
504.                                              }
505.                                              }
506.                                              }
507.                                              }
508.                                              }
509.                                              }
510.                                              }
511.                                              }
512.                                              }
513.                                              }
514.                                              }
515.                                              }
516.                                              }
517.                                              }
518.                                              }
519.                                              }
520.                                              }
521.                                              }
522.                                              }
523.                                              }
524.                                              }
525.                                              }
526.                                              }
527.                                              }
528.                                              }
529.                                              }
530.                                              }
531.                                              }
532.                                              }
533.                                              }
534.                                              }
535.                                              }
536.                                              }
537.                                              }
538.                                              }
539.                                              }
540.                                              }
541.                                              }
542.                                              }
543.                                              }
544.                                              }
545.                                              }
546.                                              }
547.                                              }
548.                                              }
549.                                              }
550.                                              }
551.                                              }
552.                                              }
553.                                              }
554.                                              }
555.                                              }
556.                                              }
557.                                              }
558.                                              }
559.                                              }
560.                                              }
561.                                              }
562.                                              }
563.                                              }
564.                                              }
565.                                              }
566.                                              }
567.                                              }
568.                                              }
569.                                              }
570.                                              }
571.                                              }
572.                                              }
573.                                              }
574.                                              }
575.                                              }
576.                                              }
577.                                              }
578.                                              }
579.                                              }
580.                                              }
581.                                              }
582.                                              }
583.                                              }
584.                                              }
585.                                              }
586.                                              }
587.                                              }
588.                                              }
589.                                              }
590.                                              }
591.                                              }
592.                                              }
593.                                              }
594.                                              }
595.                                              }
596.                                              }
597.                                              }
598.                                              }
599.                                              }
600.                                              }
601.                                              }
602.                                              }
603.                                              }
604.                                              }
605.                                              }
606.                                              }
607.                                              }
608.                                              }
609.                                              }
610.                                              }
611.                                              }
612.                                              }
613.                                              }
614.                                              }
615.                                              }
616.                                              }
617.                                              }
618.                                              }
619.                                              }
620.                                              }
621.                                              }
622.                                              }
623.                                              }
624.                                              }
625.                                              }
626.                                              }
627.                                              }
628.                                              }
629.                                              }
630.                                              }
631.                                              }
632.                                              }
633.                                              }
634.                                              }
635.                                              }
636.                                              }
637.                                              }
638.                                              }
639.                                              }
640.                                              }
641.                                              }
642.                                              }
643.                                              }
644.                                              }
645.                                              }
646.                                              }
647.                                              }
648.                                              }
649.                                              }
650.                                              }
651.                                              }
652.                                              }
653.                                              }
654.                                              }
655.                                              }
656.                                              }
657.                                              }
658.                                              }
659.                                              }
660.                                              }
661.                                              }
662.                                              }
663.                                              }
664.                                              }
665.                                              }
666.                                              }
667.                                              }
668.                                              }
669.                                              }
670.                                              }
671.                                              }
672.                                              }
673.                                              }
674.                                              }
675.                                              }
676.                                              }
677.                                              }
678.                                              }
679.                                              }
680.                                              }
681.                                              }
682.                                              }
683.                                              }
684.                                              }
685.                                              }
686.                                              }
687.                                              }
688.                                              }
689.                                              }
690.                                              }
691.                                              }
692.                                              }
693.                                              }
694.                                              }
695.                                              }
696.                                              }
697.                                              }
698.                                              }
699.                                              }
700.                                              }
701.                                              }
702.                                              }
703.                                              }
704.                                              }
705.                                              }
706.                                              }
707.                                              }
708.                                              }
709.                                              }
710.                                              }
711.                                              }
712.                                              }
713.                                              }
714.                                              }
715.                                              }
716.                                              }
717.                                              }
718.                                              }
719.                                              }
720.                                              }
721.                                              }
722.                                              }
723.                                              }
724.                                              }
725.                                              }
726.                                              }
727.                                              }
728.                                              }
729.                                              }
730.                                              }
731.                                              }
732.                                              }
733.                                              }
734.                                              }
735.                                              }
736.                                              }
737.                                              }
738.                                              }
739.                                              }
740.                                              }
741.                                              }
742.                                              }
743.                                              }
744.                                              }
745.                                              }
746.                                              }
747.                                              }
748.                                              }
749.                                              }
750.                                              }
751.                                              }
752.                                              }
753.                                              }
754.                                              }
755.                                              }
756.                                              }
757.                                              }
758.                                              }
759.                                              }
760.                                              }
761.                                              }
762.                                              }
763.                                              }
764.                                              }
765.                                              }
766.                                              }
767.                                              }
768.                                              }
769.                                              }
770.                                              }
771.                                              }
772.                                              }
773.                                              }
774.                                              }
775.                                              }
776.                                              }
777.                                              }
778.                                              }
779.                                              }
780.                                              }
781.                                              }
782.                                              }
783.                                              }
784.                                              }
785.                                              }
786.                                              }
787.                                              }
788.                                              }
789.                                              }
790.                                              }
791.                                              }
792.                                              }
793.                                              }
794.                                              }
795.                                              }
796.                                              }
797.                                              }
798.                                              }
799.                                              }
800.                                              }
801.                                              }
802.                                              }
803.                                              }
804.                                              }
805.                                              }
806.                                              }
807.                                              }
808.                                              }
809.                                              }
810.                                              }
811.                                              }
812.                                              }
813.                                              }
814.                                              }
815.                                              }
816.                                              }
817.                                              }
818.                                              }
819.                                              }
820.                                              }
821.                                              }
822.                                              }
823.                                              }
824.                                              }
825.                                              }
826.                                              }
827.                                              }
828.                                              }
829.                                              }
830.                                              }
831.                                              }
832.                                              }
833.                                              }
834.                                              }
835.                                              }
836.                                              }
837.                                              }
838.                                              }
839.                                              }
840.                                              }
841.                                              }
842.                                              }
843.                                              }
844.                                              }
845.                                              }
846.                                              }
847.                                              }
848.                                              }
849.                                              }
850.                                              }
851.                                              }
852.                                              }
853.                                              }
854.                                              }
855.                                              }
856.                                              }
857.                                              }
858.                                              }
859.                                              }
860.                                              }
861.                                              }
862.                                              }
863.                                              }
864.                                              }
865.                                              }
866.                                              }
867.                                              }
868.                                              }
869.                                              }
870.                                              }
871.                                              }
872.                                              }
873.                                              }
874.                                              }
875.                                              }
876.                                              }
877.                                              }
878.                                              }
879.                                              }
880.                                              }
881.                                              }
882.                                              }
883.                                              }
884.                                              }
885.                                              }
886.                                              }
887.                                              }
888.                                              }
889.                                              }
890.                                              }
891.                                              }
892.                                              }
893.                                              }
894.                                              }
895.                                              }
896.                                              }
897.                                              }
898.                                              }
899.                                              }
900.                                              }
901.                                              }
902.                                              }
903.                                              }
904.                                              }
905.                                              }
906.                                              }
907.                                              }
908.                                              }
909.                                              }
910.                                              }
911.                                              }
912.                                              }
913.                                              }
914.                                              }
915.                                              }
916.                                              }
917.                                              }
918.                                              }
919.                                              }
920.                                              }
921.                                              }
922.                                              }
923.                                              }
924.                                              }
925.                                              }
926.                                              }
927.                                              }
928.                                              }
929.                                              }
930.                                              }
931.                                              }
932.                                              }
933.                                              }
934.                                              }
935.                                              }
936.                                              }
937.                                              }
938.                                              }
939.                                              }
940.                                              }
941.                                              }
942.                                              }
943.                                              }
944.                                              }
945.                                              }
946.                                              }
947.                                              }
948.                                              }
949.                                              }
950.                                              }
951.                                              }
952.                                              }
953.                                              }
954.                                              }
955.                                              }
956.                                              }
957.                                              }
958.                                              }
959.                                              }
960.                                              }
961.                                              }
962.                                              }
963.                                              }
964.                                              }
965.                                              }
966.                                              }
967.                                              }
968.                                              }
969.                                              }
970.                                              }
971.                                              }
972.                                              }
973.                                              }
974.                                              }
975.                                              }
976.                                              }
977.                                              }
978.                                              }
979.                                              }
980.                                              }
981.                                              }
982.                                              }
983.                                              }
984.                                              }
985.                                              }
986.                                              }
987.                                              }
988.                                              }
989.                                              }
990.                                              }
991.                                              }
992.                                              }
993.                                              }
994.                                              }
995.                                              }
996.                                              }
997.                                              }
998.                                              }
999.                                              }
1000.                                              }
1001.                                              }
1002.                                              }
1003.                                              }
1004.                                              }
1005.                                              }
1006.                                              }
1007.                                              }
1008.                                              }
1009.                                              }
1010.                                              }
1011.                                              }
1012.                                              }
1013.                                              }
1014.                                              }
1015.                                              }
1016.                                              }
1017.                                              }
1018.                                              }
1019.                                              }
1020.                                              }
1021.                                              }
1022.                                              }
1023.                                              }
1024.                                              }
1025.                                              }
1026.                                              }
1027.                                              }
1028.                                              }
1029.                                              }
1030.                                              }
1031.                                              }
1032.                                              }
1033.                                              }
1034.                                              }
1035.                                              }
1036.                                              }
1037.                                              }
1038.                                              }
1039.                                              }
1040.                                              }
1041.                                              }
1042.                                              }
1043.                                              }
1044.                                              }
1045.                                              }
1046.                                              }
1047.                                              }
1048.                                              }
1049.                                              }
1050.                                              }
1051.                                              }
1052.                                              }
1053.                                              }
1054.                                              }
1055.                                              }
1056.                                              }
1057.                                              }
1058.                                              }
1059.                                              }
1060.                                              }
1061.                                              }
1062.                                              }
1063.                                              }
1064.                                              }
1065.                                              }
1066.                                              }
1067.                                              }
1068.                                              }
1069.                                              }
1070.                                              }
1071.                                              }
1072.                                              }
1073.                                              }
1074.                                              }
1075.                                              }
1076.                                              }
1077.                                              }
1078.                                              }
1079.                                              }
1080.                                              }
1081.                                              }
1082.                                              }
1083.                                              }
1084.                                              }
1085.                                              }
1086.                                              }
1087.                                              }
1088.                                              }
1089.                                              }
1090.                                              }
1091.                                              }
1092.                                              }
1093.                                              }
1094.                                              }
1095.                                              }
1096.                                              }
1097.                                              }
1098.                                              }
1099.                                              }
1100.                                              }
1101.                                              }
1102.                                              }
1103.                                              }
1104.                                              }
1105.                                              }
1106.                                              }
1107.                                              }
1108.                                              }
1109.                                              }
1110.                                              }
1111.                                              }
1112.                                              }
1113.                                              }
1114.                                              }
1115.                                              }
1116.                                              }
1117.                                              }
1118.                                              }
1119.                                              }
1120.                                              }
1121.                                              }
1122.                                              }
1123.                                              }
1124.                                              }
1125.                                              }
1126.                                              }
1127.                                              }
1128.                                              }
1129.                                              }
1130.                                              }
1131.                                              }
1132.                                              }
1133.                                              }
1134.                                              }
1135.                                              }
1136.                                              }
1137.                                              }
1138.                                              }
1139.                                              }
1140.                                              }
1141.                                              }
1142.                                              }
1143.                                              }
1144.                                              }
1145.                                              }
1146.                                              }
1147.                                              }
1148.                                              }
1149.                                              }
1150.                                              }
1151.                                              }
1152.                                              }
1153.                                              }
1154.                                              }
1155.                                              }
1156.                                              }
1157.                                              }
1158.                                              }
1159.                                              }
1160.                                              }
1161.                                              }
1162.                                              }
1163.                                              }
1164.                                              }
1165.                                              }
1166.                                              }
1167.                                              }
1168.                                              }
1169.                                              }
1170.                                              }
1171.                                              }
1172.                                              }
1173.                                              }
1174.                                              }
1175.                                              }
1176.                                              }
1177.                                              }
1178.                                              }
1179.                                              }
1180.                                              }
1181.                                              }
1182.                                              }
1183.                                              }
1184.                                              }
1185.                                              }
1186.                                              }
1187.                                              }
1188.                                              }
1189.                                              }
1190.                                              }
1191.                                              }
1192.                                              }
1193.                                              }
1194.                                              }
1195.                                              }
1196.                                              }
1197.                                              }
1198.                                              }
1199.                                              }
1200.                                              }
1201.                                              }
1202.                                              }
1203.                                              }
1204.                                              }
1205.                                              }
1206.                                              }
1207.                                              }
1208.                                              }
1209.                                              }
1210.                                              }
1211.                                              }
1212.                                              }
1213.                                              }
1214.                                              }
1215.                                              }
1216.                                              }
1217.                                              }
1218.                                              }
1219.                                              }
1220.                                              }
1221.                                              }
1222.                                              }
1223.                                              }
1224.                                              }
1225.                                              }
1226.                                              }
1227.                                              }
1228.                                              }
1229.                                              }
1230.                                              }
1231.                                              }
1232.                                              }
1233.                                              }
1234.                                              }
1235.                                              }
1236.                                              }
1237.                                              }
1238.                                              }
1239.                                              }
1240.                                              }
1241.                                              }
1242.                                              }
1243.                                              }
1244.                                              }
1245.                                              }
1246.                                              }
1247.                                              }
1248.                                              }
1249.                                              }
1250.                                              }
1251.                                              }
1252.                                              }
1253.                                              }
1254.                                              }
1255.                                              }
1256.                                              }
1257.                                              }
1258.                                              }
1259.                                              }
1260.                                              }
1261.                                              }
1262.                                              }
1263.                                              }
1264.                                              }
1265.                                              }
1266.                                              }
1267.                                              }
1268.                                              }
1269.                                              }
1270.                                              }
1271.                                              }
1272.                                              }
1273.                                              }
1274.                                              }
1275.                                              }
1276.                                              }
1277.                                              }
1278.                                              }
1279.                                              }
1280.                                              }
1281.                                              }
1282.                                              }
1283.                                              }
1284.                                              }
1285.                                              }
1286.                                              }
1287.                                              }
1288.                                              }
1289.                                              }
1290.                                              }
1291.                                              }
1292.                                              }
1293.                                              }
1294.                                              }
1295.                                              }
1296.                                              }
1297.                                              }
1298.                                              }
1299.                                              }
1300.                                              }
1301.                                              }
1302.                                              }
1303.                                              }
1304.                                              }
1305.                                              }
1306.                                              }
1307.                                              }
1308.                                              }
1309.                                              }
1310.                                              }
1311.                                              }
1312.                                              }
1313.                                              }
1314.                                              }
1315.                                              }
1316.                                              }
1317.                                              }
1318.                                              }
1319.                                              }
1320.                                              }
1321.                                              }
1322.                                              }
1323.                                              }
1324.                                              }
1325.                                              }
1326.                                              }
1327.                                              }
1328.                                              }
1329.                                              }
1330.                                              }
1331.                                              }
1332.                                              }
1333.                                              }
1334.                                              }
1335.                                              }
1336.                                              }
1337.                                              }
1338.                                              }
1339.                                              }
1340.                                              }
1341.                                              }
1342.                                              }
1343.                                              }
1344.                                              }
1345.                                              }
1346.                                              }
1347.                                              }
1348.                                              }
1349.                                              }
1350.                                              }
1351.                                              }
1352.                                              }
1353.                                              }
1354.                                              }
1355.                                              }
1356.                                              }
1357.                                              }
1358.                                              }
1359.                                              }
1360.                                              }
1361.                                              }
1362.                                              }
1363.                                              }
1364.                                              }
1365.                                              }
1366.                                              }
1367.                                              }
1368.                                              }
1369.                                              }
1370.                                              }
1371.                                              }
1372.                                              }
1373.                                              }
1374.                                              }
1375.                                              }
1376.                                              }
1377.                                              }
1378.                                              }
1379.                                              }
1380.                                              }
1381.                                              }
1382.                                              }
1383.                                              }
1384.                                              }
1385.                                              }
1386.                                              }
1387.                                              }
1388.                                              }
1389.                                              }
1390.                                              }
1391.                                              }
1392.                                              }
1393.                                              }
1394.                                              }
1395.                                              }
1396.                                              }
1397.                                              }
1398.                                              }
1399.                                              }
1400.                                              }
1401.                                              }
1402.                                              }
1403.                                              }
1404.                                              }
1405.                                              }
1406.                                              }
1407.                                              }
1408.                                              }
1409.                                              }
1410.                                              }
1411.                                              }
1412.                                              }
1413.                                              }
1414.                                              }
1415.                                              }
1416.                                              }
1417.                                              }
1418.                                              }
1419.                                              }
1420.                                              }
1421.                                              }
1422.                                              }
1423.                                              }
1424.                                              }
1425.                                              }
1426.                                              }
1427.                                              }
1428.                                              }
1429.                                              }
1430.                                              }
1431.                                              }
1432.                                              }
1433.                                              }
1434.                                              }
1435.                                              }
1436.                                              }
1437.                                              }
1438.                                              }
1439.                                              }
1440.                                              }
1441.                                              }
1442.                                              }
1443.                                              }
1444.                                              }
1445.                                              }
1446.                                              }
1447.                                              }
1448.                                              }
1449.                                              }
1450.                                              }
1451.                                              }
1452.                                              }
1453.                                              }
1454.                                              }
1455.                                              }
1456.                                              }
1457.                                              }
1458.                                              }
1459.                                              }
1460.                                              }
1461.                                              }
1462.                                              }
1463.                                              }
1464.                                              }
1465.                                              }
1466.                                              }
1467.                                              }
1468.                                              }
1469.                                              }
1470.                                              }
1471.                                              }
1472.                                              }
1473.                                              }
1474.                                              }
1475.                                              }
1476.                                              }
1477.                                              }
1478.                                              }
1479.                                              }
1480.                                              }
1481.                                              }
1482.                                              }
1483.                                              }
1484.                                              }
1485.                                              }
1486.                                              }
1487.                                              }
1488.                                              }
1489.                                              }
1490.                                              }
1491.                                              }
1492.                                              }
1493.                                              }
1494.                                              }
1495.                                              }
1496.                                              }
1497.                                              }
1498.                                              }
1499.                                              }
1500.                                              }
1501.                                              }
1502.                                              }
1503.                                              }
1504.                                              }
1505.                                              }
1506.                                              }
1507.                                              }
1508.                                              }
1509.                                              }
1510.                                              }
1511.                                              }
1512.                                              }
1513.                                              }
1514.                                              }
1515.                                              }
1516.                                              }
1517.                                              }
1518.                                              }
1519.                                              }
1520.                                              }
1521.                                              }
1522.                                              }
1523.                                              }
1524.                                              }
1525.                                              }
1526.                                              }
1527.                                              }
1528.                                              }
1529.                                              }
1530.                                              }
1531.                                              }
1532.                                              }
1533.                                              }
1534.                                              }
1535.                                              }
1536.                                              }
1537.                                              }
1538.                                              }
1539.                                              }
1540.                                              }
1541.                                              }
1542.                                              }
1543.                                              }
1544.                                              }
1545.                                              }
1546.                                              }
1547.                                              }
1548.                                              }
1549.                                              }
1550.                                              }
1551.                                              }
1552.                                              }
1553.                                              }
1554.                                              }
1555.                                              }
1556.                                              }
1557.                                              }
1558.                                              }
1559.                                              }
1560.                                              }
1561.                                              }
1562.                                              }
1563.                                              }
1564.                                              }
1565.                                              }
1566.                                              }
1567.                                              }
1568.                                              }
1569.                                              }
1570.                                              }
1571.                                              }
1572.                                              }
1573.                                              }
1574.                                              }
1575.                                              }
1576.                                              }
1577.                                              }
1578.                                              }
1579.                                              }
1580.                                              }
1581.                                              }
1582.                                              }
1583.                                              }
1584.                                              }
1585.                                              }
1586.                                              }
1587.                                              }
1588.                                              }
1589.                                              }
1590.                                              }
1591.                                              }
1592.                                              }
1593.                                              }
1594.                                              }
1595.                                              }
1596.                                              }
1597.                                              }
1598.                                              }
1599.                                              }
1600.                                              }
1601.                                              }
1602.                                              }
1603.                                              }
1604.                                              }
1605.                                              }
1606.                                              }
1607.                                              }
1608.                                              }
1609.                                              }
1610.                                              }
1611.                                              }
1612.                                              }
1613.                                              }
1614.                                              }
1615.                                              }
1616.                                              }
1617.                                              }
1618.                                              }
1619.                                              }
1620.                                              }
1621.                                              }
1622.                                              }
1623.                                              }
1624.                                              }
1625.                                              }
1626.                                              }
1627.                                              }
1628.                                              }
1629.                                              }
1630.                                              }
1631.                                              }
1632.                                              }
1633.                                              }
1634.                                              }
1635.                                              }
1636.                                              }
1637.                                              }
1638.                                              }
1639.                                              }
1640.                                              }
1641.                                              }
1642.                                              }
1643.                                              }
1644.                                              }
1645.                                              }
1646.                                              }
1647.                                              }
1648.                                              }
1649.                                              }
1650.                                              }
1651.                                              }
1652.                                              }
1653.                                              }
1654.                                              }
1655.                                              }
1656.                                              }
1657.                                              }
1658.                                              }
1659.                                              }
1660.                                              }
1661.                                              }
1662.                                              }
1663.                                              }
1664.                                              }
1665.                                              }
1666.                                              }
1667.                                              }
1668.                                              }
1669.                                              }
1670.                                              }
1671.                                              }
1672.                                              }
1673.                                              }
1674.                                              }
1675.                                              }
1676.                                              }
1677.                                              }
1678.                                              }
1679.                                              }
1680.                                              }
1681.                                              }
1682.                                              }
1683.                                              }
1684.                                              }
1685.                                              }
1686.                                              }
1687.                                              }
1688.                                              }
1689.                                              }
1690.                                              }
1691.                                              }
1692.                                              }
1693.                                              }
1694.                                              }
1695.                                              }
1696.                                              }
1697.                                              }
1698.                                              }
1699.                                              }
1700.                                              }
1701.                                              }
1702.                                              }
1703.                                              }
1704.                                              }
1705.                                              }
1706.                                              }
1707.                                              }
1708.                                              }
1709.                                              }
1710.                                              }
1711.                                              }
1712.                                              }
1713.                                              }
1714.                                              }
1715.                                              }
1716.                                              }
1717.                                              }
1718.                                              }
1719.                                              }
1720.                                              }
1721.                                              }
1722.                                              }
1723.                                              }
1724.                                              }
1725.                                              }
1726.                                              }
1727.                                              }
1728.                                              }
1729.                                              }
1730.                                              }
1731.                                              }
1732.                                              }
1733.                                              }
1734.                                              }
1735.                                              }
1736.                                              }
1737.                                              }
1738.                                              }
1739.                                              }
1740.                                              }
1741.                                              }
1742.                                              }
1743.                                              }
1744.                                              }
1745.                                              }
1746.                                              }
1747.                                              }
1748.                                              }
1749.                                              }
1750.                                              }
1751.                                              }
1752.                                              }
1753.                                              }
1754.                                              }
1755.                                              }
1756.                                              }
1757.                                              }
1758.                                              }
1759.                                              }
1760.                                              }
1761.                                              }
1762.                                              }
1763.                                              }
1764.                                              }
1765.                                              }
1766.                                              }
1767.                                              }
1768.                                              }
1769.                                              }
1770.                                              }
1771.                                              }
1772.                                              }
1773.                                              }
1774.                                              }
1775.                                              }
1776.                                              }
1777.                                              }
1778.                                              }
1779.                                              }
1780.                                              }
1781.                                              }
1782.                                              }
1783.                                              }
1784.                                              }
1785.                                              }
1786.                                              }
1787.                                              }
1788.                                              }
1789.                                              }
1790.                                              }
1791.                                              }
1792.                                              }
1793.                                              }
1794.                                              }
1795.                                              }
1796.                                              }
1797.                                              }
1798.                                              }
1799.                                              }
1800.                                              }
1801.                                              }
1802.                                              }
1803.                                              }
1804.                                              }
1805.                                              }
1806.                                              }
1807.                                              }
1808.                                              }
1809.                                              }
1810.                                              }
1811.                                              }
1812.                                              }
1813.                                              }
1814.                                              }
1815.                                              }
1816.                                              }
1817.                                              }
1818.                                              }
1819.                                              }
1820.                                              }
1821.                                              }
1822.                                              }
1823.                                              }
1824.                                              }
1825.                                              }
1826.                                              }
1827.                                              }
1828.                                              }
1829.                                              }
1830.                                              }
1831.                                              }
1832.                                              }
1833.                                              }
1834.                                              }
1835.                                              }
1836.                                              }
1837
```

```

261.
262.           //Set calendar event will be saved to
263.           event.calendar =
264.           eventStore.calendar(withIdentifier: DEFAULTS.object(forKey:
265.           "GoGigCalendar") as! String)
266.           do {
267.               //Save the event with completion
268.               try eventStore.save(event, span:
269.               .thisEvent)
270.           } //Process any errors
271.           } catch let e as NSError {
272.               completion?(false, e)
273.               return
274.           }
275.           completion?(true, nil)
276.       } else {
277.           completion?(false, error as NSError?)
278.       }
279.   }
280. }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
imageCache	NSCache<NSString, UIImage>	Global	To reduce data usage by storing and accessing frequently used UIImages under a string key rather than downloading them.
tap	UITapGestureRecognizer	Local	Recognise when screen is tapped to dismiss keyboard
alertController	UIAlertController	Local	To display errors and important information
first	Any	Local	For quicksort, the first element of the array.
smallerOrEqual	Array	Local	For quicksort, array of values smaller or equal to <u>first</u> .
larger	Array	Local	For quicksort, array of values larger than <u>first</u> .
photoPopup	UIAlertController	Local	To ask user what source they want images from.
cameraAction photoAction videoAction	UIAlertAction	Local	Buttons to add to photo popup to decide image source.
urlString	String	Local	String of image download url for image caching.
cachedImage	UIImage	Local	UIImage fetched from a cache.
task	NSObject	Local	Task to download images
downloadedImage	UIImage	Local	UIImage downloaded from a URL.

imageGenerator	AVAssetImageGenerator	Local	To grab a thumbnail image from a posted video.
cgImage	CGImage	Local	The thumbnail image grabbed from the video.
eventStore	EKEventStore	Local	The application's point of contact for accessing the calendar.
event	EKEvent	Local	The event object which will be added to calendar.
newCalendar	EKCalendar	Local	A unique calendar called "GoGig – My Gigs" which is created once only.

## StringExtensions.swift

```

1. import Foundation
2.
3. extension String {
4.
5.     //Swift string extension (like Java)
6.     func substring(start: Int, end: Int) -> String {
7.         //the start index
8.         let start = self.index(self.startIndex, offsetBy: start)
9.         //the end index
10.        let end = self.index(self.startIndex, offsetBy: end)
11.        //the range from start to end
12.        let range = start..

```

## UserAccountTableView.swift

```

1. import UIKit
2. import AVKit
3. import AVFoundation
4. import GoogleMaps
5. import GooglePlaces
6.
7. //MARK: UserAccountVC TableView
8. extension UserAccountVC {
9.
10.     //Now each row of the table view is a section to allow
11.     //padding
12.     override func numberOfSections(in tableView: UITableView) -> Int {
13.         //+1 is AccountHeaderCell
14.         return portfolioPosts.count + 1
15.     }
16. }
```

```
15.
16.        //In each section is one row
17.        override func tableView(_ tableView: UITableView,
18.            numberOfRowsInSection section: Int) -> Int {
19.                return 1
20.
21.        //set the appearance of each cell
22.        //We access the row using indexPath.section instead of
23.        //indexPath.row
23.        override func tableView(_ tableView: UITableView,
24.            cellForRowAt indexPath: IndexPath) -> UITableViewCell {
24.
25.                //FIRST CELL IS USER PROFILE
26.                if indexPath.section == 0 {
27.                    let headerCell =
28.                        tableView.dequeueReusableCell(withIdentifier: "AccountHeaderCell",
29.                            for: indexPath) as! AccountHeaderCell
30.
31.                    //On initial launch of the app, clean the header
32.                    if hideForLoad {
33.                        headerCell.userBioTextView.isHidden = true
34.                        headerCell.userTypeLabel.isHidden = true
35.                        headerCell.socialLinkStackView.isHidden = true
36.                    } else {
37.                        headerCell.userBioTextView.isHidden = false
38.                        headerCell.userTypeLabel.isHidden = false
39.                        headerCell.socialLinkStackView.isHidden = false
40.
41.                    //update the cell with the user data
42.                    updateUserData(cell: headerCell)
43.
44.                    return headerCell
45.
46.                //OTHER CELLS ARE PORTFOLIO POSTS
47.            } else {
48.                //instantiate a reusable post cell
49.                let postCell =
50.                    tableView.dequeueReusableCell(withIdentifier: "AccountPostCell",
51.                        for: indexPath)
52.                        as! AccountPostCell
53.
54.                //add post data to the cell
55.                updatepostData(cell: postCell, row:
56.                    indexPath.section - 1)
57.
58.                return postCell
59.
60.            } //To update the height of the row depending on the post
61.            override func tableView(_ tableView: UITableView,
62.                heightForRowAt indexPath: IndexPath) -> CGFloat {
63.                    //Account Header is always same height
```

```

61.             if indexPath.section == 0 {
62.                 return 215
63.             } else {
64.                 //this is the ratio of feed width to post width
65.                 let ratio = (tableView.frame.size.width - 32) /
66.                 ((portfolioPosts[indexPath.section-1].dimensions["width"] as?
67.                   CGFloat)!)
68.                 //set the height of the cell based off ratio
69.                 return ((portfolioPosts[indexPath.section-
70.                   1].dimensions["height"] as? CGFloat)! * ratio) + 92
71.             }
72.         }
73.     }
74. 
```

//Add padding to the cells

```

75.     override func tableView(_ tableView: UITableView,
76.     heightForHeaderInSection section: Int) -> CGFloat {
77.         if section != 0 {
78.             return 20
79.         }
80.         //No top padding for top cell
81.         return 0
82.     }
83. 
```

// Make the background color show through

```

84.     override func tableView(_ tableView: UITableView,
85.     viewForHeaderInSection section: Int) -> UIView? {
86.         let headerView = UIView()
87.         headerView.backgroundColor = UIColor.clear
88.         return headerView
89.     }
90. 
```

//when scrolling the feed

```

91.     override func scrollViewWillBeginDragging(_ scrollView:
92.     UIScrollView) {
93.         if #available(iOS 13.0, *) {
94.             //give the navigation bar an opaque white colour
95.             let navBarAppearance = UINavigationBarAppearance()
96.             navBarAppearance.configureWithOpaqueBackground()
97.             navBarAppearance.backgroundColor =
98.               UIColor.white.withAlphaComponent(0.95)
99.             self.navigationController?.navigationBar.standardAppearance =
100.               navBarAppearance
101.             self.navigationController?.navigationBar.scrollEdgeAppearance =
102.               navBarAppearance
103.         }
104.     }
105. 
```

//when stopped scrolling feed

```

106.     override func scrollViewWillEndDragging(_ scrollView:
107.     UIScrollView, withVelocity velocity: CGPoint, targetContentOffset:
108.     UnsafeMutablePointer<CGPoint>) {
109.         if #available(iOS 13.0, *) {
110.             //give the navigation bar an opaque white colour
111.         }
112.     }
113. 
```

```

103.         let navBarAppearance = UINavigationBarAppearance()
104.             navBarAppearance.configureWithOpaqueBackground()
105.             navBarAppearance.backgroundColor =
106.                 UIColor.white.withAlphaComponent(0.75)
107.             self.navigationController?.navigationBar.standardAppearance =
108.                 navBarAppearance
109.         }
110.     }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
headerCell	AccountHeaderCell	Local	Portfolio cell all profile data will be displayed on.
postCell	AccountPostCell	Local	Portfolio cell all post data will be displayed on.
ratio	CGFloat	Local	To set correct dimensions for the <u>PostContainerView</u>
headerView	UIView	Local	To display background through the portfolio table view.
navBarAppearance	UINavigationBarAppearance	Local	To customise appearance of navigation bar.

## ActivityFeedCollectionView.swift

```

1. import UIKit
2.
3. extension ActivityFeedVC {
4.
5.     //MARK: COLLECTION VIEW METHODS
6.
7.     func collectionView(_ collectionView: UICollectionView,
8.         numberOfItemsInSection section: Int) -> Int {
9.         //Each collection view cell contains a table view
10.        return 2
11.    }
12.
13.    func collectionView(_ collectionView: UICollectionView,
14.        cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
15.        let cell =
16.            collectionView.dequeueReusableCell(withIdentifier: "cvCell",
17.                for: indexPath) as! ActivityCVCCell
18.        //give each collection view cell a background
19.        setupView(tableView: cell.feedTableView)
20.    }

```

```
16.                 cell.feedTableView.reloadData()
17.
18.             return cell
19.         }
20.
21.         func collectionView(_ collectionView: UICollectionView,
22.                             layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt
23.                             indexPath: IndexPath) -> CGSize {
24.             //size of the collection view cells
25.             return CGSize(width: collectionView.frame.width,
26.                           height: collectionView.frame.height)
27.         }
28.
29.         //called before the UICollectionViewCell appears to the
30.         //view
31.         func collectionView(_ collectionView: UICollectionView,
32.                             willDisplay cell: UICollectionViewCell, forItemAt indexPath:
33.                             IndexPath) {
34.
35.             guard let collectionViewCell = cell as? ActivityCVCell
36.             else { return }
37.
38.             collectionViewCell.setTableViewDataSourceDelegate(dataSourceDelegate
39. : self as UITableViewDelegate & UITableViewDataSource, forRow:
40. indexPath.item)
41.             collectionViewCell.tableViewOffset =
42.             storedOffsets[indexPath.row] ?? 0
43.         }
44.
45.         //called after the UICollectionViewCell disappears from the
46.         //view
47.         func collectionView(_ collectionView: UICollectionView,
48.                             didEndDisplaying cell: UICollectionViewCell, forItemAt indexPath:
49.                             IndexPath) {
50.
51.             guard let collectionViewCell = cell as? ActivityCVCell
52.             else { return }
53.
54.             storedOffsets[indexPath.row] =
55.             collectionViewCell.tableViewOffset
56.         }
57.
58.         //MARK: TABLEVIEW METHODS
59.
60.         func numberOfSections(in tableView: UITableView) -> Int {
61.             //Section 1 - Activity
62.             //Section 2 - Loading Cell
63.             if tableView.tag == 0 {
64.                 return 2
65.             } else {
66.                 //Don't need loading cell on the event listings
67.                 return 1
68.             }
69.         }
70.     }
71. }
```

```
55.
56.        func tableView(tableView: UITableView,
57.                      numberOfRowsInSection section: Int) -> Int {
58.            if tableView.tag == 0 {
59.                //notifications section
60.                if section == 0 {
61.                    //amount of cells needed
62.                    return activityNotifications.count
63.
64.                //loading more section
65.            } else {
66.                //if we are fetching more then return an extra
67.                //cell (loading cell)
68.                return fetchingMore ? 1 : 0
69.            }
70.        } else {
71.            //event listing section
72.            return usersEvents.count
73.        }
74.
75.        func tableView(tableView: UITableView, cellForRowAtIndexPath: IndexPath) -> UITableViewCell {
76.            let cell =
77.                tableView.dequeueReusableCell(withIdentifier: "NotificationCell",
78.                                              for: indexPath) as! ActivityFeedCell
79.                //set nil while loading (cleaner)
80.                cell.notificationImage.image = nil
81.                //if notifications section
82.                if tableView.tag == 0 {
83.                    //update notifications
84.                    updateNotificationData(cell: cell, row:
85.                        indexPath.row)
86.                    //if 'My Events' section
87.                } else {
88.                    //update the event listings
89.                    updateEventListingData(cell: cell, row:
90.                        indexPath.row)
91.                }
92.
93.                //when cell has been tapped (selected)
94.                func tableView(tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
95.
96.                    //activity notifications section
97.                    if tableView.tag == 0 {
98.                        let selectedNotification =
99.                            activityNotifications[indexPath.row]
99.                            //Making sure only an organisier can navigate to
99.                            //review application
```

```

100.          if user!.gigs == false &&
101.            selectedNotification.getType() != "personal" &&
102.              selectedNotification.getType() == "applied" {
103.                //So data is transferred when segue is performed
104.                checkUid =
105.                  activityNotifications[indexPath.row].getSenderId()
106.                  selectedApplication =
107.                    activityNotifications[indexPath.row]
108.                    performSegue(withIdentifier:
109.                      TO REVIEW APPLICATION, sender: nil)
110.                    }
111.                    //event listings section
112.                    } else {
113.                      selectedListing = usersEvents[indexPath.row]
114.                      performSegue(withIdentifier:
115.                        TO EVENT DESCRIPTION_2, sender: nil)
116.                        }
117.                        //MARK: MENUBAR METHODS
118.                        //change the purple bar position when scrolling
119.                        func scrollViewDidScroll(_ scrollView: UIScrollView) {
120.                          //Scrolled the collection view horizontally
121.                          if scrollView == self.collectionView {
122.                            menuBar.barLeftAnchorConstraint?.constant =
123.                              scrollView.contentOffset.x / 2
124.                            //scrolled the table view vertically
125.                            } else {
126.                              //this is needed incase user has no data, causes a
127.                              crash!
128.                              if activityNotifications.count != 0 {
129.                                let offsetY = scrollView.contentOffset.y
130.                                let contentHeight =
131.                                  scrollView.contentSize.height
132.                                  if offsetY > contentHeight -
133.                                    scrollView.frame.size.height * leadingScreensForBatching {
134.                                      //if there is more activity to fetch
135.                                      if !fetchingMore && !endReached {
136.                                        getMoreNotifications()
137.                                      }
138.                                    }
139.                                  }
140.                                }
141.                                //change the state of menu bar when we finish dragging the
142.                                large collection view cells

```

```

143.         func scrollViewWillEndDragging(_ scrollView: UIScrollView,
144.             withVelocity velocity: CGPoint, targetContentOffset:
145.             UnsafeMutablePointer<CGPoint>) {
146.             if scrollView == self.collectionView {
147.                 let index = targetContentOffset.pointee.x /
148.                 view.frame.width
149.                 let indexPath = NSIndexPath(item: Int(index),
150.                     section: 0)
151.                 //go to the collection view cell at index
152.                 menuBar.collectionView.selectItem(at: indexPath as
153.                 IndexPath, animated: false, scrollPosition: .centeredHorizontally)
154.                 //For selection of what button is pressed from what
155.                 selectedCVCell = Int(index)
156.             }
157.         }
158.     }
159.     let indexPath = NSIndexPath(item: menuIndex, section:
160.         0)
161.     //automatically scroll to that collection view cell
162.     collectionView.scrollToItem(at: indexPath as IndexPath,
163.     at: .centeredHorizontally, animated: true)
164.     selectedCVCell = menuIndex
165. }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
cell	ActivityCVCell	Local	To display table views in the collection view.
cell	ActivityFeedCell	Local	To display notifications and events in the table view.
selectedNotification	ActivityNotification	Local	The ActivityNotification object corresponding to the tapped cell.
offsetY	CGPoint	Local	To help with deciding when user has scrolled enough to fetch more notifications from Database.
contentHeight	CGSize	Local	To help with deciding when user has scrolled enough to fetch more notifications from Database.
indexPath	NSIndexPath	Local	To keep track of what state the collection view and menu bar are in so they can communicate accordingly.

# Services

## AuthService.swift

```

1. import Foundation
2. import FirebaseAuth
3.
4. class AuthService {
5.
6.     //using a singleton
7.     static let instance = AuthService()
8.
9.     //subscribe the user to Firebase Authentication
10.    func registerUser(withEmail email: String, andPassword
11.        password: String, userCreationComplete: @escaping (_ status: Bool,
12.        _ error: Error?) -> ()) {
13.
14.        //adds user to Auth list
15.        Auth.auth().createUser(withEmail: email, password:
16.            password) { (authResult, error) in
17.
18.            //if there is no result of creating user
19.            guard let user = authResult?.user else {
20.
21.                //complete as error
22.                userCreationComplete(false, error)
23.
24.                return
25.            }
26.
27.            //adds user to the database
28.            let userData = ["provider": user.providerID,
29.                "email": user.email] //provider = Firebase/Facebook/Google/Email
30.
31.            //add this userData to Database
32.            DataService.instance.createDBUser(uid: user.uid ,
33.                userData: userData as Dictionary<String, Any>)
34.
35.            //complete as no error
36.            userCreationComplete(true, nil)
37.
38.        }
39.
40.        //Log the user in with Firebase Authentication
41.        func loginUser(withEmail email: String, andPassword
42.            password: String, loginComplete: @escaping (_ status: Bool, _ error: Error?) -> ()) {
43.
44.            //sign them in
45.            Auth.auth().signIn(withEmail: email, password:
46.                password) { (authResult, error) in
47.
48.                //if there is an error signing in
49.                if error != nil {
50.
51.                    //complete with error
52.                    loginComplete(false, error)
53.
54.                    return
55.                }
56.
57.                //complete with no error
58.                loginComplete(true, nil)
59.
60.            }
61.
62.        }
63.
64.    }
65.
66. }

```

```

44.
45.        //MARK: USER DEFAULTS
46.        //To keep user logged in
47.        let defaults = UserDefaults.standard
48.
49.        var isLoggedIn: Bool {
50.            get {
51.                //get the Boolean of logged in
52.                return defaults.bool(forKey: LOGGED_IN_KEY)
53.            }
54.            set {
55.                //set the Boolean of logged in
56.                defaults.set(newValue, forKey: LOGGED_IN_KEY)
57.            }
58.        }
59.
60.        var userEmail: Bool {
61.            get {
62.
63.                return defaults.bool(forKey: USER_EMAIL)
64.            }
65.            set {
66.
67.                defaults.set(newValue, forKey: USER_EMAIL)
68.            }
69.        }
70.    }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
instance	AuthService	Static Public	To initialise the AuthService class a single time to share it globally.
userData	Dictionary<String, Any>	Local	Simply contains their email and Auth provider to add to Database
isLoggedIn	Boolean	Local	Closure for UserDefaults to keep user logged in.
userEmail	Boolean	Local	Closure for UserDefaults to keep user logged in.

## DataService.swift

```

1. import Foundation
2. import FirebaseAuth
3. import FirebaseDatabase
4. import FirebaseStorage
5.
6. //location references for Database and Storage
7. let DB_BASE = Database.database().reference()
8. let ST_BASE = Storage.storage().reference()
9.
10. //handles for observing Database changes
11. var postsHandle: DatabaseHandle?
12. var activityHandle: DatabaseHandle?

```

```

13.      var eventsHandle: DatabaseHandle?
14.
15.      class DataService {
16.
17.          //using a singleton
18.          static let instance = DataService()
19.
20.          //MARK: DATABASE
21.
22.          //private for safety
23.          //specific location references
24.          private var _REF_BASE = DB_BASE
25.          private var _REF_USERS = DB_BASE.child("users")
26.          private var _REF_EVENTS = DB_BASE.child("events")
27.
28.          //closures to get references and
29.          //ensure we dont change values
30.          var REF_BASE: DatabaseReference {
31.
32.              return _REF_BASE
33.          }
34.
35.          var REF_USERS: DatabaseReference {
36.
37.              return _REF_USERS
38.          }
39.
40.          var REF_EVENTS: DatabaseReference {
41.
42.              return _REF_EVENTS
43.          }
44.
45.          //MARK: OBSERVERS
46.
47.          func removeObservers(uid: String) {
48.              //portfolio is the initial view so there will always be
49.              //an observer
50.              //may not be an observer as user may not have clicked
51.              //on those tabs since launch
52.              //store the closure under a global variable (handle)
53.              //and only remove it if it has a value
54.              //is observing)
55.              if eventsHandle != nil && activityHandle != nil {
56.                  REF_USERS.child(uid).child("events").removeObserver(withHandle:
57.                  eventsHandle!)
58.              }
59.          }
60.
61.          }
62.
63.      }
64.
65.  }

```

```
59.          //MARK: DATABASE USER PROFILE
60.
61.          //add the user under 'auth' to Database
62.          func createDBUser(uid: String, userData: Dictionary<String,
63.          Any>) {
64.              REF_USERS.child(uid).child("auth").updateChildValues(userData)
65.          }
66.          //add the user data under 'profile'
67.          func updateDBUserProfile(uid: String, userData:
68.          Dictionary<String, Any>, handler: @escaping (_ completion: Bool) ->
69.          ())
70.
71.          REF_USERS.child(uid).child("profile").updateChildValues(userData) {
72.              (error:Error?, ref:DatabaseReference) in
73.              if error != nil {
74.                  handler(false)
75.              } else {
76.                  handler(true)
77.              }
78.          }
79.          //get the user data under 'profile'
80.          func getDBUserProfile(uid: String, handler: @escaping (_
81.          user: User) -> ())
82.
83.          //Grab user profile data from the database once...
84.
85.          REF_USERS.child(uid).child("profile").observeSingleEvent(of: .value,
86.          with: { (profileSnapshot) in
87.
88.              //... and cast as a NSDictionary
89.              let profileData = profileSnapshot.value as?
90.              NSDictionary
91.
92.              //Get every value from every key of the dictionary
93.              if let currentUserName = profileData?["name"] as?
94.              String {
95.                  if let currentUserEmail = profileData?["email"]
96.                  as? String {
97.                      if let currentUserGigs =
98.                      profileData?["gigs"] as? Bool {
99.                          if let currentUserBio =
100.                          profileData?["bio"] as? String {
101.                              if let currentUserPicURLStr =
102.                              profileData?["picURL"] as? String {
103.
104.                                  if let currentUserPhone =
105.                                  profileData?["phone"] as? String {
106.
107.                                      if let currentUserFacebook =
108.                                      profileData?["facebook"] as? String {
109.
110.                                          if let
111.                                          currentUserTwitter = profileData?["twitter"] as? String {
112.
113.                                              if let
114.                                              currentUserInstagram = profileData?["instagram"] as? String {
```

```

95.                                     if let
  currentUserWebsiteURLStr = profileData?["website"] as? String {
96.                                         if let
  currentUserAppleMusicURLStr = profileData?["appleMusic"] as? String
  {
97.                                         if let
  currentUserSpotifyURLStr = profileData?["spotify"] as? String {
98.
99.
100.                                     let
  currentUserPicURL = URL(string: currentUserPicURLStr)
101.
102.
103.                                     guard let currentUserFCMToken = profileData?["FCMToken"] as? String
  else { return }
104.
105.
  //instansiate a new user object from the data grabbed
106.                                     let
  currentUser = User(uid: uid, name: currentUserName, email:
  currentUserEmail, phone: currentUserPhone, bio: currentUserBio,
  gigs: currentUserGigs, picURL: currentUserPicURL!, facebook:
  currentUserFacebook, twitter: currentUserTwitter, instagram:
  currentUserInstagram, website: currentUserWebsiteURLStr, appleMusic:
  currentUserAppleMusicURLStr, spotify: currentUserSpotifyURLStr,
  fcmToken: currentUserFCMToken)
107.
108.
  //return the user
109.
  handler(currentUser)
110.                                     }
111.                                     }
112.                                     }
113.                                     }
114.                                     }
115.                                     }
116.                                     }
117.                                     }
118.                                     }
119.                                     }
120.                                     }
121.                                     }
122.                                     //print error if there was a problem
123.                                     } { (error) in
124.                                         print("Error Getting user profile")
125.                                         print(error.localizedDescription)
126.                                     }
127.                                     }
128.
129.                                     //MARK: DATABASE USER PORTFOLIO POSTS
130.
131.                                     func updateDBPortfolioPosts(uid: String, postID: String,
  postData: Dictionary<String, Any>){

```

```
132.          //We want to build an array of posts to grab and loop
    through in table view
133.
    REF_USERS.child(uid).child("posts").child(postID).updateChildValues(
    postData)
134.        }
135.
136.        //we now store the postID in the database as well
137.        func deleteDBPortfolioPosts(uid: String, postID: String) {
138.
    REF_USERS.child(uid).child("posts").child(postID).removeValue()
139.        }
140.
141.        //We're going to return an array to loop through full of
    post objects
142.        func getDBPortfolioPosts(uid: String, handler: @escaping (_
    posts: [PortfolioPost]) -> ()) {
143.
144.            var portfolioPosts = [PortfolioPost]()
145.
146.            //Grab the array full of posts
147.            postsHandle =
    REF_USERS.child(uid).child("posts").observe(.value, with: {
    (snapshot) in
148.
149.                //Grab an array of all posts in the database
150.                if let snapshot = snapshot.children.allObjects as?
    [DataSnapshot] {
151.
152.                    //Loop through them and grab data for
    instantiation
153.                    for snap in snapshot {
154.
155.                        if let postData = snap.value as?
    NSDictionary {
156.
157.                            if let postID = postData["postID"] as?
    String {
158.
159.                                if let postURLStr =
    postData["postURL"] as? String {
160.
161.                                    if let thumbnailURLStr =
    postData["thumbnailURL"] as? String {
162.
163.                                        if let timeInterval =
    postData["timestamp"] as? TimeInterval {
164.
165.                                            if let postIsImage =
    postData["isImage"] as? Bool {
166.
167.                                                if let postCaption =
    postData["caption"] as? String {
168.
169.                                                    if let
    postLocation = postData["location"] as? String {
170.
171.                                                        if let
    postDimensions = postData["dimensions"] as? Dictionary<String, Any>
    {
172.
173.                }
174.
175.            }
176.
177.        }
178.
179.    }
180.
181.    }
182.
183.    }
184.
185.    }
186.
187.    }
188.
189.    }
190.
191.    }
192.
193.    }
194.
195.    }
196.
197.    }
198.
199.    }
200.
201.    }
202.
203.    }
204.
205.    }
206.
207.    }
208.
209.    }
210.
211.    }
212.
213.    }
214.
215.    }
216.
217.    }
218.
219.    }
220.
221.    }
222.
223.    }
224.
225.    }
226.
227.    }
228.
229.    }
230.
231.    }
232.
233.    }
234.
235.    }
236.
237.    }
238.
239.    }
240.
241.    }
242.
243.    }
244.
245.    }
246.
247.    }
248.
249.    }
250.
251.    }
252.
253.    }
254.
255.    }
256.
257.    }
258.
259.    }
260.
261.    }
262.
263.    }
264.
265.    }
266.
267.    }
268.
269.    }
270.
271.    }
272.
273.    }
274.
275.    }
276.
277.    }
278.
279.    }
280.
281.    }
282.
283.    }
284.
285.    }
286.
287.    }
288.
289.    }
290.
291.    }
292.
293.    }
294.
295.    }
296.
297.    }
298.
299.    }
300.
301.    }
302.
303.    }
304.
305.    }
306.
307.    }
308.
309.    }
310.
311.    }
312.
313.    }
314.
315.    }
316.
317.    }
318.
319.    }
320.
321.    }
322.
323.    }
324.
325.    }
326.
327.    }
328.
329.    }
330.
331.    }
332.
333.    }
334.
335.    }
336.
337.    }
338.
339.    }
340.
341.    }
342.
343.    }
344.
345.    }
346.
347.    }
348.
349.    }
350.
351.    }
352.
353.    }
354.
355.    }
356.
357.    }
358.
359.    }
360.
361.    }
362.
363.    }
364.
365.    }
366.
367.    }
368.
369.    }
370.
371.    }
372.
373.    }
374.
375.    }
376.
377.    }
378.
379.    }
380.
381.    }
382.
383.    }
384.
385.    }
386.
387.    }
388.
389.    }
390.
391.    }
392.
393.    }
394.
395.    }
396.
397.    }
398.
399.    }
400.
401.    }
402.
403.    }
404.
405.    }
406.
407.    }
408.
409.    }
410.
411.    }
412.
413.    }
414.
415.    }
416.
417.    }
418.
419.    }
420.
421.    }
422.
423.    }
424.
425.    }
426.
427.    }
428.
429.    }
430.
431.    }
432.
433.    }
434.
435.    }
436.
437.    }
438.
439.    }
440.
441.    }
442.
443.    }
444.
445.    }
446.
447.    }
448.
449.    }
450.
451.    }
452.
453.    }
454.
455.    }
456.
457.    }
458.
459.    }
460.
461.    }
462.
463.    }
464.
465.    }
466.
467.    }
468.
469.    }
470.
471.    }
472.
473.    }
474.
475.    }
476.
477.    }
478.
479.    }
480.
481.    }
482.
483.    }
484.
485.    }
486.
487.    }
488.
489.    }
490.
491.    }
492.
493.    }
494.
495.    }
496.
497.    }
498.
499.    }
500.
501.    }
502.
503.    }
504.
505.    }
506.
507.    }
508.
509.    }
510.
511.    }
512.
513.    }
514.
515.    }
516.
517.    }
518.
519.    }
520.
521.    }
522.
523.    }
524.
525.    }
526.
527.    }
528.
529.    }
530.
531.    }
532.
533.    }
534.
535.    }
536.
537.    }
538.
539.    }
540.
541.    }
542.
543.    }
544.
545.    }
546.
547.    }
548.
549.    }
550.
551.    }
552.
553.    }
554.
555.    }
556.
557.    }
558.
559.    }
560.
561.    }
562.
563.    }
564.
565.    }
566.
567.    }
568.
569.    }
570.
571.    }
572.
573.    }
574.
575.    }
576.
577.    }
578.
579.    }
580.
581.    }
582.
583.    }
584.
585.    }
586.
587.    }
588.
589.    }
590.
591.    }
592.
593.    }
594.
595.    }
596.
597.    }
598.
599.    }
599.
600.    }
601.
602.    }
603.
604.    }
605.
606.    }
607.
608.    }
609.
609.
610.
611.    }
612.
613.    }
614.
615.    }
616.
617.    }
618.
619.    }
620.
621.    }
622.
623.    }
624.
625.    }
626.
627.    }
628.
629.    }
630.
631.    }
632.
633.    }
634.
635.    }
636.
637.    }
638.
639.    }
639.
640.
641.    }
642.
643.    }
644.
645.    }
646.
647.    }
648.
649.    }
649.
650.
651.    }
652.
653.    }
654.
655.    }
656.
657.    }
658.
659.    }
659.
660.
661.    }
662.
663.    }
664.
665.    }
666.
667.    }
668.
669.    }
669.
670.
671.    }
672.
673.    }
674.
675.    }
676.
677.    }
678.
679.    }
679.
680.
681.    }
682.
683.    }
684.
685.    }
686.
687.    }
688.
689.    }
689.
690.
691.    }
692.
693.    }
694.
695.    }
696.
697.    }
698.
699.    }
699.
700.
701.    }
702.
703.    }
704.
705.    }
706.
707.    }
708.
709.    }
709.
710.
711.    }
712.
713.    }
714.
715.    }
716.
717.    }
718.
719.    }
719.
720.
721.    }
722.
723.    }
724.
725.    }
726.
727.    }
728.
729.    }
729.
730.
731.    }
732.
733.    }
734.
735.    }
736.
737.    }
738.
739.    }
739.
740.
741.    }
742.
743.    }
744.
745.    }
746.
747.    }
748.
749.    }
749.
750.
751.    }
752.
753.    }
754.
755.    }
756.
757.    }
758.
759.    }
759.
760.
761.    }
762.
763.    }
764.
765.    }
766.
767.    }
768.
769.    }
769.
770.
771.    }
772.
773.    }
774.
775.    }
776.
777.    }
778.
779.    }
779.
780.
781.    }
782.
783.    }
784.
785.    }
786.
787.    }
788.
789.    }
789.
790.
791.    }
792.
793.    }
794.
795.    }
796.
797.    }
798.
799.    }
799.
800.
801.    }
802.
803.    }
804.
805.    }
806.
807.    }
808.
809.    }
809.
810.
811.    }
812.
813.    }
814.
815.    }
816.
817.    }
818.
819.    }
819.
820.
821.    }
822.
823.    }
824.
825.    }
826.
827.    }
828.
829.    }
829.
830.
831.    }
832.
833.    }
834.
835.    }
836.
837.    }
838.
839.    }
839.
840.
841.    }
842.
843.    }
844.
845.    }
846.
847.    }
848.
849.    }
849.
850.
851.    }
852.
853.    }
854.
855.    }
856.
857.    }
858.
859.    }
859.
860.
861.    }
862.
863.    }
864.
865.    }
866.
867.    }
868.
869.    }
869.
870.
871.    }
872.
873.    }
874.
875.    }
876.
877.    }
877.
878.
879.    }
879.
880.
881.    }
882.
883.    }
884.
885.    }
886.
887.    }
887.
888.
889.    }
889.
890.
891.    }
892.
893.    }
894.
895.    }
896.
897.    }
898.
899.    }
899.
900.
901.    }
902.
903.    }
904.
905.    }
906.
907.    }
908.
909.    }
909.
910.
911.    }
912.
913.    }
914.
915.    }
916.
917.    }
918.
919.    }
919.
920.
921.    }
922.
923.    }
924.
925.    }
926.
927.    }
928.
929.    }
929.
930.
931.    }
932.
933.    }
934.
935.    }
936.
937.    }
938.
939.    }
939.
940.
941.    }
942.
943.    }
944.
945.    }
946.
947.    }
948.
949.    }
949.
950.
951.    }
952.
953.    }
954.
955.    }
956.
957.    }
958.
959.    }
959.
960.
961.    }
962.
963.    }
964.
965.    }
966.
967.    }
968.
969.    }
969.
970.
971.    }
972.
973.    }
974.
975.    }
976.
977.    }
977.
978.
979.    }
979.
980.
981.    }
982.
983.    }
984.
985.    }
986.
987.    }
987.
988.
989.    }
989.
990.
991.    }
992.
993.    }
994.
995.    }
996.
997.    }
997.
998.
999.    }
999.
1000.
1001.    }
1002.
1003.    }
1004.
1005.    }
1006.
1007.    }
1008.
1009.    }
1009.
1010.
1011.    }
1012.
1013.    }
1014.
1015.    }
1016.
1017.    }
1018.
1019.    }
1019.
1020.
1021.    }
1022.
1023.    }
1024.
1025.    }
1026.
1027.    }
1028.
1029.    }
1029.
1030.
1031.    }
1032.
1033.    }
1034.
1035.    }
1036.
1037.    }
1038.
1039.    }
1039.
1040.
1041.    }
1042.
1043.    }
1044.
1045.    }
1046.
1047.    }
1048.
1049.    }
1049.
1050.
1051.    }
1052.
1053.    }
1054.
1055.    }
1056.
1057.    }
1058.
1059.    }
1059.
1060.
1061.    }
1062.
1063.    }
1064.
1065.    }
1066.
1067.    }
1068.
1069.    }
1069.
1070.
1071.    }
1072.
1073.    }
1074.
1075.    }
1076.
1077.    }
1078.
1079.    }
1079.
1080.
1081.    }
1082.
1083.    }
1084.
1085.    }
1086.
1087.    }
1087.
1088.
1089.    }
1089.
1090.
1091.    }
1092.
1093.    }
1094.
1095.    }
1096.
1097.    }
1097.
1098.
1099.    }
1099.
1100.
1101.    }
1102.
1103.    }
1104.
1105.    }
1106.
1107.    }
1108.
1109.    }
1109.
1110.
1111.    }
1112.
1113.    }
1114.
1115.    }
1116.
1117.    }
1118.
1119.    }
1119.
1120.
1121.    }
1122.
1123.    }
1124.
1125.    }
1126.
1127.    }
1128.
1129.    }
1129.
1130.
1131.    }
1132.
1133.    }
1134.
1135.    }
1136.
1137.    }
1138.
1139.    }
1139.
1140.
1141.    }
1142.
1143.    }
1144.
1145.    }
1146.
1147.    }
1148.
1149.    }
1149.
1150.
1151.    }
1152.
1153.    }
1154.
1155.    }
1156.
1157.    }
1158.
1159.    }
1159.
1160.
1161.    }
1162.
1163.    }
1164.
1165.    }
1166.
1167.    }
1168.
1169.    }
1169.
1170.
1171.    }
1172.
1173.    }
1174.
1175.    }
1176.
1177.    }
1178.
1179.    }
1179.
1180.
1181.    }
1182.
1183.    }
1184.
1185.    }
1186.
1187.    }
1187.
1188.
1189.    }
1189.
1190.
1191.    }
1192.
1193.    }
1194.
1195.    }
1196.
1197.    }
1197.
1198.
1199.    }
1199.
1200.
1201.    }
1202.
1203.    }
1204.
1205.    }
1206.
1207.    }
1208.
1209.    }
1209.
1210.
1211.    }
1212.
1213.    }
1214.
1215.    }
1216.
1217.    }
1218.
1219.    }
1219.
1220.
1221.    }
1222.
1223.    }
1224.
1225.    }
1226.
1227.    }
1228.
1229.    }
1229.
1230.
1231.    }
1232.
1233.    }
1234.
1235.    }
1236.
1237.    }
1238.
1239.    }
1239.
1240.
1241.    }
1242.
1243.    }
1244.
1245.    }
1246.
1247.    }
1248.
1249.    }
1249.
1250.
1251.    }
1252.
1253.    }
1254.
1255.    }
1256.
1257.    }
1258.
1259.    }
1259.
1260.
1261.    }
1262.
1263.    }
1264.
1265.    }
1266.
1267.    }
1268.
1269.    }
1269.
1270.
1271.    }
1272.
1273.    }
1274.
1275.    }
1276.
1277.    }
1278.
1279.    }
1279.
1280.
1281.    }
1282.
1283.    }
1284.
1285.    }
1286.
1287.    }
1287.
1288.
1289.    }
1289.
1290.
1291.    }
1292.
1293.    }
1294.
1295.    }
1296.
1297.    }
1297.
1298.
1299.    }
1299.
1300.
1301.    }
1302.
1303.    }
1304.
1305.    }
1306.
1307.    }
1308.
1309.    }
1309.
1310.
1311.    }
1312.
1313.    }
1314.
1315.    }
1316.
1317.    }
1318.
1319.    }
1319.
1320.
1321.    }
1322.
1323.    }
1324.
1325.    }
1326.
1327.    }
1328.
1329.    }
1329.
1330.
1331.    }
1332.
1333.    }
1334.
1335.    }
1336.
1337.    }
1338.
1339.    }
1339.
1340.
1341.    }
1342.
1343.    }
1344.
1345.    }
1346.
1347.    }
1348.
1349.    }
1349.
1350.
1351.    }
1352.
1353.    }
1354.
1355.    }
1356.
1357.    }
1358.
1359.    }
1359.
1360.
1361.    }
1362.
1363.    }
1364.
1365.    }
1366.
1367.    }
1368.
1369.    }
1369.
1370.
1371.    }
1372.
1373.    }
1374.
1375.    }
1376.
1377.    }
1378.
1379.    }
1379.
1380.
1381.    }
1382.
1383.    }
1384.
1385.    }
1386.
1387.    }
1387.
1388.
1389.    }
1389.
1390.
1391.    }
1392.
1393.    }
1394.
1395.    }
1396.
1397.    }
1397.
1398.
1399.    }
1399.
1400.
1401.    }
1402.
1403.    }
1404.
1405.    }
1406.
1407.    }
1408.
1409.    }
1409.
1410.
1411.    }
1412.
1413.    }
1414.
1415.    }
1416.
1417.    }
1418.
1419.    }
1419.
1420.
1421.    }
1422.
1423.    }
1424.
1425.    }
1426.
1427.    }
1428.
1429.    }
1429.
1430.
1431.    }
1432.
1433.    }
1434.
1435.    }
1436.
1437.    }
1438.
1439.    }
1439.
1440.
1441.    }
1442.
1443.    }
1444.
1445.    }
1446.
1447.    }
1448.
1449.    }
1449.
1450.
1451.    }
1452.
1453.    }
1454.
1455.    }
1456.
1457.    }
1458.
1459.    }
1459.
1460.
1461.    }
1462.
1463.    }
1464.
1465.    }
1466.
1467.    }
1468.
1469.    }
1469.
1470.
1471.    }
1472.
1473.    }
1474.
1475.    }
1476.
1477.    }
1478.
1479.    }
1479.
1480.
1481.    }
1482.
1483.    }
1484.
1485.    }
1486.
1487.    }
1487.
1488.
1489.    }
1489.
1490.
1491.    }
1492.
1493.    }
1494.
1495.    }
1496.
1497.    }
1497.
1498.
1499.    }
1499.
1500.
1501.    }
1502.
1503.    }
1504.
1505.    }
1506.
1507.    }
1508.
1509.    }
1509.
1510.
1511.    }
1512.
1513.    }
1514.
1515.    }
1516.
1517.    }
1518.
1519.    }
1519.
1520.
1521.    }
1522.
1523.    }
1524.
1525.    }
1526.
1527.    }
1528.
1529.    }
1529.
1530.
1531.    }
1532.
1533.    }
1534.
1535.    }
1536.
1537.    }
1538.
1539.    }
1539.
1540.
1541.    }
1542.
1543.    }
1544.
1545.    }
1546.
1547.    }
1548.
1549.    }
1549.
1550.
1551.    }
1552.
1553.    }
1554.
1555.    }
1556.
1557.    }
1558.
1559.    }
1559.
1560.
1561.    }
1562.
1563.    }
1564.
1565.    }
1566.
1567.    }
1568.
1569.    }
1569.
1570.
1571.    }
1572.
1573.    }
1574.
1575.    }
1576.
1577.    }
1578.
1579.    }
1579.
1580.
1581.    }
1582.
1583.    }
1584.
1585.    }
1586.
1587.    }
1587.
1588.
1589.    }
1589.
1590.
1591.    }
1592.
1593.    }
1594.
1595.    }
1596.
1597.    }
1597.
1598.
1599.    }
1599.
1600.
1601.    }
1602.
1603.    }
1604.
1605.    }
1606.
1607.    }
1608.
1609.    }
1609.
1610.
1611.    }
1612.
1613.    }
1614.
1615.    }
1616.
1617.    }
1618.
1619.    }
1619.
1620.
1621.    }
1622.
1623.    }
1624.
1625.    }
1626.
1627.    }
1628.
1629.    }
1629.
1630.
1631.    }
1632.
1633.    }
1634.
1635.    }
1636.
1637.    }
1638.
1639.    }
1639.
1640.
1641.    }
1642.
1643.    }
1644.
1645.    }
1646.
1647.    }
1648.
1649.    }
1649.
1650.
1651.    }
1652.
1653.    }
1654.
1655.    }
1656.
1657.    }
1658.
1659.    }
1659.
1660.
1661.    }
1662.
1663.    }
1664.
1665.    }
1666.
1667.    }
1668.
1669.    }
1669.
1670.
1671.    }
1672.
1673.    }
1674.
1675.    }
1676.
1677.    }
1678.
1679.    }
1679.
1680.
1681.    }
1682.
1683.    }
1684.
1685.    }
1686.
1687.    }
1687.
1688.
1689.    }
1689.
1690.
1691.    }
1692.
1693.    }
1694.
1695.    }
1696.
1697.    }
1697.
1698.
1699.    }
1699.
1700.
1701.    }
1702.
1703.    }
1704.
1705.    }
1706.
1707.    }
1708.
1709.    }
1709.
1710.
1711.    }
1712.
1713.    }
1714.
1715.    }
1716.
1717.    }
1718.
1719.    }
1719.
1720.
1721.    }
1722.
1723.    }
1724.
1725.    }
1726.
1727.    }
1728.
1729.    }
1729.
1730.
1731.    }
1732.
1733.    }
1734.
1735.    }
1736.
1737.    }
1738.
1739.    }
1739.
1740.
1741.    }
1742.
1743.    }
1744.
1745.    }
1746.
1747.    }
1748.
1749.    }
1749.
1750.
1751.    }
1752.
1753.    }
1754.
1755.    }
1756.
1757.    }
1758.
1759.    }
1759.
1760.
1761.    }
1762.
1763.    }
1764.
1765.    }
1766.
1767.    }
1768.
1769.    }
1769.
1770.
1771.    }
1772.
1773.    }
1774.
1775.    }
1776.
1777.    }
1778.
1779.    }
1779.
1780.
1781.    }
1782.
1783.    }
1784.
1785.    }
1786.
1787.    }
1787.
1788.
1789.    }
1789.
1790.
1791.    }
1792.
1793.    }
1794.
1795.    }
1796.
1797.    }
1797.
1798.
1799.    }
1799.
1800.
1801.    }
1802.
1803.    }
1804.
1805.    }
1806.
1807.    }
1808.
1809.    }
1809.
1810.
1811.    }
1812.
1813.    }
1814.
1815.    }
1816.
1817.    }
1818.
1819.    }
1819.
1820.
1821.    }
1822.
1823.    }
1824.
1825.    }
1826.
1827.    }
1828.
1829.    }
1829.
1830.
1831.    }
1832.
1833.    }
1834.
1835.    }
1836.
1837.    }
1838.
1839.    }
1839.
1840.
1841.    }
1842.
1843.    }
1844.
1845.    }
1846.
1847.    }
1848.
1849.    }
1849.
1850.
1851.    }
1852.
1853.    }
1854.
1855.    }
1856.
1857.    }
1858.
1859.    }
1859.
1860.
1861.    }
1862.
1863.    }
1864.
1865.    }
1866.
1867.    }
1868.
1869.    }
1869.
1870.
1871.    }
1872.
1873.    }
1874.
1875.    }
1876.
1877.    }
1878.
1879.    }
1879.
1880.
1881.    }
1882.
1883.    }
1884.
1885.    }
1886.
1887.    }
1887.
1888.
1889.    }
1889.
1890.
1891.    }
1892.
1893.    }
1894.
1895.    }
1896.
1897.    }
1897.
1898.
1899.    }
1899.
1900.
1901.    }
1902.
1903.    }
1904.
1905.    }
1906.
1907.    }
1908.
1909.    }
1909.
1910.
1911.    }
1912.
1913.    }
1914.
1915.    }
1916.
1917.    }
1918.
1919.    }
1919.
1920.
1921.    }
1922.
1923.    }
1924.
1925.    }
1926.
1927.    }
1928.
1929.    }
1929.
1930.
1931.    }
1932.
1933.    }
1934.
1935.    }
1936.
1937.    }
1938.
1939.    }
1939.
1940.
1941.    }
1942.
1943.    }
1944.
1945.    }
1946.
1947.    }
1948.
1949.    }
1949.
1950.
1951.    }
1952.
1953.    }
1954.
1955.    }
1956.
1957.    }
1958.
1959.    }
1959.
1960.
1961.    }
1962.
1963.    }
1964.
1965.    }
1966.
1967.    }
1968.
1969.    }
1969.
1970.
1971.    }
1972.
1973.    }
1974.
1975.    }
1976.
1977.    }
1978.
1979.    }
1979.
1980.
1981.    }
1982.
1983.    }
1984.
1985.    }
1986.
1987.    }
1987.
1988.
1989.    }
1989.
1990.
1991.    }
1992.
1993.    }
1994.
1995.    }
1996.
1997.    }
1997.
1998.
1999.    }
1999.
2000.
2001.    }
2002.
2003.    }
2004.
2005.    }
2006.
2007.    }
2008.
2009.    }
2009.
2010.
2011.    }
2012.
2013.    }
2014.
2015.    }
2016.
2017.    }
2018.
2019.    }
2019.
2020.
2021.    }
2022.
2023.    }
2024.
2025.    }
2026.
2027.    }
2028.
2029.    }
2029.
2030.
2031.    }
2032.
2033.    }
2034.
2035.    }
2036.
2037.    }
2038.
2039.    }
2039.
2040.
2041.    }
2042.
2043.    }
2044.
2045.    }
2046.
2047.    }
2048.
2049.    }
2049.
2050.
2051.    }
2052.
2053.    }
2054.
2055.    }
2056.
2057.    }
2058.
2059.    }
2059.
2060.
2061.    }
2062.
2063.    }
2064.
2065.    }
2066.
2067.    }
2068.
2069.    }
2069.
2070.
2071.    }
2072.
2073.    }
2074.
2075.    }
2076.
2077.    }
2078.
2079.    }
2079.
2080.
2081.    }
2082.
2083.    }
2084.
2085.    }
2086.
2087.    }
2087.
2088.
2089.    }
2089.
2090.
2091.    }
2092.
2093.    }
2094.
2095.    }
2096.
2097.    }
2097.
2098.
2099.    }
2099.
2100.
2101.    }
2102.
2103.    }
2104.
2105.    }
2106.
2107.    }
2108.
2109.    }
2109.
2110.
2111.    }
2112.
2113.    }
2114.
2115.    }
2116.
2117.    }
2118.
2119.    }
2119.
2120.
2121.    }
2122.
2123.    }
2124.
2125.    }
2126.
2127.    }
2128.
2129.    }
2129.
2130.
2131.    }
2132.
2133.    }
2134.
2135.    }
2136.
2
```

```

167.     postURL = URL(string: postURLStr)
168.     thumbnailURL = postURL
169.     it has a thumbnail
170.     tURL = URL(string: thumbnailURLStr) {
171.         thumbnailURL = tURL
172.     }
173.
174.     //Convert time to NSDate
175.     postTime = NSDate(timeIntervalSince1970: timeInterval)
176.
177.     let
178.         post = PortfolioPost(uid: uid, id: postID, location: postLocation,
179.             caption: postCaption, isImage: postIsImage, postURL: postURL!,
180.             thumbnailURL: thumbnailURL!, time: postTime, dimensions:
181.             postDimensions)
182.     porfolioPosts.append(post)
183. }
184. }
185. }
186. }
187. }
188. }
189. }
190. }
191.     //return it outside the list
192.     handler(porfolioPosts)
193. }
194. }
195.
196.     //MARK: DATABASE EVENTS
197.
198.     func updateDBEvents(uid: String, eventID: String,
199.         eventData: Dictionary<String, Any>){
200.         //we want to build an array of events to grab and loop
201.         through in table view
202.         REF_EVENTS.child(eventID).updateChildValues(eventData)
203.     }
204.     func updateDBEventsInteractedUsers(uid: String, eventID:
205.         String, eventData: Dictionary<String, Bool>){
206.

```

```

205.          //setValue because we are updating a new array with all
206.          // the interacted users, therefore we want the array to replace each
207.          // time, not update
208.
209.          //delete the event from Database
210.          func deleteDBEvents(uid: String, eventID: String){
211.              REF_EVENTS.child(eventID).removeValue()
212.          }
213.
214.          //Return an array to loop through all the event objects
215.          func getDBEvents(uid: String, handler: @escaping (_ events:
216.          [GigEvent]) -> ()) {
217.              var gigEvents = [GigEvent]()
218.
219.              //Grab the array full of events
220.              REF_EVENTS.observeSingleEvent(of: .value, with: {
221.                  (snapshot) in
222.                      //Grab an array of events in database
223.                      if let snapshot = snapshot.children.allObjects as?
224.                      [DataSnapshot] {
225.                          //Loop through them and grab data for
226.                          // instantiation
227.                          for snap in snapshot {
228.                              if let eventData = snap.value as?
229.                              NSDictionary {
230.                                  if let appliedUsers =
231.                                  eventData["appliedUsers"] as? [String: Bool] {
232.                                      if
233.                                      self.checkAppliedUsers(appliedUsers: appliedUsers) {
234.                                          if let eventID =
235.                                          eventData["eventID"] as? String {
236.                                              if let eventTitle =
237.                                              eventData["title"] as? String {
238.                                                  if let timestamp =
239.                                                  eventData["timestamp"] as? String {
240.                                                      if let
241.                                                      eventDescription = eventData["description"] as? String {
242.                                                          //  

243.                                                          if let
244.                                                          eventLatitude = eventData["latitude"] as? Double {
245.                                                              if let
246.                                                              eventLongitude = eventData["longitude"] as? Double {
247.                                                                  if let
248.                                                                  eventLocationName = eventData["locationName"] as? String {
249.                                                          }
250.          }
251.      }
252.  }
253. }
254. }
255. 
```

```
243.                                         if
244.     let eventPostcode = eventData["postcode"] as? String {
245.         if let eventPayment = eventData["payment"] as? Double {
246.             if let eventOrganiserUid = eventData["uid"] as? String {
247.                 if let eventName = eventData["name"] as? String {
248.                     if let eventEmail = eventData["email"] as? String {
249.                         if let eventPhone = eventData["phone"] as? String {
250.                             if let eventPhotoURLStr = eventData["eventPhotoURL"] as? String {
251.                                 let eventPhotoURL = URL(string: eventPhotoURLStr)
252.
253.
254.                                 let gigEvent = GigEvent(uid: eventOrganiserUid, id: eventID, title:
255.                                         eventTitle, timestamp: timestamp, description: eventDescription,
256.                                         latitude: eventLatitude, longitude: eventLongitude, locationName:
257.                                         eventLocationName, postcode: eventPostcode, payment: eventPayment,
258.                                         name: eventName, email: eventEmail, phone: eventPhone,
259.                                         eventPhotoURL: eventPhotoURL!, appliedUsers: appliedUsers)
260.
261.
262.
263.                                         }
264.                                         }
265.                                         }
266.                                         }
267.                                         }
268.                                         }
269.                                         }
270.                                         }
271.                                         }
272.                                         }
273.                                         }
274.                                         }
275.                                         }
276.                                         //return it outside the list
```

```

277.             handler(gigEvents)
278.         }
279.     }
280.
281.     //So that the same gig doesn't appear twice to a musician
282.     // that has already seen it
283.     func checkAppliedUsers(appliedUsers: [String: Bool]) ->
284.     Bool {
285.
286.         for (uid, _) in appliedUsers {
287.             if uid == Auth.auth().currentUser?.uid {
288.
289.                 //print("denied access to gig")
290.                 return false
291.             }
292.
293.             //print("granted access to gig")
294.             return true
295.         }
296.
297.         //Get a single GigEvent upon request rather than array of
298.         // GigEvents
299.         func getDBSingleEvent(uid: String, eventID: String,
300.         handler: @escaping (_ events: GigEvent, _ success: Bool) -> ()) {
301.
302.             REF_EVENTS.child(eventID).observeSingleEvent(of:
303.             .value, with: { (snapshot) in
304.
305.                 if snapshot.exists() {
306.
307.                     if let eventData = snapshot.value as?
308.                     NSDictionary {
309.
310.                         if let appliedUsers =
311.                         eventData["appliedUsers"] as? [String: Bool] {
312.                             if let eventID = eventData["eventID"]
313.                             as? String {
314.
315.                                 if let eventTitle =
316.                                 eventData["title"] as? String {
317.
318.                                     if let timestamp =
319.                                     eventData["timestamp"] as? String {
320.
321.                                         if let eventDescription =
322.                                         eventData["description"] as? String {
323.
324.                                             if let eventLatitude =
325.                                             eventData["latitude"] as? Double {
326.
327.                                                 if let
328.                                                 eventLongitude = eventData["longitude"] as? Double {
329.
330.                                                     if let
331.                                                     eventLocationName = eventData["locationName"] as? String {
332.
333.                                                         if let
334.                                                         eventPostcode = eventData["postcode"] as? String {

```

```

316.                                     if let
317.             eventPayment = eventData["payment"] as? Double {
318.                 let eventOrganiserUid = eventData["uid"] as? String {
319.                     if let eventName = eventData["name"] as? String {
320.                         if let eventEmail = eventData["email"] as? String {
321.                             if let eventPhone = eventData["phone"] as? String {
322.                                 if let eventPhotoURLStr = eventData["eventPhotoURL"] as? String {
323.                                     let eventPhotoURL = URL(string: eventPhotoURLStr)
324.
325.                                     let gigEvent = GigEvent(uid: eventOrganiserUid, id: eventID, title:
326.                                         eventTitle, timestamp: timestamp, description: eventDescription,
327.                                         latitude: eventLatitude, longitude: eventLongitude, locationName:
328.                                         eventLocationName, postcode: eventPostcode, payment: eventPayment,
329.                                         name: eventName, email: eventEmail, phone: eventPhone,
330.                                         eventPhotoURL: eventPhotoURL!, appliedUsers: appliedUsers)
331.
332.                                     handler(gigEvent, true)
333.
334.                                 }
335.                             }
336.                         }
337.                     }
338.                 }
339.             }
340.         }
341.     }
342. }
343. }
344. } else {
345.     //We couldn't find the GigEvent in the database
346.     //It has been deleted)
347.     //Just filler URL
348.     let nilURL = URL(string: "https://chilly-
designs.com/")
349.     let nilGigEvent = GigEvent(uid: "", id: "",
title: "", timestamp: "", description: "", latitude: 0.00,
longitude: 0.00, locationName: "", postcode: "", payment: 0.00,

```

```

        name: "", email: "", phone: "", eventPhotoURL: nilURL!,
        appliedUsers: ["": false])
349.            handler(nilGigEvent, false)
350.        }
351.    }
352. }
353.
354. //MARK: USERS EVENTS
355.
356. //Simply keep record of which events this user has
   interacted with to list in the 'My Events' section
357. func updateDBUserEvents(uid: String, eventID: String) {
358.     //Get the current array of user's events
359.     //And append to it to update the database with
360.     getDBUserEvents(uid: uid) { (returnedEvents) in
361.         var recordedEvents = returnedEvents
362.         //recordedEvents.insert(eventID, at: 0)
363.         recordedEvents.append(eventID)
364.
365.
366.         self.REF_USERS.child(uid).child("events").setValue(recordedEvents)
367.     }
368.     //We delete by updating with a new array without the value
   to be deleted
369.     func deleteDBUserEvents(uid: String, eventIDs: [String]) {
370.
371.         self.REF_USERS.child(uid).child("events").setValue(eventIDs)
372.     }
373.     func getDBUserEvents(uid: String, handler: @escaping (_
   events: [String]) -> ()) {
374.
375.         var recordedEvents = [String]()
376.
377.         //Grab the array full of events
378.
379.         REF_USERS.child(uid).child("events").observeSingleEvent(of: .value,
   with: { (snapshot) in
380.
381.             if let snapshot = snapshot.children.allObjects as?
   [DataSnapshot] {
382.
383.                 for snap in snapshot {
384.                     if let recordedEvent = snap.value as?
   String {
385.                         recordedEvents.insert(recordedEvent,
   at: 0)
386.                     }
387.                 }
388.             }
389.         })
390.     }
391.

```

```

392.          //MARK: DATABASE USER ACTIVITY
393.
394.          //Using a completion handler now so the feed updates
395.          //correctly
395.          func updateDBActivityFeed(uid: String, notificationID:
396.          String, notificationData: Dictionary<String, Any>, handler:
397.          @escaping (_ completion: Bool) -> ())
398.
399.
400.          REF_USERS.child(uid).child("activity").child(notificationID).setValu
401.          e(notificationData) {
402.              (error:Error?, ref:DatabaseReference) in
403.              if error != nil {
404.                  handler(false)
405.              } else {
406.                  //set activity in Database successfully
407.                  handler(true)
408.              }
409.
410.          func deleteDBActivityFeed(uid: String, notificationID:
411.          String) {
412.              REF_USERS.child(uid).child("activity").child(notificationID).removeV
413.              alue()
414.
415.          func getDBActivityFeed(uid: String, currentActivity:
416.          [ActivityNotification], handler: @escaping (_ events:
417.          [ActivityNotification]) -> ())
418.              let lastActivity = currentActivity.last
419.              var queryRef: DatabaseQuery
420.
421.              if lastActivity == nil || paginationGateOpen {
422.                  //Needed as doesn't refresh properly on sign in and
423.                  //sign out, fetches incorrect starting 10
424.                  paginationGateOpen = false
425.                  //Fetch first 10 if the initial query
426.                  queryRef =
427.                  REF_USERS.child(uid).child("activity").queryOrdered(byChild:
428.                  "timestamp").queryLimited(toLast: 10)
429.              } else {
430.                  let lastTimestamp =
431.                  lastActivity?.getTime().timeIntervalSince1970
432.                  //fetch another 10 starting at the last one in the
433.                  //array, progressing another 10
434.                  queryRef =
435.                  REF_USERS.child(uid).child("activity").queryOrdered(byChild:
436.                  "timestamp").queryEnding(atValue:
437.                  lastTimestamp).queryLimited(toLast: 10)
438.              }
439.

```

```

429.          //This contents of this array is appended when returned
    to display in table
430.          var activityNotifications = [ActivityNotification]()
431.
432.          queryRef.observeSingleEvent(of: .value, with: {
    (snapshot) in
433.
434.          //Grab an array of all posts in the database
435.          if let snapshot = snapshot.children.allObjects as?
    [DataSnapshot] {
436.
437.          //Loop through them and grab data for
    instantiation
438.          for snap in snapshot {
439.
440.          if let activityData = snap.value as?
    NSDictionary {
441.
442.          if let notificationID =
    activityData["notificationID"] as? String {
443.
444.          if notificationID !=

    lastActivity?.getId() {
445.
446.          if let relatedEventID =
    activityData["relatedEventID"] as? String {
447.          if let notificationType =
    activityData["type"] as? String {
448.          if let senderUid =
    activityData["sender"] as? String {
449.          if let recieverUid =
    = activityData["reciever"] as? String {
450.          if let
    senderName = activityData["senderName"] as? String {
451.          if let
    notificationPhotoURLStr = activityData["picURL"] as? String {
452.          if let
    notificationDescription = activityData["description"] as? String {
453.          if
    let timeInterval = activityData["timestamp"] as? TimeInterval {
454.
455.          let notificationPhotoURL = URL(string: notificationPhotoURLStr)
456.
457.          let notificationTime = NSDate(timeIntervalSince1970: timeInterval)
458.
459.          let activityNotification = ActivityNotification(id: notificationID,
    relatedEventId: relatedEventID, type: notificationType, senderUid:
    senderUid, recieverUid: recieverUid, senderName: senderName, picURL:
    notificationPhotoURL!, description: notificationDescription, time:
    notificationTime)
460.

```

```
461.     //Insert at 0 (not append) to be in correct order
462.     activityNotifications.insert(activityNotification, at: 0)
463.
464.
465.
466. }
467. }
468. }
469. }
470. }
471. }
472. }
473. }
474. }
475. }
476. }
477.     handler(activityNotifications)
478. }
479. }
480.
481.     func observeDBActivityFeed(uid: String, handler: @escaping
482.     (_ events: ActivityNotification) -> ()) {
483.         //will run this closure anytime a .childAdded to
484.         Database
485.         activityHandle =
486.             REF_USERS.child(uid).child("activity").observe(.childAdded, with: {
487.                 snapshot) in
488.                     //grab an array of all notifications in the
489.                     database
490.                     if let activityData = snapshot.value as?
491.                         NSDictionary {
492.                             if let notificationID =
493.                                 activityData["notificationID"] as? String {
494.                                     if let relatedEventID =
495.                                         activityData["relatedEventID"] as? String {
496.                                             if let notificationType =
497.                                                 activityData["type"] as? String {
498.                                                     if let senderUid =
499.                                                         activityData["sender"] as? String {
500.                                                             if let recieverUid =
501.                                                               activityData["reciever"] as? String {
502.                                                                   if let senderName =
503.                                                                     activityData["senderName"] as? String {
504.                                                                         if let
505.                             notificationPhotoURLStr = activityData["picURL"] as? String {
506.                                 if let
507.                             notificationDescription = activityData["description"] as? String {
508.                                 if let
509.                             timeInterval = activityData["timestamp"] as? TimeInterval {
```

```

498.                                     let
499.             notificationPhotoURL = URL(string: notificationPhotoURLStr)
500.                                     let
501.             notificationTime = NSDate(timeIntervalSince1970: timeInterval)
502.                                     let
503.             activityNotification = ActivityNotification(id: notificationID,
504.             relatedEventID: relatedEventID, type: notificationType, senderUid:
505.             senderUid, recieverUid: recieverUid, senderName: senderName, picURL:
506.             notificationPhotoURL!, description: notificationDescription, time:
507.             notificationTime)
508.             handler(activityNotification)
509.         }
510.     }
511. }
512. }
513. }
514. }
515. }
516. }
517.
518. //MARK: CLOUD STORAGE
519.
520. //reference Storage location
521. private var _REF_ST = ST_BASE
522.
523. var REF_ST: StorageReference {
524.
525.     return _REF_ST
526. }
527.
528. func updateSTPic(uid: String, directory: String,
529.     imageContent: UIImage, imageID: String, uploadComplete: @escaping (
530.     _ status: Bool, _ error: Error?) -> () {
531.     //specify MIME type
532.     let metadata = StorageMetadata()
533.     metadata.contentType = "image/jpeg"
534.     //Converting the imageData to JPEG to be stored
535.     if let imageData =
536.         imageContent.jpegData(compressionQuality: 0.1) {
537.             REF_ST.child(uid).child(directory).child(imageID).putData(imageData,
538.             metadata: metadata, completion: { (metadata, error) in
539.                 if error != nil {
540.                     uploadComplete(false, error)

```

```

540.                     return
541.                 }
542.             uploadComplete(true, nil)
543.         }
544.     }
545. }
546.
547. func updateSTVid(uid: String, directory: String,
548.     vidContent: URL, imageID: String, uploadComplete: @escaping (_
549.     status: Bool, _ error: Error) -> () {
550.         //Uploading the content with unique string ID
551.
552.         //Upload Bug
553.         //This time we use .putFile to upload the URL and not
554.         // imageData
555.         // REF_ST.child(uid).child(directory).child(imageID).putFile(from:
556.         // vidContent, metadata: nil, completion: { (metadata, error) in
557.             // if error != nil {
558.             //     uploadComplete(false, error)
559.             //     return
560.             // }
561.             // uploadComplete(true, nil)
562.         // }
563.         //Uploads the video, but not as a video. Therefore
564.         // won't play when requested
565.         //Therefore need to change the MIME type
566.         let metadata = StorageMetadata()
567.         metadata.contentType = "video/quicktime"
568.         //Convert to data and upload the data to Storage
569.         if let videoData = NSData(contentsOf: vidContent) as
570.             Data? {
571.             REF_ST.child(uid).child(directory).child(imageID).putData(videoData,
572.                 metadata: metadata, completion: { (metadata, error) in
573.                     if error != nil {
574.                         uploadComplete(false, error)
575.                         return
576.                     }
577.                     uploadComplete(true, nil)
578.                 })
579.         }
580.     }
581.     //Reference to the folder
582.     let ref =
583.         REF_ST.child(uid).child(directory).child(imageID)

```

```

584.          //Get's the url of the image or video file
585.          ref.downloadURL(completion: { (url, error) in
586.              if error != nil {
587.
588.
589.                  print("Couldn't get imageURL
590.                      \(error!.localizedDescription)")
591.              } else {
592.
593.                  handler(url!)
594.
595.              }
596.          })
597.
598.
599.      func deleteSTFile(uid: String, directory: String, fileID:
600. String){
600.
601.      REF_ST.child(uid).child(directory).child(fileID).delete(completion:
602.      { error in
603.          if error != nil {
604.              print("Couldn't delete image from ST
605.                  \(error!.localizedDescription)")
606.          }
607.      }
608.
609.      //MARK: CLOUD MESSAGING
610.      // (and database)
611.
612.      func updateDBUserFCMToken(uid: String, token: String) {
613.
614.          REF_USERS.child(uid).child("profile").child("FCMToken").setValue(token)
615.
616.          //Send a notification from a device to another device
617.          func sendPushNotification(to token: String, title: String,
618.          body: String) {
619.              //URL to send notification
620.              let urlString = "https://fcm.googleapis.com/fcm/send"
621.              let url = NSURL(string: urlString)!
622.              //Parameters of the notification
623.              let paramString: [String : Any] = ["to" : token,
624.                                              "notification" :
625.                                              ["title" : title, "body" : body],
626.                                              "data" : ["user" :
627.                                              "test_id"]]
628.
629.              //Create request object using url
630.              let request = NSMutableURLRequest(url: url as URL)
631.              request.httpMethod = "POST" //Http method is POST

```

```

628.          //Convert paramString to JSON and set it as request
  body
629.          request.httpBody = try?
  JSONSerialization.data(withJSONObject:paramString, options:
  [.prettyPrinted])
630.          request.setValue("application/json",
  forHTTPHeaderField: "Content-Type")
631.          //Server key
632.
  request.setValue("key=AAAAb8BHzs:APA91bEBkZ3IfE6dU4xclXlP4qGVqyFhML
  QEuCTA8NtFjKC7WGN_L8LeuaH_t7142RWGLbuYqjSHozuiz7HtmAADhGEz67yM0jN416
  Z-EdbIE9FXJ-0uyI37mcQ6bcMzbohxSzF4nCUJ", forHTTPHeaderField:
  "Authorization")
633.          //Data task, send the request to the server in a
  completion handler (executed when the request's response is returned
  to the app
634.          let task = URLSession.shared.dataTask(with: request as
  URLRequest) { (data, response, error) in
635.              do {
636.                  //If there is JSON data
637.                  if let jsonData = data {
638.                      //Try and convert to a dictionary
639.                      if let jsonDataDict = try
  JSONSerialization.jsonObject(with: jsonData, options:
  JSONSerialization.ReadingOptions.allowFragments) as? [String:
  AnyObject] {
640.                          NSLog("Received
  data:\n\(jsonDataDict)")
641.                      }
642.                  }
643.                  //If failed catch it
644.              } catch let err as NSError {
645.                  //Print the error
646.                  print(err.debugDescription)
647.              }
648.          }
649.          //Resume the task
650.          task.resume()
651.      }
652.
653.          //FCM Server key:
  AAAAb8BHzs:APA91bEBkZ3IfE6dU4xclXlP4qGVqyFhMLQEuCTA8NtFjKC7WGN_L8Le
  uaH_t7142RWGLbuYqjSHozuiz7HtmAADhGEz67yM0jN416Z-EdbIE9FXJ-
  0uyI37mcQ6bcMzbohxSzF4nCUJ
654.      }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
DB_BASE	DatabaseReference	Global	To encapsulate location references in a constant and remove human error in development.
ST_BASE	StorageReference		

postsHandle activityHandle eventHandle	DatabaseHandle	Global	Handles the observed changes to the database.
instance	DataService	Public Static	To initialise the DataService class a single time to share it globally.
_REF_BASE _REF_USERS _REF_EVENTS	DatabaseReference	Private	More specific Database location references to remove human error in development.
profileData	NSDictionary	Local	To convert the Database JSON profile data to a dictionary.
currentUser currentUserEmail currentUserGigs currentUserBio currentUserPicURLStr currentUserPhone currentUserFacebook currentUserTwitter currentUserInstagram currentUserWebsiteURLStr currentUserAppleMusicURLS tr currentUserSpotifyURLStr currentUserPicURL currentUserFCMToken	String . . . URL String	Local	The fetched user data from profileData to instantiate a User object with and return.
currentUser	User	Local	The User object to be returned.
portfolioPosts	Array<PortfolioPost>	Local	Array of all post objects fetched from Database.
snapshot	Array<DataSnapshot>	Local	Array of all the JSON portfolio post objects fetched.
postData	NSDictionary	Local	To convert the Database JSON post data to a dictionary.
postID postURLStr thumbnailURLStr timeInterval postIsImage postCaption	String String String TimeInterval Boolean String	Local	The fetched post data from postData to instantiate a post object and append it to portfolioPosts.

postLocation	String		
postDimensions	Dictionary<String, Any>		
postURL	URL		
thumbnailURL	URL		
postTime	NSDate		
post	PortfolioPost	Local	The PortfolioPost object to be appended to array.
gigEvents	Array<GigEvent>	Local	Array of all event objects fetched from Database.
eventData	NSDictionary	Local	To convert the Database JSON event data to a dictionary.
appliedUsers	Dictionary<String, Boolean>	Local	The fetched event data from eventData to instantiate an event object and append it to gigEvents.
eventID	String		
eventTitle	String		
timestamp	String		
eventDescription	String		
eventLatitude	Double		
eventLongitude	Double		
eventLocationName	String		
eventPostcode	String		
eventPayment	Double		
eventOrganiserUid	String		
eventName	.		
eventEmail	.		
eventPhone	.		
eventPhotoURLStr	URL		
eventPhotoURL			
recordedEvents	Array<String>	Local	Array of event identifiers which are associated to the user fetched from Database.
lastActivity	ActivityNotification	Local	The last notification object to be fetched, so that it knows where to fetch more from.
queryRef	DatabaseQuery	Local	To check the same query to fetch more notifications (pagination).
lastTimestamp	TimeInterval	Local	To know which notification object was the last to be

			fetched as part of the query.
activityNotifications	Array<ActivityNotification>	Local	Array of all notification objects fetched from Database.
activityData	NSDictionary	Local	To convert the Database JSON notification data to a dictionary.
notificationID relatedEventID notificationType senderUid receiverUid senderName notificationPhotoURLStr notificationDescription timeInterval notificationPhotoURL notificationTime	String String String String String String String String TimeInterval URL NSDate	Local	The fetched notification data from activityData to instantiate a notification object and append it to activityNotifications.
activityNotification	ActivityNotification	Local	The ActivityNotification object to be appended to array.
metadata	StorageMetadata	Local	To specify the file type in Storage.
imageData videoData	Data	Local	The data to be uploaded to Storage.
ref	StorageReference	Local	Location in Storage to get download URL from.

## Models

### GigEvent.swift

```

1. import Foundation
2. import CoreLocation
3.
4. class GigEvent: Comparable {
5.
6.     //attributes
7.     private var uid: String
8.     private var id: String
9.     private var title: String
10.    private var timestamp: String
11.    private var description: String
12.    private var latitude: Double

```

```
13.         private var longitude: Double
14.         private var distance: Double
15.         private var locationName: String
16.         private var postcode: String
17.         private var payment: Double
18.         private var name: String
19.         private var email: String
20.         private var phone: String
21.         private var eventPhotoURL: URL
22.         private var appliedUsers: [String: Bool]
23.
24.         //instantiate
25.         init(uid: String, id: String, title: String, timestamp:
26.               String, description: String, latitude: Double, longitude: Double,
27.               locationName: String, postcode: String, payment: Double, name:
28.               String, email: String, phone: String, eventPhotoURL: URL,
29.               appliedUsers: [String: Bool]) {
30.             self.uid = uid
31.             self.id = id
32.             self.title = title
33.             self.timestamp = timestamp
34.             self.description = description
35.             self.locationName = locationName
36.             self.latitude = latitude
37.             self.longitude = longitude
38.             self.postcode = postcode
39.             self.payment = payment
40.             self.name = name
41.             self.email = email
42.             self.phone = phone
43.             self.eventPhotoURL = eventPhotoURL
44.             self.distance = 0.00
45.             self.appliedUsers = appliedUsers
46.         }
47.
48.         //getters and setters
49.         func getuid() -> String {
50.             return uid
51.         }
52.         func getid() -> String {
53.             return id
54.         }
55.         func getTitle() -> String {
56.             return title
57.         }
58.         func getTimestamp() -> String {
59.             return timestamp
60.         }
61.         func getDescription() -> String {
62.             return description
63.         }
64.         func getLocationName() -> String {
65.             return locationName
66.         }
67.         func getLatitude() -> Double {
```

```
64.            return latitude
65.        }
66.        func getLongitude() -> Double {
67.            return longitude
68.        }
69.        func getDistance() -> Double {
70.            return distance
71.        }
72.        func setDistance(distanceFromUser: Double) {
73.            self.distance = distanceFromUser
74.        }
75.        func getPostcode() -> String {
76.            return postcode
77.        }
78.        func getPayment() -> Double {
79.            return payment
80.        }
81.        func getName() -> String {
82.            return name
83.        }
84.        func getEmail() -> String {
85.            return email
86.        }
87.        func getPhone() -> String {
88.            return phone
89.        }
90.        func getEventPhotoURL() -> URL {
91.            return eventPhotoURL
92.        }
93.
94.        func getAppliedUsers() -> [String: Bool] {
95.            return appliedUsers
96.        }
97.
98.        //MARK: timestamp string manipulation for date
99.        //Get the actual date of data type Date
100.       func getDate() -> Date {
101.           let dateFormatter = DateFormatter()
102.           dateFormatter.dateFormat = "yyyy-MM-dd HH:mm:ss Z"
103.           let date = dateFormatter.date(from: timestamp)!
104.           return date
105.       }
106.       func getDayDate() -> String {
107.           return timestamp.substring(start: 8, end: 10)
108.       }
109.       func getMonthYearDate() -> String {
110.           return timestamp.substring(start: 0, end: 7)
111.       }
112.       func getTime() -> String {
113.           return timestamp.substring(start: 11, end: 16)
114.       }
115.       func getLongMonthYearDate() -> String {
116.           let month = getMonthYearDate().substring(start: 5, end:
7)
```

```

117.         let year = getMonthYearDate().substring(start: 0, end:
118.                                         4)
119.         switch month {
120.             case "01":
121.                 return "January " + year
122.             case "02":
123.                 return "February " + year
124.             case "03":
125.                 return "March " + year
126.             case "04":
127.                 return "April " + year
128.             case "05":
129.                 return "May " + year
130.             case "06":
131.                 return "June " + year
132.             case "07":
133.                 return "July " + year
134.             case "08":
135.                 return "August " + year
136.             case "09":
137.                 return "September " + year
138.             case "10":
139.                 return "October " + year
140.             case "11":
141.                 return "November " + year
142.             case "12":
143.                 return "December " + year
144.             default:
145.                 return "error getting month"
146.         }
147.     }
148.
149.     //MARK: get exact point location using the latitude and the
150.     longitude
151.     func getGigEventLocation() -> CLLocation {
152.         let gigEventLocation = CLLocation(latitude: latitude,
153.                                         longitude: longitude)
154.         return gigEventLocation
155.     }
156.     //quicksort based on location - nearest is first
157.     static func < (lhs: GigEvent, rhs: GigEvent) -> Bool {
158.         return lhs.getDistance() < rhs.getDistance()
159.     }
160.
161.     static func == (lhs: GigEvent, rhs: GigEvent) -> Bool {
162.         return lhs.getDistance() == rhs.getDistance()
163.     }
164. }
```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
------------	-----------	-------	---------

uid	String	Private	The user identifier that created the event.
id	String	Private	The unique identifier of the event.
title	String	Private	The title the event's creator has given.
timestamp	String	Private	To do various string manipulations on to display the time and date of the event
description	String	Private	The description the event's creator has given.
latitude	Double	Private	The latitude coordinate of the event's location for distance.
longitude	Double	Private	The longitude coordinate of the event's location for distance.
distance	Double	Private	The distance between musician and the event location, calculated from coordinates for quicksort.
locationName	String	Private	Name of the event's location for display.
postcode	String	Private	Postcode of the event's location for display.
payment	Double	Private	Amount musician will be paid for display.
name	String	Private	The name of the organiser user for display.
email	String	Private	The email of the organiser user for display.
phone	String	Private	The phone number of the organiser user for display.
eventPhotoURL	URL	Private	URL to download image of event for display.
appliedUsers	Dictionary<String, Bool>	Private	To keep track of which musicians have already seen the event in <u><a href="#">FindGigVC</a></u> .
month	String	Local	A substring to determine how to display the date of the event with a switch statement.
year	String	Local	A substring to concatenate to the end of a month for displaying a date of the event
gigEventLocation	CLLocation	Local	A location instantiated from latitude and longitude coordinates to ultimately calculate a distance between user and this point.

## PortfolioPost.swift

1. `import Foundation`
2. `import UIKit`

```

3.
4. class PortfolioPost: Comparable {
5.
6.     var uid: String
7.     private var id: String
8.     var location: String
9.     var caption: String
10.    var isImage: Bool
11.    var postURL: URL
12.    var thumbnailURL: URL
13.    private var time: NSDate
14.    var dimensions: Dictionary<String, Any>
15.
16.    init(uid: String, id: String, location: String, caption: String, isImage: Bool, postURL: URL, thumbnailURL: URL, time: NSDate, dimensions: Dictionary<String, Any>) {
17.        self.uid = uid
18.        self.id = id
19.        self.location = location
20.        self.caption = caption
21.        self.isImage = isImage
22.        self.postURL = postURL
23.        self.thumbnailURL = thumbnailURL
24.        self.time = time
25.        self.dimensions = dimensions
26.    }
27.
28.    func getId() -> String {
29.        return id
30.    }
31.    func getTime() -> NSDate {
32.        return time
33.    }
34.
35.    //quicksort based on timestamp
36.    static func <(lhs: PortfolioPost, rhs: PortfolioPost) -> Bool {
37.        //inverse so that quick sort of feed shows most recent first
38.        return rhs.getTime().compare(lhs.getTime() as Date) == .orderedAscending
39.    }
40.
41.    static func == (lhs: PortfolioPost, rhs: PortfolioPost) -> Bool {
42.        return lhs.getTime() == rhs.getTime() ||
43.        lhs.getTime().compare(rhs.getTime() as Date) == .orderedSame
44.    }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
uid	String	Public	The user identifier that created the post to refer back to.

id	String	Private	The unique identifier of the post so it is possible to delete.
location	String	Public	The location name of where post was taken for display in portfolio.
caption	String	Public	Post caption added by user for display in portfolio.
isImage	Boolean	Public	To determine whether post is an image or video to set up portfolio cell accordingly.
postURL	URL	Public	The download URL of post image or video so that it can be displayed in the portfolio.
thumbnailURL	URL	Public	The download URL of a video thumbnail to display before user tries to play a video.
time	NSDate	Private	The attribute for which the posts will be quick sorted by.
dimensions	Dictionary<String, Any>	var	Dimensions of the post image so that <u>PostContainerView</u> can be scaled accordingly.

## User.swift

```

1. import Foundation
2.
3. var uniqueID: String?
4.
5. class User {
6.
7.     var uid: String
8.     var name: String
9.     var email: String
10.    var phone: String
11.    var bio: String
12.    var gigs: Bool
13.    var picURL: URL
14.    private var facebook: String
15.    private var twitter: String
16.    private var instagram: String
17.    private var website: String
18.    private var appleMusic: String
19.    private var spotify: String
20.    private var fcmToken: String
21.
22.    init(uid: String, name: String, email: String, phone:
23.         String, bio: String, gigs: Bool, picURL: URL, facebook: String,
24.              twitter: String, instagram: String, website: String, appleMusic:
25.              String, spotify: String, fcmToken: String) {
26.        self.uid = uid
27.        self.name = name
28.        self.email = email

```

```

26.          self.phone = phone
27.          self.bio = bio
28.          self.gigs = gigs
29.          self.picURL = picURL
30.          self.facebook = facebook
31.          self.twitter = twitter
32.          self.instagram = instagram
33.          self.website = website
34.          self.appleMusic = appleMusic
35.          self.spotify = spotify
36.          self.fcmToken = fcmToken
37.      }
38.
39.    func getFacebook() -> String {
40.        return facebook
41.    }
42.    func getTwitter() -> String {
43.        return twitter
44.    }
45.    func getInstagram() -> String {
46.        return instagram
47.    }
48.    func getWebsite() -> String {
49.        return website
50.    }
51.    func getAppleMusic() -> String {
52.        return appleMusic
53.    }
54.    func getSpotify() -> String {
55.        return spotify
56.    }
57.    func getFCMToken() -> String {
58.        return fcmToken
59.    }
60.
61.    }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
uid	String	Public	The user identifier used to get data from the Database.
name	String	Public	The name of the user for display.
email	String	Public	The email of the user for display.
phone	String	Public	The phone of the user for display.
bio	String	Public	The biography of the user for display.
gigs	Boolean	Public	To determine whether the user is a musician or an event organiser.
picURL	URL	Public	The download URL of profile picture so that it can be displayed in the portfolio and notifications.
facebook	String	Private	The pageID input by the user so that there is a hyperlink to Facebook.

twitter	String	Private	The twitter username input by the user so that there is a hyperlink to Twitter.
instagram	String	Private	The Instagram username input by the user so that there is a hyperlink to Instagram.
website	String	Private	The URL of a website to hyperlink to it.
appleMusic	String	Private	The URL input by the user so that there is a hyperlink to AppleMusic profile.
Spotify	String	Private	The URL input by the user so that there is a hyperlink to Spotify profile.
fcmToken	String	Private	To keep track of which device gets push notifications corresponding to the user logged in on it.

## ActivityNotification.swift

```

1. import Foundation
2.
3. class ActivityNotification: Comparable {
4.
5.     private var id: String
6.     private var relatedEventId: String
7.     private var type: String
8.     private var senderUid: String
9.     private var receiverUid: String
10.    private var senderName: String
11.    private var picURL: URL
12.    private var description: String
13.    private var time: NSDate
14.
15.    init(id: String, relatedEventId: String, type: String,
16.         senderUid: String, receiverUid: String, senderName: String, picURL:
17.              URL, description: String, time: NSDate) {
18.        self.id = id
19.        self.relatedEventId = relatedEventId
20.        self.type = type
21.        self.senderUid = senderUid
22.        self.receiverUid = receiverUid
23.        self.senderName = senderName
24.        self.picURL = picURL
25.        self.description = description
26.        self.time = time
27.    }
28.
29.    func getId() -> String {
30.        return id
31.    }
32.    func getRelatedEventId() -> String {
33.        return relatedEventId
34.    }
35.    func getType() -> String {
36.        return type
37.    }
38.    func getSenderUid() -> String {
39.        return senderUid

```

```

38.          }
39.          func getRecieverUid() -> String {
40.              return recieverUid
41.          }
42.          func getSenderName() -> String {
43.              return senderName
44.          }
45.          func getNotificationPicURL() -> URL {
46.              return picURL
47.          }
48.          func getNotificationDescription() -> String {
49.              return description
50.          }
51.          func getTime() -> NSDate {
52.              return time
53.          }
54.
55.          //quicksort to most recent notification first (based from
56.          //timestamp)
56.          static func < (lhs: ActivityNotification, rhs:
57.              ActivityNotification) -> Bool {
58.                  //inverse so that quick sort of feed shows most recent
59.                  //first
60.                  return rhs.getTime().compare(lhs.getTime() as Date) ==
61.                      .orderedAscending
62.                  }
63.
64.          static func == (lhs: ActivityNotification, rhs:
65.              ActivityNotification) -> Bool {
66.                  return lhs.getTime() == rhs.getTime() ||
67.                      lhs.getTime().compare(rhs.getTime() as Date) == .orderedSame
68.          }
69.      }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
id	String	Private	The ActivityNotification identifier to delete the object from Database if wanted.
relatedEventId	String	Private	The ID of the event that is associated with the notification so that if event data is needed, there can be a Database query.
type	String	Private	To help distinguish which users can see and interact with certain notifications.
senderUid	String	Private	The user identifier which sent the notification so there can be a Database query to get the profile data.
receiverUid	String	Private	The identifier of user that gets the notification. To keep track of who the notification is for and where it is stored in Database
senderName	String	Private	So that the sender's name is displayed on the notification, they know who it's from.
picURL	URL	Private	To download and display the profile image of user that sent the notification.

description	String	Private	The notification content to be displayed to the receiver.
time	NSDate	Private	So that the activity notifications can be sorted chronologically for the receiver.

## Views

### AccountHeaderCell.swift

```

1. import UIKit
2. import FirebaseAuth
3.
4. class AccountHeaderCell: UITableViewCell {
5.
6.     @IBOutlet weak var userTypeLabel: UILabel!
7.     @IBOutlet weak var userBioTextView: MyTextView!
8.     @IBOutlet weak var profilePicView: UIImageView!
9.     @IBOutlet weak var userEmailLabel: UILabel!
10.    @IBOutlet weak var userPhoneLabel: UILabel!
11.    @IBOutlet weak var facebookLinkButton: UIButton!
12.    @IBOutlet weak var twitterLinkButton: UIButton!
13.    @IBOutlet weak var instagramLinkButton: UIButton!
14.    @IBOutlet weak var websiteLinkButton: UIButton!
15.    @IBOutlet weak var appleMusicLinkButton: UIButton!
16.    @IBOutlet weak var spotifyLinkButton: UIButton!
17.    @IBOutlet weak var socialLinkStackView: UIStackView!
18.
19.    override func awakeFromNib() {
20.        //Give the profile image a small border and make it
21.        //round
22.        profilePicView.layer.borderWidth = 0.1
23.        profilePicView.layer.masksToBounds = false
24.        profilePicView.layer.cornerRadius =
25.            profilePicView.frame.height/2
26.        profilePicView.clipsToBounds = true
27.        //Set cell to be an opaque white colour
28.        self.backgroundColor = UIColor(white: 1, alpha: 0.75)
29.    }
30.}

```

### AccountPostCell.swift

```

1. import UIKit
2.
3. class AccountPostCell: UITableViewCell {
4.
5.     @IBOutlet weak var postLocationLabel: UILabel!
6.     @IBOutlet weak var postCaptionTextView: MyTextView!
7.     @IBOutlet weak var postContainerView: PostContainerView!
8.     @IBOutlet weak var postMoreButton: UIButton!
9.
10.    override func awakeFromNib() {
11.        //so user cannot edit the caption text view
12.        postCaptionTextView.setEditable = false
13.        //slightly more opaque than the AccountHeaderCell

```

```
14.           self.backgroundColor = UIColor(white: 1, alpha: 0.5)
15.       }
16.
17.   }
```

## AutoComplete.swift

```
1. import UIKit
2. import GoogleMaps
3. import GooglePlaces
4.
5. class AutoComplete: UIViewController,
    GMSAutocompleteViewControllerDelegate {
6.
7.     //default location string
8.     var locationResult = "Location"
9.
10.    //present the VC
11.    func presentAutocompleteVC(){
12.
13.        let autocompleteController =
    GMSAutocompleteViewController()
14.        autocompleteController.delegate = self
15.
16.        //MUST PAY FOR API USAGE
17.        // specify the place data types to return
18.        let fields: GMSPlaceField = GMSPlaceField(rawValue:
    UInt(GMSPlaceField.name.rawValue) |
    UInt(GMSPlaceField.placeID.rawValue))!
19.        autocompleteController.placeFields = fields
20.
21.        // specify a filter
22.        let filter = GMSAutocompleteFilter()
23.        filter.type = .address
24.        autocompleteController.autocompleteFilter = filter
25.
26.        // display the autocomplete view controller
27.        present(autocompleteController, animated: true,
28.        completion: nil)
29.    }
30.
31.    // handle the user's selection
32.    func viewController(_ viewController:
    GMSAutocompleteViewController, didAutocompleteWith place: GMSPlace)
    {
33.        //location string is what user has chosen and returned
34.        locationResult = place.name!
35.        //dismiss the view
36.        dismiss(animated: true, completion: nil)
37.    }
38.
39.    func viewController(_ viewController:
    GMSAutocompleteViewController, didFailAutocompleteWithError error:
    Error) {
40.        //handle the error if there is one
41.        print("Error: ", error.localizedDescription)
```

```

42.         }
43.
44.         //user canceled the operation
45.         func wasCancelled(viewController:
46.             GMSAutocompleteViewController) {
47.             dismiss(animated: true, completion: nil)
48.
49.             //turn the network activity indicator on and off again
50.             func didRequestAutocompletePredictions(viewController:
51.                 GMSAutocompleteViewController) {
52.                 UIApplication.shared.isNetworkActivityIndicatorVisible
53.                 = true
54.             }
55.
56.             func didUpdateAutocompletePredictions(viewController:
57.                 GMSAutocompleteViewController) {
58.                 UIApplication.shared.isNetworkActivityIndicatorVisible
59.                 = false
60.             }
61.         }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
locationResult	String	Public	Default value is “Location” and holds the value of string returned from Google Places API if used.
autocompleteController	<b>GMSAutocompleteViewController</b>	Local	The view controller instance which will be presented.
fields	<b>GMSPlaceField</b>	Local	To specify the place data types to return.
filter	<b>GMSAutocompleteFilter</b>	Local	To specify the filter of searching to addresses.

### LoginField.swift

```

1. import UIKit
2.
3. class LoginField: UITextField {
4.
5.     //to allow object to communicate with its owner in a decoupled
6.     //way
6.     let textFieldDelegate = TextFieldDelegate()
7.     var characterLimit: Int?

```

```

8.    required init?(coder aDecoder: NSCoder) {
9.        super.init(coder: aDecoder)
10.
11.        self.smartInsertDeleteType =
12.            UITextSmartInsertDeleteType.no
13.            //set the delegate so that we can control character
14.            limit
15.            self.delegate = textFieldDelegate
16.            characterLimit = textFieldDelegate.characterLimit
17.
18.        //customisation goes here
19.        override func awakeFromNib() {
20.            //make a rounded opaque text field
21.            clipsToBounds = true
22.            layer.cornerRadius = self.frame.size.height/2
23.            layer.borderWidth = 2.0
24.            layer.borderColor = colorLiteral(red: 0.9652684959,
25.            green: 0.9729685758, blue: 1, alpha: 1)
26.            backgroundColor = UIColor.white.withAlphaComponent(0.1)
27.            font = .systemFont(ofSize: 15)
28.        }
29.        //custom function to limit the length of input
30.        func updateCharacterLimit(limit: Int) {
31.            textFieldDelegate.characterLimit = limit
32.        }
33.    }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
textFieldDelegate	TextFieldDelegate	Public	Instance made to access delegate methods of UITextField and edit them (set a character a limit).
characterLimit	Integer	Public	To specify the maximum input length for the text field.

### MyTextView.swift

```

1. import UIKit
2.
3. class MyTextView: UITextView {
4.
5.    //delegation is a design pattern that enables a class or
structure to hand off (or delegate) some of its responsibilities to
an instance of another type
6.    let textViewDelegate = TextViewDelegate()
7.    var placeholder: String?
8.    required init?(coder aDecoder: NSCoder) {
9.        super.init(coder: aDecoder)
10.           //set the delegate so that I can control placeholder
11.           self.delegate = textViewDelegate
12.           placeholder = textViewDelegate.placeholder

```

```

13.         }
14.
15.         //customisation goes here
16.         override func awakeFromNib() {
17.             clipsToBounds = true
18.             layer.cornerRadius = 15.0
19.             backgroundColor = colorLiteral(red: 0.9652684959,
20.                                         green: 0.9729685758, blue: 1, alpha: 1)
21.         }
22.
23.         //set the placeholder
24.         func updatePlaceholder(placeholder: String) {
25.             textViewDelegate.placeholder = placeholder
26.         }
27.         //to allow access to the editing delegate methods
28.         class TextViewDelegate: NSObject, UITextViewDelegate {
29.
30.             var placeholder = ""
31.
32.             //clear the placeholder when editing
33.             func textViewDidBeginEditing(_ textView: UITextView) {
34.                 if textView.textColor == UIColor.lightGray {
35.                     textView.text = nil
36.                     //text is now black
37.                     textView.textColor = UIColor.black
38.                 }
39.             }
40.             //if empty, add the placeholder
41.             func textViewDidEndEditing(_ textView: UITextView) {
42.                 if textView.text.isEmpty {
43.                     textView.text = placeholder
44.                     //text is back to gray
45.                     textView.textColor = UIColor.lightGray
46.                 }
47.             }
48.         }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
textViewDelegate	TextViewDelegate	Public	Instance made to access delegate methods of UITextView and edit them (custom placeholder).
placeholder	String	Public	The custom string that occupies the text view when there is no input from the user.

## MyTextField.swift

```

1. import UIKit
2.
3. class MyTextField: UITextField {
4.
5.     let textFieldDelegate = TextFieldDelegate()

```

```
6.      var characterLimit: Int?
7.      required init?(coder aDecoder: NSCoder) {
8.          super.init(coder: aDecoder)
9.          //set delegate so that I can set a character limit
10.         self.smartInsertDeleteType =
11.             UITextSmartInsertDeleteType.no
12.         self.delegate = textFieldDelegate
13.         characterLimit = textFieldDelegate.characterLimit
14.
15.     }
16.
17.     //customisation goes here
18.     override func awakeFromNib() {
19.         clipsToBounds = true
20.         layer.cornerRadius = self.frame.size.height/2
21.         layer.borderWidth = 2.0
22.         layer.borderColor = colorLiteral(red: 0.9652684959,
23.             green: 0.9729685758, blue: 1, alpha: 1)
24.             //backgroundColor = #colorLiteral(red: 0.9652684959,
25.             green: 0.9729685758, blue: 1, alpha: 1)
26.             backgroundColor = UIColor.white.withAlphaComponent(0.7)
27.             font = .systemFont(ofSize: 15)
28.     }
29.     //set a character limit
30.     func updateCharacterLimit(limit: Int) {
31.         textFieldDelegate.characterLimit = limit
32.     }
33.     //to allow access to the editing delegate methods
34.     class TextFieldDelegate: NSObject, UITextFieldDelegate {
35.
36.         //default character limit (if not specified)
37.         var characterLimit = 100
38.
39.         //put a limit on the number of characters allowed to be
40.         //entered in the textField
41.         //called: when characters are being changed
42.         func textField(_ textField: UITextField,
43.             shouldChangeCharactersIn range: NSRange, replacementString string:
44.             String) -> Bool {
45.             guard let textFieldText = textField.text,
46.                 let rangeOfTextToReplace = Range(range, in:
47.                     textFieldText) else {
48.                 return false
49.             }
50.             let substringToReplace =
51.                 textFieldText[rangeOfTextToReplace]
```

## PostContainerView.swift

```
1. import UIKit
2. import AVKit
3. import AVFoundation
4.
5. class PostContainerView: UIView {
6.
7.     private var dimensionHeight: CGFloat = 0.00
8.     private var dimensionWidth: CGFloat = 0.00
9.
10.    private var imageView: UIImageView!
11.    private var avPlayer: AVPlayer!
12.    private var avPlayerLayer: AVPlayerLayer!
13.
14.    //play button for videos (setup in a closure)
15.    let playButton: UIImageView = {
16.        let pb = UIImageView(image: UIImage(named:
17.            "playButton"))
18.        pb.translatesAutoresizingMaskIntoConstraints = false
19.        return pb
20.    }()
21.
22.    override func awakeFromNib() {
23.        //default dimensions
24.        dimensionHeight = self.frame.size.height
25.        dimensionWidth = self.frame.size.width
26.
27.        imageView = UIImageView()
28.        //give a shadow, border and rounded edges
29.        layer.borderColor = colorLiteral(red: 0.8039215803,
30.            green: 0.8039215803, blue: 0.8039215803, alpha: 1)
31.        layer.shadowColor = colorLiteral(red: 0.6000000238,
32.            green: 0.6000000238, blue: 0.6000000238, alpha: 1)
33.        layer.shadowRadius = 10
34.        layer.shadowOpacity = 0.5
35.        layer.cornerRadius = 30
36.        //make the UIView clear
37.        self.backgroundColor = UIColor.clear
38.
39.    func clearView(fit: Bool){
40.        //clear the container view if there is anything
41.        //if there is a video, close it
42.        if avPlayer != nil {
43.            closePlayer()
44.        }
45.        //if there is an image remove it
46.        if self.subviews.count > 0 {
47.            imageView.removeFromSuperview()
48.        }
49.        //if choosing to fit, return to default dimensions
50.        if fit {
```

```
50.             self.frame.size.height = dimensionHeight
51.             self.frame.size.width = dimensionWidth
52.         }
53.     }
54.
55.     func addPhoto(imageContent: UIImage, fit: Bool){
56.
57.         //clear the container view if there is anything
58.         clearView(fit: fit)
59.
60.         //fit = true means UIView decides dimensions of post
61.         //fit = false means post decides dimensions of UIView
62.         if fit {
63.             imageView.frame.size.height = dimensionHeight
64.             imageView.frame.size.width = dimensionWidth
65.             imageView.translatesAutoresizingMaskIntoConstraints = true
66.             imageView.contentMode = .scaleAspectFit
67.             imageView.image = imageContent
68.
69.             //resize for the feed
70.         } else {
71.             imageView = UIImageView(image: imageContent)
72.             layer.cornerRadius = 0
73.
74.         }
75.         //hide until sizing is right
76.         imageView.isHidden = true
77.
78.         //get ratio of how much to shrink the image by by using
79.         //the width of the UIView
80.         //(width is set using constraints)
81.         //height is not set, because that is what is changing
82.         let ratio = self.frame.size.width /
83.             imageView.frame.size.width
84.         //change the height of the UIView by setting it to the
85.         //new height of the imageView
86.         self.frame.size.height = imageView.frame.size.height *
87.             ratio
88.
89.         //fill the UIView with the imageView
90.         imageView.frame = self.bounds
91.
92.         //add image to view and show it
93.         self.addSubview(imageView)
94.         imageView.isHidden = false
95.
96.     }
97.     func addVideo(url: URL, fit: Bool){
98.
99.         //clear
100.        clearView(fit: fit)
101.        //add a video player
102.        avPlayer = AVPlayer(playerItem: AVPlayerItem(url: url))
```

```

100.         avPlayerLayer = AVPlayerLayer(player: avPlayer)
101.         //fill the UIView with the video player
102.         avPlayerLayer.frame = self.bounds
103.         //add it
104.         self.layer.insertSublayer(avPlayerLayer, at: 0)
105.         //show the play button
106.         playButton.isHidden = true
107.     }
108.
109.     func addPlayButton(){
110.         //add the playbutton in the middle of the UIView
111.         playButton.isHidden = false
112.         self.addSubview(playButton)
113.         NSLayoutConstraint.activate([
114.             playButton.centerXAnchor.constraint(equalTo:
115.             self.centerXAnchor),
116.             playButton.centerYAnchor.constraint(equalTo:
117.             self.centerYAnchor),
118.             playButton.widthAnchor.constraint(equalToConstant:
119.             60),
120.             playButton.heightAnchor.constraint(equalToConstant:
121.             60)
122.         ])
123.     }
124.
125.     func playPlayer(){
126.         avPlayer.play()
127.     }
128.     //remove the video player
129.     func closePlayer(){
130.         avPlayer.pause()
131.         avPlayer = nil
132.         avPlayerLayer.removeFromSuperlayer()
133.     }
134.
135.     //load images from NSCache
136.     var imageUrlString: NSString?
137.     func loadImageCache(url: URL, isImage: Bool) {
138.         //url as a string
139.         let urlString = url.absoluteString as NSString
140.         imageUrlString = urlString
141.         //check for a chached image
142.         if let cachedImage = imageCache.object(forKey:
143.             urlString) {
144.             //if there is one, add it to UIView
145.             self.addPhoto(imageContent: cachedImage, fit:
146.             false)
147.             //if post is a video, add a playbutton too
148.             if !isImage {
149.                 self.addPlayButton()
150.             }
151.         }
152.     }

```

```

149.          //not cached, need to download
150.      } else {
151.
152.          //to avoid flashing of images
153.          self.clearView(fit: false)
154.          //download task
155.          let task = URLSession.shared.dataTask(with: url) {
156.              data, response, error in
157.                  if let error = error {
158.                      print(error.localizedDescription)
159.                  } else {
160.
161.                      DispatchQueue.main.async {
162.                          if let downloadedImage = UIImage(data:
163.                              data!) {
164.                              if self.imageUrlString == urlString
165.                                  self.addPhoto(imageContent:
166.                                      downloadedImage, fit: false)
167.                              }
168.                          if !isImage {
169.                              self.addPlayButton()
170.                          }
171.                          //add it to cache once downloaded
172.
173.                          imageCache.setObject(downloadedImage, forKey: urlString)
174.                      }
175.                  }
176.              }
177.              task.resume()
178.          }
179.      }
180.  }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
dimensionHeight	CGFloat	Private	The post dimensions to scale it up or down with a ratio when displaying it.
dimensionWidth	CGFloat	Private	
imageView	UIImageView	Private	To display post images in.
avPlayer	AVPlayer	Private	To display post videos in.
avPlayerLayer	AVPlayerLayer	Private	
playButton	UIImageView	Public	The play button to be displayed over a video thumbnail.
ratio	CGFloat	Local	Ratio of how much to shrink the image by.

## GigEventView.swift

1. `import UIKit`

```
2.
3. class GigEventView: UIView {
4.
5.     @IBOutlet weak var dayDateLabel: UILabel!
6.     @IBOutlet weak var monthYearDateLabel: UILabel!
7.     @IBOutlet weak var timeLabel: UILabel!
8.     @IBOutlet weak var titleLabel: UILabel!
9.     @IBOutlet weak var paymentLabel: UILabel!
10.    @IBOutlet weak var eventPhotoImageView: UIImageView!
11.
12.    override func awakeFromNib() {
13.        //rounded corners like shape of card
14.        layer.shadowColor = colorLiteral(red: 0.2549019754,
15.        green: 0.2745098174, blue: 0.3019607961, alpha: 1)
16.        layer.shadowRadius = 10.0
17.        layer.shadowOpacity = 0.5
18.        layer.cornerRadius = 10.0
19.        //create a gradient of stops orange to purple
20.        let gradient: CAGradientLayer = CAGradientLayer()
21.        gradient.colors = [UIColor(red: 255.0/255.0, green:
22.        159.0/255.0, blue: 2.0/255.0, alpha: 0.5).cgColor, UIColor(red:
23.        104.0/255.0, green: 35.0/255.0, blue: 128.0/255.0, alpha:
24.        0.6).cgColor]
25.        gradient.locations = [0.0 , 1.0]
26.        gradient.frame = CGRect(x: 0.0, y: 0.0, width:
27.        self.frame.size.width, height: self.frame.size.height)
28.        gradient.cornerRadius = 10.0
29.        self.layer.insertSublayer(gradient, at: 0)
30.
31.    }
32.}
```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
gradient	CAGradientLayer	Local	A programmatic gradient which is displayed on the gig cards.

## ActivityCollectionViewCell.swift

```
1. import UIKit
2.
3. class ActivityCVCell: UICollectionViewCell {
4.     //notification feed table view
5.     @IBOutlet weak var feedTableView: UITableView!
6.
7.     //link the table view datasource and delegate to the view
8.     //controller rather than the collection view cell with a tag
9.     //so that I can stick by the model-view-controller pattern,
10.    //rather than working from the ActivityCVCell class
11.    func setTableViewDataSourceDelegate(dataSourceDelegate:
12.        UITableViewDataSource & UITableViewDelegate, forRow row: Int) {
13.        feedTableView.delegate = dataSourceDelegate
14.        feedTableView.dataSource = dataSourceDelegate
15.    }
16. }
```

```

13.           //tag allows us to distinguish between the table views
   within the cv cells for datasource
14.           feedTableView.tag = row
15.           feedTableView.reloadData()
16.           feedTableView.addSubview(refreshControl)
17.
18.           //press tab to scroll to top
19.           NotificationCenter.default.addObserver(self, selector:
   #selector(scrollToTop), name: NSNotification.Name(rawValue:
   "scrollToTop"), object: nil)
20.       }
21.
22.           //pull to refresh
23.           lazy var refreshControl: UIRefreshControl = {
24.               let rc = UIRefreshControl()
25.               rc.addTarget(self, action: #selector(pullToRefresh),
   for: .valueChanged)
26.               rc.tintColor = UIColor.purple
27.               return rc
28.           }()
29.           //refresh the activity feed when pulled
30.           @objc func pullToRefresh() {
31.               NotificationCenter.default.post(name:
   NSNotification.Name(rawValue: "refreshAllActivity"), object: nil)
32.               refreshControl.endRefreshing()
33.           }
34.
35.           @objc func scrollToTop() {
36.               feedTableView.setContentOffset(.zero, animated: true)
37.           }
38.
39.           var tableViewOffset: CGFloat {
40.               get {
41.                   return feedTableView.contentOffset.x
42.               }
43.
44.               set {
45.                   feedTableView.contentOffset.x = newValue
46.               }
47.           }
48.       }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
refreshControl	UIRefreshControl	Lazy	Lazy – delaying objects creation until needed because it is an expensive process. To refresh the activity feed when pulled downwards.

### ActivityMenuBar.swift

1. import UIKit
- 2.

```
3. class MenuBar: UIView, UICollectionViewDelegate,  
    UICollectionViewDataSource, UICollectionViewDelegateFlowLayout {  
4.  
5.     var activityFeedVC: ActivityFeedVC?  
6.  
7.     let cellID = "menuBarCell"  
8.     let titleNames = ["Notifications", "My Events"]  
9.  
10.    override init(frame: CGRect) {  
11.        super.init(frame: frame)  
12.        //collection view is the titles  
13.        setupCollectionView()  
14.        //horizontal bar is the purple slider  
15.        setupHorizontalBar()  
16.        //to keep track of what cell is selected  
17.        let selectedIndexPath = NSIndexPath(item: 0, section:  
    0)  
18.        collectionView.selectItem(at: selectedIndexPath as  
    IndexPath, animated: false, scrollPosition: .left)  
19.    }  
20.  
21.    //when user taps that one, then take to another table view  
22.    //this moves the large collection view  
23.    func collectionView(_ collectionView: UICollectionView,  
        didSelectItemAt indexPath: IndexPath) {  
24.        activityFeedVC?.scrollToMenuIndex(menuIndex:  
        indexPath.item)  
25.    }  
26.  
27.  
28.    //MARK: SETUP  
29.  
30.    //UI with swift closures  
31.    lazy var collectionView: UICollectionView = {  
32.        let layout = UICollectionViewFlowLayout()  
33.        let cv = UICollectionView(frame: .zero,  
        collectionViewLayout: layout)  
34.        cv.backgroundColor = .white  
35.        cv.dataSource = self  
36.        cv.delegate = self  
37.        cv.translatesAutoresizingMaskIntoConstraints = false  
38.        return cv  
39.    }()  
40.  
41.    let separator: UIView = {  
42.        let view = UIView()  
43.        view.backgroundColor = .lightGray  
44.        view.translatesAutoresizingMaskIntoConstraints = false  
45.        return view  
46.    }()  
47.  
48.    //this changes to create animation  
49.    var barLeftAnchorConstraint: NSLayoutConstraint?  
50.  
51.    func setupHorizontalBar() {
```

```
52.          let barView = UIView()
53.          barView.backgroundColor = .purple
54.          barView.translatesAutoresizingMaskIntoConstraints =
55.          false
56.          addSubview(barView)
57.
58.          barView.bottomAnchor.constraint(equalTo:
59.          self.bottomAnchor).isActive = true
60.          barView.heightAnchor.constraint(equalToConstant:
61.          3).isActive = true
62.          //multiplier 0.5 so takes half of device view
63.          barView.widthAnchor.constraint(equalTo:
64.          self.widthAnchor, multiplier: 0.5).isActive = true
65.          barLeftAnchorConstraint =
66.          barView.leftAnchor.constraint(equalTo: self.leftAnchor)
67.          barLeftAnchorConstraint?.isActive = true
68.      }
69.
70.      private func setupCollectionView() {
71.          addSubview(collectionView)
72.          collectionView.topAnchor.constraint(equalTo:
73.          self.topAnchor).isActive = true
74.          collectionView.bottomAnchor.constraint(equalTo:
75.          self.bottomAnchor).isActive = true
76.          collectionView.leftAnchor.constraint(equalTo:
77.          self.leftAnchor).isActive = true
78.          collectionView.rightAnchor.constraint(equalTo:
79.          self.rightAnchor).isActive = true
80.          addSubview(separator)
81.          separator.bottomAnchor.constraint(equalTo:
82.          self.bottomAnchor).isActive = true
83.          separator.leftAnchor.constraint(equalTo:
84.          self.leftAnchor).isActive = true
85.          separator.rightAnchor.constraint(equalTo:
86.          self.rightAnchor).isActive = true
87.          separator.heightAnchor.constraint(equalToConstant:
88.          0.5).isActive = true
89.          collectionView.register(MenuCell.self,
90.          forCellWithReuseIdentifier: cellID)
91.      }
92.
93.      //MARK: MENU BAR COLLECTION VIEW
94.
95.      //two titles needed
96.      func collectionView(_ collectionView: UICollectionView,
97.      numberOfItemsInSection section: Int) -> Int {
98.          return 2
99.      }
100.     //set appearance of each menubar cell
101.     func collectionView(_ collectionView: UICollectionView,
102.     cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
```

```

91.         let cell =
92.             collectionView.dequeueReusableCell(withReuseIdentifier: cellID, for:
93.             indexPath) as! MenuCell
94.
95.         cell.title.text = titleNames[indexPath.item]
96.     }
97.     //set size of cell
98.     func collectionView(_ collectionView: UICollectionView,
99.         layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt
100.            indexPath: IndexPath) -> CGSize {
101.         return CGSize(width: frame.width / 2, height:
102.             frame.height)
103.     }
104.
105.     func collectionView(_ collectionView: UICollectionView,
106.         layout collectionViewLayout: UICollectionViewLayout,
107.         minimumInteritemSpacingForSectionAt section: Int) -> CGFloat {
108.         return 0
109.     }
110.
111.
112. //MARK: MENU CELL
113.
114. class MenuCell: UICollectionViewCell {
115.
116.     //add the title
117.     override init(frame: CGRect) {
118.         super.init(frame: frame)
119.
120.         addTitle()
121.     }
122.     //set appearance of titles
123.     let title: UILabel = {
124.         let label = UILabel()
125.         label.translatesAutoresizingMaskIntoConstraints = false
126.         label.textAlignment = .left
127.         label.font = UIFont(name: "Avenir-Medium", size: 20.0)
128.         label.textColor = UIColor.lightGray
129.         return label
130.     }()
131.
132.     //becomes purple when active...
133.     override var isHighlighted: Bool {
134.         didSet {
135.             title.textColor = isHighlighted ? UIColor.purple :
136.                 UIColor.lightGray
137.         }
138.     }

```

```

138.
139.        //...and selected
140.        override var isSelected: Bool {
141.            didSet {
142.                title.textColor = isSelected ? UIColor.purple :
143.                    UIColor.lightGray
144.            }
145.        //add the title to the cell
146.        func addTitle() {
147.            addSubview(title)
148.            title.centerXAnchor.constraint(equalTo:
149.                self.centerXAnchor).isActive = true
150.            title.centerYAnchor.constraint(equalTo:
151.                self.centerYAnchor).isActive = true
152.            title.heightAnchor.constraint(equalToConstant:
153.                28).isActive = true
154.        }
155.    }
156.}

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
activityFeedVC	ActivityFeedVC	Public	Instance of the activity feed so that menu bar can communicate to the view controller.
cellID	String	Public	The reuse identifier of the menu bar cell.
titleNames	Array<String>	Public	The title's that will appear on the menu bar cells (names of two sections).
selectedIndexPath	NSIndexPath	Local	To keep track of what menu bar cell is selected.
collectionView	UICollectionView	Lazy	The menu bar collection view object.
separator	UIView	Public	A view that separates the two menu bar cells.
barLeftAnchorConstraint	NSLayoutConstraint	Public	This constraint will change to animate the sliding of the barView.
barView	UIView	Local	The purple bar which slides to indicate what section of ActivityFeedVC user is on.
title	UILabel	Local	The label which displays the text from titleNames in the cells.

## ActivityFeedCell.swift

1. **import** UIKit

```

2.
3. class ActivityFeedCell: UITableViewCell {
4.
5.     @IBOutlet weak var notificationImage: UIImageView!
6.     @IBOutlet weak var eventNameButton: UIButton!
7.     @IBOutlet weak var notificationDescriptionLabel: UILabel!
8.     @IBOutlet weak var deleteNotificationButton: UIButton!
9.
10.    override func awakeFromNib() {
11.        //make the notification image a circle with a small
12.        //border
13.        notificationImage.layer.borderWidth = 0.1
14.        notificationImage.layer.masksToBounds = false
15.        notificationImage.layer.cornerRadius =
16.            notificationImage.frame.height/2
17.        notificationImage.clipsToBounds = true
18.        //cell itself is slightly opaque
19.        self.backgroundColor = UIColor(white: 1, alpha: 0.75)
20.    }

```

## SpinnerViewController.swift

```

1. import UIKit
2.
3. class SpinnerViewController: UIViewController {
4.
5.     var spinner = UIActivityIndicatorView(style: .whiteLarge)
6.
7.     override func loadView() {
8.         //create a which is black with 0.5 opacity
9.         view = UIView()
10.        view.backgroundColor = UIColor(white: 0, alpha: 0.5)
11.        //start and add spinner
12.        spinner.translatesAutoresizingMaskIntoConstraints =
13.            false
14.        spinner.startAnimating()
15.        view.addSubview(spinner)
16.        //center the spinner
17.        spinner.centerXAnchor.constraint(equalTo:
18.            view.centerXAnchor).isActive = true
19.        spinner.centerYAnchor.constraint(equalTo:
19.            view.centerYAnchor).isActive = true
20.    }
21. }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
spinner	UIActivityIndicatorView	Public	Spinner will be presented when uploading/fetching data to/from Database to indicate loading and stop user interaction.

# Controllers

## TabBarController.swift

```

1. import UIKit
2. import FirebaseAuth
3. import FirebaseStorage
4. import FirebaseDatabase
5.
6. class TabBarController: UITabBarController {
7.     override func viewDidLoad() {
8.         super.viewDidLoad()
9.         //notification to refresh the tabs
10.         NotificationCenter.default.addObserver(self, selector:
11.             #selector(refreshTabs), name: NSNotification.Name(rawValue:
12.             "refreshTabs"), object: nil)
13.     }
14.     //make sure that navigation bar (TabBarController is part
15.     //of navigation stack)
16.     //is hidden, but tabBar is shown
17.     override func viewWillAppear(_ animated: Bool) {
18.         self.navigationController?.navigationBar.isHidden =
19.             true
20.         self.tabBarController?.tabBar.isHidden = false
21.     }
22.
23.     var userGigs: Bool?
24.     //first launch only (use of gate so that tabs don't refresh
25.     //everytime view is shown)
26.     override func viewDidAppear(_ animated: Bool) {
27.         if tabGateOpen {
28.             refreshTabs()
29.         }
30.     }
31.     //logic to decide what tabs are shown
32.     @objc func refreshTabs(){
33.         print("Tabs have been refreshed")
34.         //set the original state of the tabs (all four)
35.         if tabGateOpen {
36.             tabs = self.viewControllers!
37.         }
38.         //IF USER IS RESUMING
39.         if let userGigsDefaults = DEFAULTS.object(forKey:
40.             "gigs") as? Bool {
41.             //remove the tabs that shouldn't be seen by
42.             //musician/organiser
43.             if tabGateOpen {
44.                 if userGigsDefaults == true {
45.                     self.viewControllers?.remove(at: 0)
46.                 } else {
47.                     self.viewControllers?.remove(at: 1)
48.                 }
49.             }
50.         }
51.     }
52. }

```

```

45.
46.                                //safety so they don't refresh again
47.                                tabGateOpen = false
48.
49.                                //makes the initial tab the portfolio tab
50.                                self.selectedIndex = 1;
51.                            }
52.
53.                        } else {
54.
55.                            if tabGateOpen {
56.                                //remove the tabs that shouldn't be seen by
57.                                //musician/organiser
58.                                if let uid = Auth.auth().currentUser?.uid {
59.
60.                                    //IF USER IS LOGGING IN
61.                                    if userGigs == nil {
62.
63.                                        DataService.instance.getDBUserProfile(uid: uid) { (returnedUser) in
64.                                            //remember the tab state when
65.                                            //logged in
66.                                            self.setDefaults(userGigsCondition:
67.                                            returnedUser.gigs)
68.                                        }
69.
70.                                    //IF THE USER IS SIGNING UP FOR THE FIRST
71.                                    //TIME OR EDITING THEIR PROFILE
72.                                    else {
73.                                        //remember the tab state when logged in
74.                                        self.setDefaults(userGigsCondition:
75.                                        userGigs!)
76.
77.                                    }
78.
79.                                //refresh the portfolio when tabs are ready
80.                                NotificationCenter.default.post(name:
81.                                    NSNotification.Name(rawValue: "refreshPortfolio"), object: nil)
82.
83.                                //UserDefaults to set remembered value
84.                                func setDefaults(userGigsCondition: Bool) {
85.                                    if userGigsCondition == true {
86.                                        //remove the create tab and view
87.                                        self.viewControllers?.remove(at: 0)
88.
89.                                        //write value to device
90.                                        DEFAULTS.set(true, forKey: "gigs")
91.

```

```

92.             } else {
93.                 //remove the find tab and view
94.                 self.viewControllers?.remove(at: 1)
95.
96.                 //write value to device
97.                 DEFAULTS.set(false, forKey: "gigs")
98.             }
99.
100.            tabGateOpen = false
101.
102.            //Makes the initial tab the profile tab
103.            self.selectedIndex = 1;
104.        }
105.    }
106.
107.    //Tab Bar Original State
108.    var tabs = [UIViewController]()

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
userGigs	Boolean	Public	Boolean value to help determine what type of user is logged in and what tabs to display.
userGigsDefaults	Boolean	Local	To check if user type is saved in defaults to save querying the database (quicker).
tabs	Array<UIViewController>	Global	Array that holds all of the tabs. (The tab bar's original state to refer back to at reset)

## LoginSignupVC.swift

```

1. import UIKit
2. import FirebaseAuth
3. import FirebaseDatabase
4. import FirebaseMessaging
5. import FirebaseInstanceID
6.
7. class LoginSignupVC: UIViewController {
8.
9.     @IBOutlet weak var fieldsStack: UIStackView!
10.    @IBOutlet weak var emailField: MyTextField!
11.    @IBOutlet weak var passwordField: MyTextField!
12.    @IBOutlet weak var confirmPasswordField: MyTextField!
13.    @IBOutlet weak var topLSButton: UIButton!
14.    @IBOutlet weak var bottomLSButton: UIButton!
15.    @IBOutlet weak var switchPrompt: UILabel!
16.
17.    var userData: Dictionary<String, Any>?
18.

```

```
19.         var email: String?
20.         var password: String?
21.         //keep track of what state view is in
22.         var logInMode = false
23.
24.         override func viewDidLoad() {
25.             super.viewDidLoad()
26.             //for iOS 13
27.             self.modalPresentationStyle = .overFullScreen
28.             setupView()
29.             hideKeyboard()
30.             emailField.updateCharacterLimit(limit: 62)
31.             passwordField.updateCharacterLimit(limit: 30)
32.             confirmPasswordField.updateCharacterLimit(limit: 30)
33.
34.             logInMode = false
35.             //initially will be Log In
36.             logInMode = switchMode(logInMode: logInMode)
37.         }
38.         override func viewDidAppear(_ animated: Bool) {
39.             self.navigationController?.navigationBar.isHidden =
40.                 true
41.         }
42.         func switchMode(logInMode: Bool) -> Bool{
43.
44.             //hide everything for Sign up
45.             if logInMode == false {
46.
47.                 emailField.text = ""
48.                 passwordField.text = ""
49.                 //change from "create password"
50.                 passwordField.placeholder = "password"
51.                 confirmPasswordField.text = ""
52.
53.                 confirmPasswordField.isHidden = true
54.
55.                 //change the images of the top and bottom login and
56.                 //signup buttons
57.                 for: .normal)
58.                     topLSButton.setImage(UIImage(named: "loginButton"),
59.                     bottomLSButton.setImage(UIImage(named:
60.                     "signupButton"), for: .normal)
61.
62.                     //change the prompt so the user knows what the
63.                     //bottom button does
64.                     switchPrompt.text = "New to GoGig? Create an
65.                     account"
66.
67.                     //return the new logInMode
68.                     //logInMode = true is logging in state
69.                     return true
70.
71.             //show everything for Sign up
```

```
68.        } else {
69.
70.            //reset input when changing mode
71.            emailField.text = ""
72.            passwordField.text = ""
73.            passwordField.placeholder = "create password"
74.            confirmPasswordField.text = ""
75.
76.            //show the confirm password field
77.            confirmPasswordField.isHidden = false
78.
79.            //swap the images for the buttons
80.            topLSButton.setImage(UIImage(named:
81.                "signupButton"), for: .normal)
82.            bottomLSButton.setImage(UIImage(named:
83.                "loginButton"), for: .normal)
84.
85.            switchPrompt.text = "Already have an account?"
86.
87.        }
88.    }
89.
90.    @IBAction func topLSButton(_ sender: Any) {
91.
92.        //email
93.        if let userEmail = emailField.text {
94.            if userEmail != "" && userEmail.contains("@") &&
95.                userEmail.count > 3 {
96.                    //password
97.                    if let userPassword = passwordField.text {
98.                        if userPassword != "" {
99.                            //USER IS SIGNING UP
100.                           if logInMode == false {
101.                               //password validation checks
102.                               if userPassword.count > 6 {
103.                                   if let confirmPassword =
104.                                       confirmPasswordField.text {
105.                                           if confirmPassword != ""
106.                                           && confirmPassword == userPassword {
107.                                               //data dictionary for
108.                                               Database
109.                                               userData = ["email": user
110.                                                 "email", "bio": "", "gigs": true, "name": "", "picURL": "", "website": "", "phone": "", "instagram": "", "twitter": "", "facebook": "", "appleMusic": "", "spotify": ""]
111.                                               self.performSegue(withIdentifier: T0_CREATE_PROFILE, sender: nil)
112.                                           } else {
113.                                               displayError(title: "Passwords", message: "The password confirmation does not match")
114.                                           }
115.                                           }
116.                                           }
117.                                           }
118.                                           }
119.                                           }
120.                                           }
121.                                           }
122.                                           }
123.                                           }
124.                                           }
125.                                           }
126.                                           }
127.                                           }
128.                                           }
129.                                           }
130.                                           }
131.                                           }
132.                                           }
133.                                           }
134.                                           }
135.                                           }
136.                                           }
137.                                           }
138.                                           }
139.                                           }
140.                                           }
141.                                           }
142.                                           }
143.                                           }
144.                                           }
145.                                           }
146.                                           }
147.                                           }
148.                                           }
149.                                           }
150.                                           }
151.                                           }
152.                                           }
153.                                           }
154.                                           }
155.                                           }
156.                                           }
157.                                           }
158.                                           }
159.                                           }
160.                                           }
161.                                           }
162.                                           }
163.                                           }
164.                                           }
165.                                           }
166.                                           }
167.                                           }
168.                                           }
169.                                           }
170.                                           }
171.                                           }
172.                                           }
173.                                           }
174.                                           }
175.                                           }
176.                                           }
177.                                           }
178.                                           }
179.                                           }
180.                                           }
181.                                           }
182.                                           }
183.                                           }
184.                                           }
185.                                           }
186.                                           }
187.                                           }
188.                                           }
189.                                           }
190.                                           }
191.                                           }
192.                                           }
193.                                           }
194.                                           }
195.                                           }
196.                                           }
197.                                           }
198.                                           }
199.                                           }
200.                                           }
201.                                           }
202.                                           }
203.                                           }
204.                                           }
205.                                           }
206.                                           }
207.                                           }
208.                                           }
209.                                           }
210.                                           }
211.                                           }
212.                                           }
213.                                           }
214.                                           }
215.                                           }
216.                                           }
217.                                           }
218.                                           }
219.                                           }
220.                                           }
221.                                           }
222.                                           }
223.                                           }
224.                                           }
225.                                           }
226.                                           }
227.                                           }
228.                                           }
229.                                           }
230.                                           }
231.                                           }
232.                                           }
233.                                           }
234.                                           }
235.                                           }
236.                                           }
237.                                           }
238.                                           }
239.                                           }
240.                                           }
241.                                           }
242.                                           }
243.                                           }
244.                                           }
245.                                           }
246.                                           }
247.                                           }
248.                                           }
249.                                           }
250.                                           }
251.                                           }
252.                                           }
253.                                           }
254.                                           }
255.                                           }
256.                                           }
257.                                           }
258.                                           }
259.                                           }
260.                                           }
261.                                           }
262.                                           }
263.                                           }
264.                                           }
265.                                           }
266.                                           }
267.                                           }
268.                                           }
269.                                           }
270.                                           }
271.                                           }
272.                                           }
273.                                           }
274.                                           }
275.                                           }
276.                                           }
277.                                           }
278.                                           }
279.                                           }
280.                                           }
281.                                           }
282.                                           }
283.                                           }
284.                                           }
285.                                           }
286.                                           }
287.                                           }
288.                                           }
289.                                           }
290.                                           }
291.                                           }
292.                                           }
293.                                           }
294.                                           }
295.                                           }
296.                                           }
297.                                           }
298.                                           }
299.                                           }
300.                                           }
301.                                           }
302.                                           }
303.                                           }
304.                                           }
305.                                           }
306.                                           }
307.                                           }
308.                                           }
309.                                           }
310.                                           }
311.                                           }
312.                                           }
313.                                           }
314.                                           }
315.                                           }
316.                                           }
317.                                           }
318.                                           }
319.                                           }
320.                                           }
321.                                           }
322.                                           }
323.                                           }
324.                                           }
325.                                           }
326.                                           }
327.                                           }
328.                                           }
329.                                           }
330.                                           }
331.                                           }
332.                                           }
333.                                           }
334.                                           }
335.                                           }
336.                                           }
337.                                           }
338.                                           }
339.                                           }
340.                                           }
341.                                           }
342.                                           }
343.                                           }
344.                                           }
345.                                           }
346.                                           }
347.                                           }
348.                                           }
349.                                           }
350.                                           }
351.                                           }
352.                                           }
353.                                           }
354.                                           }
355.                                           }
356.                                           }
357.                                           }
358.                                           }
359.                                           }
360.                                           }
361.                                           }
362.                                           }
363.                                           }
364.                                           }
365.                                           }
366.                                           }
367.                                           }
368.                                           }
369.                                           }
370.                                           }
371.                                           }
372.                                           }
373.                                           }
374.                                           }
375.                                           }
376.                                           }
377.                                           }
378.                                           }
379.                                           }
380.                                           }
381.                                           }
382.                                           }
383.                                           }
384.                                           }
385.                                           }
386.                                           }
387.                                           }
388.                                           }
389.                                           }
390.                                           }
391.                                           }
392.                                           }
393.                                           }
394.                                           }
395.                                           }
396.                                           }
397.                                           }
398.                                           }
399.                                           }
400.                                           }
401.                                           }
402.                                           }
403.                                           }
404.                                           }
405.                                           }
406.                                           }
407.                                           }
408.                                           }
409.                                           }
410.                                           }
411.                                           }
412.                                           }
413.                                           }
414.                                           }
415.                                           }
416.                                           }
417.                                           }
418.                                           }
419.                                           }
420.                                           }
421.                                           }
422.                                           }
423.                                           }
424.                                           }
425.                                           }
426.                                           }
427.                                           }
428.                                           }
429.                                           }
430.                                           }
431.                                           }
432.                                           }
433.                                           }
434.                                           }
435.                                           }
436.                                           }
437.                                           }
438.                                           }
439.                                           }
440.                                           }
441.                                           }
442.                                           }
443.                                           }
444.                                           }
445.                                           }
446.                                           }
447.                                           }
448.                                           }
449.                                           }
450.                                           }
451.                                           }
452.                                           }
453.                                           }
454.                                           }
455.                                           }
456.                                           }
457.                                           }
458.                                           }
459.                                           }
460.                                           }
461.                                           }
462.                                           }
463.                                           }
464.                                           }
465.                                           }
466.                                           }
467.                                           }
468.                                           }
469.                                           }
470.                                           }
471.                                           }
472.                                           }
473.                                           }
474.                                           }
475.                                           }
476.                                           }
477.                                           }
478.                                           }
479.                                           }
480.                                           }
481.                                           }
482.                                           }
483.                                           }
484.                                           }
485.                                           }
486.                                           }
487.                                           }
488.                                           }
489.                                           }
490.                                           }
491.                                           }
492.                                           }
493.                                           }
494.                                           }
495.                                           }
496.                                           }
497.                                           }
498.                                           }
499.                                           }
500.                                           }
501.                                           }
502.                                           }
503.                                           }
504.                                           }
505.                                           }
506.                                           }
507.                                           }
508.                                           }
509.                                           }
510.                                           }
511.                                           }
512.                                           }
513.                                           }
514.                                           }
515.                                           }
516.                                           }
517.                                           }
518.                                           }
519.                                           }
520.                                           }
521.                                           }
522.                                           }
523.                                           }
524.                                           }
525.                                           }
526.                                           }
527.                                           }
528.                                           }
529.                                           }
530.                                           }
531.                                           }
532.                                           }
533.                                           }
534.                                           }
535.                                           }
536.                                           }
537.                                           }
538.                                           }
539.                                           }
540.                                           }
541.                                           }
542.                                           }
543.                                           }
544.                                           }
545.                                           }
546.                                           }
547.                                           }
548.                                           }
549.                                           }
550.                                           }
551.                                           }
552.                                           }
553.                                           }
554.                                           }
555.                                           }
556.                                           }
557.                                           }
558.                                           }
559.                                           }
560.                                           }
561.                                           }
562.                                           }
563.                                           }
564.                                           }
565.                                           }
566.                                           }
567.                                           }
568.                                           }
569.                                           }
570.                                           }
571.                                           }
572.                                           }
573.                                           }
574.                                           }
575.                                           }
576.                                           }
577.                                           }
578.                                           }
579.                                           }
580.                                           }
581.                                           }
582.                                           }
583.                                           }
584.                                           }
585.                                           }
586.                                           }
587.                                           }
588.                                           }
589.                                           }
590.                                           }
591.                                           }
592.                                           }
593.                                           }
594.                                           }
595.                                           }
596.                                           }
597.                                           }
598.                                           }
599.                                           }
600.                                           }
601.                                           }
602.                                           }
603.                                           }
604.                                           }
605.                                           }
606.                                           }
607.                                           }
608.                                           }
609.                                           }
610.                                           }
611.                                           }
612.                                           }
613.                                           }
614.                                           }
615.                                           }
616.                                           }
617.                                           }
618.                                           }
619.                                           }
620.                                           }
621.                                           }
622.                                           }
623.                                           }
624.                                           }
625.                                           }
626.                                           }
627.                                           }
628.                                           }
629.                                           }
630.                                           }
631.                                           }
632.                                           }
633.                                           }
634.                                           }
635.                                           }
636.                                           }
637.                                           }
638.                                           }
639.                                           }
640.                                           }
641.                                           }
642.                                           }
643.                                           }
644.                                           }
645.                                           }
646.                                           }
647.                                           }
648.                                           }
649.                                           }
650.                                           }
651.                                           }
652.                                           }
653.                                           }
654.                                           }
655.                                           }
656.                                           }
657.                                           }
658.                                           }
659.                                           }
660.                                           }
661.                                           }
662.                                           }
663.                                           }
664.                                           }
665.                                           }
666.                                           }
667.                                           }
668.                                           }
669.                                           }
670.                                           }
671.                                           }
672.                                           }
673.                                           }
674.                                           }
675.                                           }
676.                                           }
677.                                           }
678.                                           }
679.                                           }
680.                                           }
681.                                           }
682.                                           }
683.                                           }
684.                                           }
685.                                           }
686.                                           }
687.                                           }
688.                                           }
689.                                           }
690.                                           }
691.                                           }
692.                                           }
693.                                           }
694.                                           }
695.                                           }
696.                                           }
697.                                           }
698.                                           }
699.                                           }
700.                                           }
701.                                           }
702.                                           }
703.                                           }
704.                                           }
705.                                           }
706.                                           }
707.                                           }
708.                                           }
709.                                           }
710.                                           }
711.                                           }
712.                                           }
713.                                           }
714.                                           }
715.                                           }
716.                                           }
717.                                           }
718.                                           }
719.                                           }
720.                                           }
721.                                           }
722.                                           }
723.                                           }
724.                                           }
725.                                           }
726.                                           }
727.                                           }
728.                                           }
729.                                           }
730.                                           }
731.                                           }
732.                                           }
733.                                           }
734.                                           }
735.                                           }
736.                                           }
737.                                           }
738.                                           }
739.                                           }
740.                                           }
741.                                           }
742.                                           }
743.                                           }
744.                                           }
745.                                           }
746.                                           }
747.                                           }
748.                                           }
749.                                           }
750.                                           }
751.                                           }
752.                                           }
753.                                           }
754.                                           }
755.                                           }
756.                                           }
757.                                           }
758.                                           }
759.                                           }
760.                                           }
761.                                           }
762.                                           }
763.                                           }
764.                                           }
765.                                           }
766.                                           }
767.                                           }
768.                                           }
769.                                           }
770.                                           }
771.                                           }
772.                                           }
773.                                           }
774.                                           }
775.                                           }
776.                                           }
777.                                           }
778.                                           }
779.                                           }
780.                                           }
781.                                           }
782.                                           }
783.                                           }
784.                                           }
785.                                           }
786.                                           }
787.                                           }
788.                                           }
789.                                           }
790.                                           }
791.                                           }
792.                                           }
793.                                           }
794.                                           }
795.                                           }
796.                                           }
797.                                           }
798.                                           }
799.                                           }
800.                                           }
801.                                           }
802.                                           }
803.                                           }
804.                                           }
805.                                           }
806.                                           }
807.                                           }
808.                                           }
809.                                           }
810.                                           }
811.                                           }
812.                                           }
813.                                           }
814.                                           }
815.                                           }
816.                                           }
817.                                           }
818.                                           }
819.                                           }
820.                                           }
821.                                           }
822.                                           }
823.                                           }
824.                                           }
825.                                           }
826.                                           }
827.                                           }
828.                                           }
829.                                           }
830.                                           }
831.                                           }
832.                                           }
833.                                           }
834.                                           }
835.                                           }
836.                                           }
837.                                           }
838.                                           }
839.                                           }
840.                                           }
841.                                           }
842.                                           }
843.                                           }
844.                                           }
845.                                           }
846.                                           }
847.                                           }
848.                                           }
849.                                           }
850.                                           }
851.                                           }
852.                                           }
853.                                           }
854.                                           }
855.                                           }
856.                                           }
857.                                           }
858.                                           }
859.                                           }
860.                                           }
861.                                           }
862.                                           }
863.                                           }
864.                                           }
865.                                           }
866.                                           }
867.                                           }
868.                                           }
869.                                           }
870.                                           }
871.                                           }
872.                                           }
873.                                           }
874.                                           }
875.                                           }
876.                                           }
877.                                           }
878.                                           }
879.                                           }
880.                                           }
881.                                           }
882.                                           }
883.                                           }
884.                                           }
885.                                           }
886.                                           }
887.                                           }
888.                                           }
889.                                           }
890.                                           }
891.                                           }
892.                                           }
893.                                           }
894.                                           }
895.                                           }
896.                                           }
897.                                           }
898.                                           }
899.                                           }
900.                                           }
901.                                           }
902.                                           }
903.                                           }
904.                                           }
905.                                           }
906.                                           }
907.                                           }
908.                                           }
909.                                           }
910.                                           }
911.                                           }
912.                                           }
913.                                           }
914.                                           }
915.                                           }
916.                                           }
917.                                           }
918.                                           }
919.                                           }
920.                                           }
921.                                           }
922.                                           }
923.                                           }
924.                                           }
925.                                           }
926.                                           }
927.                                           }
928.                                           }
929.                                           }
930.                                           }
931.                                           }
932.                                           }
933.                                           }
934.                                           }
935.                                           }
936.                                           }
937.                                           }
938.                                           }
939.                                           }
940.                                           }
941.                                           }
942.                                           }
943.                                           }
944.                                           }
945.                                           }
946.                                           }
947.                                           }
948.                                           }
949.                                           }
950.                                           }
951.                                           }
952.                                           }
953.                                           }
954.                                           }
955.                                           }
956.                                           }
957.                                           }
958.                                           }
959.                                           }
960.                                           }
961.                                           }
962.                                           }
963.                                           }
964.                                           }
965.                                           }
966.                                           }
967.                                           }
968.                                           }
969.                                           }
970.                                           }
971.                                           }
972.                                           }
973.                                           }
974.                                           }
975.                                           }
976.                                           }
977.                                           }
978.                                           }
979.                                           }
980.                                           }
981.                                           }
982.                                           }
983.                                           }
984.                                           }
985.                                           }
986.                                           }
987.                                           }
988.                                           }
989.                                           }
990.                                           }
991.                                           }
992.                                           }
993.                                           }
994.                                           }
995.                                           }
996.                                           }
997.                                           }
998.                                           }
999.                                           }
999.                                           }
```

```

112.                                     }
113.                                     } else {
114.                                         displayError(title: "Password
Length", message: "Choose a good, strong password more than 6
characters")
115.                                     }
116.
117.
118.                                     //USER IS LOGGING IN
119.                                     } else {
120.                                         //log the user in
121.
122.                                         AuthService.instance.loginUser(withEmail: userEmail, andPassword:
userPassword, loginComplete: {(user, error) in
123.                                             if error != nil {
124.                                                 //if it goes wrong
125.                                                 self.displayError(title:
"Couldn't Log In", message: error!.localizedDescription)
126.                                             }
127.
128.                                             showing UserAccountVC
129.                                         }
130.                                         //dismiss the LoginVC
131.                                         self.dismiss(animated:
true, completion: nil)
132.                                         //call notification to
refresh the tabs
133.                                         //update the deviceFCMToken
134.                                         for push notifications
135.                                         InstanceID.instanceID().instanceID { (result, error) in
136.                                             if let error = error {
137.                                                 print("Error
fetching remote instance ID: \(error)")
138.                                             } else if let result =
139.                                                 result {
140.                                                 print("Remote
instance ID token: \(result.token)")
141.                                                 deviceFCMToken =
142.                                                 result.token
143.                                             }
144.                                         }
145.
146.                                         } else {
147.                                             displayError(title: "Oops", message:
"Please enter a password")
148.                                         }
149.                                     }

```

```

150.             } else {
151.                 displayError(title: "Oops", message: "Please
152.                     enter a valid email address")
152.             }
153.         }
154.     }
155.
156.     @IBAction func bottomLSButton(_ sender: Any) {
157.         //when bottom button pressed, the returned Bool value
158.         //is the new state of view
159.         logInMode = switchMode(logInMode: logInMode)
160.     }
161.     //prepare for CreateProfileCAVC
162.     override func prepare(for segue: UIStoryboardSegue, sender:
163.     Any?) {
164.         if segue.identifier == TO_CREATE_PROFILE {
165.
166.             //need this line to pass information between view
167.             controllers
168.             controllers
169.             let createProfileCAVC = segue.destination as!
170.             CreateProfileCAVC
171.             createProfileCAVC.userData = self.userData
172.             createProfileCAVC.email = emailField.text
173.             createProfileCAVC.password = passwordField.text
174.         }
175.     }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
userData	Dictionary<String, Any>	Public	To collect all the profile input data in when signing up for GoGig. Will be uploaded to Database.
logInMode	Boolean	Public	To keep track of whether user is going to sign up or log in and what state the view is in.
userEmail	String	Local	The input from emailField.
userPassword	String	Local	The input from passwordField.
confirmUserPassword	String	Local	The input from confirmPassword.
createProfileCAVC	CreateProfileCAVC	Local	Instance of view CreateProfileCAVC to change the value of its variables from LoginSignupVC before a segue.

## CreateProfileCAVC.swift

```
1. import UIKit
2. import FirebaseStorage
3. import FirebaseAuth
4. import FirebaseDatabase
5.
6.
7. class CreateProfileCAVC: UIViewController {
8.
9.     @IBOutlet weak var nameBioStack: UIStackView!
10.    @IBOutlet weak var playGigsButton: UIButton!
11.    @IBOutlet weak var hireMusiciansButton: UIButton!
12.    @IBOutlet weak var iWantToLabel: UILabel!
13.    @IBOutlet weak var usernameField: MyTextField!
14.    @IBOutlet weak var userBioTextView: MyTextView!
15.    @IBOutlet weak var profileImageView: UIImageView!
16.    @IBOutlet weak var musicianIcon: UIImageView!
17.    @IBOutlet weak var organiserIcon: UIImageView!
18.
19.    var user: User?
20.    var editingGate = true
21.
22.    var userData: Dictionary<String, Any>?
23.
24.    var email: String?
25.    var password: String?
26.
27.    var userGigs: Bool?
28.
29.    var placeholder = "Write a bio... |"
30.
31.    var imagePicker: UIImagePickerController?
32.    var imageID = ""
33.    var imageAdded = false
34.
35.    override func viewDidLoad() {
36.        super.viewDidLoad()
37.        setupView()
38.        hideKeyboard()
39.
40.        usernameField.updateCharacterLimit(limit: 50)
41.        userBioTextView.updatePlaceholder(placeholder:
42.            placeholder)
43.        userBioTextView.text = placeholder
44.        userBioTextView.textColor = UIColor.lightGray
45.        //to get profile image from sources
46.        imagePicker = UIImagePickerController()
47.        imagePicker?.delegate = self
48.        //make profile image round with slight border
49.        profileImageView.layer.borderWidth = 0.1
50.        profileImageView.layer.masksToBounds = false
51.        profileImageView.layer.cornerRadius =
52.            profileImageView.frame.height/2
53.        profileImageView.clipsToBounds = true
```

```

52.
53.        }
54.        //is part of navigation stack so hide the navigation bar
55.        override func viewWillAppear(_ animated: Bool) {
56.            self.navigationController?.navigationBar.isHidden =
57.                true
58.        }
59.        override func viewDidAppear(_ animated: Bool) {
60.            //if editing account
61.            if editingProfile && editingGate {
62.                //get the current user data...
63.                if let uid = Auth.auth().currentUser?.uid {
64.                    DataService.instance.getDBUserProfile(uid: uid)
65.                    { (returnedUser) in
66.                        //..as a dictionary
67.                        let returnedUserData = ["email": returnedUser.email, "bio": returnedUser.bio, "gigs": returnedUser.gigs, "name": returnedUser.name, "picURL": returnedUser.picURL, "website": returnedUser.getWebsite(), "phone": returnedUser.phone, "instagram": returnedUser.getInstagram(), "twitter": returnedUser.getTwitter(), "facebook": returnedUser.getFacebook(), "appleMusic": returnedUser.getAppleMusic(), "spotify": returnedUser.getSpotify()]
68.                        as [String : Any]
69.                        self.user = returnedUser
70.                        //userData is the returnedUserData until
71.                        //auto-fill the UI elements with current
72.                        self.userData = returnedUserData
73.                        self.usernameField.text = returnedUser.name
74.                        self.userBioTextView.text =
75.                            returnedUser.bio
76.                        self.userGigs = returnedUser.gigs
77.                        self.email = returnedUser.email
78.                        self.password = ""
79.                        if returnedUser.gigs {
80.                            self.formatGigHireButtons(chosenButton: self.playGigsButton, ignoredButton: self.hireMusiciansButton)
81.                        } else {
82.                            self.formatGigHireButtons(chosenButton: self.hireMusiciansButton, ignoredButton: self.playGigsButton)
83.                        }
84.                        //editing gate stops user going back in the
85.                        //navigation stack and
86.                        //filled out again.
87.                        self.editingGate = false
88.                    }
89.                }
90.            }
91.        }
92.    }
93.}
94.
95.//MARK: STAGE 1: TYPE OF USER

```

```
90.
91.        //set the appearance of the user type buttons
92.        func formatGigHireButtons(chosenButton: UIButton,
93.                                     ignoredButton: UIButton){
94.            chosenButton.alpha = 1
95.            //ignored button will be see-through
96.            ignoredButton.alpha = 0.5
97.            //set icons accordingly
98.            if chosenButton == playGigsButton {
99.                musicianIcon.alpha = 1
100.               organiserIcon.alpha = 0.5
101.               //record type of user
102.               userGigs = true
103.            } else {
104.                musicianIcon.alpha = 0.5
105.                organiserIcon.alpha = 1
106.                userGigs = false
107.            }
108.            //dont allow user to edit their type of user
109.            //when editing profile
110.            if editingProfile {
111.                iWantToLabel.isHidden = true
112.                playGigsButton.isHidden = true
113.                musicianIcon.isHidden = true
114.                hireMusiciansButton.isHidden = true
115.                organiserIcon.isHidden = true
116.            } else {
117.                iWantToLabel.isHidden = false
118.                playGigsButton.isHidden = false
119.                musicianIcon.isHidden = false
120.                hireMusiciansButton.isHidden = false
121.                organiserIcon.isHidden = false
122.            }
123.
124.        @IBAction func userGigs(_ sender: Any) {
125.
126.            formatGigHireButtons(chosenButton: playGigsButton,
127.                                   ignoredButton: hireMusiciansButton)
128.        }
129.        @IBAction func userHires(_ sender: Any) {
130.
131.            formatGigHireButtons(chosenButton: hireMusiciansButton,
132.                                   ignoredButton: playGigsButton)
133.
134.            //MARK: STAGE 2: ADD PROFILE PICTURE
135.
136.            //open action sheet to choose or take photo
137.            @IBAction func addProfilePhoto(_ sender: Any) {
138.                openPhotoPopup(video: false, imagePicker: imagePicker!,
139.                               title: "Profile Picture", message: "This picture is seen by other
users")
```

```
140.          //dismiss the imagePicker once one has been selected
141.          func imagePickerController(_ picker:
142.          UIImagePickerController, didFinishPickingMediaWithInfo info:
143.          [UIImagePickerController.InfoKey : Any]) {
144.          guard let selectedImage = info[.originalImage] as?
145.          UIImage else {
146.              fatalError("Expected a dictionary containing an
147.              image, but was provided the following: \(info)")
148.          }
149.          //set the UIImageView to the image picked
150.          profileImageView.image = selectedImage
151.          imageAdded = true
152.          imageID = "\(\(NSUUID()).uuidString).jpg"
153.          //dismiss ImagePicker Controller
154.          dismiss(animated: true, completion: nil)
155.      }
156.
157.
158.
159.
160.      //MARK: STAGE 3: UPLOAD PROFILE
161.
162.      @IBAction func continueButton(_ sender: Any) {
163.          //check username
164.          if let userName = usernameField.text {
165.              if userName.count >= 2 {
166.
167.                  //check Bio, it is optional
168.                  if let userBiography = userBioTextView.text {
169.                      var userBio = userBiography
170.                      //so if the text is the default...
171.                      if userBiography == "Write a bio... |" {
172.                          //...then just store in database as an
173.                          //empty string
174.                          userBio = ""
175.                      }
176.                      //check necessary data
177.                      if imageAdded && userGigs != nil {
178.                          //add to userData and segue
179.                          self.userData!["name"] = userName
180.                          self.userData!["bio"] = userBio
181.                          self.userData!["gigs"] = userGigs
182.
183.                          self.performSegue(withIdentifier:
184.                          TO_SOCIAL_LINKS, sender: nil)
185.                      } else {
186.                          displayError(title: "Oops", message:
"Please provide all necessary information")
```

```

187.                     }
188.                 }
189.
190.             } else {
191.                 displayError(title: "Oops", message: "Please
192.                     enter your name")
193.                 }
194.             }
195.
196.         override func prepare(for segue: UIStoryboardSegue, sender:
197.             Any?) {
198.             if segue.identifier == TO_SOCIAL_LINKS {
199.
200.                 //need this line to pass information between view
201.                 controllers
202.                 let socialLinksCAVC = segue.destination as!
203.                     SocialLinksCAVC
204.                 //changes it
205.                 socialLinksCAVC.userData = self.userData
206.                 socialLinksCAVC.email = self.email
207.                 socialLinksCAVC.password = self.password
208.                 socialLinksCAVC.userGigs = self.userGigs
209.                 socialLinksCAVC.imageID = self.imageID
210.                 socialLinksCAVC.profileImage =
211.                     self.profileImageView.image
212.                 socialLinksCAVC.user = self.user
213.             }
214.         }
215.     }
216.
217. }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
user	User	Public	Object to autofill out profile data when editing an account.
editingGate	Boolean	Public	So that data is not auto filled out again when user returns back to this view but one time only.
email	String	Public	To subscribe user to Authentication with when finished creating account.
password	String	Public	To subscribe user to Authentication with when finished creating account.
userGigs	Boolean	Public	To determine what type user is when signing up or editing account.
placeholder	String	Public	The custom placeholder text of the userBioTextView.

imagePicker	UIImagePickerController	Public	The view controller to choose/take images and videos from.
imageID	String	Public	Will be a unique string to store the profile image in Storage with.
imageAdded	Boolean	Public	To keep track of whether there is a profile image to be dealt with or not.
returnedUserData	Dictionary<String, Any>	Local	Profile data fetched from Database which will autofill the data entry fields for editing an account.
userName	String	Local	The input from usernameField.
userBiography	String	Local	The input from userBioTextView.
socialLinksCAVC	SocialLinksCAVC	Local	Instance of view socialLinksCAVC to change the value of its variables from CreateProfileCAVC before a segue.

## SocialLinksCAVC.swift

```

1. import UIKit
2. import FirebaseAuth
3. import FirebaseDatabase
4. import FirebaseMessaging
5. import FirebaseInstanceID
6.
7. class SocialLinksCAVC: UIViewController {
8.
9.     @IBOutlet weak var fieldsStack: UIStackView!
10.    @IBOutlet weak var phoneNumberField: MyTextField!
11.    @IBOutlet weak var websiteField: MyTextField!
12.    @IBOutlet weak var instagramField: MyTextField!
13.    @IBOutlet weak var twitterField: MyTextField!
14.    @IBOutlet weak var facebookField: MyTextField!
15.    let loadingSpinner = SpinnerViewController()
16.
17.    var user: User?
18.
19.    var userData: Dictionary<String, Any>?
20.
21.    var email: String?
22.    var password: String?
23.
24.    var userGigs: Bool?
25.
26.    var imageID = ""

```

```

27.         var profileImage: UIImage?
28.
29.         override func viewDidLoad() {
30.             super.viewDidLoad()
31.             setupView()
32.             hideKeyboard()
33.             phoneNumberField.updateCharacterLimit(limit: 16)
34.             websiteField.updateCharacterLimit(limit: 100)
35.             instagramField.updateCharacterLimit(limit: 30)
36.             twitterField.updateCharacterLimit(limit: 15)
37.             facebookField.updateCharacterLimit(limit: 20)
38.         }
39.         override func viewDidAppear(_ animated: Bool) {
40.             self.navigationController?.navigationBar.isHidden =
41.                 true
42.             self.tabBarController?.tabBar.isHidden = true
43.         }
44.         //view Did Appear is only called when in a navigation
45.         controller
45.         override func viewDidAppear(_ animated: Bool) {
46.             //autofill fields when editing
47.             if editingProfile == true && user != nil {
48.                 websiteField.text = user?.getWebsite()
49.                 phoneNumberField.text = user?.phone
50.                 instagramField.text = user?.getInstagram()
51.                 twitterField.text = user?.getTwitter()
52.                 facebookField.text = user?.getFacebook()
53.             }
54.         }
55.
56.         //MARK: STAGE 4: UPLOAD PROFILE PICTURE (If event
57.         organiser)
58.         //put the profile pic in firebase storage
59.         func picUpload(uid: String, handler: @escaping (_ url: URL)
60.             -> ()) {
61.             if let userPic = profileImage {
62.                 //upload the picture to Firebase
63.                 DataService.instance.updateSTPic(uid: uid,
64.                     directory: "profilePic", imageContent: userPic, imageID: imageID,
65.                     uploadComplete: { (success, error) in
66.                         if error != nil {
67.                             //allow user to interact with screen again
68.                             self.removeSpinnerView(self.loadingSpinner)
69.                             self.displayError(title: "There was an
70.                                 Error", message: error!.localizedDescription)
71.                         } else {
72.                             //get the URL of the stored image, to store
73.                             //in the database
74.                             DataService.instance.getSTURL(uid: uid,
75.                                 directory: "profilePic", imageID: self.imageID) { (returnedURL) in
76.                                 self.imageURL = returnedURL
77.                                 self.displaySuccess(title: "Profile Picture
78.                                     uploaded successfully!")
79.                             }
80.                         }
81.                     }
82.                 }
83.             }
84.         }
85.     }
86. 
```

```

73.                                handler(returnedURL)
74.                            }
75.                        }
76.                    }
77.                }
78.            }
79.
80.        //MARK: STAGE 5: SIGN UP IF ORGANISER, PROGRESS IF MUSICIAN
81.
82.        @IBAction func continueButton(_ sender: Any) {
83.            //all social links are optional inputs
84.            //but flag if an input does not follow validation rules
85.            var continueFine = true
86.            if let userWebsite = websiteField.text {
87.                //do checks
88.                if (userWebsite.contains(".")) && userWebsite.count
89.                    > 5) || userWebsite == "" {
90.                    //add to dictionary if okay
91.                    userData!["website"] = userWebsite
92.                } else {
93.                    //if not okay then cannot continue
94.                    displayError(title: "Oops", message: "Please
95.                        enter a valid website (optional)")
96.                    continueFine = false
97.                }
98.            if let userPhoneNumber = phoneNumberField.text {
99.                if userPhoneNumber.count > 10 || userPhoneNumber ==
100.                    "" {
101.                    userData!["phone"] = userPhoneNumber
102.                } else {
103.                    displayError(title: "Oops", message: "Please
104.                        enter a valid phone number (optional)")
105.                    continueFine = false
106.                }
107.            if let userInstagramStr = instagramField.text {
108.                //make variable from constant
109.                var userInstagram = userInstagramStr
110.                //check first char
111.                if userInstagram != "" && userInstagram.count > 1
112.                    && userInstagram[userInstagram.startIndex] == "@" {
113.                        //make a substring (remove the @)
114.                        userInstagram = userInstagram.substring(start:
115.                            1, end: userInstagram.count)
116.                    }
117.                    if (!userInstagram.contains("@") &&
118.                        !userInstagram.contains(" ")) || userInstagram == "" {
119.                        userData!["instagram"] = userInstagram
120.                    } else {
121.                        displayError(title: "Oops", message: "Please
122.                            enter a valid Instagram username (optional)")
123.                        continueFine = false
124.                    }
125.                }
126.            }
127.        }
128.
129.    }

```

```

120.             if let userTwitterStr = twitterField.text {
121.                 //make variable from constant
122.                 var userTwitter = userTwitterStr
123.                 //check first char
124.                 if userTwitter != "" && userTwitter.count > 1 &&
125.                     userTwitter[userTwitter.startIndex] == "@" {
126.                         //make a substring (remove the @)
127.                         userTwitter = userTwitter.substring(start: 1,
128.                                         end: userTwitter.count)
129.                         }
130.                         if (!userTwitter.contains("@") &&
131.                             !userTwitter.contains(" ")) || userTwitter == "" {
132.                             userData!["twitter"] = userTwitter
133.                         } else {
134.                             displayError(title: "Oops", message: "Please
135.                             enter a valid Twitter username (optional)")
136.                             continueFine = false
137.                         }
138.                         }
139.                         if let userFacebookID = facebookField.text {
140.                             if (userFacebookID.count > 10 &&
141.                                 !userFacebookID.contains(" ")) || userFacebookID == "" {
142.                                     userData!["facebook"] = userFacebookID
143.                                     }
144.                                     if continueFine {
145.                                         //if musician, allow them to add Spotify and Apple
146.                                         Music
147.                                         if userGigs! {
148.                                             self.performSegue(withIdentifier:
149.                                                 TO_MUSIC_LINKS, sender: nil)
150.                                             //organiser is done
151.                                         } else {
152.                                             //stop user interaction
153.                                             self.createSpinnerView(self.loadingSpinner)
154.                                             if editingProfile == false {
155.                                                 //sign the user up
156.                                                 AuthService.instance.registerUser(withEmail: email!, andPassword:
157.                                                     password!, userCreationComplete: { (success, error) in
158.                                                         if error != nil {
159.                                                             //allow user interaction again
160.                                                             self.removeSpinnerView(self.loadingSpinner)
161.                                                             //display a UIAlertController if
162.                                                             something goes wrong
163.                                                             self.displayError(title: "There was
164.                                                               an Error", message: error!.localizedDescription)

```

```
162.
163.                } else {
164.                    //successfully registered and added
165.                    to database
166.                } //now log user in
167.
168.                AuthService.instance.loginUser(withEmail: self.email!, andPassword:
169.                    self.password!, loginComplete: { (success, nil) in
170.                        //send their profile data to
171.                        Database
172.                        self.updateUserData()
173.                    }
174.                })
175.            } //user is editing their profile, not
176.            //creating an account
177.        }
178.    }
179.}
180.
181.func updateUserData(){
182.    if let uid = Auth.auth().currentUser?.uid {
183.        //upload the pic to cloud storage
184.        self.picUpload(uid: uid) { (returnedURL) in
185.
186.            self.userData!["picURL"] =
187.                returnedURL.absoluteString
188.                DataService.instance.updateDBUserProfile(uid:
189.                    uid, userData: self.userData!) { (complete) in
190.
191.                    if complete {
192.                        accountGateOpen = true
193.
194.                        self.removeSpinnerView(self.loadingSpinner)
195.                    }
196.                    if !editingProfile {
197.                        //creating account
198.                        self.performSegue(withIdentifier:
199.                            TO_MAIN, sender: nil)
200.                    }
201.                    else {
202.                        self.dismiss(animated: true)
203.                        //done with editing refresh the tab
204.                        NotificationCenter.default.post(name: NSNotification.Name(rawValue:
205.                            "refreshTabs"), object: nil)
206.                    }
207.                }
208.            }
209.        }
210.    }
211.}
```

```

204.                                     //update FCM Token for push
205.     notifications
206.     (result, error) in
207.     instance ID: \(error\")
208.
209.     token: \(result.token\")
210.
211.     InstanceID.instanceID().instanceID {
212.         if let error = error {
213.             print("Error fetching remote
214.             } else if let result = result {
215.                 print("Remote instance ID
216.                 deviceFCMToken = result.token
217.             }
218.
219.         override func prepare(for segue: UIStoryboardSegue, sender:
220.             Any?) {
221.             if segue.identifier == TO_MUSIC_LINKS {
222.
223.                 //need this line to pass information between view
224.                 controllers
225.                 let musicLinksCAVC = segue.destination as!
226.                 MusicLinksCAVC
227.                 //changes it
228.                 musicLinksCAVC.userData = self.userData
229.                 musicLinksCAVC.email = self.email
230.                 musicLinksCAVC.password = self.password
231.                 musicLinksCAVC.userGigs = self.userGigs
232.                 musicLinksCAVC.imageID = self.imageID
233.                 musicLinksCAVC.profileImage = self.profileImage
234.                 musicLinksCAVC.user = self.user
235.             } else if segue.identifier == TO_MAIN {
236.
237.                 let tabBarController = segue.destination as!
238.                 TabBarController
239.                 tabBarController.userGigs = self.userGigs
240.                 //done with editing
241.                 editingProfile = false
242.             }
243.         }
244.     }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
loadingSpinner	SpinnerViewController	Public	View controller is presented when data is being processed

			to show user can't interact with the screen.
profileImage	UIImage	Public	The image object which will be converted to data and uploaded to Storage.
continueFine	Boolean	Local	A variable that flags up when one of the optional inputs do not meet with validation rules to correct the user.
userWebsite userPhoneNumber userInstagramStr userTwitterStr userFacebookID	String	Local	The input from social link UITextField outlets.
userInstagram userTwitter	String	Local	To check for and remove “@” from input. Need to make a variable value of the input as userInstagramStr and userTwitterStr are constants.
musicLinksCAVC	MusicLinksCAVC	Local	Instance of view controller MusicLinksCAVC to change the value of its variables from SocialLinksCAVC before a segue.
tabBarController	TabBarController	Local	Instance of TabBarController to change the value of its variables from SocialLinksCAVC before a segue.

## MusicLinksCAVC.swift

```

1. import UIKit
2. import FirebaseAuth
3. import FirebaseDatabase
4. import FirebaseMessaging
5. import FirebaseInstanceID
6.
7. class MusicLinksCAVC: UIViewController {
8.
9.     @IBOutlet weak var fieldsStack: UIStackView!
10.    @IBOutlet weak var appleMusicField: MyTextField!
11.    @IBOutlet weak var spotifyField: MyTextField!
12.    let loadingSpinner = SpinnerViewController()
13.
14.    var user: User?
15.
16.    var userData: Dictionary<String, Any>?
17.
18.    var email: String?

```

```
19.         var password: String?
20.
21.         var userGigs: Bool?
22.
23.         var imageID = ""
24.         var profileImage: UIImage?
25.
26.         override func viewDidLoad() {
27.             super.viewDidLoad()
28.             setupView()
29.             hideKeyboard()
30.         }
31.         override func viewDidAppear(_ animated: Bool) {
32.             self.navigationController?.navigationBar.isHidden =
33.                 true
34.             self.tabBarController?.tabBar.isHidden = true
35.         }
36.         //auto-fill if editing
37.         override func viewDidAppear(_ animated: Bool) {
38.             if editingProfile == true && user != nil {
39.                 appleMusicField.text = user?.getAppleMusic()
40.                 spotifyField.text = user?.getSpotify()
41.             }
42.
43.
44.             //MARK: STAGE 6: UPLOAD PROFILE PICTURE (If musician)
45.
46.             //put the profile pic in firebase storage
47.             func picUpload(uid: String, handler: @escaping (_ url: URL)
48. -> ()) {
49.
50.                 if let userPic = profileImage {
51.                     DataService.instance.updateSTPic(uid: uid,
52.                     directory: "profilePic", imageContent: userPic, imageID: imageID,
53.                     uploadComplete: { (success, error) in
54.                         if error != nil {
55.                             self.removeSpinnerView(self.loadingSpinner)
56.                             self.displayError(title: "There was an
57. Error", message: error!.localizedDescription)
58.                         } else {
59.                             DataService.instance.getSTURL(uid: uid,
60.                             directory: "profilePic", imageID: self.imageID) { (returnedURL) in
61.                                 handler(returnedURL)
62.                             }
63.                         }
64.                     }
65.                 }
66.             }
67.
```

```

68.         @IBAction func continueButton(_ sender: Any) {
69.             //check streaming links
70.             var continueFine = true
71.             if let userAppleMusic = appleMusicField.text {
72.                 if (userAppleMusic.contains(".")) &&
73.                     userAppleMusic.count > 5) || userAppleMusic == "" {
74.                         userData!["appleMusic"] = userAppleMusic
75.                     } else {
76.                         displayError(title: "Oops", message: "Please
77.                             enter a valid Apple Music URL (optional)")
78.                         continueFine = false
79.                     }
80.                     if let userSpotify = spotifyField.text {
81.                         if (userSpotify.contains(".")) && userSpotify.count
82.                             > 5) || userSpotify == "" {
83.                                 userData!["spotify"] = userSpotify
84.                             } else {
85.                                 displayError(title: "Oops", message: "Please
86.                                     enter a valid Spotify URL (optional)")
87.                                     continueFine = false
88.                                 }
89.             }
90.             //do same as we did for organiser
91.             if continueFine {
92.                 createSpinnerView(self.loadingSpinner)
93.                 if editingProfile == false {
94.                     //sign the user up
95.                     AuthService.instance.registerUser(withEmail:
96.                         email!, andPassword: password!, userCreationComplete: { (success,
97.                         error) in
98.                             if error != nil {
99.                                 self.removeSpinnerView(self.loadingSpinner)
100.                                 self.displayError(title: "There was an
101.                                     Error", message: error!.localizedDescription)
102.                             } else {
103.                                 //successfully registered and added to
104.                                 database
105.                                 //now log user in
106.                                 AuthService.instance.loginUser(withEmail: self.email!, andPassword:
107.                                     self.password!, loginComplete: { (success, nil) in })
108.                                 self.updateUserData()
109.                             }
110.                         } else {
111.                             //user is editing their profile
112.                             self.updateUserData()
113.                         }
114.                     }
115.                 }
116.             }
117.         }
118.     }
119. }

```

```

112.             }
113.         }
114.     }
115.
116.     func updateUserData(){
117.         if let uid = Auth.auth().currentUser?.uid {
118.             //upload the pic to cloud storage
119.             self.picUpload(uid: uid) { (returnedURL) in
120.
121.                 self.userData!["picURL"] =
122.                     returnedURL.absoluteString
123.             DataService.instance.updateDBUserProfile(uid:
124.                 uid, userData: self.userData!) { (complete) in
125.                     if complete {
126.
127.                         accountGateOpen = true
128.
129.                         self.removeSpinnerView(self.loadingSpinner)
130.
131.                         //now do a dismiss rather than a
132.                         //perform segue, this is because when performing a segue, we
133.                         //instantiate a new tab controller
134.                         if !editingProfile {
135.                             //creating account
136.                             self.performSegue(withIdentifier:
137.                                 TO_MAIN_2, sender: nil)
138.                         } else {
139.                             //dismiss for editing
140.                             self.dismiss(animated: true)
141.                             editingProfile = false
142.                         }
143.                         NotificationCenter.default.post(name: NSNotification.Name(rawValue:
144.                             "refreshTabs"), object: nil)
145.                     }
146.                     //update FCM Token for push
147.                     notifications
148.                     InstanceID.instanceID().instanceID {
149.                         if let error = error {
150.                             print("Error fetching remote
151.                             instance ID: \(error)")
152.                         } else if let result = result {
153.                             print("Remote instance ID
154.                             token: \(result.token)")
155.                         }
156.                     }
157.                 }
158.             }
159.         }
160.     }
161. }
```

```

155.
156.         override func prepare(for segue: UIStoryboardSegue, sender:
157.             Any?) {
158.             if segue.identifier == TO_MAIN_2 {
159.
160.                 let tabBarController = segue.destination as!
161.                     TabBarController
162.                     tabBarController.userGigs = self.userGigs
163.                     editingProfile = false
164.             }
165.         }
166.     }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
userAppleMusic	String	Local	The inputs from the music link UITextField outlets.
userSpotify			

### UserAccountVC.swift

```

1. import UIKit
2. import AVFoundation
3. import FirebaseAuth
4. import FirebaseStorage
5. import FirebaseDatabase
6.
7. class UserAccountVC: UITableViewController {
8.
9.     @IBOutlet weak var settingsBarButton: UIBarButtonItem!
10.    @IBOutlet weak var addPortfolioBarButton: UIBarButtonItem!
11.
12.    //keep track of user viewing someone else's portfolio or
13.    not
14.    var observingPortfolio = false
15.    var hideForLoad = true
16.
17.    var user: User?
18.    var uid: String?
19.    var portfolioPosts = [PortfolioPost]()
20.
21.    override func viewDidLoad() {
22.        setupView()
23.
24.        NotificationCenter.default.addObserver(self, selector:
25.            #selector(refreshPortfolio), name: NSNotification.Name(rawValue:
26.                "refreshPortfolio"), object: nil)
27.
28.        //prevents crash if we get to this view without a
29.        logged in user
30.        if Auth.auth().currentUser != nil {
31.            refreshPortfolio()
32.        }

```

```

29.        }
30.        //refresh token for push notifications
31.        override func viewDidAppear(_ animated: Bool) {
32.            refreshFCMToken()
33.        }
34.        //had issue where audio was heard after a segue
35.        override func viewDidDisappear(_ animated: Bool) {
36.            //close player when view disappears
37.            if playingAVPlayer != nil {
38.                playingAVPlayer!.closePlayer()
39.            }
40.        }
41.
42.        //MARK: FETCH DATA
43.        //reuse this VC when we want to look at someone else's
44.        //profile
45.        //if we didn't click on anything when the view appears, use
46.        //the current user uid
47.        @objc func refreshPortfolio(){
48.
49.            //user is looking at themself
50.            //gate is needed incase user signs in and signs out
51.            //again
52.            if uid == nil || accountGateOpen {
53.                accountGateOpen = false
54.                uid = Auth.auth().currentUser?.uid
55.            }
56.            //user is looking at another
57.
58.            //when observing
59.            if observingPortfolio {
60.                //hide the settings
61.                navigationItem.leftBarButtonItem = nil
62.                //hide the add post button
63.                navigationItem.rightBarButtonItem = nil
64.            }
65.
66.            //get the profile data
67.            DataService.instance.getDBUserProfile(uid: uid!) {
68.                (returnedUser) in
69.                    //set the user who owns the portfolio
70.                    self.user = returnedUser
71.                    //load profile picture (from cache or download)
72.                    self.loadImageCache(url: returnedUser.picURL,
73.                        isImage: true) { (returnedProfileImage) in
74.                            self.profilePic = returnedProfileImage
75.                            DataService.instance.getDBPortfolioPosts(uid:
76.                                self.uid!) { (returnedPosts) in
77.                                    //quick sort the posts by reverse
78.                                    //chronological
79.                                    self.portfolioPosts =
80.                                    self.quickSort(array:returnedPosts)
81.                                    //show profile
82.                                    self.hideForLoad = false
83.                                    //show all the data in table view

```

```

76.                                self.tableView.reloadData()
77.                            }
78.                        }
79.                    }
80.                }
81.
82.                @IBAction func settingsButton(_ sender: Any) {
83.                    let settingsPopup = UIAlertController(title:
84.                        "Settings", message: "What would you like to do?", preferredStyle:
85.                            .actionSheet)
86.                    //go and edit the account
87.                    let editProfileAction = UIAlertAction(title: "Edit
88.                        profile", style: .default) { (buttonTapped) in
89.                            editingProfile = true
90.                            if let tabBarController = self.tabBarController {
91.                                tabBarController.viewControllers = tabs
92.                                tabGateOpen = true //Reset tabs
93.                                accountGateOpen = true //Reset portfolio
94.                                refresh
95.                                refresh
96.                                refresh
97.                                pagination
98.                                pushNotificationGateOpen = true
99.                            }
100.                           //go and log out of the account
101.                           let logoutAction = UIAlertAction(title: "Log out",
102.                               style: .destructive) { (buttonTapped) in
103.                                   //provide a double check
104.                                   let alertController = UIAlertController(title: "Log
105.                                       out", message: "Are you sure you want to log out of your account?", preferredStyle:
106.                                           .alert)
107.                                   alertController.addAction(UIAlertAction(title:
108.                                       "Cancel", style: .default))
109.                                   alertController.addAction(UIAlertAction(title: "Log
110.                                       out", style: .destructive, handler: { (buttonPressed) in
111.                                           do {
112.                                               if let uid = Auth.auth().currentUser?.uid {
113.                                                   //so does not see things in account
114.                                                   which is not theirs
115.                                                   DataService.instance.removeObservers(uid: uid)
116.                                               //change the FCM token so the iPhone
117.                                               stops receiving notifications
118.                                               DataService.instance.updateDBUserFCMToken(uid: uid, token:
119.                                                   "empty_token")
120.                                           }
121.                                       }
122.                                       }
123.                                       }
124.                                       }
125.                                       }
126.                                       }
127.                                       }
128.                                       }
129.                                       }
130.                                       }
131.                                       }
132.                                       }
133.                                       }
134.                                       }
135.                                       }
136.                                       }
137.                                       }
138.                                       }
139.                                       }
140.                                       }
141.                                       }
142.                                       }
143.                                       }
144.                                       }
145.                                       }
146.                                       }
147.                                       }
148.                                       }
149.                                       }
150.                                       }
151.                                       }
152.                                       }
153.                                       }
154.                                       }
155.                                       }
156.                                       }
157.                                       }
158.                                       }
159.                                       }
160.                                       }
161.                                       }
162.                                       }
163.                                       }
164.                                       }
165.                                       }
166.                                       }
167.                                       }
168.                                       }
169.                                       }
170.                                       }
171.                                       }
172.                                       }
173.                                       }
174.                                       }
175.                                       }
176.                                       }
177.                                       }
178.                                       }
179.                                       }
180.                                       }
181.                                       }
182.                                       }
183.                                       }
184.                                       }
185.                                       }
186.                                       }
187.                                       }
188.                                       }
189.                                       }
190.                                       }
191.                                       }
192.                                       }
193.                                       }
194.                                       }
195.                                       }
196.                                       }
197.                                       }
198.                                       }
199.                                       }
200.                                       }
201.                                       }
202.                                       }
203.                                       }
204.                                       }
205.                                       }
206.                                       }
207.                                       }
208.                                       }
209.                                       }
210.                                       }
211.                                       }
212.                                       }
213.                                       }
214.                                       }
215.                                       }
216.                                       }
217.                                       }
218.                                       }
219.                                       }
220.                                       }
221.                                       }
222.                                       }
223.                                       }
224.                                       }
225.                                       }
226.                                       }
227.                                       }
228.                                       }
229.                                       }
230.                                       }
231.                                       }
232.                                       }
233.                                       }
234.                                       }
235.                                       }
236.                                       }
237.                                       }
238.                                       }
239.                                       }
240.                                       }
241.                                       }
242.                                       }
243.                                       }
244.                                       }
245.                                       }
246.                                       }
247.                                       }
248.                                       }
249.                                       }
250.                                       }
251.                                       }
252.                                       }
253.                                       }
254.                                       }
255.                                       }
256.                                       }
257.                                       }
258.                                       }
259.                                       }
260.                                       }
261.                                       }
262.                                       }
263.                                       }
264.                                       }
265.                                       }
266.                                       }
267.                                       }
268.                                       }
269.                                       }
270.                                       }
271.                                       }
272.                                       }
273.                                       }
274.                                       }
275.                                       }
276.                                       }
277.                                       }
278.                                       }
279.                                       }
280.                                       }
281.                                       }
282.                                       }
283.                                       }
284.                                       }
285.                                       }
286.                                       }
287.                                       }
288.                                       }
289.                                       }
290.                                       }
291.                                       }
292.                                       }
293.                                       }
294.                                       }
295.                                       }
296.                                       }
297.                                       }
298.                                       }
299.                                       }
300.                                       }
301.                                       }
302.                                       }
303.                                       }
304.                                       }
305.                                       }
306.                                       }
307.                                       }
308.                                       }
309.                                       }
310.                                       }
311.                                       }
312.                                       }
313.                                       }
314.                                       }
315.                                       }
316.                                       }
317.                                       }
318.                                       }
319.                                       }
320.                                       }
321.                                       }
322.                                       }
323.                                       }
324.                                       }
325.                                       }
326.                                       }
327.                                       }
328.                                       }
329.                                       }
330.                                       }
331.                                       }
332.                                       }
333.                                       }
334.                                       }
335.                                       }
336.                                       }
337.                                       }
338.                                       }
339.                                       }
340.                                       }
341.                                       }
342.                                       }
343.                                       }
344.                                       }
345.                                       }
346.                                       }
347.                                       }
348.                                       }
349.                                       }
350.                                       }
351.                                       }
352.                                       }
353.                                       }
354.                                       }
355.                                       }
356.                                       }
357.                                       }
358.                                       }
359.                                       }
360.                                       }
361.                                       }
362.                                       }
363.                                       }
364.                                       }
365.                                       }
366.                                       }
367.                                       }
368.                                       }
369.                                       }
370.                                       }
371.                                       }
372.                                       }
373.                                       }
374.                                       }
375.                                       }
376.                                       }
377.                                       }
378.                                       }
379.                                       }
380.                                       }
381.                                       }
382.                                       }
383.                                       }
384.                                       }
385.                                       }
386.                                       }
387.                                       }
388.                                       }
389.                                       }
390.                                       }
391.                                       }
392.                                       }
393.                                       }
394.                                       }
395.                                       }
396.                                       }
397.                                       }
398.                                       }
399.                                       }
400.                                       }
401.                                       }
402.                                       }
403.                                       }
404.                                       }
405.                                       }
406.                                       }
407.                                       }
408.                                       }
409.                                       }
410.                                       }
411.                                       }
412.                                       }
413.                                       }
414.                                       }
415.                                       }
416.                                       }
417.                                       }
418.                                       }
419.                                       }
420.                                       }
421.                                       }
422.                                       }
423.                                       }
424.                                       }
425.                                       }
426.                                       }
427.                                       }
428.                                       }
429.                                       }
430.                                       }
431.                                       }
432.                                       }
433.                                       }
434.                                       }
435.                                       }
436.                                       }
437.                                       }
438.                                       }
439.                                       }
440.                                       }
441.                                       }
442.                                       }
443.                                       }
444.                                       }
445.                                       }
446.                                       }
447.                                       }
448.                                       }
449.                                       }
450.                                       }
451.                                       }
452.                                       }
453.                                       }
454.                                       }
455.                                       }
456.                                       }
457.                                       }
458.                                       }
459.                                       }
460.                                       }
461.                                       }
462.                                       }
463.                                       }
464.                                       }
465.                                       }
466.                                       }
467.                                       }
468.                                       }
469.                                       }
470.                                       }
471.                                       }
472.                                       }
473.                                       }
474.                                       }
475.                                       }
476.                                       }
477.                                       }
478.                                       }
479.                                       }
480.                                       }
481.                                       }
482.                                       }
483.                                       }
484.                                       }
485.                                       }
486.                                       }
487.                                       }
488.                                       }
489.                                       }
490.                                       }
491.                                       }
492.                                       }
493.                                       }
494.                                       }
495.                                       }
496.                                       }
497.                                       }
498.                                       }
499.                                       }
500.                                       }
501.                                       }
502.                                       }
503.                                       }
504.                                       }
505.                                       }
506.                                       }
507.                                       }
508.                                       }
509.                                       }
510.                                       }
511.                                       }
512.                                       }
513.                                       }
514.                                       }
515.                                       }
516.                                       }
517.                                       }
518.                                       }
519.                                       }
520.                                       }
521.                                       }
522.                                       }
523.                                       }
524.                                       }
525.                                       }
526.                                       }
527.                                       }
528.                                       }
529.                                       }
530.                                       }
531.                                       }
532.                                       }
533.                                       }
534.                                       }
535.                                       }
536.                                       }
537.                                       }
538.                                       }
539.                                       }
540.                                       }
541.                                       }
542.                                       }
543.                                       }
544.                                       }
545.                                       }
546.                                       }
547.                                       }
548.                                       }
549.                                       }
550.                                       }
551.                                       }
552.                                       }
553.                                       }
554.                                       }
555.                                       }
556.                                       }
557.                                       }
558.                                       }
559.                                       }
560.                                       }
561.                                       }
562.                                       }
563.                                       }
564.                                       }
565.                                       }
566.                                       }
567.                                       }
568.                                       }
569.                                       }
570.                                       }
571.                                       }
572.                                       }
573.                                       }
574.                                       }
575.                                       }
576.                                       }
577.                                       }
578.                                       }
579.                                       }
580.                                       }
581.                                       }
582.                                       }
583.                                       }
584.                                       }
585.                                       }
586.                                       }
587.                                       }
588.                                       }
589.                                       }
590.                                       }
591.                                       }
592.                                       }
593.                                       }
594.                                       }
595.                                       }
596.                                       }
597.                                       }
598.                                       }
599.                                       }
600.                                       }
601.                                       }
602.                                       }
603.                                       }
604.                                       }
605.                                       }
606.                                       }
607.                                       }
608.                                       }
609.                                       }
610.                                       }
611.                                       }
612.                                       }
613.                                       }
614.                                       }
615.                                       }
616.                                       }
617.                                       }
618.                                       }
619.                                       }
620.                                       }
621.                                       }
622.                                       }
623.                                       }
624.                                       }
625.                                       }
626.                                       }
627.                                       }
628.                                       }
629.                                       }
630.                                       }
631.                                       }
632.                                       }
633.                                       }
634.                                       }
635.                                       }
636.                                       }
637.                                       }
638.                                       }
639.                                       }
640.                                       }
641.                                       }
642.                                       }
643.                                       }
644.                                       }
645.                                       }
646.                                       }
647.                                       }
648.                                       }
649.                                       }
650.                                       }
651.                                       }
652.                                       }
653.                                       }
654.                                       }
655.                                       }
656.                                       }
657.                                       }
658.                                       }
659.                                       }
660.                                       }
661.                                       }
662.                                       }
663.                                       }
664.                                       }
665.                                       }
666.                                       }
667.                                       }
668.                                       }
669.                                       }
670.                                       }
671.                                       }
672.                                       }
673.                                       }
674.                                       }
675.                                       }
676.                                       }
677.                                       }
678.                                       }
679.                                       }
680.                                       }
681.                                       }
682.                                       }
683.                                       }
684.                                       }
685.                                       }
686.                                       }
687.                                       }
688.                                       }
689.                                       }
690.                                       }
691.                                       }
692.                                       }
693.                                       }
694.                                       }
695.                                       }
696.                                       }
697.                                       }
698.                                       }
699.                                       }
700.                                       }
701.                                       }
702.                                       }
703.                                       }
704.                                       }
705.                                       }
706.                                       }
707.                                       }
708.                                       }
709.                                       }
710.                                       }
711.                                       }
712.                                       }
713.                                       }
714.                                       }
715.                                       }
716.                                       }
717.                                       }
718.                                       }
719.                                       }
720.                                       }
721.                                       }
722.                                       }
723.                                       }
724.                                       }
725.                                       }
726.                                       }
727.                                       }
728.                                       }
729.                                       }
730.                                       }
731.                                       }
732.                                       }
733.                                       }
734.                                       }
735.                                       }
736.                                       }
737.                                       }
738.                                       }
739.                                       }
740.                                       }
741.                                       }
742.                                       }
743.                                       }
744.                                       }
745.                                       }
746.                                       }
747.                                       }
748.                                       }
749.                                       }
750.                                       }
751.                                       }
752.                                       }
753.                                       }
754.                                       }
755.                                       }
756.                                       }
757.                                       }
758.                                       }
759.                                       }
760.                                       }
761.                                       }
762.                                       }
763.                                       }
764.                                       }
765.                                       }
766.                                       }
767.                                       }
768.                                       }
769.                                       }
770.                                       }
771.                                       }
772.                                       }
773.                                       }
774.                                       }
775.                                       }
776.                                       }
777.                                       }
778.                                       }
779.                                       }
780.                                       }
781.                                       }
782.                                       }
783.                                       }
784.                                       }
785.                                       }
786.                                       }
787.                                       }
788.                                       }
789.                                       }
790.                                       }
791.                                       }
792.                                       }
793.                                       }
794.                                       }
795.                                       }
796.                                       }
797.                                       }
798.                                       }
799.                                       }
800.                                       }
801.                                       }
802.                                       }
803.                                       }
804.                                       }
805.                                       }
806.                                       }
807.                                       }
808.                                       }
809.                                       }
810.                                       }
811.                                       }
812.                                       }
813.                                       }
814.                                       }
815.                                       }
816.                                       }
817.                                       }
818.                                       }
819.                                       }
820.                                       }
821.                                       }
822.                                       }
823.                                       }
824.                                       }
825.                                       }
826.                                       }
827.                                       }
828.                                       }
829.                                       }
830.                                       }
831.                                       }
832.                                       }
833.                                       }
834.                                       }
835.                                       }
836.                                       }
837.                                       }
838.                                       }
839.                                       }
840.                                       }
841.                                       }
842.                                       }
843.                                       }
844.                                       }
845.                                       }
846.                                       }
847.                                       }
848.                                       }
849.                                       }
850.                                       }
851.                                       }
852.                                       }
853.                                       }
854.                                       }
855.                                       }
856.                                       }
857.                                       }
858.                                       }
859.                                       }
860.                                       }
861.                                       }
862.                                       }
863.                                       }
864.                                       }
865.                                       }
866.                                       }
867.                                       }
868.                                       }
869.                                       }
870.                                       }
871.                                       }
872.                                       }
873.                                       }
874.                                       }
875.                                       }
876.                                       }
877.                                       }
878.                                       }
879.                                       }
880.                                       }
881.                                       }
882.                                       }
883.                                       }
884.                                       }
885.                                       }
886.                                       }
887.                                       }
888.                                       }
889.                                       }
890.                                       }
891.                                       }
892.                                       }
893.                                       }
894.                                       }
895.                                       }
896.                                       }
897.                                       }
898.                                       }
899.                                       }
900.                                       }
901.                                       }
902.                                       }
903.                                       }
904.                                       }
905.                                       }
906.                                       }
907.                                       }
908.                                       }
909.                                       }
910.                                       }
911.                                       }
912.                                       }
913.                                       }
914.                                       }
915.                                       }
916.                                       }
917.                                       }
918.                                       }
919.                                       }
920.                                       }
921.                                       }
922.                                       }
923.                                       }
924.                                       }
925.                                       }
926.                                       }
927.                                       }
928.                                       }
929.                                       }
930.                                       }
931.                                       }
932.                                       }
933.                                       }
934.                                       }
935.                                       }
936.                                       }
937.                                       }
938.                                       }
939.                                       }
940.                                       }
941.                                       }
942.                                       }
943.                                       }
944.                                       }
945.                                       }
946.                                       }
947.                                       }
948.                                       }
949.                                       }
950.                                       }
951.                                       }
952.                                       }
953.                                       }
954.                                       }
955.                                       }
956.                                       }
957.                                       }
958.                                       }
959.                                       }
960.                                       }
961.                                       }
962.                                       }
963.                                       }
964.                                       }
965.                                       }
966.                                       }
967.                                       }
968.                                       }
969.                                       }
970.                                       }
971.                                       }
972.                                       }
973.                                       }
974.                                       }
975.                                       }
976.                                       }
977.                                       }
978.                                       }
979.                                       }
980.                                       }
981.                                       }
982.                                       }
983.                                       }
984.                                       }
985.                                       }
986.                                       }
987.                                       }
988.                                       }
989.                                       }
990.                                       }
991.                                       }
992.                                       }
993.                                       }
994.                                       }
995.                                       }
996.                                       }
997.                                       }
998.                                       }
999.                                       }
1000.                                       }
1001.                                       }
1002.                                       }
1003.                                       }
1004.                                       }
1005.                                       }
1006.                                       }
1007.                                       }
1008.                                       }
1009.                                       }
1010.                                       }
1011.                                       }
1012.                                       }
1013.                                       }
1014.                                       }
1015.                                       }
1016.                                       }
1017.                                       }
1018.                                       }
1019.                                       }
1020.                                       }
1021.                                       }
1022.                                       }
1023.                                       }
1024.                                       }
1025.                                       }
1026.                                       }
1027.                                       }
1028.                                       }
1029.                                       }
1030.                                       }
1031.                                       }
1032.                                       }
1033.                                       }
1034.                                       }
1035.                                       }
1036.                                       }
1037.                                       }
1038.                                       }
1039.                                       }
1040.                                       }
1041.                                       }
1042.                                       }
1043.                                       }
1044.                                       }
1045.                                       }
1046.                                       }
1047.                                       }
1048.                                       }
1049.                                       }
1050.                                       }
1051.                                       }
1052.                                       }
1053.                                       }
1054.                                       }
1055.                                       }
1056.                                       }
1057.                                       }
1058.                                       }
1059.                                       }
1060.                                       }
1061.                                       }
1062.                                       }
1063.                                       }
1064.                                       }
1065.                                       }
1066.                                       }
1067.                                       }
1068.                                       }
1069.                                       }
1070.                                       }
1071.                                       }
1072.                                       }
1073.                                       }
1074.                                       }
1075.                                       }
1076.                                       }
1077.                                       }
1078.                                       }
1079.                                       }
1080.                                       }
1081.                                       }
1082.                                       }
1083.                                       }
1084.                                       }
1085.                                       }
1086.                                       }
1087.                                       }
1088.                                       }
1089.                                       }
1090.                                       }
1091.                                       }
1092.                                       }
1093.                                       }
1094.                                       }
1095.                                       }
1096.                                       }
1097.                                       }
1098.                                       }
1099.                                       }
1100.                                       }
1101.                                       }
1102.                                       }
1103.                                       }
1104.                                       }
1105.                                       }
1106.                                       }
1107.                                       }
1108.                                       }
1109.                                       }
1110.                                       }
1111.                                       }
1112.                                       }
1113.                                       }
1114.                                       }
1115.                                       }
1116.                                       }
1117.                                       }
1118.                                       }
1119.                                       }
1120.                                       }
1121.                                       }
1122.                                       }
1123.                                       }
1124.                                       }
1125.                                       }
1126.                                       }
1127.                                       }
1128.                                       }
1129.                                       }
1130.                                       }
1131.                                       }
1132.                                       }
1133.                                       }
1134.                                       }
1135.                                       }
1136.                                       }
1137.                                       }
1138.                                       }
1139.                                       }
1140.                                       }
1141.                                       }
1142.                                       }
1143.                                       }
1144.                                       }
1145.                                       }
1146.                                       }
1147.                                       }
1148.                                       }
1149.                                       }
1150.                                       }
1151.                                       }
1152.                                       }
1153.                                       }
1154.                                       }
1155.                                       }
1156.                                       }
1157.                                       }
1158.                                       }
1159.                                       }
1160.                                       }
1161.                                       }
1162.                                       }
1163.                                       }
1164.                                       }
1165.                                       }
1166.                                       }
1167.                                       }
1168.                                       }
1169.                                       }
1170.                                       }
1171.                                       }
1172.                                       }
1173.                                       }
1174.                                       }
1175.                                       }
1176.                                       }
1177.                                       }
1178.                                       }
1179.                                       }
1180.                                       }
1181.                                       }
1182.                                       }
1183.                                       }
1184.                                       }
1185.                                       }
1186.                                       }
1187.                                       }
1188.                                       }
1189.                                       }
1190.                                       }
1191.                                       }
1192.                                       }
1193.                                       }
1194.                                       }
1195.                                       }
1196.                                       }
1197.                                       }
1198.                                       }
1199.                                       }
1200.                                       }
1201.                                       }
1202.                                       }
1203.                                       }
1204.                                       }
1205.                                       }
1206.                                       }
1207.                                       }
1208.                                       }
1209.                                       }
1210.                                       }
1211.                                       }
1212.                                       }
1213.                                       }
1214.                                       }
1215.                                       }
1216.                                       }
1217.                                       }
1218.                                       }
1219.                                       }
1220.                                       }
1221.                                       }
1222.                                       }
1223.                                       }
1224.                                       }
1225.                                       }
1226.                                       }
1227.                                       }
1228.                                       }
1229.                                       }
1230.                                       }
1231.                                       }
1232.                                       }
1233.                                       }
1234.                                       }
1235.                                       }
1236.                                       }
1237.                                       }
1238.                                       }
1239.                                       }
1240.                                       }
1241.                                       }
1242.                                       }
1243.                                       }
1244.                                       }
1245.                                       }
1246.                                       }
1247.                                       }
1248.                                       }
1249.                                       }
1250.                                       }
1251.                                       }
1252.                                       }
1253.                                       }
1254.                                       }
1255.                                       }
1256.                                       }
1257.                                       }
1258.                                       }
1259.                                       }
1260.                                       }
1261.                                       }
1262.                                       }
1263.                                       }
1264.                                       }
1265.                                       }
1266.                                       }
1267.                                       }
1268.                                       }
1269.                                       }
1270.                                       }
1271.                                       }
1272.                                       }
1273.                                       }
1274.                                       }
1275.                                       }
1276.                                       }
1277.                                       }
1278.                                       }
1279.                                       }
1280.                                       }
1281.                                       }
1282.                                       }
1283.                                       }
1284.                                       }
1285.                                       }
1286.                                       }
1287.                                       }
1288.                                       }
1289.                                       }
1290.                                       }
1291.                                       }
1292.                                       }
1293.                                       }
1294.                                       }
1295.                                       }
1296.                                       }
1297.                                       }
1298.                                       }
1299.                                       }
1300.                                       }
1301.                                       }
1302.                                       }
1303.                                       }
1304.                                       }
1305.                                       }
1306.                                       }
1307.                                       }
1308.                                       }
1309.                                       }
1310.                                       }
1311.                                       }
1312.                                       }
1313.                                       }
1314.                                       }
1315.                                       }
1316.                                       }
1317.                                       }
1318.                                       }
1319.                                       }
1320.                                       }
1321.                                       }
1322.                                       }
1323.                                       }
1324.                                       }
1325.                                       }
1326.                                       }
1327.                                       }
1328.                                       }
1329.                                       }
1330.                                       }
1331.                                       }
1332.                                       }
1333.                                       }
1334.                                       }
1335.                                       }
1336.                                       }
1337.                                       }
1338.                                       }
1339.                                       }
1340.                                       }
1341.                                       }
1342.                                       }
1343.                                       }
1344.                                       }
1345.                                       }
1346.                                       }
1347.                                       }
1348.                                       }
1349.                                       }
1350.                                       }
1351.                                       }
1352.                                       }
1353.                                       }
1354.                                       }
1355.                                       }
1356.                                       }
1357.                                       }
1358.                                       }
1359.                                       }
1360.                                       }
1361.                                       }
1362.                                       }
1363.                                       }
1364.                                       }
1365.                                       }
1366.                                       }
1367.                                       }
1368.                                       }
1369.                                       }
1370.                                       }
1371.                                       }
1372.                                       }
1373.                                       }
1374.                                       }
1375.                                       }
1376.                                       }
1377.                                       }
1378.                                       }
1379.                                       }
1380.                                       }
1381.                                       }
1382.                                       }
1383.                                       }
1384.                                       }
1385.                                       }
1386.                                       }
1387.                                       }
1388.                                       }
1389.                                       }
1390.                                       }
1391.                                       }
1392.                                       }
1393.                                       }
1394.                                       }
1395.                                       }
1396.                                       }
1397.                                       }
1398.                                       }
1399.                                       }
1400.                                       }
1401.                                       }
1402.                                       }
1403.                                       }
1404.                                       }
1405.                                       }
1406.                                       }
1407.                                       }
1408.                                       }
1409.                                       }
1410.                                       }
1411.                                       }
1412.                                       }
1413.                                       }
1414.                                       }
1415.                                       }
1416.                                       }
1417.                                       }
1418.                                       }
1419.                                       }
1420.                                       }
1421.                                       }
1422.                                       }
1423.                                       }
1424.                                       }
1425.                                       }
1426.                                       }
1427.                                       }
1428.                                       }
1429.                                       }
1430.                                       }
1431.                                       }
1432.                                       }
1433.                                       }
1434.                                       }
1435.                                       }
1436.                                       }
1437.                                       }
1438.                                       }
1439.                                       }
1440.                                       }
1441.                                       }
1442.                                       }
1443.                                       }
1444.                                       }
1445.                                       }
1446.                                       }
1447.                                       }
1448.                                       }
1449.                                       }
1450.                                       }
1451.                                       }
1452.                                       }
1453.                                       }
1454.                                       }
1455.                                       }
1456.                                       }
1457.                                       }
1458.                                       }
1459.                                       }
1460.                                       }
1461.                                       }
1462.                                       }
1463.                                       }
1464.                                       }
1465.                                       }
1466.                                       }
1467.                                       }
1468.                                       }
1469.                                       }
1470.                                       }
1471.                                       }
1472.                                       }
1473.                                       }
1474.                                       }
1475.                                       }
1476.                                       }
1477.                                       }
1478.                                       }
1479.                                       }
1480.                                       }
1481.                                       }
1482.                                       }
1483.                                       }
1484.                                       }
1485.                                       }
1486.                                       }
1487.                                       }
1488.                                       }
1489.                                       }
1490.                                       }
1491.                                       }
1492.                                       }
1493.                                       }
1494.                                       }
1495.                                       }
1496.                                       }
1497.                                       }
1498.                                       }
1499.                                       }
1500.                                       }
1501.                                       }
1502.                                       }
1503.                                       }
1504.                                       }
1505.                                       }
1506.                                       }
1507.                                       }
1508.                                       }
1509.                                       }
1510.                                       }
1511.                                       }
1512.                                       }
1513.                                       }
1514.                                       }
1515.                                       }
1516.                                       }
1517.                                       }
1518.                                       }
1519.                                       }
1520.                                       }
1521.                                       }
1522.                                       }
1523.                                       }
1524.                                       }
1525.                                       }
1526.                                       }
1527.                                       }
1528.                                       }
1529.                                       }
1530.                                       }
1531.                                       }
1532.                                       }
1533.                                       }
1534.                                       }
1535.                                       }
1536.                                       }
1537.                                       }
1538.                                       }
1539.                                       }
1540.                                       }
1541.                                       }
1542.                                       }
1543.                                       }
1544.                                       }
1545.                                       }
1546.                                       }
1547.                                       }
1548.                                       }
1549.                                       }
1550.                                       }
1551.                                       }
1552.                                       }
1553.                                       }
1554.                                       }
1555.                                       }
1556.                                       }
1557.                                       }
1558.                                       }
1559.                                       }
1560.                                       }
1561.                                       }
1562.                                       }
1563.                                       }
1564.                                       }
1565.                                       }
1566.                                       }
1567.                                       }
1568.                                       }
1569.                                       }
1570.                                       }
1571.                                       }
1572.                                       }
1573.                                       }
1574.                                       }
1575.                                       }
1576.                                       }
1577.                                       }
1578.                                       }
1579.                                       }
1580.                                       }
1581.                                       }
1582.                                       }
1583.                                       }
1584.                                       }
1585.                                       }
1586.                                       }
1587.                                       }
1588.                                       }
1589.                                       }
1590.                                       }
1591.                                       }
1592.                                       }
1593.                                       }
1594.                                       }
1595.                                       }
1596.                                       }
1597.                                       }
1598.                                       }
1599.                                       }
1600.                                       }
1601.                                       }
1602.                                       }
1603.                                       }
1604.                                       }
1605.                                       }
1606.                                       }
1607.                                       }
1608.                                       }
1609.                                       }
1610.                                       }
1611.                                       }
1612.                                       }
1613.                                       }
1614.                                       }
1615.                                       }
1616.                                       }
1617.                                       }
1618.                                       }
1619.                                       }
1620.                                       }
1621.                                       }
1622.                                       }
1623.                                       }
1624.                                       }
1625.                                       }
1626.                                       }
1627.                                       }
1628.                                       }
1629.                                       }
1630.                                       }
1631.                                       }
1632.                                       }
1633.                                       }
1634.                                       }
1635.                                       }
1636.                                       }
1637.                                       }
1638.                                       }
1639.                                       }
1640.                                       }
1641.                                       }
1642.                                       }
1643.                                       }
1644.                                       }
1645.                                       }
1646.                                       }
1647.                                       }
1648.                                       }
1649.                                       }
1650.                                       }
1651.                                       }
1652.                                       }
1653.                                       }
1654.                                       }
1655.                                       }
1656.                                       }
1657.                                       }
1658.                                       }
1659.                                       }
1660.                                       }
1661.                                       }
1662.                                       }
1663.                                       }
1664.                                       }
1665.                                       }
1666.                                       }
1667.                                       }
1668.                                       }
1669.                                       }
1670.                                       }
1671.                                       }
1672.                                       }
1673.                                       }
1674.                                       }
1675.                                       }
1676.                                       }
1677.                                       }
1678.                                       }
1679.                                       }
1680.                                       }
1681.                                       }
1682.                                       }
1683.                                       }
1684.                                       }
1685.                                       }
1686.                                       }
1687.                                       }
1688.                                       }
1689.                                       }
1690.                                       }
1691.                                       }
1692.                                       }
1693.                                       }
1694.                                       }
1695.                                       }
1696.                                       }
1697.                                       }
1698.                                       }
1699.                                       }
1700.                                       }
1701.                                       }
1702.                                       }
1703.                                       }
1704.                                       }
1705.                                       }
1706.                                       }
1707.                                       }
1708.                                       }
1709.                                       }
1710.                                       }
1711.                                       }
1712.                                       }
1713.                                       }
1714.                                       }
1715.                                       }
1716.                                       }
1717.                                       }
1718.                                       }
1719.                                       }
1720.                                       }
1721.                                       }
1722.                                       }
1723.                                       }
1724.                                       }
1725.                                       }
1726.                                       }
1727.                                       }
1728.                                       }
1729.                                       }
1730.                                       }
1731.                                       }
1732.                                       }
1733.                                       }
1734.                                       }
1735.                                       }
1736.                                       }
1737.                                       }
1738.                                       }
1739.                                       }
1740.                                       }
1741.                                       }
1742.                                       }
1743.                                       }
1744.                                       }
1745.                                       }
1746.                                       }
1747.                                       }
1748.                                       }
1749.                                       }
1750.                                       }
1751.                                       }
1752.                                       }
1753.                                       }
1754.                                       }
1755.                                       }
1756.
```

```

113.
114.                                //sign them out and present the log in page
115.                                try Auth.auth().signOut()
116.                                let loginVC =
117.                                self.storyboard?.instantiateViewController(withIdentifier:
118.                                    "LoginSignupVC") as? LoginSignupVC
119.                                self.present(loginVC!, animated: true,
120.                                    completion: nil)
121.                                //when the user logs out we need to return
122.                                //the tab bar to its original state ready for either type of user to
123.                                //log in
124.                                if let tabBarController =
125.                                self.tabBarController {
126.                                    //reset tabs
127.                                    tabBarController.viewControllers = tabs
128.                                    tabGateOpen = true
129.                                    accountGateOpen = true
130.                                    cardGateOpen = true
131.                                    feedGateOpen = true
132.                                    observeGateOpen = true
133.                                    paginationGateOpen = true
134.                                    pushNotificationGateOpen = true
135.                                    self.uid = nil
136.                                    //set defaults back to normal
137.                                    //do not know what type of user will
138.                                    //log in next
139.                                    DEFAULTS.set(nil, forKey: "gigs")
140.                                }
141.                                } catch {
142.                                    self.displayError(title: "There was an
143.                                    error", message: "Something went wrong, please try again")
144.                                }
145.                                self.present(alertController, animated: true,
146.                                    completion: nil)
147.                                }
148.                                settingsPopup.addAction(editProfileAction)
149.                                settingsPopup.addAction(logoutAction)
150.                                settingsPopup.addAction(cancelAction)
151.                                present(settingsPopup, animated: true, completion: nil)
152.                                }
153.                                var profilePic = UIImage(named: "icons8-user") //have a
154.                                //placeholder image
155.                                func updateUserData(cell: AccountHeaderCell){
156.                                    //set the navigation bar title to username
157.                                    self.navigationController?.navigationBar.topItem?.title
158.                                    = user?.name

```

```
156.
157.        //set the account header cell outlets
158.        cell.userBioTextView.text = user?.bio
159.        if user?.gigs == true {
160.            cell.userTypeLabel.text = "Looking to play"
161.        } else {
162.            cell.userTypeLabel.text = "Hiring entertainment"
163.        }
164.
165.        cell.profilePicView.image = profilePic
166.        cell.userEmailLabel.text = user?.email
167.        cell.userPhoneLabel.text = user?.phone
168.
169.        //if user hasn't got Facebook
170.        if user?.getFacebook() == "" {
171.            //put it at the right of the horizontal stack
172.
173.            cell.socialLinkStackView.insertArrangedSubview(cell.facebookLinkButton, at: 5)
174.            //disable the button
175.            cell.facebookLinkButton.isEnabled = false
176.            //and hide it
177.            cell.facebookLinkButton.alpha = 0.0
178.            //has facebook
179.        } else {
180.            //enable it
181.            cell.facebookLinkButton.isEnabled = true
182.            //show it
183.            cell.facebookLinkButton.alpha = 1.0
184.
185.            if user?.getTwitter() == "" {
186.
187.                cell.socialLinkStackView.insertArrangedSubview(cell.twitterLinkButton, at: 5)
188.                cell.twitterLinkButton.isEnabled = false
189.                cell.twitterLinkButton.alpha = 0.0
190.            } else {
191.                cell.twitterLinkButton.isEnabled = true
192.                cell.twitterLinkButton.alpha = 1.0
193.            }
194.            if user?.getInstagram() == "" {
195.
196.                cell.socialLinkStackView.insertArrangedSubview(cell.instagramLinkButton, at: 5)
197.                cell.instagramLinkButton.isEnabled = false
198.                cell.instagramLinkButton.alpha = 0.0
199.            } else {
200.                cell.instagramLinkButton.isEnabled = true
201.                cell.instagramLinkButton.alpha = 1.0
202.            }
203.            if user?.getWebsite() == "" {
204.
205.                cell.socialLinkStackView.insertArrangedSubview(cell.websiteLinkButton, at: 5)
```

```

203.            cell.websiteLinkButton.isEnabled = false
204.            cell.websiteLinkButton.alpha = 0.0
205.        } else {
206.            cell.websiteLinkButton.isEnabled = true
207.            cell.websiteLinkButton.alpha = 1.0
208.        }
209.        if user?.getAppleMusic() == "" {
210.
211.            cell.socialLinkStackView.insertArrangedSubview(cell.appleMusicLinkBu
212.                tton, at: 5)
213.                cell.appleMusicLinkButton.isEnabled = false
214.                cell.appleMusicLinkButton.alpha = 0.0
215.            } else {
216.                cell.appleMusicLinkButton.isEnabled = true
217.                cell.appleMusicLinkButton.alpha = 1.0
218.            }
219.            if user?.getSpotify() == "" {
220.
221.                cell.spotifyLinkButton.isEnabled = false
222.                cell.spotifyLinkButton.alpha = 0.0
223.            } else {
224.                cell.spotifyLinkButton.isEnabled = true
225.                cell.spotifyLinkButton.alpha = 1.0
226.            }
227.        //MARK: PORTFOLIO POST CELLS
228.        func updatepostData(cell: AccountPostCell, row: Int) {
229.            //if observing
230.            if uid != Auth.auth().currentUser?.uid {
231.                //stop user deleting the post
232.                cell.postMoreButton.isHidden = true
233.            }
234.
235.            //portfolioPosts[row] means the right post for that
236.            //cell in the table
237.            cell.postLocationLabel.text =
238.                portfolioPosts[row].location
239.
240.            //if no caption, then hide the text view
241.            if portfolioPosts[row].caption != "" {
242.                cell.postCaptionTextView.isHidden = false
243.                cell.postCaptionTextView.text =
244.                    portfolioPosts[row].caption
245.            } else {
246.                cell.postCaptionTextView.isHidden = true
247.            }
248.            //add a tag to the button so we know what cell the
249.            //button belongs to when it is tapped
250.            cell.postMoreButton.tag = row
251.
252.            //load image from cache or download
253.            if portfolioPosts[row].isImage {

```

```

250.
251.            cell.postContainerView.removePlayButton()
252.            cell.postContainerView.loadImageCache(url:
253.                portfolioPosts[row].postURL, isImage: portfolioPosts[row].isImage)
254.                //load video thumbnail from cache or download
255.            } else {
256.
257.                cell.postContainerView.loadImageCache(url:
258.                    portfolioPosts[row].thumbnailURL, isImage:
259.                        portfolioPosts[row].isImage)
260.            }
261.
262.            //to play a video when cell is tapped
263.            var playingAVPlayer: PostContainerView?
264.            override func tableView(_ tableView: UITableView,
265.                didSelectRowAt indexPath: IndexPath) {
266.
267.                if indexPath.section != 0 { //so no crash when profile
268.                    tapped
269.
270.                    let tappedCell = tableView.cellForRow(at:
271.                        indexPath) as! AccountPostCell
272.
273.                    //if it's a video and tapped
274.                    if !portfolioPosts[indexPath.section - 1].isImage {
275.
276.                        //remove the thumbnail and play the video
277.
278.                        tappedCell.postContainerView.addVideo(url:
279.                            portfolioPosts[indexPath.section - 1].postURL, fit: false)
280.
281.                        tappedCell.postContainerView.playPlayer()
282.                        playingAVPlayer = tappedCell.postContainerView
283.
284.                    }
285.
286.                }
287.
288.                //MARK: MORE (DELETION)
289.
290.                @IBAction func postMoreButton(_ sender: UIButton) {
291.                    //learn cell row from the button pressed
292.                    let row = sender.tag
293.                    //ask user if they want to delete
294.                    let morePopup = UIAlertController(title: "More",
295.                        message: "What would you like to do with your post?",
296.                        preferredStyle: .actionSheet)
297.                    let deletePostAction = UIAlertAction(title: "Delete
298.                        Post", style: .destructive) { (buttonTapped) in
299.
300.                        //delete the post from Database and from Storage
301.                        DataService.instance.deleteDBPortfolioPosts(uid:
302.                            self.user!.uid, postID: self.portfolioPosts[row].getid())

```

```

293.             DataService.instance.deleteSTFile(uid:
294.             self.user!.uid, directory: "portfolioPost", fileID:
295.             self.portfolioPosts[row].getId())
296.             if !(self.portfolioPosts[row].isImage) {
297.                 DataService.instance.deleteSTFile(uid:
298.                 self.user!.uid, directory: "portfolioThumbnail", fileID:
299.                 self.portfolioPosts[row].getId())
300.             }
301.             }
302.             let cancelPostAction = UIAlertAction(title: "Cancel",
303.             style: .cancel) { (buttonTapped) in
303.                 print("operation aborted")
304.             }
305.             }
306.             morePopup.addAction(deletePostAction)
307.             morePopup.addAction(cancelPostAction)
308.             present(morePopup, animated: true, completion: nil)
309.         }
310.         }
311.         //refresh the FCM Token for push notifications
312.         func refreshFCMToken() {
313.             if deviceFCMToken != nil && pushNotificationGateOpen ==
314.                 true {
315.                     if let uid = Auth.auth().currentUser?.uid {
316.                         DataService.instance.updateDBUserFCMToken(uid:
317.                         uid, token: deviceFCMToken!)
318.                         //so it does not update everytime this view
319.                         appears
320.                     }
321.                     pushNotificationGateOpen = false
322.                 }
323.             }
324.             //MARK: SOCIAL LINKS
325.             @IBAction func facebookLink(_ sender: Any) {
326.                 let fbPageID = user?.getFacebook()
327.                 if let appURL = URL(string:
328.                 "fb://profile/(\(fbPageID!))" {
329.                     let application = UIApplication.shared
330.                     if application.canOpenURL(appURL) {
331.                         application.open(appURL)
332.                     } else {
333.                         // if Facebook app is not installed, open URL
334.                         inside Safari
335.                         let webURL = URL(string:
336.                         "http://www.facebook.com/(\(fbPageID!))"!
337.                         if application.canOpenURL(webURL) {
338.                             application.open(webURL)
339.                         //something went wrong with the URL

```

```

337.             } else {
338.                 displayError(title: "", message: "Couldn't
339.                     open Facebook account")
340.                 }
341.             }
342.         }
343.     @IBAction func twitterLink(_ sender: Any) {
344.         //get the username
345.         let username = user?.getTwitter()
346.         //make a url from username
347.         if let appURL = URL(string:
348.             "twitter://user?screen_name=\(username!)") {
349.             let application = UIApplication.shared
350.             //if app installed...
351.             if application.canOpenURL(appURL) {
352.                 //...open in app
353.                 application.open(appURL)
354.             } else {
355.                 //if not installed, open in Safari browser
356.                 let webURL = URL(string:
357.                     "https://twitter.com/\(username!)")!
358.                 if application.canOpenURL(webURL){
359.                     application.open(webURL)
360.                     //URL doesn't work, display error
361.                 } else {
362.                     displayError(title: "", message: "Couldn't
363.                     open Twitter account")
364.                 }
365.             }
366.         @IBAction func instagramLink(_ sender: Any) {
367.             let username = user?.getInstagram()
368.             if let appURL = URL(string:
369.                 "instagram://user?username=\(username!)") {
370.                 let application = UIApplication.shared
371.                 if application.canOpenURL(appURL) {
372.                     application.open(appURL)
373.                 } else {
374.                     let webURL = URL(string:
375.                         "https://instagram.com/\(username!)")!
376.                     if application.canOpenURL(webURL){
377.                         application.open(webURL)
378.                     } else {
379.                         displayError(title: "", message: "Couldn't
380.                         open Instagram account")
381.                     }
382.                 }
383.             }
384.         @IBAction func websiteLink(_ sender: Any) {
385.             let userWebsite = user?.getWebsite()
386.             if let webURL = URL(string: userWebsite!) {

```

```

385.             let application = UIApplication.shared
386.             if application.canOpenURL(webURL) {
387.                 application.open(webURL)
388.             } else {
389.                 displayError(title: "", message: "Couldn't find
390.                     website")
391.             }
392.         } else {
393.             displayError(title: "", message: "Couldn't find
394.                     website")
395.         }
396.     @IBAction func appleMusicLink(_ sender: Any) {
397.         let userStreaming = user?.getAppleMusic()
398.         if let webURL = URL(string: userStreaming!) {
399.             let application = UIApplication.shared
400.             if application.canOpenURL(webURL) {
401.                 application.open(webURL)
402.             } else {
403.                 displayError(title: "", message: "Couldn't find
404.                     Apple Music profile")
405.             }
406.         }
407.     }
408.     @IBAction func spotifyLink(_ sender: Any) {
409.         //get spotify string url
410.         let userStreaming = user?.getSpotify()
411.         //instantiate a url object
412.         if let webURL = URL(string: userStreaming!) {
413.             let application = UIApplication.shared
414.             //if can open in browser (or app)
415.             if application.canOpenURL(webURL) {
416.                 application.open(webURL)
417.             } else {
418.                 displayError(title: "", message: "Couldn't find
419.                     Spotify profile")
420.             }
421.         } //if instantiation failed
422.     } else {
423.         displayError(title: "", message: "Couldn't find
424.                     Spotify profile")
425.     }
426. }
427. var deviceFCMToken: String?

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
observingPortfolio	Boolean	Public	To keep track of whether current user is viewing their

			own portfolio or not to display UI elements appropriately.
hideForLoad	Boolean	Public	To keep track of hiding the UI elements while refreshing the portfolio for a good user experience.
user	User	Public	The User object that corresponds to the portfolio data current user will be looking at.
uid	String	Public	User identifier to fetch portfolio data from Database.
portfolioPosts	Array<PortfolioPost>	Public	Array of posts to display iteratively in portfolio table view cells.
settingsPopup	UIAlertController	Local	The action sheet that will give user options when they click on “Settings” button.
editProfileAction logoutAction cancelAction	UIAlertAction	Local	The buttons on the settingsPopup.
loginVC	LoginSignupVC	Local	An instance of LoginSignupVC to be presented when the user logs out.
profilePic	UIImage	Public	The profile image that will be presented in the AccountHeaderCell.
playingAVPlayer	PostContainerView	Public	To keep track of whether there is a video playing or not, so that I can stop the audio when the portfolio disappears from view.
tappedCell	AccountPostCell	Local	Keep track of what cell was tapped to play a video in that cell.
row	Integer	Local	To keep track of what cell is being interacted with when the postMoreButton has been pressed.
morePopup deletePostAction cancelPostAction	UIAlertController UIAlertAction UIAlertAction	Local	Ask user if they want to delete a post before deleting it.
fbPageID username userWebsite userStreaming	String	Local	Constant holding the user's social inputs to embed in a URL.

appURL webURL	URL	Local	The URL which opens the social link in an app or Safari browser.
------------------	-----	-------	--

## PortfolioPostVC.swift

```

1. import UIKit
2. import AVKit
3. import AVFoundation
4. import FirebaseDatabase
5. import FirebaseStorage
6. import FirebaseAuth
7. import GooglePlaces
8. import GoogleMaps
9.
10.    //NSCache is used to store objects like a UIImage, NSCache requires the type to be AnyObject so you can store your cache using UIImage() if you incline to do so. Smart enough to know when to drop the images if too much memory is being used.
11.
12.    //subclass of the AutoComplete
13.    //inherits methods to present the LocationAutoCompleteVC
14.    class PortfolioPostVC: AutoComplete {
15.
16.        @IBOutlet weak var locationLabel: UILabel!
17.        @IBOutlet weak var postContainerView: PostContainerView!
18.        @IBOutlet weak var captionContentView: MyTextView!
19.        let loadingSpinner = SpinnerViewController()
20.
21.        var postData: Dictionary<String, Any>?
22.
23.        var imagePicker: UIImagePickerController?
24.
25.        var postID = ""
26.        var imageID = ""
27.        var imageAdded = false
28.        var videoAdded = false
29.        var imageContent: UIImage?
30.        var videoContent: URL?
31.        var videoThumbnail: UIImage?
32.        var thumbnailURL: URL?
33.
34.        //dimensions to keep track of how to present the post in the portfolio
35.        var dimensions: [String : CGFloat] = ["height": 0.00, "width": 0.00]
36.        func updateDimensions(image: UIImage){
37.            dimensions["height"] = image.size.height
38.            dimensions["width"] = image.size.width
39.        }
40.
41.        var height: CGFloat?
42.        var width: CGFloat?
43.

```

```
44.         var placeholder = "Write a caption... |"
45.
46.         override func viewDidLoad() {
47.             super.viewDidLoad()
48.             setupView()
49.             captionContentView.updatePlaceholder(placeholder:
50.             placeholder)
51.             captionContentView.text = placeholder
52.             captionContentView.textColor = UIColor.lightGray
53.             hideKeyboard()
54.
55.             imagePicker = UIImagePickerController()
56.             imagePicker?.delegate = self
57.         }
58.
59.         //go back to portfolio
60.         @IBAction func popView(_ sender: Any) {
61.             dismiss(animated: true, completion: nil)
62.         }
63.
64.         //MARK: STAGE 1: OPTIONAL POST LOCATION
65.
66.         //present Google Places view and return choice result
67.         @IBAction func addLocationButton(_ sender: Any) {
68.             presentAutocompleteVC()
69.             locationLabel.text = locationResult
70.         }
71.
72.
73.         //MARK: STAGE 2: POST CONTENT
74.
75.         @IBAction func chooseImage(_ sender: Any) {
76.             //allow user to pick what to post
77.             openPhotoPopup(video: true, imagePicker: imagePicker!,,
78.             title: "Post", message: "Take or choose a portfolio post")
79.         }
80.
81.         func imagePickerController(_ picker:
82.             UIImagePickerController, didFinishPickingMediaWithInfo info:
83.             [UIImagePickerController.InfoKey : Any]) {
84.
85.             //the info dictionary may contain multiple
86.             //representations of the image. Use the original.
87.             if let selectedImage = info[.originalImage] as? UIImage
88.             {
89.                 //add photo to preview
90.                 postContainerView.addPhoto(imageContent:
91.                 selectedImage, fit: true)
```

```
92.             updateDimensions(image: imageContent!)
93.             //unique ID for the post
94.             imageID = "\(UUID().uuidString).jpg"
95.         }
96.
97.         if let selectedVideo =
98.             info[UIImagePickerController.InfoKey.mediaURL] as? URL {
99.
100.            postContainerView.addVideo(url: selectedVideo, fit:
101.                true)
102.            postContainerView.playPlayer()
103.            videoContent = selectedVideo
104.            videoAdded = true
105.            imageAdded = false
106.
107.            let thumbnail = generateThumbnail(url:
108.                videoContent!)
109.            updateDimensions(image: thumbnail)
110.            imageID = "\(UUID().uuidString).mov"
111.        }
112.
113.        //dismiss ImagePicker Controller
114.        dismiss(animated: true, completion: nil)
115.    }
116.    //upload the post to Storage
117.    func contentUpload(uid: String, handler: @escaping (_
118.        returnedURLs: [URL]) ->()){
119.        //for video, there will be two urls (thumbnail and
120.        //video itself)
121.        var urls = [URL]()
122.        if imageAdded {
123.            if let postImage = imageContent {
124.                //add pic to Storage
125.                DataService.instance.updateSTPic(uid: uid,
126.                    directory: "portfolioPost", imageContent: postImage, imageID:
127.                        imageID, uploadComplete: { (success, error) in
128.                            if error != nil {
129.                                self.removeSpinnerView(self.loadingSpinner)
130.                                self.displayError(title: "There was an
131.                                    Error", message: error!.localizedDescription)
132.                            } else {
133.                                //get the url
134.                                DataService.instance.getSTURL(uid: uid,
135.                                    directory: "portfolioPost", imageID: self.imageID) { (returnedURL)
136.                                        in
137.                                            urls.append(returnedURL)
138.                                            //return the url
139.                                       
140.                                    }
141.                                }
142.                            }
143.                        }
144.                    }
145.                }
146.            }
147.        }
148.    }
149.    //return the url
150.    urls
151.}
```

```

136.                               handler(urls)
137.                           }
138.                           }
139.                           }
140.                           }
141.                           }
142.                           }
143.                           //if video posting
144.                           if videoAdded {
145.                               if let postVideo = videoContent {
146.                                   //add the video to Storage
147.                                   DataService.instance.updateSTVid(uid: uid,
148.                                       directory: "portfolioPost", vidContent: postVideo, imageID: imageID,
149.                                       uploadComplete: { (success, error) in
150.                                           if error != nil {
151.                                               self.removeSpinnerView(self.loadingSpinner)
152.                                               self.displayError(title: "There was an
153.                                                   Error", message: error!.localizedDescription)
154.                                           } else {
155.                                               //get the url
156.                                               DataService.instance.getSTURL(uid: uid,
157.                                   directory: "portfolioPost", imageID: self.imageID) { (returnedURL)
158.                                       in
159.                                           //and upload the thumbnail as well
160.                                           self.thumbnailUpload(uid: uid, url:
161.                                               postVideo) { (returnedThumbnailURL) in
162.                                                   //add both the video and
163.                                                   //thumbnail url to array
164.                                                   urls.append(returnedURL)
165.                                               }
166.                                               }
167.                                               }
168.                                               }
169.                                           }
170.                                       }
171.                                       func thumbnailUpload(uid: String, url: URL, handler:
172.                                         @escaping (_ returnedThumbnailURL: URL) -> ()) {
173.                                         //generate a thumbnail from the video
174.                                         self.videoThumbnail = generateThumbnail(url: url)
175.                                         //upload the thumbnail
176.                                         DataService.instance.updateSTPic(uid: uid, directory:
177.                                         "portfolioThumbnail", imageContent: videoThumbnail!, imageID:
178.                                         imageID, uploadComplete: { (success, error) in
179.                                             if error != nil {
180.                                                 self.removeSpinnerView(self.loadingSpinner)

```

```

179.                     self.displayError(title: "There was an Error",
180.                         message: error!.localizedDescription)
181.                     } else {
182.                         DataService.instance.getSTURL(uid: uid,
183.                         directory: "portfolioThumbnail", imageID: self.imageID) {
184.                             (returnedURL) in
185.                                 }
186.                         }
187.                     }
188.
189.                     //MARK: STAGE 3: UPLOAD POST
190.
191. @IBAction func postComplete(_ sender: Any) {
192.
193.     if imageAdded || videoAdded {
194.
195.         //postID needed for deletion
196.         postID = imageID
197.
198.         let range = postID.index(postID.endIndex, offsetBy:
199.             -4)..

```

```

224.                                     //set the post URL (first index of returned
225.                                     array
226.                                     self.postData!["postURL"] =
227.                                     returnedURLs[0].absoluteString
228.                                     //if there is two in array (will be video)
229.                                     if returnedURLs.count == 2 {
230.                                         //add the vid thumbnail if there is one
231.                                         self.postData!["thumbnailURL"] =
232.                                         returnedURLs[1].absoluteString
233.                                     }
234.                                     }
235.                                     //dismiss view
236.                                     self.dismiss(animated: true, completion:
237.                                         nil)
238.                                     //and refresh portfolio to show new post
239.                                     NotificationCenter.default.post(name:
240.                                     NSNotification.Name(rawValue: "refreshPortfolio"), object: nil)
241.                                     }
242.                                     } else {
243.                                     }
244.                                     //user not added a photo or video
245.                                     displayError(title: "Oops", message: "Please add a
246.                                     photo or video to post")
247.                                     }
248.                                     }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
postData	Dictionary<String, Any>	Public	Dictionary to store input data about the post and upload to Database.
postID	String	Public	The unique identifier of the post object so that it can be deleted later on.
imageID	String	Public	Equal to the postID, the unique identifier of which the post data is stored under in Storage.
imageAdded videoAdded	Boolean	Public	To keep track of what type of content has been added to be posted.
imageContent videoContent thumbnailURL	UIImage URL URL	Public	The image and video that will be converted to data for Storage upload.

dimensions	Dictionary<String, CGFloat>	Public	Dictionary added to postData to help with scaling down the image to be displayed later.
thumbnail	UIImage	Local	UIImage returned from grabbing the first frame of the video to be posted.
urls	Array<URL>	Local	Array to put Storage download URLs in after uploading them to Storage.
postImage	UIImage	Local	The image to be posted.
postVideo	URL	Local	The video URL to be posted.
caption	String	Local	The input from captionContentView to display in the portfolio
location	String	Local	The input from locationLabel to display in the portfolio.

## CreateGigVC.swift

```

1. import UIKit
2.
3. class CreateGigVC: UIViewController {
4.
5.     @IBOutlet weak var descriptionStack: UIStackView!
6.
7.     override func viewDidLoad() {
8.         super.viewDidLoad()
9.         setupView()
10.    }
11.    //Is in a navigation stack, hide bar as this is the root
12.    override func viewDidAppear(_ animated: Bool) {
13.        self.navigationController?.navigationBar.isHidden =
14.            true
15.    }
16.    override func viewDidAppear(_ animated: Bool) {
17.        //Bug after adding photo of the event and going back
18.        self.navigationController?.navigationBar.isHidden =
19.            true
20.        //event data dictionary for Database
21.        let eventData = ["uid": "", "eventID": "", "title": "",
22. "timestamp": "", "latitude": 0.0, "longitude": 0.0,
23. "locationName": "", "postcode": "", "payment": 0.0, "description": "",
24. "name": "", "email": "", "phone": "", "eventPhotoURL": "",
25. "appliedUsers": [String: Bool].self] as [String : Any]
26.        @IBAction func continueButton(_ sender: Any) {
27.            performSegue(withIdentifier: TO_TITLE_DATE, sender:
28. nil)
29.        }
30.        //take the dictionary and fill it from the forms and inputs

```

```
27.         override func prepare(for segue: UIStoryboardSegue, sender:  
Any?) {  
28.             if segue.identifier == TO_TITLE_DATE {  
29.                 let titleDateCGVC = segue.destination as!  
TitleDateCGVC  
30.                 titleDateCGVC.eventData = self.eventData  
31.             }  
32.         }  
33.     }
```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
eventData	Dictionary<String, Any>	Public	Dictionary to store input data about the event and upload to Database.
titleDateCGVC	TitleDateCGVC	Local	Instance of TitleDateCGVC to change the value of its variables from CreateGigVC before a segue.

## TitleDateCGVC.swift

```

31.        }
32.        //so user can pick, date/day and time from the picker
33.        dateFormatter.dateFormat = "yyyy-MM-dd HH:mm:ss +zzzz"
34.
35.        //set restrictions on the Date Picker
36.        let calendar = Calendar(identifier: .gregorian)
37.        let currentDate = Date()
38.        var components = DateComponents()
39.        components.calendar = calendar
40.
41.        //can only choose a date up to 10 years in advance
42.        components.year = 10
43.        components.month = 0
44.        let maxDate = calendar.date(byAdding: components, to:
45.            currentDate)!
46.
47.        //cannot choose date or time before current time
48.        components.year = 0
49.        let minDate = calendar.date(byAdding: components, to:
50.            currentDate)!
51.
52.        //assign these restrictions to the picker
53.        datePicker.minimumDate = minDate
54.        datePicker.maximumDate = maxDate
55.    }
56.    //hidden navigation bar is inherited, show it this time
57.    override func viewWillAppears(_ animated: Bool) {
58.        self.navigationController?.navigationBar.isHidden =
59.            false
60.    }
61.    override func viewDidAppear(_ animated: Bool) {
62.        //we are creating not editing, this causes crash if
63.        //user starts editing, then decides to create by clicking on tab
64.        if self.tabBarController?.selectedIndex == 0 {
65.            editingGigEvent = false
66.        }
67.        //if editing the GigEvent object
68.        if editingGigEvent && editingGate {
69.            //get all the data about it...
70.            if let uid = Auth.auth().currentUser?.uid {
71.                DataService.instance.getDBSingleEvent(uid: uid,
72.                eventID: editEventID) { (returnedGigEvent, success) in
73.                    //...put it in a dictionary...
74.                    let returnedGigEventData = ["uid": uid,
75.                    "eventID": returnedGigEvent.getid(), "title":
76.                    returnedGigEvent.getTitle(), "timestamp":
77.                    returnedGigEvent.getTimestamp(), "latitude":
78.                    returnedGigEvent.getLatitude(), "longitude":
79.                    returnedGigEvent.getLongitude(), "locationName":
80.                    returnedGigEvent.getLocationName(), "postcode":
81.                    returnedGigEvent.getPostcode(), "payment":
82.                    returnedGigEvent.getPayment(), "description":
83.                    returnedGigEvent.getDescription(), "name":
84.                    returnedGigEvent.getName(), "email": returnedGigEvent.getEmail(),
85.                    "phone": returnedGigEvent.getPhone(), "eventPhotoURL":
```

```

        returnedGigEvent.getEventPhotoURL(), "appliedUsers":
        returnedGigEvent.getAppliedUsers()) as [String : Any]
70.                                self.eventData = returnedGigEventData
71.                                self.gigEvent = returnedGigEvent
72.                                //...auto fill the UI inputs
73.                                self.eventTitleField.text =
        returnedGigEvent.getTitle()
74.                                //remove an hour from time (is an hour
        ahead)
75.                                let date =
        returnedGigEvent.getDate().addingTimeInterval(-3600)
76.                                //set the date to be edited
77.                                self.datePicker.setDate(date, animated:
        false)
78.                                //so doesn't auto-fill again
79.                                self.editingGate = false
80.                            }
81.                        }
82.                    }
83.                }
84.
85.                @IBAction func continueButton(_ sender: Any) {
86.
87.                    //let eventID = NSUUID().uuidString
88.                    //will make the eventID the same as the imageID
89.
90.                    //raw value from picker - Hour behind, add an hour
        (3600s)
91.                    //NOT NEEDED ANYMORE?
92.                    //let chosenTime =
        datePicker.date.addingTimeInterval(3600)
93.                    //string for Database
94.                    let timestamp = "\(datePicker.date)"
95.
96.                    if let eventTitle = eventTitleField.text{
97.                        //check valid
98.                        if eventTitle != "" && eventTitle.count <= 60 {
99.                            //add all the inputs to dictionary
100.                           self.eventData!["uid"] = self.user?.uid as Any
101.                           self.eventData!["title"] = eventTitle
102.                           self.eventData!["timestamp"] = timestamp
103.
104.                           performSegue(withIdentifier:
        TO_LOCATION_PRICING, sender: nil)
105.
106.                    } else {
107.                        // user not written a title
108.                        displayError(title: "Add a title", message:
        "Please add the title of your event to continue")
109.                    }
110.                }
111.            }
112.
113.            //take the eventDate through every view controller as user
        progresses

```

```

114.         override func prepare(for segue: UIStoryboardSegue, sender:
115.             Any?) {
116.             if segue.identifier == TO_LOCATION_PRICING {
117.                 //need this line to pass information between view
118.                 controllers
119.                 let locationPriceCGVC = segue.destination as!
120.                     LocationPriceCGVC
121.                     //changes it
122.                     locationPriceCGVC.user = user
123.                     locationPriceCGVC.eventData = eventData
124.                     locationPriceCGVC.gigEvent = gigEvent
125.                     locationPriceCGVC.editingGate = true
126.
127.             }
128.         }
129.     }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
user	User	Public	Instantiated object after getting user profile data to help autofill information for event editing.
editingGate	Boolean	Public	So that data is not auto filled out again when user returns back to this view but one time only.
editEventID	String	Public	To get current event data with from Database so that it can be edited.
gigEvent	GigEvent	Public	Instantiated object from fetched data about the event to be edited.
dateFormatter	Formatter	Public	To convert string representations to NSDate objects and vice-versa.
calendar currentDate components	Calendar Date DateComponents	Local	So that I can set date restrictions on the date picker.
maxDate minDate	Date Date	Local	Maximum date on picker is 10 years in advance, minimum date is the current date as error prevention.
returnedGigEventData	Dictionary<String, Any>	Local	Event data fetched from Database which will autofill the data entry fields for editing the event.

timestamp	String	Local	String value of the date picker input to store in Database.
eventTitle	String	Local	Input from eventTitleField.
locationPriceCGVC	LocationPriceCGVC	Local	Instance of LocationPriceCGVC to change the value of its variables from TitleDateCGVC before a segue.

## LocationPriceCGVC.swift

```

1. import UIKit
2. import CoreLocation
3.
4. //is subclass of AutoComplete to inherit function to present Google
Places view
5. class LocationPriceCGVC: AutoComplete, CLLocationManagerDelegate {
6.
7.     @IBOutlet weak var confirmationImageView: UIImageView!
8.     @IBOutlet weak var locationNameField: MyTextField!
9.     @IBOutlet weak var postcodeField: MyTextField!
10.    @IBOutlet weak var paymentField: MyTextField!
11.
12.    //editing
13.    var editingGate = true
14.    var gigEvent: GigEvent?
15.
16.    var user: User?
17.    var eventData: Dictionary<String, Any>?
18.    let locationManager: CLLocationManager = {
19.        let lm = CLLocationManager()
20.        //to nearest hundred metres to save battery
21.        lm.desiredAccuracy = kCLLocationAccuracyHundredMeters
22.        lm.requestWhenInUseAuthorization()
23.        return lm
24.    }()
25.
26.    override func viewDidLoad() {
27.        super.viewDidLoad()
28.        setupView()
29.        hideKeyboard()
30.        //input restrictions
31.        postcodeField.updateCharacterLimit(limit: 8)
32.        locationNameField.updateCharacterLimit(limit: 64)
33.    }
34.    override func viewDidDisappear(_ animated: Bool) {
35.        //stop updating when view disappears to conserve
battery life
36.        locationManager.stopUpdatingLocation()
37.    }
38.    override func viewDidAppear(_ animated: Bool) {

```

```
39.          //if editing
40.          if editingGate && editingGigEvent && gigEvent != nil {
41.              //if the user chose to use the location services
42.              if !(gigEvent?.getLongitude() == 0.00 &&
43.                  gigEvent?.getLatitude() == 0.00) {
44.                  //continue to use them
45.                  useCurrentLocation(true)
46.              }
47.              //auto fill from the GigEvent object
48.              locationNameField.text =
49.                  gigEvent?.getLocationName()
50.              postcodeField.text = gigEvent?.getPostcode()
51.              paymentField.text = String(gigEvent!.getPayment())
52.          }
53.      }
54.
55. //MARK: GET LOCATION
56. var currentLocationOn = false
57. @IBAction func useCurrentLocation(_ sender: Any) {
58.     //start updating location
59.     if currentLocationOn == false {
60.         currentLocationOn = true
61.         locationManager.delegate = self
62.         locationManager.startUpdatingLocation()
63.         //show it's being used
64.         confirmationImageView.alpha = 1.0
65.         confirmationImageView.isHidden = false
66.         //stop updating the location
67.     } else {
68.         currentLocationOn = false
69.         locationManager.stopUpdatingLocation()
70.         //set coordinates back to zero if they
71.         //don't want to use location
72.         eventLatitude = 0.00
73.         eventLongitude = 0.00
74.         //show it's not being used
75.         confirmationImageView.alpha = 0.3
76.     }
77. }
78.
79. var eventLatitude = 0.00
80. var eventLongitude = 0.00
81. func locationManager(_ manager: CLLocationManager,
82. didUpdateLocations locations: [CLLocation]) {
83.     //grab the most up to date coordinates (first in array)
84.     let userLocation: CLLocation = locations[0]
85.     eventLatitude = userLocation.coordinate.latitude
86.     eventLongitude = userLocation.coordinate.longitude
87.
88.     //info about user location usage
89.     @IBAction func locationInfoButton(_ sender: Any) {
```

```
90.            displayError(title: "Using Current Location", message:
91.                "will make your event appear first to applicants nearby")
92.
93.            //search and return location String with Google Places API
94.            @IBAction func searchLocationName(_ sender: Any) {
95.                presentAutocompleteVC()
96.                locationNameField.text = locationResult
97.            }
98.
99.            @IBAction func continueButton(_ sender: Any) {
100.
101.                if let locationName = locationNameField.text {
102.                    if let postcode = postcodeField.text {
103.                        if let strPayment = paymentField.text {
104.                            //validation
105.                            if locationName.count > 2 {
106.                                if (postcode.count == 7 ||
107.                                    postcode.count == 8) {
108.                                        //check payment can be made into a
109.                                        //Double data type
110.                                        if let payment = Double(strPayment)
111.                                            //add inputs to dictionary
112.                                            self.eventData!["latitude"] =
113.                                            self.eventData!["longitude"] =
114.                                            self.eventData!["locationName"] =
115.                                            self.eventData!["postcode"] =
116.                                            self.eventData!["payment"] =
117.                                            performSegue(withIdentifier:
118.                                                TO_INFO_CONTACT, sender: nil)
119.                                            } else {
120.                                                //cannot be converted to Double
121.                                                displayError(title: "Payment",
122.                                                message: "Please enter the chosen amount in a suitable format")
123.                                            }
124.                                            } else {
125.                                                displayError(title: "Postcode",
126.                                                message: "Please enter the correct 7 character postcode of the
127.                                                event")
128.                                            }
129.                                            } else {
130.                                                displayError(title: "Location",
131.                                                message: "Please search for or type in a location")
```

```

131.                     }
132.                 }
133.             }
134.         }
135.     }
136.
137.     override func prepare(for segue: UIStoryboardSegue, sender:
138.     Any?) {
139.         if segue.identifier == T0_INFO_CONTACT {
140.
141.             let infoContactCGVC = segue.destination as!
142.             InfoContactCGVC
143.             infoContactCGVC.user = user
144.             infoContactCGVC.eventData = eventData
145.             infoContactCGVC.gigEvent = gigEvent
146.             infoContactCGVC.editingGate = true
147.         }
148.     }
149. }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
locationManager	CLLocationManager	Public	To get the device's location with so that it can be attached to the event at creation.
currentLocationOn	Boolean	Public	To keep track of whether user has asked to use location services or not.
eventLatitude eventLongitude	Double	Public	The coordinates of the user's current location to add to eventData dictionary.
userLocation	CLLocation	Local	The most recent location update from the locationManager.
locationName postcode strPayment	String	Local	The input from the UITextField outlets.
payment	Double	Local	Checking strPayment can be converted to a double data type as a validation rule.
infoContactCGVC	InfoContactCGVC	Local	Instance of InfoContactCGVC to change the value of its variables from LocationPriceCGVC before a segue.

### InfoContactCGVC.swift

1. **import** UIKit
- 2.

```

3. class InfoContactCGVC: UIViewController {
4.
5.     @IBOutlet weak var descriptionTextView: MyTextView!
6.     @IBOutlet weak var nameTextField: MyTextField!
7.     @IBOutlet weak var emailTextField: MyTextField!
8.     @IBOutlet weak var phoneTextField: MyTextField!
9.
10.    var editingGate = true
11.    var gigEvent: GigEvent?
12.    var user: User?
13.    var eventData: Dictionary<String, Any>?
14.
15.    var placeholder = "''''"
16.    Write a description...
17.    Things to think about:
18.    •The style of music wanted
19.    •How big the event will be
20.    •The length of their performance
21.    •Any equipment you cannot supply
22.    '''''"
23.
24.    override func viewDidLoad() {
25.        super.viewDidLoad()
26.        setupView()
27.        hideKeyboard()
28.
29.        //setup placeholder of the description text view
30.        descriptionTextView.updatePlaceholder(placeholder:
31.            placeholder)
32.        descriptionTextView.text = placeholder
33.        descriptionTextView.textColor = UIColor.lightGray
34.        //set restrictions on contact inputs
35.        nameTextField.updateCharacterLimit(limit: 50)
36.        emailTextField.updateCharacterLimit(limit: 62)
37.        phoneTextField.updateCharacterLimit(limit: 16)
38.
39.        //auto-filled
40.        nameTextField.text = user?.name
41.        emailTextField.text = user?.email
42.        phoneTextField.text = user?.phone
43.    }
44.    override func viewDidAppear(_ animated: Bool) {
45.        //if editing, auto fill the description
46.        if editingGigEvent && editingGate && gigEvent != nil {
47.            descriptionTextView.text =
48.                gigEvent?.getDescription()
49.            editingGate = false
50.        }
51.        //separate function because this is an optional method of
52.        func checkPhoneField() {
53.            //less checks needed as number pad keyboard is used
54.            if let phone = phoneTextField.text {

```

```

55.             if phone.count > 7 {
56.                 eventData!["phone"] = phone
57.             }
58.         }
59.     }
60.
61.
62.     @IBAction func continueButton(_ sender: Any) {
63.         if let description = descriptionTextView.text {
64.             if let name = nameTextField.text {
65.                 if let email = emailTextField.text {
66.                     if let phone = phoneTextField.text {
67.                         //validation checks
68.                         if name != "" {
69.                             if description.count > 10 &&
70.                                 !(description.contains("Write a description... |")) {
71.                                     //add to inputs to dictionary
72.                                     eventData!["name"] = name
73.                                     eventData!["description"] =
74.                                         description
75.                                         //add email...
76.                                         if email.contains("@") &&
77.                                         email.contains(".") && email.count >= 5 {
78.                                             eventData!["email"] = email
79.                                         //...and phone
80.                                         checkPhoneField()
81.
82.
83.             performSegue(withIdentifier: TO_ADD_PHOTO, sender: nil)
84.
85.             //just add phone
86.             } else if phone.count >= 7 {
87.
88.                 eventData!["phone"] = phone
89.
90.
91.             performSegue(withIdentifier: TO_ADD_PHOTO, sender: nil)
92.             } else {
93.
94.                 displayError(title:
95. "Contact Information", message: "please enter a valid email address
96. or phone number")
97.             }
98.             } else {
99.                 displayError(title: "Event
100. Description", message: "please enter a description of your event
outlining the suggested points")
101.             }
102.         } else {

```

```

101.                                     displayError(title: "Name",
  message: "please enter your name")
102.                                 }
103.                               }
104.                             }
105.                           }
106.                         }
107.                       }
108.
109.         override func prepare(for segue: UIStoryboardSegue, sender:
  Any?) {
110.
111.           if segue.identifier == T0_ADD_PHOTO {
112.
113.             let photoCGVC = segue.destination as! PhotoCGVC
114.
115.             photoCGVC.user = user
116.             photoCGVC.eventData = eventData
117.             photoCGVC.gigEvent = gigEvent
118.             photoCGVC.editingGate = true
119.           }
120.         }
121.       }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
phone name email	String	Public	The input from UITextField outlets.
description	String	Public	The input from descriptionTextView.
photoCGVC	PhotoCGVC	Public	Instance of PhotoCGVC to change the value of its variables from InfoContactCGVC before a segue.

### PhotoCGVC.swift

```

1. import UIKit
2.
3. class PhotoCGVC: UIViewController {
4.
5.   @IBOutlet weak var eventPicView: UIImageView!
6.   let loadingSpinner = SpinnerViewController()
7.
8.   var editingGate = true
9.   var gigEvent: GigEvent?
10.   var user: User?
11.   var eventData: Dictionary<String, Any>?
12.   var notificationData: Dictionary<String, Any>?
13.
14.   var imagePicker: UIImagePickerController?

```

```
15.
16.        var eventID = ""
17.        var imageID = ""
18.        var imageAdded = false
19.        var imageContent: UIImage?
20.
21.        override func viewDidLoad() {
22.            super.viewDidLoad()
23.            setupView()
24.
25.            imagePicker = UIImagePickerController()
26.            imagePicker?.delegate = self
27.
28.        }
29.
30.        override func viewDidAppear(_ animated: Bool) {
31.            //if editing
32.            if editingGigEvent && editingGate && gigEvent != nil {
33.                //get the event image id
34.                imageID = "\(gigEvent!.getid()).jpg"
35.                //download the image and put it in the view to
36.                //preview
37.                downloadImage(url: gigEvent!.getEventPhotoURL()) {
38.                    (returnedImage) in
39.                        self.eventPicView.image = returnedImage
40.                        self.imageAdded = true
41.                }
42.            }
43.        }
44.
45.        @IBAction func chooseImage(_ sender: Any) {
46.            //allow user to choose an image (camera or library)
47.            openPhotoPopup(video: false, imagePicker: imagePicker!, title: "Event", message: "Take or choose a photo of the event venue")
48.        }
49.
50.        func imagePickerController(_ picker:
51.            UIImagePickerController, didFinishPickingMediaWithInfo info:
52.            [UIImagePickerController.InfoKey : Any]) {
53.
54.            if let selectedImage = info[.originalImage] as? UIImage
55.            {
56.                eventPicView.image = selectedImage
57.                //returned image
58.                imageContent = selectedImage
59.                //set as one added
60.                imageAdded = true
61.                //give the image a new ID if creating event
62.                if !editingGigEvent {
```

```

63.                     imageID = "\(\NSUUID().uuidString).jpg"
64.                 }
65.             }
66.             //dismiss the image picker
67.             dismiss(animated: true, completion: nil)
68.         }
69.
70.         func picUpload(uid: String, handler: @escaping (_ url: URL)
-> ())
71.
72.             //delete the old event photo from Storage if editing
73.             if editingGigEvent && gigEvent != nil {
74.                 DataService.instance.deleteSTFile(uid: user!.uid,
75.                     directory: "events", fileID: gigEvent!.getid()) //efficient Storage
76.                     usage
77.             }
78.             //upload the photo
79.             if let eventPic = eventPicView.image {
80.
81.                 DataService.instance.updateSTPic(uid: uid,
82.                     directory: "events", imageContent: eventPic, imageID: imageID,
83.                     uploadComplete: { (success, error) in
84.                         if error != nil {
85.                             //allow user to interact with screen again
86.                             if error
87.                                 self.removeSpinnerView(self.loadingSpinner)
88.                                 self.displayError(title: "There was an
89.                         Error", message: error!.localizedDescription)
90.
91.                         } else {
92.                             //get the download URL
93.                             DataService.instance.getSTURL(uid: uid,
94.                                 directory: "events", imageID: self.imageID) { (returnedURL) in
95.                                     //return it
96.                                     handler(returnedURL)
97.                                 }
98.                             }
99.                         }
100.                     }
101.                 let range = imageID.index(imageID.endIndex,
102.                     offsetBy: -4)..<imageID.endIndex
103.                     imageID.removeSubrange(range)//'remove the .jpg
104.                     from the imageID
105.
106.                     //eventID needed for deletion
107.                     eventID = imageID
108.                     //add id to dictionary
109.                     eventData!["eventID"] = eventID

```

```

107.          //stop user interaction with screen (show
108.          uploading)           createSpinnerView(loadingSpinner)
109.          //upload the picture
110.          self.picUpload(uid: user!.uid) {
111.              (returnedURL) in
112.                  //add photo url to dictionary
113.                  self.eventData!["eventPhotoURL"] =
114.                      returnedURL.absoluteString
115.                      //start a dictionary to keep track of musicians
116.                      applied to the event
117.                      self.eventData!["appliedUsers"] =
118.                          ["CREATOR:\(self.user!.uid)": true]
119.                      //create an event object in Database
120.                      DataService.instance.updateDBEvents(uid:
121.                          self.user!.uid, eventID: self.eventID, eventData: self.eventData!)
122.                      //add the event under user to the database
123.                      //no need to update if the user is editing,
124.                      //otherwise we get a duplicate
125.                      if !editingGigEvent {
126.                          DataService.instance.updateDBUserEvents(uid: self.user!.uid,
127.                              eventID: self.eventID)
128.                      }
129.                      //update the activity feed in a completion
130.                      // handler so it updates correctly
131.                      DataService.instance.updateDBActivityFeed(uid:
132.                          self.notificationData!["reciever"] as! String, notificationID:
133.                          self.notificationData!["notificationID"] as! String,
134.                          notificationData: self.notificationData!) { (complete) in
135.                              if complete {
136.                                  self.removeSpinnerView(self.loadingSpinner)
137.                                  //take user to Activity tab to see
138.                                  // their posted event
139.                                  //without completion handler we
140.                                  // were jumping to the view controller before the activity had updated
141.                                  self.tabBarController?.selectedIndex = 2
142.                                  //clear the event creation and pop
143.                                  // to root of the navigation stack
144.                                  self.navigationController?.popToRootViewControllerAnimated(animated: true)
145.                              }
146.                          }
147.                      }
148.                  }
149.              }
150.          }
151.      }
152.  }
153. }

```

```

143.                                //if editing only
144.                                self.removeSpinnerView(self.loadingSpinner)
145.                                //clear the event creation and pop to root
   of the navigation stack (which is the EventDescriptionVC this time)
146.                                self.navigationController?.popToRootViewController(animated: true)
147.                                //done editing
148.                                editingGigEvent = false
149.                                //show the new event features in the
   activity feed
150.                                NotificationCenter.default.post(name:
   NSNotification.Name(rawValue: "refreshAllActivity"), object: nil)
151.                                }
152.                                }
153.                            } else {
154.                                displayError(title: "Oops", message: "Please take
   or add a photo of the venue to post event")
155.                                }
156.                            }
157.                        }
158.
159.                        //create dictionary for activity notification
160.                        func updateActivity() {
161.                            let notificationID = NSUUID().uuidString
162.                            let senderUid = user!.uid
163.                            //receiver is also the sender for a personal
   notification
164.                            let receiverUid = senderUid
165.                            let senderName = "You"
166.                            //users profile picture
167.                            let notificationPicURL = user!.picURL.absoluteString
168.                            let notificationDescription = "Created the event:
   \((eventData!["title"])!)"
169.                            //timestamp for quicksort
170.                            let timestamp = NSDate().timeIntervalSince1970
171.                            notificationData = ["notificationID": notificationID,
   "relatedEventID": eventID, "type": "personal", "sender": senderUid,
   "receiver": receiverUid, "senderName": senderName, "picURL":
   notificationPicURL, "description": notificationDescription,
   "timestamp": timestamp]
172.                        }
173.                    }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
notificationData	Dictionary<String, Any>	Public	Dictionary to store data about a notification and upload to Database.
eventID	String	Public	The unique identifier of the event object so it can be fetched from Database again.
imageAdded	Boolean	Public	To keep track of whether user has added an event image or not.

eventPic	UIImage	Local	The UIImage displayed in eventPicView to be uploaded to Storage.
notificationID	String	Local	Unique identifier of the notification to fetch and delete it from Database.
senderUid	String	Local	The unique identifier of user that is sending the notification, to display their profile data when requested.
receiverUid	String	Local	The unique identifier of user that will receive the notification so that it is stored in Database at the correct location.
senderName	String	Local	The name that will be displayed on the notification cell.
notificationPicURL	String	Local	Download URL of profile image to be displayed on notification cell.
notificationDescription	String	Local	Notification content to be displayed on the cell.
timestamp	TimeInterval	Local	To quicksort the notifications chronologically.

## FindGigVC.swift

```

1. import UIKit
2. import CoreLocation
3. import FirebaseAuth
4. import FirebaseStorage
5. import FirebaseDatabase
6.
7. class FindGigVC: UIViewController, CLLocationManagerDelegate {
8.
9.     @IBOutlet weak var contactStack: UIStackView!
10.    @IBOutlet weak var nameLabel: UILabel!
11.    @IBOutlet weak var emailLabel: UILabel!
12.    @IBOutlet weak var phoneLabel: UILabel!
13.    @IBOutlet weak var currentGigEventView: GigEventView!
14.    @IBOutlet weak var nextGigEventView: GigEventView!
15.    @IBOutlet weak var refreshButton: UIButton!
16.
17.
18.    var user: User?
19.    var gigEvents = [GigEvent]()
20.    var notificationData: Dictionary<String, Any>?
21.    //create a location manager for sorting events
22.    let locationManager: CLLocationManager = {
23.        let lm = CLLocationManager()
24.        lm.desiredAccuracy = kCLLocationAccuracyBest

```

```
25.             lm.requestWhenInUseAuthorization()
26.             return lm
27.         }()
28.
29.         override func viewDidLoad() {
30.             super.viewDidLoad()
31.             navigationController?.navigationBar.isHidden = true
32.             cardGateOpen = false
33.
34.             //function is called when gesture is recognised
35.             let dragGesture = UIPanGestureRecognizer(target: self,
36.             action: #selector(self.gigEventWasDragged(gestureRecogniser:)))
37.
38.             //assign the drag gesture to the view
39.             currentGigEventView.isUserInteractionEnabled = true
40.             currentGigEventView.addGestureRecognizer(dragGesture)
41.             //front card is in the centre of the view
42.             currentGigEventView.center = CGPoint(x:
43.             self.view.frame.width / 2, y: (self.view.frame.height / 2) + 60)
44.             self.view.sendSubviewToBack(currentGigEventView)
45.             //back card will be slightly above the front card
46.             // (looks like stack)
47.             nextGigEventView.center = CGPoint(x:
48.             self.view.bounds.width / 2, y: (self.view.bounds.height / 2) + 30)
49.             nextGigEventView.alpha = 0.6
50.             self.view.sendSubviewToBack(nextGigEventView)
51.             //refresh button for when there are no more cards
52.             closeButton.center = CGPoint(x:
53.             self.view.bounds.width / 2, y: self.view.bounds.height / 2)
54.             self.view.sendSubviewToBack(closeButton)
55.             //hide everything before refreshing
56.             nameLabel.isHidden = true
57.             emailLabel.isHidden = true
58.             phoneLabel.isHidden = true
59.             nextGigEventView.isHidden = true
60.             currentGigEventView.isHidden = true
61.
62.             refresh()
63.
64.             setupView()
65.         }
66.         override func viewDidAppear(_ animated: Bool) {
67.             navigationController?.navigationBar.isHidden = true
68.         }
69.         override func viewDidDisappear(_ animated: Bool) {
70.             //so it doesn't refresh everytime view appears (only
71.             once)
72.             if cardGateOpen {
73.                 cardGateOpen = false
74.                 refresh()
75.             }
76.         }
77.         override func viewDidDisappear(_ animated: Bool) {
78.             //stop using location to save battery
79.             locationManager.stopUpdatingLocation()
80.         }
81.     }
82. 
```

```

74.        }
75.        @IBAction func refreshButton(_ sender: Any) {
76.            refresh()
77.        }
78.        func refresh() {
79.            //get the current user profile
80.            if let uid = Auth.auth().currentUser?.uid {
81.                DataService.instance.getDBUserProfile(uid: uid) {
82.                    (returnedUser) in
83.                        self.user = returnedUser
84.                        //and start updating location if they have
85.                        //authorised it
86.                        self.locationManager.delegate = self
87.                        self.locationManager.startUpdatingLocation()
88.                        //get all the GigEvents in an array...
89.                        DataService.instance.getDBEvents(uid: uid) {
90.                            (returnedGigEvents) in
91.                                //...and sort them by locality
92.                                self.gigEvents =
93.                                self.setGigEventDistances(gigs: returnedGigEvents)
94.                                //show the cards to musician
95.                                self.updateCards()
96.                            }
97.                        }
98.                    }
99.                    }
100.                    }
101.                    func locationManager(_ manager: CLLocationManager,
102.                        didUpdateLocations locations: [CLLocation]) {
103.                            //grab the coordinates of the current user
104.                            let userLocation: CLLocation = locations[0]
105.                            userLatitude = userLocation.coordinate.latitude
106.                            userLongitude = userLocation.coordinate.longitude
107.                        }
108.                        //compare the two coordinates of the gig and the user
109.                        //update the distance attribute
110.                        //then use the distance to do a quick sort
111.                        func setGigEventDistances(gigs: [GigEvent]) -> [GigEvent] {
112.                            //instantiate a location out of coordinates
113.                            let userLocation = CLLocation(latitude: userLatitude,
114.                                longitude: userLongitude)
115.                                //for each gig there is...
116.                                for gig in gigs {
117.                                    //...get the location of the GigEvent object
118.                                    let gigEventLocation = gig.getGigEventLocation()
119.                                    //find the distance between the two
120.                                    let distance = gigEventLocation.distance(from:
121.                                        userLocation) as Double
122.                                        //set the distance from user of the GigEvent object

```

```

122.                 gig.setDistance(distanceFromUser: distance)
123.             }
124.             //sort the objects by distance and return the new array
125.             return quickSort(array: gigs)
126.         }
127.
128.         //MARK: GIG EVENT CARDS
129.
130.         //keep track of what object has been swiped
131.         var interactedGigEvent: GigEvent?
132.         //keep track of what object is next
133.         var nextEventImage: UIImage?
134.
135.         func displayGigEventInfo(gigEventView: GigEventView,
136.                                     gigEvent: GigEvent) {
136.             //show data about the event on the card
137.             gigEventView.dayDateLabel.text = gigEvent.getDayDate()
138.             gigEventView.monthYearDateLabel.text =
139.                 gigEvent.getLongMonthYearDate()
139.             gigEventView.timeLabel.text = gigEvent.getTime()
140.             gigEventView.titleLabel.text = gigEvent.getTitle()
141.             gigEventView.paymentLabel.text = "For:
142.                 £\u20ac(gigEvent.getPayment())"
142.         }
143.
144.         func updateCards() {
145.             //show the contact information
146.             nameLabel.isHidden = false
147.             emailLabel.isHidden = false
148.             phoneLabel.isHidden = false
149.             refreshButton.isHidden = true
150.             nextGigEventView.isHidden = true
151.
152.             //if there are gigs to apply for
153.             if gigEvents.count >= 1 {
154.
155.                 //the gig upfront
156.                 if let currentGigEvent = gigEvents.first {
157.                     //will be interacted with
158.                     interactedGigEvent = currentGigEvent
159.
160.                     nameLabel.text = currentGigEvent.getName()
161.                     emailLabel.text = currentGigEvent.getEmail()
162.                     phoneLabel.text = currentGigEvent.getPhone()
163.
164.                     //set the UI for the first in array
165.                     displayGigEventInfo(gigEventView:
166.                         currentGigEventView, gigEvent: currentGigEvent)
167.
168.                     //get image from nextGigEventView or download
169.                     one
168.                     if nextEventImage != nil {
169.                         //reuse the image from card behind
170.
170.                         currentGigEventView.eventPhotoImageView.image = nextEventImage

```

```
171.             } else {
172.                 //download one if its first time loading
173.                 array
174.                 downloadImage(url:
175.                     currentGigEvent.getEventPhotoURL()) { (returnedImage) in
176.                         self.currentGigEventView.eventPhotoImageView.image = returnedImage
177.                     }
178.                 }
179.                 //show the card upfront
180.                 currentGigEventView.isHidden = false
181.             }
182.             //if more than one gigEvent
183.             if gigEvents.count > 1 {
184.                 //display the nextGigEventView behind with the
185.                 //next gigEvent in line
186.                 nextGigEventView.isHidden = false
187.                 let nextGigEvent = gigEvents[1]
188.                 displayGigEventInfo(gigEventView:
189.                     nextGigEventView, gigEvent: nextGigEvent)
190.             }
191.             //this image is always downloaded
192.             downloadImage(url:
193.                 nextGigEvent.getEventPhotoURL()) { (returnedImage) in
194.                     self.nextEventImage = returnedImage
195.                 }
196.             }
197.         }
198.     }
199.     } else {
200.         //no gigs to apply for, tell user
201.         nameLabel.text = "No Gigs Around"
202.         emailLabel.text = "Share GoGig"
203.         nextEventImage = nil
204.         phoneLabel.isHidden = true
205.         currentGigEventView.isHidden = true
206.         nextGigEventView.isHidden = true
207.         refreshButton.isHidden = false
208.     }
209. }
210.
211.
212.     @IBAction func checkEvent(_ sender: Any) {
213.         performSegue(withIdentifier: TO_EVENT_DESCRIPTION,
214.             sender: nil)
215.
216.
217.     //MARK: UPDATE APPLIED USERS
```

```

218.
219.    func didChoose(applied: Bool){
220.
221.        //get the interacted users of that event
222.        var gigEventAppliedUsers =
223.            interactedGigEvent?.getAppliedUsers()
224.            //and add a new key with the current users uid
225.            gigEventAppliedUsers![user!.uid] = applied
226.
227.            //update the dictionary in the database
228.            DataService.instance.updateDBEventsInteractedUsers(uid:
229.                user!.uid, eventID: interactedGigEvent!.getId(), eventData:
230.                gigEventAppliedUsers!)
231.
232.            if applied {
233.                //tell musician that they have applied with a
234.                //notification
235.                updateActivity()
236.
237.            //remove the card from the gigEvent stack
238.            gigEvents.remove(at:0)
239.
240.
241.            func updateActivity() {
242.                let notificationID = NSUUID().uuidString
243.                let relatedEventID = interactedGigEvent!.getId()
244.                let senderUid = user!.uid
245.                //reciever is user that created gig
246.                let recieverUid = interactedGigEvent!.getUid()
247.                let senderName = user!.name
248.                let notificationPicURL = user!.picURL.absoluteString
249.                let notificationDescription = "applied for the event:
250.                    \(interactedGigEvent!.getTitle())"
251.                    let timestamp = NSDate().timeIntervalSince1970
252.                    notificationData = ["notificationID": notificationID,
253.                    "relatedEventID": relatedEventID, "type": "applied", "sender":
254.                    senderUid, "reciever": recieverUid, "senderName": senderName,
255.                    "picURL": notificationPicURL, "description":
256.                    notificationDescription, "timestamp": timestamp]
257.
258.                    //send a push notification to event creator
259.                    DataService.instance.getDBUserProfile(uid: recieverUid)
260.                    { (returnedUser) in
261.                        DataService.instance.sendPushNotification(to:
262.                        returnedUser.getFCMToken(), title: "Application pending", body:
263.                            "\(senderName) has applied for your event")
264.                    }
265.                    //notify creator
266.                    DataService.instance.updateDBActivityFeed(uid:
267.                    recieverUid, notificationID: notificationID, notificationData:
268.                    notificationData!) { (complete) in

```

```

259.             if complete {
260.                 //notify current user about their action
261.                 (sender is themself to reciever themself)
262.                 self.notificationData!["senderName"] = "You"
263.                 self.notificationData!["reciever"] = senderUid
264.                 self.notificationData!["type"] = "personal"
265.                 DataService.instance.updateDBActivityFeed(uid:
266.                     senderUid, notificationID: notificationID, notificationData:
267.                     self.notificationData!) { (complete) in
268.                         }
269.                     }
270.                 }
271.             }
272.             //MARK: GESTURE METHOD
273.             //called method when the gesture is recognised
274.             //Pan Gesture - swipe across screen
275.             @objc func gigEventWasDragged(gestureRecogniser:
276.                 UIPanGestureRecognizer) {
277.                 //returns a vector of where user drags to
278.                 let translation = gestureRecogniser.translation(in:
279.                     view)
280.                 let theView = gestureRecogniser.view!
281.                 //move the view to where the user is dragging
282.                 theView.center = CGPoint(x: self.view.bounds.width / 2
283.                     + translation.x, y: self.view.bounds.height / 2 + translation.y)
284.                 //calculate distance of view from the centre
285.                 let xFromCenter = theView.center.x -
286.                     self.view.bounds.width / 2
287.                     //view will rotate as it moved further from centre
288.                     (radians)
289.                     //will rotate less the further from the centre it goes
290.                     //view won't go upside down
291.                     var rotation = CGAffineTransform(rotationAngle:
292.                         xFromCenter / 200)
293.                     //shrink the card as it reaches screen edge - set the
294.                     scale
295.                     let scale = min(abs(100 / xFromCenter), 1)
296.                     //the stretch and rotation effect set by the scale
297.                     var stretchAndRotation = rotation.scaledBy(x: scale, y:
298.                         scale)
299.                     //apply the rotation and stretch to the view
300.                     theView.transform = stretchAndRotation
301.                     //when the user finished dragging

```

```
301.             if gestureRecogniser.state ==  
302.                 UIGestureRecognizer.State.ended {  
303.                     //the area at which a definite choice has been  
304.                     //made:  
305.                     if theView.center.x < 40 {  
306.                         //call function to do actions based on their  
307.                         choice  
308.                         didChoose(applied: false)  
309.                         //call function to show confirmation (musician  
310.                         knows what they have done)  
311.                         confirmChoiceAnimation(applied: false)  
312.                         //return the view to the centre  
313.                         rotation = CGAffineTransform(rotationAngle: 0)  
314.                         stretchAndRotation = rotation.scaledBy(x: 1, y:  
315.                                         1)  
316.                         theView.transform = stretchAndRotation  
317.                         theView.center = CGPoint(x:  
318.                                         self.view.bounds.width / 2, y: (self.view.bounds.height / 2) + 60)  
319.                     //dragged right  
320.                     } else if theView.center.x > self.view.bounds.width  
321.                         - 40 {  
322.                         didChoose(applied: true)  
323.                         confirmChoiceAnimation(applied: true)  
324.                         rotation = CGAffineTransform(rotationAngle: 0)  
325.                         stretchAndRotation = rotation.scaledBy(x: 1, y:  
326.                                         1)  
327.                         theView.transform = stretchAndRotation  
328.                         theView.center = CGPoint(x:  
329.                                         self.view.bounds.width / 2, y: (self.view.bounds.height / 2) + 60)  
330.                     }  
331.             func confirmChoiceAnimation(applied: Bool) {  
332.                 var confirmationImageView: UIImageView?  
333.                 if applied {  
334.                     //will be a tick image  
335.                     confirmationImageView = UIImageView(image:  
336.                         UIImage(named: "appliedGigEvent"))  
337.                 } else {  
338.                     //will be a cross image  
339.                     confirmationImageView = UIImageView(image:  
340.                         UIImage(named: "ignoredGigEvent"))  
341.                     confirmationImageView!.frame = CGRect.init(x: 0, y: 0,  
342.                         width: 100, height: 100)  
343.                     confirmationImageView!.center = self.view.center  
344.                     //subtle confirmation
```

```

344.         confirmationImageView!.alpha = 0.5
345.         view.addSubview(confirmationImageView!)
346.         //grow and fade
347.         UIView.animate(withDuration: 0.2, animations: {
348.             //grow x1.3
349.             confirmationImageView!.transform =
350.                 CGAffineTransform(scaleX: 1.3, y: 1.3)
351.         }) { (complete) in
352.             //fade in 0.2 seconds from 0.5 opacity to 0.0
353.             UIView.animate(withDuration: 0.2, delay: 0.2,
354.                 options: .curveEaseOut, animations: {
355.                     confirmationImageView!.alpha = 0.0
356.                 }) { (complete) in
357.                     //after animation, remove it from the view
358.                     confirmationImageView!.removeFromSuperview()
359.                 }
360.
361.         override func prepare(for segue: UIStoryboardSegue, sender:
362.             Any?) {
363.             if segue.identifier == TO_EVENT_DESCRIPTION {
364.
365.                 let eventDescriptionVC = segue.destination as!
366.                     EventDescriptionVC
367.                 //make sure there is a purple back button
368.                 let backItem = UIBarButtonItem()
369.                 backItem.tintColor = colorLiteral(red:
370.                     0.4942619801, green: 0.1805444658, blue: 0.5961503386, alpha: 1)
371.                 backItem.title = "Back"
372.                 navigationItem.backBarButtonItem = backItem
373.             }
374.         }
375.     }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
gigEvents	Array<GigEvent>	Public	Array of gigEvents which will be displayed in a 'stack' of cards.
dragGesture	UIPanGestureRecognizer	Local	Calls a function when user drags finger across the screen.
userLatitude userLongitude	Double	Public	Coordinates of the current user to create a location out of.
userLocation	CLLocation	Local	Location object instantiated from coordinates. Used to calculate a distance between user and the event location.

gigEventLocation	CLLocation	Local	Location of event. Will be used to calculate a distance between it and the musician.
distance	Double	Local	Length between the user's current location and the event's proposed location. Will be used to quicksort the gigEvents array.
interactedGigEvent	GigEvent	Public	To keep track of what GigEvent object has been swiped.
nextGigEventImage	UIImage	Public	The image that should be shown on the next gig card.
currentGigEvent	GigEvent	Local	The GigEvent object the user is interacting with.
nextGigEvent	GigEvent	Local	The GigEvent object the user will interact with after the current one.
gigEventAppliedUsers	Array<String>	Local	Array of all the uids that have interacted with this GigEvent object. To append the current user uid so that they will not see it again after they have finished with it.
translation	CGPoint	Local	Returns a vector of where the user drags to.
theView	UIView	Local	The card that will be dragged across the screen.
xFromCenter	CGFloat	Local	To calculate the distance the view has been dragged from the centre.
rotation	CGAffineTransform	Local	To rotate the card more as it is dragged further from the centre.
scale	CGFloat	Local	To shrink the card as it reaches the screen edge.
stretchAndRotation	CGAffineTransform	Local	The stretch and rotation effect set by scale to be applied to the card.
confirmationImageView	UIImageView	Local	To display a tick or cross to let user know how they responded to the gig opportunity.
eventDescriptionVC	EventDescriptionVC	Local	Instance of EventDescriptionVC to change the value of its variables from FindGigVC before a segue.

## EventDescriptionVC.swift

```

1. import UIKit
2.
3. class EventDescriptionVC: UIViewController {
4.
5.     @IBOutlet weak var nameButton: UIButton!
6.     @IBOutlet weak var dayDateLabel: UILabel!
7.     @IBOutlet weak var monthYearDateLabel: UILabel!
8.     @IBOutlet weak var timeLabel: UILabel!
9.     @IBOutlet weak var locationLabel: UILabel!
10.    @IBOutlet weak var paymentLabel: UILabel!
11.    @IBOutlet weak var descriptionTextView: MyTextView!
12.    @IBOutlet weak var editBarItem: UIBarButtonItem!
13.
14.    //observing description (creator will have an edit button)
15.    var observingGigEvent = true
16.
17.    var gigEvent: GigEvent?
18.
19.    override func viewDidLoad() {
20.        super.viewDidLoad()
21.        setupView()
22.        //show the navigation bar
23.        navigationController?.navigationBar.isHidden = false
24.
25.        refresh()
26.    }
27.    override func viewDidAppear(_ animated: Bool) {
28.        //specifiy that large title is not wanted
29.        navigationController?.navigationBar.prefersLargeTitles
30.        = false
31.    }
32.    override func viewDidDisappear(_ animated: Bool) {
33.        //if musician observing event
34.        if observingGigEvent {
35.            //hide the edit button
36.            navigationItem.rightBarButtonItem = nil
37.        }
38.
39.        func refresh() {
40.            //set all outlets with information about the gig
41.            self.navigationItem.title = gigEvent?.getTitle()
42.            nameButton.setTitle("Check out \(gigEvent!.getName())",
43.            for: .normal)
44.            dayDateLabel.text = gigEvent?.getDayDate()
45.            monthYearDateLabel.text =
46.                gigEvent?.getLongMonthYearDate()
47.            timeLabel.text = gigEvent?.getTime()
48.            locationLabel.text = gigEvent!.getLocationName() + " "
49.            + gigEvent!.getPostcode()
50.            paymentLabel.text = "For: £\(gigEvent!.getPayment())"
51.            descriptionTextView.text = gigEvent?.getDescription()
52.        }

```

```

50.
51.        //look at the portfolio of the event creator
52.        var checkUid: String?
53.        @IBAction func checkUid(_ sender: Any) {
54.            //uid is to refresh portfolio of that user
55.            checkUid = gigEvent?.getuid()
56.            performSegue(withIdentifier: TO_CHECK_PORTFOLIO_3,
57.            sender: nil)
58.
59.        @IBAction func editGigEvent(_ sender: Any) {
60.            //if creator wants to edit
61.            if observingGigEvent == false {
62.                //globally declare that user is editing event
63.                editingGigEvent = true
64.                //and go back to create event navigation stack to
65.                //edit the event
66.                self.performSegue(withIdentifier:
67.                TO_EDIT_GIG_EVENT, sender: nil)
68.
69.        override func prepare(for segue: UIStoryboardSegue, sender:
70.        Any?) {
71.            if segue.identifier == TO_CHECK_PORTFOLIO_3 {
72.
73.                let userAccountVC = segue.destination as!
74.                UserAccountVC
75.                //portfolio view
76.                let backItem = UIBarButtonItem()
77.                backItem.tintColor = colorLiteral(red:
78.                0.4942619801, green: 0.1805444658, blue: 0.5961503386, alpha: 1)
79.                backItem.title = "Back"
80.                navigationItem.backBarButtonItem = backItem
81.                //uid needed to refresh portfolio for that user
82.                userAccountVC.uid = checkUid!
83.                //mark a user is observing that portfolio
84.                userAccountVC.observingPortfolio = true
85.                //refresh the portfolio ready for the segue
86.                userAccountVC.refreshPortfolio()
87.
88.            } else if segue.identifier == TO_EDIT_GIG_EVENT {
89.                //set the create event views ready to edit the
90.                //event
91.                let titleDateCGVC = segue.destination as!
92.                TitleDateCGVC
93.                titleDateCGVC.editEventID = gigEvent!.getid()
94.                titleDateCGVC.editingGate = true
95.            }
96.        }

```

### Data Dictionary

Identifier	Data Type	Scope	Purpose
------------	-----------	-------	---------

observingGigEvent	Boolean	Public	To keep track of whether the event's creator is looking at it in detail or not to display view appropriately.
gigEvent	GigEvent	Public	The object that will be used to display the event in detail using the outlets.
checkUid	String	Public	The uid of the user whos portfolio is about to be observed so it can refresh it.
userAccountVC	UserAccountVC	Local	Instance of UserAccountVC to change the value of its variables from EventDescriptionVC before a segue.
titleDateVC	TitleDateVC	Local	Instance of TitleDateVC to change the value of its variables from EventDescriptionVC before a segue.

## ActivityFeedVC.swift

```

1. import UIKit
2. import FirebaseAuth
3. import FirebaseDatabase
4.
5. class ActivityFeedVC: UIViewController, UICollectionViewDelegate,
   UICollectionViewDataSource, UICollectionViewDelegateFlowLayout,
   UITableViewDelegate, UITableViewDataSource,
   UITabBarControllerDelegate {
6.
7.     @IBOutlet weak var editBarButton: UIBarButtonItem!
8.     @IBOutlet weak var collectionView: UICollectionView!
9.
10.    var selectedCVCell: Int = 0 //initially show notifications
11.    var storedOffsets = [Int: CGFloat]()
12.
13.    //instantiation of menubar in a closure
14.    lazy var menuBar: MenuBar = {
15.        let mb = MenuBar()
16.        mb.translatesAutoresizingMaskIntoConstraints = false
17.        mb.activityFeedVC = self
18.        return mb
19.    }()
20.
21.    private func setupMenuBar() {
22.        //dont show the scroll bar for horizontal scroll
23.        collectionView.showsHorizontalScrollIndicator = false
24.        //so scroll will stop on either cell and not sit
   halfway between them
25.        collectionView.isPagingEnabled = true

```

```
26.          //constrain the menuBar to the top of the
27.          collectionView
28.          view.addSubview(menuBar)
29.          menuBar.leftAnchor.constraint(equalTo:
30.          view.leftAnchor).isActive = true
31.          menuBar.rightAnchor.constraint(equalTo:
32.          view.rightAnchor).isActive = true
33.          menuBar.bottomAnchor.constraint(equalTo:
34.          collectionView.topAnchor).isActive = true
35.          menuBar.heightAnchor.constraint(equalToConstant:
36.          40).isActive = true
37.      }
38.
39.      var user: User?
40.      var activityNotifications = [ActivityNotification]()
41.      var usersEvents = [GigEvent]()
42.      var eventIDs = [String]()
43.
44.      override func viewDidLoad() {
45.          super.viewDidLoad()
46.          self.tabBarController?.delegate = self
47.          setupMenuBar()
48.          feedGateOpen = false
49.          refreshActivityFeed()
50.          //to refresh activity in other classes
51.          NotificationCenter.default.addObserver(self, selector:
52. #selector(refreshAll), name: NSNotification.Name(rawValue:
53. "refreshAllActivity"), object: nil)
54.      }
55.      override func viewDidAppear(_ animated: Bool) {
56.          navigationController?.navigationBar.topItem?.title =
57.          "Activity"
58.      }
59.      override func viewDidDisappear(_ animated: Bool) {
60.          //so multiple observers aren't opened when view appears
61.          if observeGateOpen {
62.              observeGateOpen = false
63.              observeActivityNotifications()
64.          }
65.          //so feed doesn't refresh everytime view appears
66.          if feedGateOpen {
67.              //need to remove all on sign in otherwise it
68.              //doesn't refresh
69.              //what has been 'observed' since view did load
70.              activityNotifications.removeAll()
71.              usersEvents.removeAll()
72.              eventIDs.removeAll()
73.              feedGateOpen = false
74.              refreshActivityFeed()
75.          }
76.      }
77.      @objc func refreshAll() {
78.          //will be called after editing an event
79.          //clear everything
80.      }
81.  }
```

```

72.          activityNotifications.removeAll()
73.          usersEvents.removeAll()
74.          eventIDs.removeAll()
75.          //reload the collectionview
76.          self.collectionView.reloadData()
77.          //and refresh it again
78.          refreshActivityFeed()
79.      }
80.
81.      //MARK: FETCH DATA
82.      func refreshActivityFeed() {
83.          //get the current user profile
84.          if let uid = Auth.auth().currentUser?.uid {
85.              DataService.instance.getDBUserProfile(uid: uid) {
86.                  (returnedUser) in
87.                      self.user = returnedUser
88.                      //get the Activity Notifications
89.                      self.fetchingMore = true
90.                      DataService.instance.getDBActivityFeed(uid:
91.                          uid, currentActivity: self.activityNotifications) {
92.                          (returnedActivityNotifications) in
93.                              self.activityNotifications =
94.                                  returnedActivityNotifications
95.                                  //keep track of how many new ones were
96.                                  //added (pagination)
97.                                  print("Got back
98. \(\self.activityNotifications.count)")
99.                                  //no more new notifications
100.                                 self.endReached =
101.                                     (returnedActivityNotifications.count == 0)
102.                                     //stop fetching more
103.                                     self.fetchingMore = false
104.                                     //reload the tables
105.                                     self.collectionView.reloadData()
106.                                 }
107.                                 //get and observe the events associate to that
108.                                 user
109.                                 eventsHandle =
110.                                 DataService.instance.REF_USERS.child(uid).child("events").observe(.value) { (snapshot) in
111.                                     DataService.instance.getDBUserEvents(uid:
112.                                         uid) { (returnedEventIDs) in
113.                                             var eventListings = [GigEvent]()
114.                                             //one was getting appended, the other
115.                                             was getting inserted at 0. deleting didn't delete the correct
116.                                             gigEvent
117.                                             self.eventIDs =
118.                                             returnedEventIDs.reversed()
119.                                             //iterate through the list of eventIDs
120.                                             associated to user, get each one from public events
121.                                             //and insert at 0 of local array
122.                                             for eventID in returnedEventIDs {
123.                                                 
```

```

111.
112.    DataService.instance.getDBSingleEvent(uid: uid, eventID: eventID) {
113.        (returnedGigEvent, success) in
114.            if success {
115.                eventListings.insert(returnedGigEvent, at: 0)
116.                self.usersEvents =
117.                    eventListings
118.                self.collectionView.reloadData()
119.            } //couldn't get GigEvent (been
120.                deleted)
121.        } //user (should be musician)
122.        has an eventID listed which the organiser has already deleted, clean
123.            up the DB
124.                if let index =
125.                    self.eventIDs.firstIndex(of: eventID) {
126.                        self.eventIDs.remove(at: index)
127.                    DataService.instance.deleteDBUserEvents(uid: uid, eventIDs:
128.                        self.eventIDs)
129.                    }
130.                }
131.            }
132.        }
133.    }
134.
135.        //needed so that the user profile pictures don't flash and
136.        //display the wrong image
137.        var timer: Timer?
138.        func attemptReload() {
139.            //stop the timer
140.            self.timer?.invalidate()
141.            //reload the collection view and all its data 0.1
142.            //seconds after timer has started
143.            self.timer = Timer.scheduledTimer(timeInterval: 0.1,
144.                target: self, selector: #selector(handleReload), userInfo: nil,
145.                repeats: false) //doesn't repeat
146.        }
147.        @objc func handleReload(){
148.            self.collectionView.reloadData()
149.        }
150.
151.        //MARK: FETCH MORE DATA
152.        //((Pagination)
153.        var fetchingMore = false

```

```

150.          //if reached the end, don't bother fetching anymore posts
151.          var endReached = false
152.          //start loading notifications 1 cell in advance
153.          var leadingScreensForBatching: CGFloat = 1.0
154.
155.          func getMoreNotifications(){
156.              fetchingMore = true
157.
158.              DataService.instance.getDBActivityFeed(uid: user!.uid,
159.                  currentActivity: activityNotifications) {
160.                      (returnedActivityNotifications) in
161.                          //we are appending the contents of the array, not
162.                          //the array itself
163.                          self.activityNotifications.append(contentsOf:
164.                              returnedActivityNotifications)
165.                          //if no more notifications, we have reached the end
166.                          self.endReached =
167.                              (returnedActivityNotifications.count == 0)
168.                          self.fetchingMore = false
169.                          self.collectionView.reloadData()
170.                  }
171.          }
172.          //MARK: OBSERVE ACTIVITY CHANGES
173.          func observeActivityNotifications(){
174.              if let uid = Auth.auth().currentUser?.uid {
175.                  //observe Activity Notifications
176.                  DataService.instance.observeDBActivityFeed(uid:
177.                      uid) { (returnedActivityNotification) in
178.                          //double check it is a new notification
179.                          if
180.                              self.activityNotifications.contains(returnedActivityNotification) ==
181.                                  false {
182.                                  self.activityNotifications.insert(returnedActivityNotification, at:
183.                                      0)
184.                              }
185.                              //reload to show new notification
186.                              self.collectionView.reloadData()
187.                          }
188.          }
189.          //MARK: NOTIFICATION CELL
190.          func updateNotificationData(cell: ActivityFeedCell, row:
191.              Int) {
192.              //bug of cell repeat
193.              cell.notificationImage.alpha = 1
194.              cell.eventNameButton.isEnabled = true
195.              cell.eventNameButton.alpha = 1
196.              cell.notificationDescriptionLabel.alpha = 1
197.              //set all the notification data in the cell

```

```

194.
  cell.eventNameButton.setTitle(activityNotifications[row].getSenderName(), for: .normal)
195.          cell.eventNameButton.tintColor = colorLiteral(red:
  0.4942619801, green: 0.1805444658, blue: 0.5961503386, alpha: 1)
196.          cell.eventNameButton.tag = row
197.          cell.deleteNotificationButton.tag = row
198.          cell.notificationDescriptionLabel.text =
  activityNotifications[row].getNotificationDescription()
199.          //try load the image from cache
200.          loadImageCache(url:
  activityNotifications[row].getNotificationPicURL(), isImage: true) {
  (returnedImage) in
201.              cell.notificationImage.image = nil
202.              cell.notificationImage.image = returnedImage
203.          }
204.          //if user clicked edit, show the 'minus' delete buttons
205.          if editingNotifications {
206.              cell.deleteNotificationButton.isHidden = false
207.          } else {
208.              cell.deleteNotificationButton.isHidden = true
209.          }
210.      }
211.
212.      //MARK: EVENT CELL
213.
214.      func updateEventListingData(cell: ActivityFeedCell, row:
  Int) {
215.          //bug of cell repeat
216.          cell.notificationImage.alpha = 1
217.          cell.eventNameButton.isEnabled = true
218.          cell.eventNameButton.alpha = 1
219.          cell.notificationDescriptionLabel.alpha = 1
220.          //set all the event listing data in the cell
221.          cell.notificationImage.isHidden = false
222.
  cell.eventNameButton.setTitle("\(usersEvents[row].getMonthYearDate())
  -\(usersEvents[row].getDayDate())", for: .normal)
223.          cell.eventNameButton.tintColor = colorLiteral(red:
  0.4942619801, green: 0.1805444658, blue: 0.5961503386, alpha: 1)
224.          cell.notificationDescriptionLabel.text =
  "\(usersEvents[row].getTitle())"
225.          cell.eventNameButton.tag = row
226.          cell.deleteNotificationButton.tag = row
227.          loadImageCache(url:
  usersEvents[row].getEventPhotoURL(), isImage: true) {
  (returnedImage) in
228.              cell.notificationImage.image = returnedImage
229.          }
230.          //only show delete if editing
231.          if editingNotifications {
232.              cell.deleteNotificationButton.isHidden = false
233.          } else {
234.              cell.deleteNotificationButton.isHidden = true
235.          }

```

```

236.
237.        //if old event change the UI
238.        if checkOld(gigEventToCompare: usersEvents[row]) ==
239.            true {
240.                //faded - set alpha to 0.3
241.                cell.notificationImage.alpha = 0.3
242.                //cannot interact with the date button
243.                cell.eventNameButton.isEnabled = false
244.                cell.eventNameButton.alpha = 0.3
245.                cell.notificationDescriptionLabel.alpha = 0.3
246.            }
247.
248.        //MARK: NOTIFICATION CELL ACTIONS
249.        var checkUid: String?
250.        @IBAction func checkOut(_ sender: UIButton) {
251.            //get the row of the cell with a tag from button
252.            let row = sender.tag
253.            //notifications section
254.            if selectedCVCell == 0 {
255.                //and get the uid of user that sent the
256.                //notification
257.                checkUid =
258.                    activityNotifications[row].getSenderId()
259.                //perform segue to observe portfolio refreshed with
260.                //this uid
261.                performSegue(withIdentifier: TO_CHECK_PORTFOLIO,
262.                sender: nil)
263.            }
264.            //get the GigEvent object
265.            let calendarEvent = usersEvents[row]
266.            //use it to add an event to the calendar
267.            addEventToCalendar(title: calendarEvent.getTitle(),
268.                description: calendarEvent.getDescription(), startDate:
269.                    calendarEvent.getDate().addingTimeInterval(-3600), endDate:
270.                    calendarEvent.getDate())
271.            //notify the user that it has been added
272.            displayError(title: "Added to Calendar", message:
273.                "This event was added to your device calendar")
274.        }
275.
276.        //MARK: COMPARE TIME AND DATE
277.
278.        //to decide if event listing should be faded or not
279.        func checkOld(gigEventToCompare: GigEvent) -> Bool {
280.            //get current date
281.            let date0bject = Date()
282.            let currentDate = date0bject.addingTimeInterval(3600)
283.            //hour behind
284.            //if the date of the GigEvent is old (less than the
285.            //current date)

```

```
280.             if gigEventToCompare.getDate() < currentDate {
281.                 return true
282.             }
283.             return false
284.         }
285.
286.         //MARK: DELETE EVENTS
287.
288.         //user choosing to delete table cells
289.         var editingNotifications = false
290.         @IBAction func editBarButton(_ sender: Any) {
291.             //change UI so user knows what is going on
292.             if editingNotifications {
293.                 editingNotifications = false
294.                 editBarButton.title = "Edit"
295.                 collectionView.reloadData()
296.             } else {
297.                 editingNotifications = true
298.                 editBarButton.title = "Done"
299.                 collectionView.reloadData()
300.             }
301.         }
302.
303.         @IBAction func deleteNotification(_ sender: UIButton) {
304.             let row = sender.tag
305.             //delete a notification
306.             if selectedCVCell == 0 {
307.                 DataService.instance.deleteDBActivityFeed(uid:
308.                     user!.uid, notificationID: activityNotifications[row].getId())
309.                     activityNotifications.remove(at: row)
310.                     collectionView.reloadData()
311.             } else {
312.                 //provide a warning message
313.                 var title = ""
314.                 var message = ""
315.                 //warn the musician
316.                 if user!.gigs {
317.                     title = "Forget the Gig"
318.                     message = "You will have no association with
319.                         this event"
320.                     //warn the organiser
321.                 } else {
322.                     title = "Delete your Event"
323.                     message = "It will no longer exist to all
324.                         users"
325.                 }
326.                 //add the UIAlerts
327.                 let alertController = UIAlertController(title:
328.                     title, message: message, preferredStyle: .alert)
329.                 alertController.addAction(UIAlertAction(title:
330.                     "Cancel", style: .cancel))
331.                 alertController.addAction(UIAlertAction(title:
332.                     "Delete", style: .destructive, handler: { (buttonPressed) in
```

```

328.                                //if organiser, delete the event object and the
   picture in Storage
329.                                if self.user!.gigs == false {
330.                                    DataService.instance.deleteDBEvents(uid:
   self.user!.uid, eventID: self.eventIDs[row])
331.                                    DataService.instance.deleteSTFile(uid:
   self.user!.uid, directory: "events", fileID: self.eventIDs[row])
332.                                }
333.                                //for everyone remove it from event listings
   under user in Database
334.                                self.eventIDs.remove(at: row)
335.                                self.usersEvents.remove(at: row)
336.                                DataService.instance.deleteDBUserEvents(uid:
   self.user!.uid, eventIDs: self.eventIDs)
337.                                self.collectionView.reloadData()
338.                            })
339.                            self.present(alertController, animated: true,
   completion: nil)
340.                        }
341.                    }
342.
343.                    //pressed the tab, scroll to the top of the table view
344.                    func tabBarController(_ tabBarController:
   UITabBarController, didSelect viewController: UIViewController) {
345.                        let tabBarIndex = tabBarController.selectedIndex
346.                        if tabBarIndex == 2 {
347.                            NotificationCenter.default.post(name:
   NSNotification.Name(rawValue: "scrollToTop"), object: nil)
348.                        }
349.                    }
350.
351.                    //MARK: SEGUES
352.
353.                    //to keep track of what object belongs to the tapped cell
354.                    var selectedApplication: ActivityNotification?
355.                    var selectedListing: GigEvent?
356.                    override func prepare(for segue: UIStoryboardSegue, sender:
   Any?) {
357.
358.                        //set up destination ready to observe portfolio
359.                        if segue.identifier == TO_CHECK_PORTFOLIO {
360.
361.                            let userAccountVC = segue.destination as!
   UserAccountVC
362.
363.                            let backItem = UIBarButtonItem()
364.                            backItem.tintColor = colorLiteral(red:
   0.4942619801, green: 0.1805444658, blue: 0.5961503386, alpha: 1)
365.                            backItem.title = "Back"
366.                            navigationItem.backBarButtonItem = backItem
367.
368.                            userAccountVC.uid = checkUid!
369.                            userAccountVC.observingPortfolio = true
370.                            userAccountVC.refreshPortfolio()
371.

```

```

372.          //setup destination ready to respond to application
373.          } else if segue.identifier == TO REVIEW APPLICATION {
374.
375.              let reviewApplicationVC = segue.destination as!
376.              ReviewApplicationVC
377.              reviewApplicationVC.uid = checkUid!
378.              reviewApplicationVC.application =
379.              selectedApplication
380.              reviewApplicationVC.refresh()
381.          //setup destination ready to look at the GigEvent
382.          // object in more detail
382.          } else if segue.identifier == TO EVENT DESCRIPTION 2 {
383.
384.              let backItem = UIBarButtonItem()
385.              backItem.tintColor = colorLiteral(red:
386. 0.4942619801, green: 0.1805444658, blue: 0.5961503386, alpha: 1)
386.              backItem.title = "Back"
387.              navigationItem.backBarButtonItem = backItem
388.
389.              let eventDescriptionVC = segue.destination as!
390.              EventDescriptionVC
390.              eventDescriptionVC.gigEvent = selectedListing
391.              eventDescriptionVC.observingGigEvent = user!.gigs
392.          }
393.      }
394.  }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
selectedCVCell	Int	Public	The index of the collection view cell selected (is initially activity notifications section).
storedOffsets	Dictionary<Integer, CGFloat>	Public	Keep track of how far user has scrolled to then fetch more notifications from Database.
menuBar	MenuBar	Lazy	Instance of the menuBar which will sit at the top of the collection view for user to pick between the two sections.
user	User	Public	The current user object to get profile data from.
usersEvents	Array<GigEvent>	Public	Array of events associated to the user that will be iteratively displayed in the cells of "My Events" section.
eventIDs	Array<String>	Public	Array of identifiers of events associated to the

			user to fetch the GigEvent objects with in the first place.
timer	Timer	Public	To refresh the collection view with after 0.1 seconds so that images load into the correct cells.
fetchingMore endReached	Boolean	Public	To keep track of whether we are fetching more notification objects from Database and if there is no more to be gotten.
leadingScreenForBatching	CGFloat	Public	The number of cells in advance before loading more notifications while scrolling.
calendarEvent	GigEvent	Local	The GigEvent object that is about to be added to the device's calendar.
dateObject currentDate	Date	Local	To check if events in "My Events" are expired against, to then display as faded.
editingNotifications	Boolean	Public	To keep track of whether delete buttons should be displayed or not when the user clicks "Edit" in the navigation bar.
title message	String	Local	The title and message to be displayed as a warning when user is about to delete an event listing associated to them.
tabBarIndex	Integer	Local	So that when the user clicks the activity tab, the feed scrolls to the top.
selectedApplication selectedListing	ActivityNotification GigEvent	Public	The object corresponding to the cell tapped on in each section.
userAccountVC	UserAccountVC	Local	Instance of UserAccountVC to change the value of its variables from ActivityFeedVC before a segue.
reviewApplicationVC	ReviewApplicationVC	Local	Instance of ReviewApplicationVC to change the value of its variables from ActivityFeedVC before a segue.

eventDescriptionVC	EventDescriptionVC	Local	Instance of EventDescriptionVC to change the value of its variables from ActivityFeedVC before a segue.
--------------------	--------------------	-------	---

## ReviewApplicationVC.swift

```
40.    DataService.instance.getDBSingleEvent(uid: returnedCurrentUser.uid,
41.    eventID: self.application!.getRelatedEventId()) { (returnedGigEvent,
42.    sucess) in
43.        self.relatedEvent =
44.        returnedGigEvent
45.        //update the UI
46.        self.eventLabel.text = "Wants to
47.        play for \(returnedGigEvent.getTitle())"
48.    }
49.}
50.
51.
52.    //back
53.    @IBAction func popView(_ sender: Any) {
54.
55.        self.navigationController?.popToRootViewController(animated: true)
56.    }
57.    //look at musician's portfolio
58.    @IBAction func checkUid(_ sender: Any) {
59.        performSegue(withIdentifier: TO_CHECK_PORTFOLIO_2,
60.        sender: nil)
61.    }
62.
63.    @IBAction func rejectUser(_ sender: Any) {
64.        updateActivity(accepted: false)
65.        //don't delete the notification on rejection because
66.        user may change their mind
67.    }
68.    @IBAction func acceptUser(_ sender: Any) {
69.        updateActivity(accepted: true)
70.        //need to delete the notification so that you cannot
71.        //accept the user again and again
72.        DataService.instance.deleteDBActivityFeed(uid:
73.            currentUser!.uid, notificationID: application!.getId())
74.
75.        //The user has been accepted, so add that to their 'My
76.        Events List'
77.        DataService.instance.updateDBUserEvents(uid: user!.uid,
78.            eventID: application!.getRelatedEventId())
79.
80.        func updateActivity(accepted: Bool) {
81.            //this line stops the button being pressed twice
82.            //sending two activity updates
83.            self.view.isUserInteractionEnabled = false
84.
85.            //set all the data for the notification
86.            let notificationID = NSUUID().uuidString
87.            guard let senderUid = currentUser?.uid else { return }
88.            guard let recieverUid = uid else { return }
```

```

82.             guard let senderName = currentUser?.name else { return
83.         }
84.             guard let notificationPicURL =
85. currentUser?.picURL.absoluteString else { return }
86.             guard let relatedEventTitle = relatedEvent?.getTitle()
87.         else { return }
88.             guard let relatedEventID = relatedEvent?.getId() else {
89.         return }
90.             //grab the event as well
91.             var notificationDescription: String?
92.             if accepted {
93.                 notificationDescription = "hired you for the event:
94. \n(relatedEventTitle)"
95.             }
96.             //send a push notification musician
97.             DataService.instance.getDBUserProfile(uid:
98. recieverUid) { (returnedUser) in
99.                 DataService.instance.sendPushNotification(to:
100. returnedUser.getFCMToken(), title: "You got the gig!", body:
101. "\n(senderName) hired you for the event: \n(relatedEventTitle)")
102.             }
103.             } else {
104.                 notificationDescription = "declined you for the
105. event: \n(relatedEventTitle)"
106.             }
107.             let timestamp = NSDate().timeIntervalSince1970
108.             notificationData = ["notificationID": notificationID,
109. "relatedEventID": relatedEventID, "type": "reply", "sender":
110. senderUid, "reciever": recieverUid, "senderName": senderName,
111. "picURL": notificationPicURL, "description":
112. notificationDescription!, "timestamp": timestamp]
113.             }
114.             }
115.             }
116.             //allow interaction again
117.             self.view.isUserInteractionEnabled = true

```

```

118.          //go back to activity feed
119.
120.          self.navigationController?.popToRootViewControllerAnimated(animated: true)
121.
122.    override func prepare(for segue: UIStoryboardSegue, sender:
123.      Any?) {
124.          //setup destination ready to observe musician's
125.          //portfolio
126.          if segue.identifier == TO_CHECK_PORTFOLIO_2 {
127.
128.              let backItem = UIBarButtonItem()
129.              backItem.tintColor = colorLiteral(red:
130.                0.4942619801, green: 0.1805444658, blue: 0.5961503386, alpha: 1)
131.              backItem.title = "Back"
132.              navigationItem.backBarButtonItem = backItem
133.
134.              userAccountVC.uid = uid
135.              userAccountVC.observingPortfolio = true
136.              userAccountVC.refreshPortfolio()
137.          }
138.      }

```

## Data Dictionary

Identifier	Data Type	Scope	Purpose
user	User	Public	The musician whose information will be displayed in the application.
currentUser	User	Public	The organiser who will make a decision and respond back to the musician.
uid	String	Public	The user identifier of the musician so that I can fetch their profile data from Database.
application	ActivityNotification	Public	To display all the application information within the view.
relatedEvent	GigEvent	Public	The event that the musician is applying for.
userAccountVC	UserAccountVC	Local	Instance of UserAccountVC to change the value of its variables from ReviewApplicationVC before a segue.

