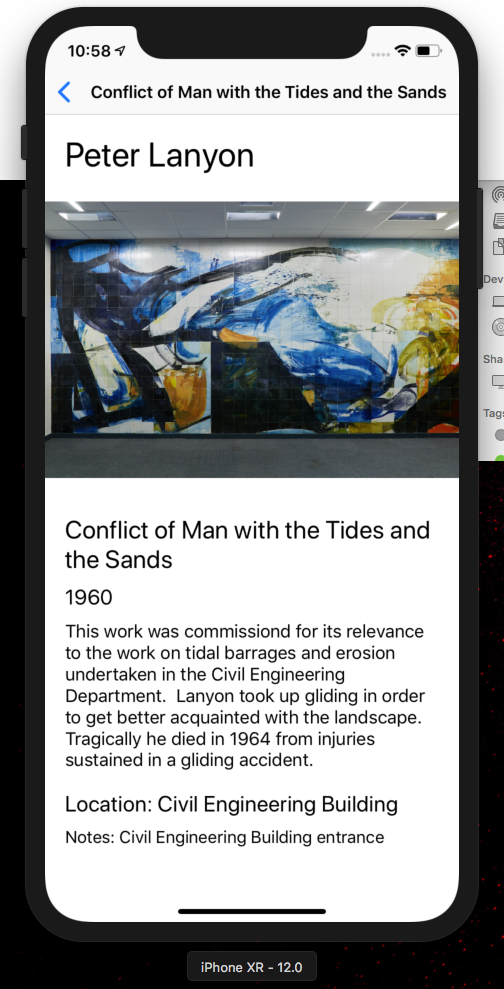
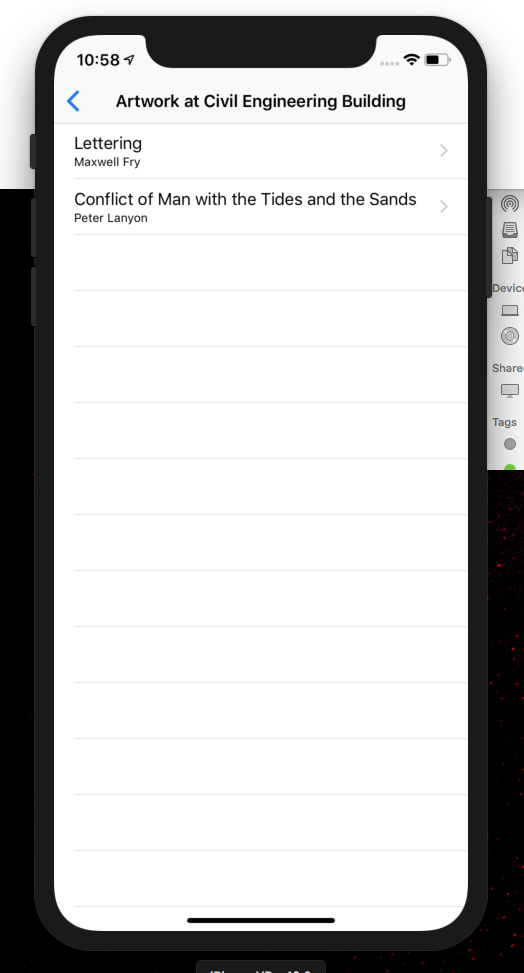
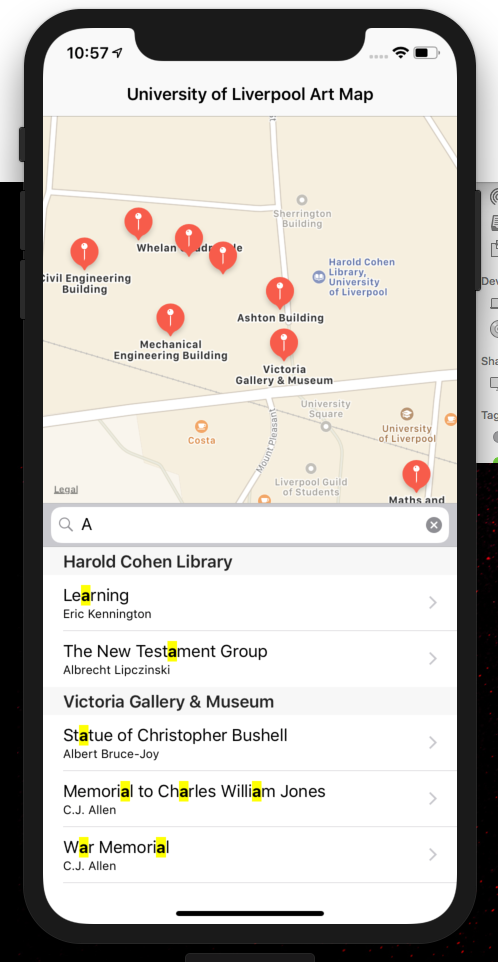
**COMP327 Assignment 2 Report**

**Ben Hawley - Student ID: 201113100**

**Application Views**

My application has 3 views: the main view, the artwork selector view and the artwork view.

****

**Main View**

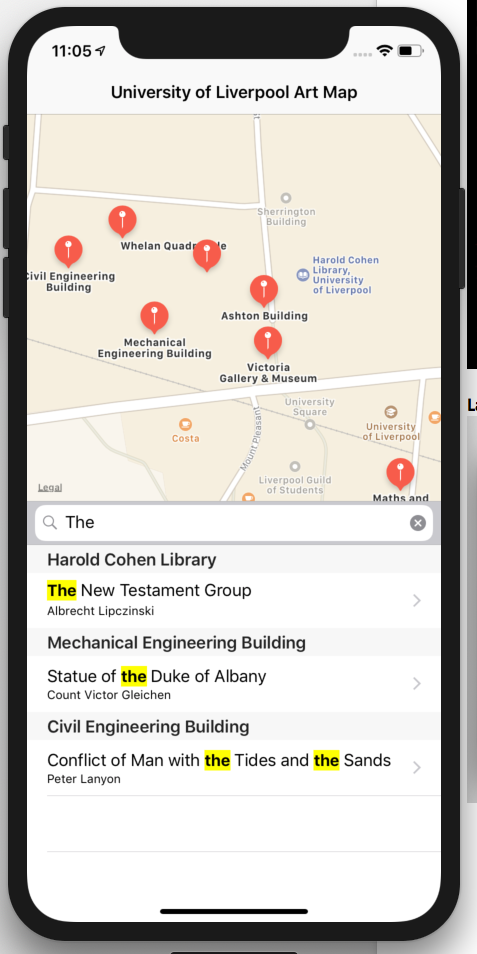
The main view consists of a MapView and TableView. The map view has various annotations of locations of artworks in campus, these are added using the longitude and latitude found in the artwork data. The initial positioning of the map is determined using the current location of the user and zooms in to show the surrounding streets. When an annotation is clicked, one of two things can happen. If there is only one artwork at the location, we transition to the artwork view to display that artworks data. If there is more than one artwork we transition to the artwork selector view to select the artwork we wish to view.

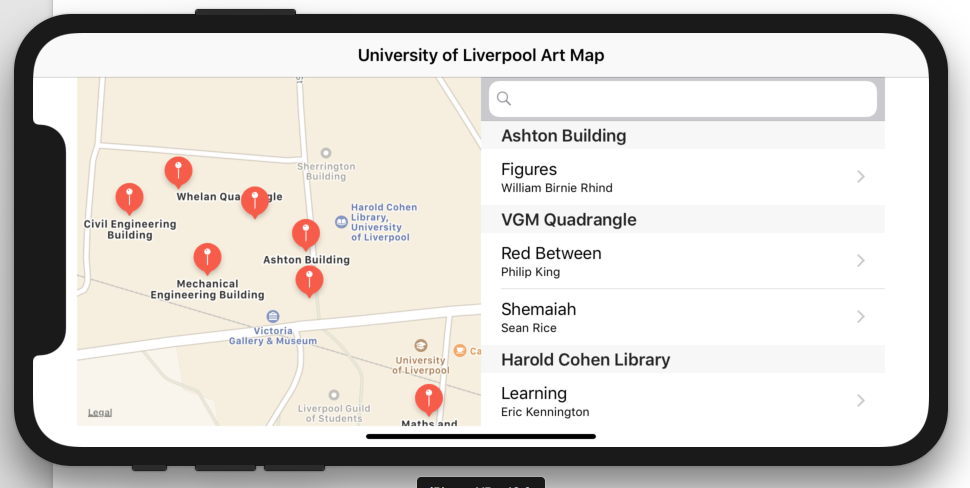
The TableView shows all of the artworks grouped in sections of their location, when an artwork is selected we transition to the artwork view. If we know the user’s current location, we sort the location headings by the distance from the user, i.e. the closest location to user. The table view also has a search bar for filtering the data.

**Artwork Selector View**

If the user selects a map annotation at a location which has more than one artwork, they are segued to the artwork selector view. This view is a UITableView which lists the multiple artworks at the location, the user can then select the artwork to transition to the artwork view.

**Alternative Layout**

****To satisfy item 4 of the additional marks, I expanded the layout of the main view to create to position the map on the left and table on the right when in landscape mode. I achieved this using the constraints and thresholds discussed in the 5th Lab Task.

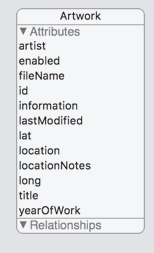
****

**Search Bar**

I have integrated a search bar on top of the table of the main view which allows users to filter the artworks in the table by title. I make use of regular expressions to find range matches between the search string and title string, and then filter out artworks which do not have a match, this does this in real time as the user is typing. When I initially implemented this, I found the experience difficult to see where the search text matches are. To improve this, I made used of attributed text strings for the titles with bold and yellow formatting indicating where the search string is matching the title, this improved the UX greatly and was a nice addition.

**Synchronisation and Caching**

The application fully supports caching and synchronisation and caching as specified in the additional assignment requirements. When the application loads, it checks **CoreData** for any cached artwork data, if there is data we will load it and set it to be in the cached Artwork array.

If there is no artwork cached, then we need to go to network and download the artwork JSON **without the last update parameter –** this will get all of the available data**.** If the download is successful, I set a key in **UserDefaults** named “lastUpdated” which is the value of the date we last downloaded the data. If there is artwork cached, there is no point in redownloading any artwork we already have cached, so we make use “lastUpdated” value in **UserDefaults** and append it as a search query on the Artwork URL endpoint, this results in all of the artwork which has been added or updated after the specified date to be returned.

Whenever we download artwork data, we need to cache it, the cached data is stored in core data, this image shows the attributes of the CoreData entity. When the data is successfully downloaded, I add the artwork data to core data. I remove the possibility of duplicate data by deleting items in core data which have a matching ID to any of the artworks we have just downloaded. When the data is downloaded, we add it to the table with the orginally cached data.

The app also caches the images by writing them to file, we don’t do this on start-up as downloading a large number of images on start up could require a lot of data. Instead, we only download and cache the image when the user navigates to an artwork view. In the download function, we initially check if the image name is a file in the directory, if it is, we opt to use this file rather than re-downloading it.