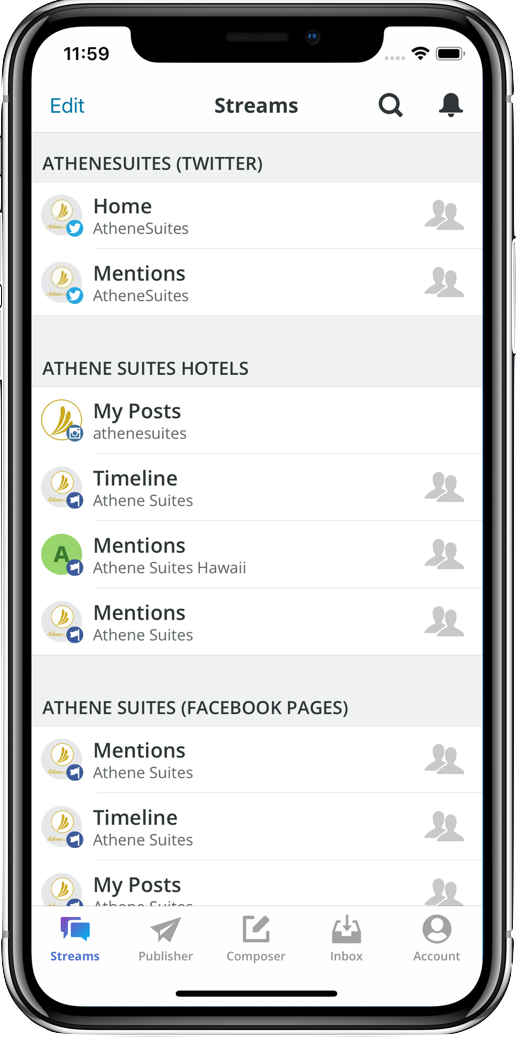
Mobile Applications Design Document for

“TweetCards”

TweetCards is built to enhance the user experience whilst using twitter. Previously if the user wanted to only see a certain set of users’ tweets, they would have to create a new account to follow those specific users. Unless they used an alternative such as the in browser TweetDeck or one of the more social marketing focused apps such as HootSuite.

The focus of this App will be to make twitter feel more like the classic RSS feed. Where you can just specify a set number of users or hashtags you would like to be updated on without having to create an account. The user could then set up multiple of these Cards for different specialised timelines. For example, one card could be for following local news accounts and then a separate one for status updates for the busses in the user’s city. This allows the user to quickly be able to check the whether or not the busses are running in the morning without having to filter through the news at the same time and vice versa.

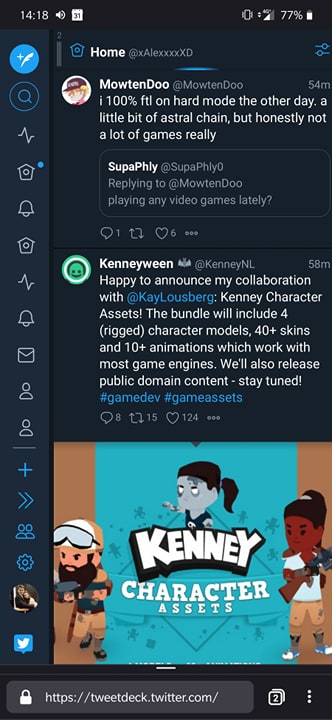
# Research

Whilst looking for related products to the proposed idea I have found some that do operate in similar manners but have some downfall or differ in some way that doesn’t quite line up to the specific use case TweetCards fulfils.

Firstly, there are the social marketing apps such as HootSuite. These apps revolve around pushing messages out to all social media platforms at once (Facebook, Twitter, etc) and keeping all customer facing social media platforms for a company updated with the latest information. These also include analytics for the interactions on each social media platform. These only really focus on keeping the specific users’ feeds across social media platforms updated, as opposed to collating tweets from multiple accounts on a single feed. As seen in Figure 1 HootSuite supports multiple social media accounts which will not be required in TweetCards. HootSuites interface also looks very outdated which will be improved upon in TweetCards.

Secondly, there are apps that focus on managing multiple accounts for twitter such as Seesmic. Allowing the user to technically curate custom feeds through the use of multiple accounts, however, if the only thing you wanted was the custom feeds and none of the account specific features then this is a very cumbersome way of getting these custom feeds.

Figure 1: HootSuite account overview



Finally, the main inspiration for this project, TweekDeck. TweetDeck is a website which allows the user to create multiple curated feeds of various twitter accounts and hashtags and view them all simultaneously. Whilst this website is more feature rich and fulfils the use cases TweetCards proposes it has a fatal flaw. Due to being designed with primarily desktop in mind it feels very clunky to use on a mobile device. The elements used to distinguish timelines are repeated multiple times as seen in Figure 2. Making it more difficult to see which timeline is which at a glance and requiring the user to open an extra menu. TweetDeck’s feature set is also a double-edged sword, whilst it does contain more features and use cases it also ends up being cluttered and confusing to users using it for the first time. Making the learning curve a longer and steeper process. By keeping TweetCards simple and fulfilling one purpose well, it will have a much shallower learning curve.

TweetCards’ will target professionals aged 20 onwards. The primary reason is due to the app allowing the professional using it to keep updated on various topics that are related to their profession or interests. Through this the design of the app is going to be focused around three main paradigms; ease of use, clarity and fast navigation. The reasoning behind these paradigms is that professionals generally do not have much free time throughout their day. By creating TweetCards with that in mind it allows for an app which will be a practical addition to their smart phone.

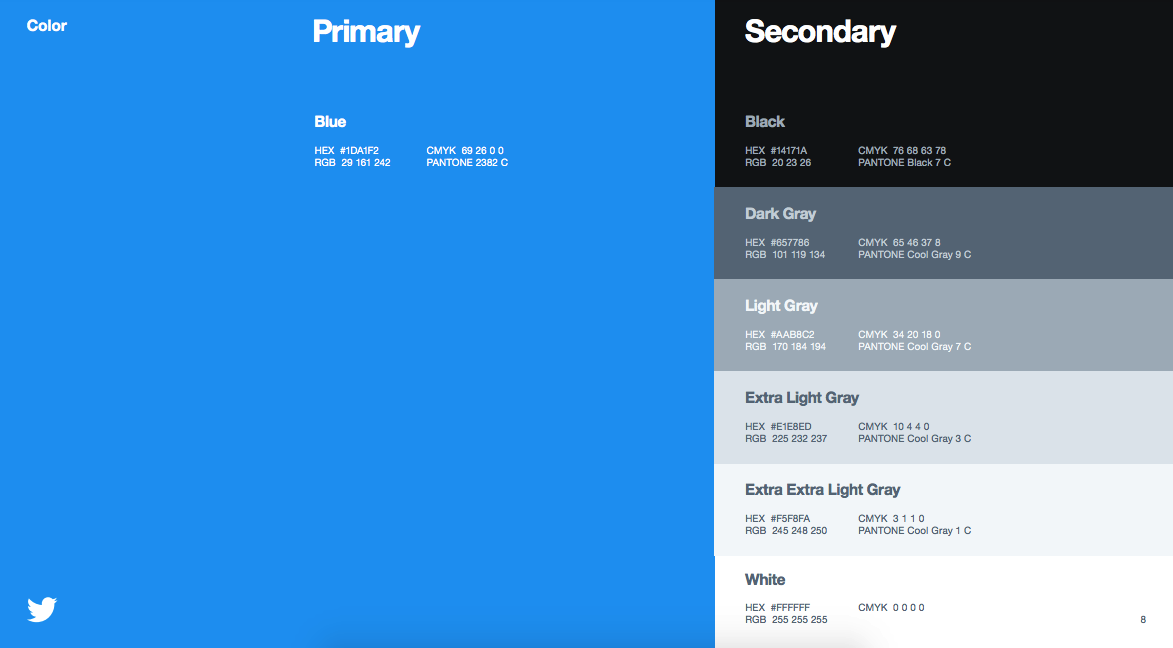
Figure 2: TweetDeck's Mobile Interface

# Design

TweetCards design will utilise a minimalistic style, making sure to only populate the screen with elements that will have a practical use. This ensures that clarity of app is maximised and that the app does not get in the way of using the app itself.

## Colour and Typeface

Since TweetCards will purely focus on the Twitter experience; to keep it familiar to users who already use Twitter, TweetCards will adopt Twitters design specifications of Colour and typeface.

TweetCards will mostly utilise “White” for background and whitespace. Any buttons or text that are used for navigating through the app or any accenting in the app will utilise Twitters primary “Blue” and this will be the case regardless of dark mode. The dark mode version of the app will also utilise Twitters dark mode colour palette. It will use # 141d26 for background and whitespace in the app but where elements were previously contrasted by going darker than the background colour this time, they will be contrasted by going lighter to #243447. The colour palettes can be seen below in Figure 3 and 4.

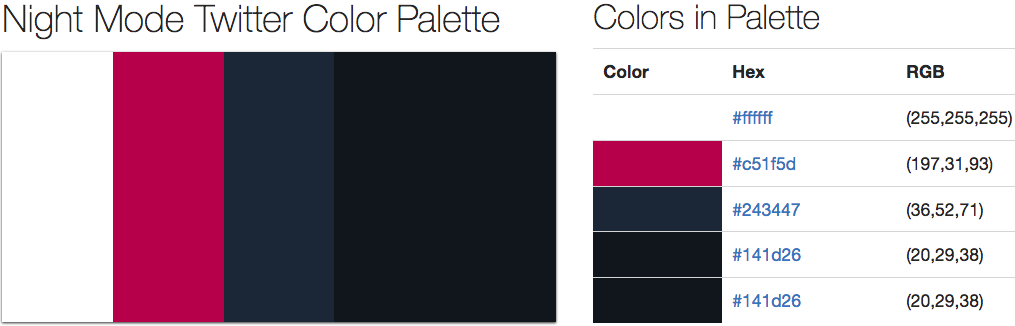
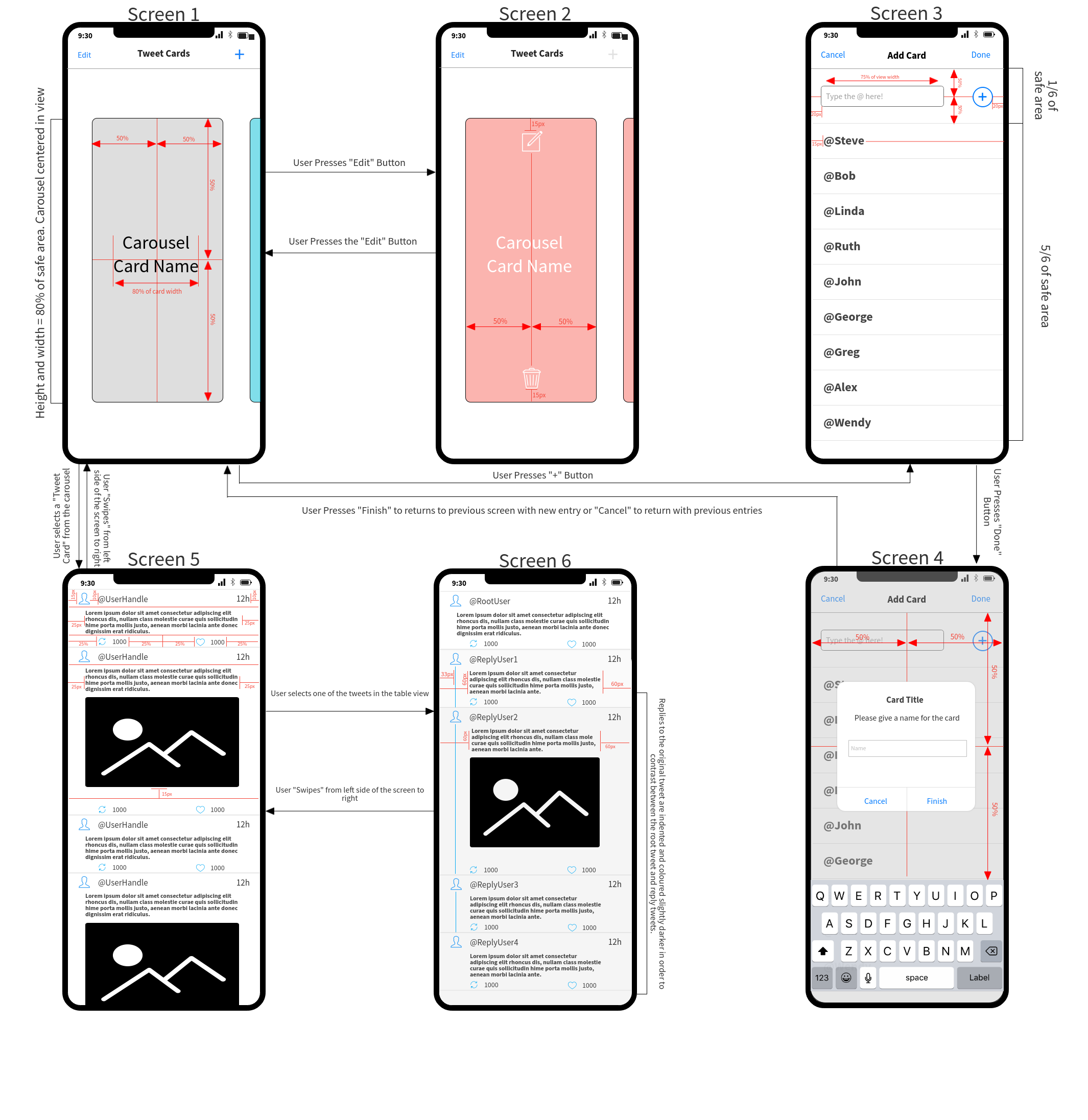


Figure 3 and 4: The Light and Dark mode colour palettes for Twitter

The typeface of the app will also follow Twitters brand guidelines. This means that for Tweet text, timestamps and users name will use “Helvetica Neue Bold”. For the users @, “Helvetica Neue Regular” will to used.

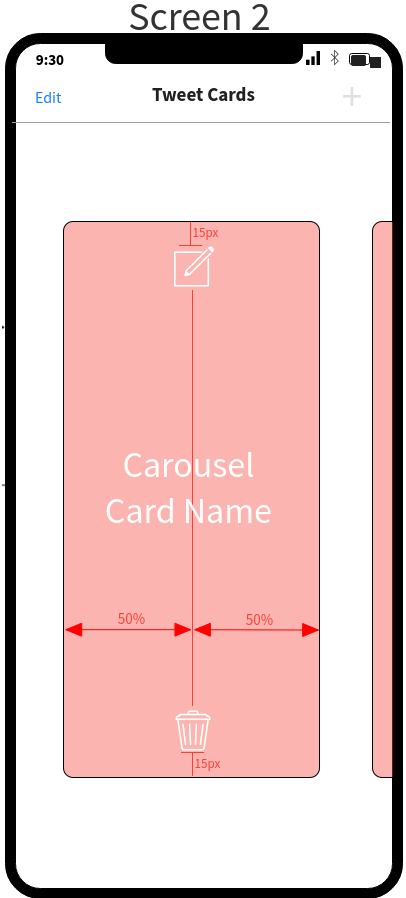
## Layout

In order for the user experience of the app to feel natural there has been a focus on using gestures and animations. Gestures will be used for navigating around the app and the animations will act as affordances on which gestures to use to navigate. For example, a new view being displayed by sliding onto the screen from the right side and being dismissible by using a “Swipe” gesture from the left side of the screen to the right side of the screen. The technical wireframe for the app can be seen below:



### Screen 1

Screen 1 is the entry point to the app and the main screen for navigating between cards. The screen itself is taken up by a carousel of the users curated timelines and a navigation bar. The carousel items can either be one of two things. A cached screenshot of last time the user was viewing the timeline on the card or it can be a picture of their choice. The latter choice would help the user identify the card they are currently looking at with a glance as opposed to reading the title for said card in the upper third of the screen. To traverse the carousel the user simply just swipes it left or right and the currently viewed card will snap into the centre of the screen once the swipe gesture ends. In the navigation bar there are two buttons. The “Edit” button will change the view of the carousel cells and change functionality as described below in section “Screen 2”. The “+” button will take the user to screen 3 in which the user can create a new card to add to the carousel.

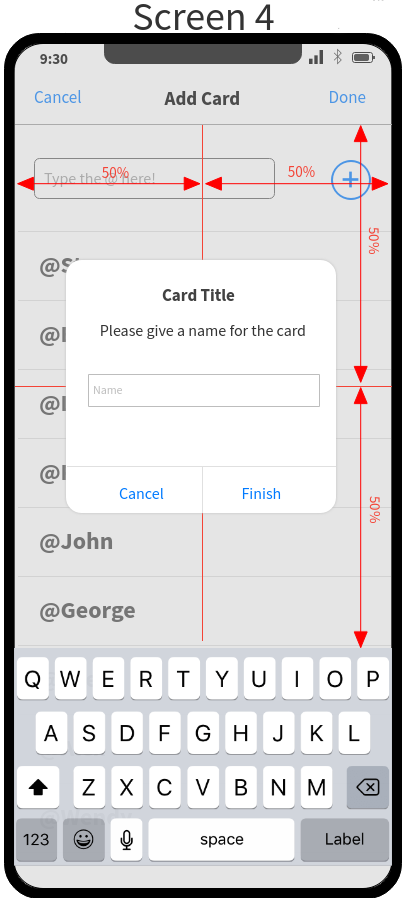


### Screen 2

Screen 2 in itself is just Screen 1 but with a different layout on the carousel cells. All the cards in the carousel get tinted red to signify that the user is in Edit mode and they also gain two new buttons. Edit mode itself is a toggle so pressing the “Edit” button once again will return the user to Screen 1. Whilst in editing mode creation of cards is disabled, therefore, the “+” button is greyed out. The Bin button on the carousel cell is used to delete a card if it’s no longer wanted. The Paper and Pencil button is used to edit the respective card; it will take the user to Screen 3 but it will already be prepopulated with the Twitter handles associated with the card the user selected to edit.

### Screen 3

Screen 3 is where the user can create cards to add to Screen 1’s carousel. Below the navigation bar there is a text field where the user can input Twitter handles to add to the new card. Once the user presses the “+” button to the right of the text field; the handle gets added to the table view below. If the user wanted to remove a handle after adding it, they can simply swipe left on the appropriate cell and a delete button will appear. This is in similar fashion to the IOS clock app when deleting alarms. In the navigation bar there are two buttons, “Cancel” and “Done”. If the “Cancel” button is selected the user is returned to Screen 1 without any changes to the carousel. If the “Done” button is selected the user is then taken to Screen 4.



### Screen 4

Screen 4 is simply an alert view over screen 3. In this alert view the user has two buttons as options. The “Cancel” button dismisses the alert and allows the user to add more Twitter handles by returning to Screen 3. If the “Finish” button is pressed, then the user is taken back to Screen 1 with a new card in the carousel named what was entered into the alerts text field.

### Screen 5

Screen 5 is the first screen to display content received from Twitter. To navigate to this screen the user selects the desired card on Screen 1; the new screen then slides in from the right side of the screen. Once its fully in view a loading animation will be played whilst the data is being fetched. The Tweets and the images themselves will be fetched separately to each other. This is mainly so that the app can quickly populate the table view with raw text, so the user isn’t stuck on a loading screen for a long time. As the images get returned from their requests, their corresponding rows in the table view will get updated to add the images into them.

This screen will request and load the latest 100 Tweets from the Twitter handles and populate the rows of the Table view. Once the user scrolls down around 80 Tweets, another request will be sent to get the next 100 Tweets and so on. The Table view’s rows will dynamically scale depending on the size of the content. For example, the row will be significantly larger if it contains an image in comparison to a Tweet that contains only 4 words. In order for constraints to be laid out correctly there will be a minimum of 150 pixels height for a cell without an image and 350 pixels height for a cell with an image.

By using the transition of sliding into view the impression has been given that this screens view is simply stacked on top of the previous one. Therefore, to navigate back to Screen 1 the user dismisses this screen simply by doing a Swipe gesture from the left side of the screen to the right. The user can also view replies to a Tweet by pressing a Tweet in the timeline, this will then take them to Screen 6.

### Screen 6

Screen 6 in essence is very similar to Screen 5 with a few key differences. The first tweet in the table view is populated with the Tweet previously selected in Screen 5. The first cell follows the same constraint rules as it would in Screen 5. All the cells following the first one will be constructed of replies to the first tweet. This Table view will work similarly to the one in Screen 5 where the replies will dynamically resize depending on the content and gets populated with more content the further the user scrolls through it. This Table view has some differences to the one found in Screen 5 and all are in appearance.

Firstly, the backgrounds of reply tweets are tinted slightly grey. This is done to make it much more obvious to the user that they are looking at a reply to a Tweet as opposed to the root Tweet. This is further reinforced by the content in the cell also being indented further in towards the middle of the screen in comparison to the root Tweet. Finally, the user icons of the reply Tweets are also connected by a blue line to signify that all the replies are linked to the root Tweet and not actual tweets themselves. By tapping a reply in this screen, it makes a new view of this screen but with the reply as the root Tweet and the replies to the reply below it. This can be nested as many times as necessary. To return to the previous screen the user will Swipe from left to right to dismiss the current view and return to the previous in the stack.

## Technical Notes

There are a few technical choices that have been made in order to keep a consistent layout and a positive user experience.

Firstly, the app will be locked into a vertical orientation. This is done so that the table views for Tweets feel more natural to scroll through and there is more efficient use of space. The exception to this rule however will be when selecting images/videos in tweets. When viewing these the device will allow the user to go into a horizontal orientation but once they leave this screen the app will return to being locked to a vertical orientation.

Secondly, the app will support from the 5s to the Xs Max on launch with newer model support following shortly after launch. The initial version of the app will support IOS 12 with support for more versions coming shortly afterwards.

Finally, to use this app to its full capabilities an internet connection will be required. Whilst no user authentication is required; an internet connection is needed to fetch Tweets from the Twitter API.

# Requirements

To assist in the creation of this app, various existing resources will be utilised. This will help speed up production of the app and allow for quicker iteration of certain elements if changes that diverge from the design document are required. For example, if something is found to be counterintuitive in user testing then that aspect of the app can be adjusted.

## CocoaPods

CocoaPods are small libraries that extend upon the existing Swift codebase. Some of these make certain aspects of the app much easier to develop for as the interface is either built for you or acts as a wrapper around more difficult interfaces to simplify them. The ones planned for this app can be seen below:

### Alamofire

Alamofire is a networking library that wraps around Swifts networking protocol. It simplifies the GET and POST requests into easy to understand function calls that are chainable, which will save on a lot of code. Alamofire will be used primarily for the GET requests received from the Twitter API. This returns JSON strings which then can be processed to populate various parts of the app such as the Tweet Table views.

### SwiftyJSON

SwiftyJSON is a JSON library that allows for much more ease when working with JSON objects. The difficulty mainly arises as Swift is strict about types but by nature JSON files are quite the opposite. SwiftyJSON allows you to construct a JSON object which you can simply access elements in the JSON object using [] and either a key as a string or an index if it’s an array. This will make handling the data received back from Twitters API much each as SwiftyJSON can serialize the response into an object which we can then just query for the data we require.

### TweenKit

TweenKit is a simple but powerful animation library that lets you animate various screen elements with different speeds, easings and the potential for scrubbing between animations. This will make elements on screen have the ability to transition with a nice flourish.

### Transition

Transition is a library which allows for custom animations to be used to traverse between screens in an app. This pod will be leveraged in screen transitions as much of the navigation in the app is based around gestures for the best possible user experience. These custom transitions can be used to give affordance to the user on which gesture they should use to navigate around various points of the app.

## Twitter API

The Twitter API will be a key part of the app as all the timelines will be populated via their API. There is one main request that will be followed every time the app is launch and then custom requests based upon the data being fetched.

### Authentication

For near all of the requests made to the Twitter API; an oauth2 token is required. To get this token the app will build a post request; based of the consumer key and secret found in the Twitter API developer accounts page. The token from this response will then be cached for the rest of the app’s lifetime. If the token becomes invalidated over the app’s lifetime; the app will request for a new token using the same process previously described. All this will be done without the user’s intervention on app start-up.

### Users Tweets

For creating the timelines based of a specific set of predefined handles the “GET statuses/user\_timeline” get request will be leveraged. This header of this request simply takes a Twitter handle and the number of tweets to be requested. This request then returns a JSON string with the specified number of tweets and their associated data. This request will be used for each handle specified in the specific set of handles. The results of these will then be collected ready for viewing in a table view.

### Replies to a Tweet

To get the replies to a Tweet a slightly more convoluted process has to be undertaken. The “Standard search API” get request will be used as there isn’t a specific one for getting replies to a Tweet. Firstly, in the header for the request the “since\_id” parameter will be given the id of the Tweet we are looking for replies to and the query “q” will be the Twitter handle the Tweet belongs to. This gives all the tweets related to that Twitter handle from their Tweet onwards. Once the reply to the request is received all the Tweets in this response JSON will be interrogated. Each Tweet will contain an “in\_reply\_to\_status\_id”; if this id matches the id of the initial Tweet, the reply will be added to a list of reply Tweets. Finally, when all the response Tweets have been interrogated the array of reply Tweets will be returned for use in a table view.

The initial implementation of this will use the “Standard search API” which will only return results within the last 7 days, which may result in not all replies being returned. In the full version of the app the “Premium search API” will be leveraged, which allows for full retrieval of Tweets dating back to 2006.

## Assets

For this app not many external Assets will be required. The icons for Twitters retweet and like buttons will be utilised for displaying such figures in the table view cells. All buttons and other icons will be from the XCode defaults.

# Strategy

For develop of this app the timeline described by the Gannt chart below will be followed: