Student Number - 140150025

Notebook report

Assignment 2 – SEM2220 Mobile Solutions

# Introduction

This report is a write up of the second assignment for SEM2220 Mobile Solutions. The assignment was to create a native mobile application in either Java for Android or Swift for IOS. In short, the application needed to be able to take notes, search Guardian content using the Guardian API [] and link articles to the notes previously taken.

The application was written in swift using XCode. Some swift code was provided to help access the Guardian API. This code was adapted to aid in finding a solution to the assignment.

This report will cover the following:

1. How the assignment was implemented in Swift including problems encountered.
2. Testing procedure and results
3. Evaluation of the solution

# Implementation

I began by looking at the previous workshops that SEM2220 had provided to refresh my memory of how to use XCode and program in Swift. After scanning through the workshops and workshop solutions on GitLab [], I found the solutions that would aid me in completing the assignments based off of the function requirements shown in Appendix 1.

## Note Functionality

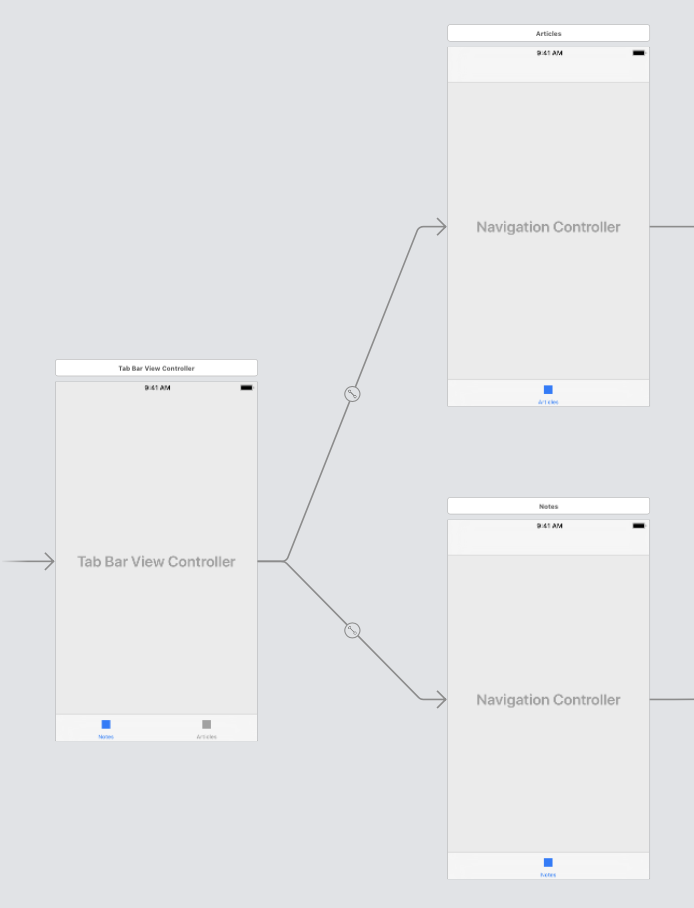


Figure 1 - Tab Bar Controller connecting to Two Navigation Controllers

I started by creating a tab bar view controller, a navigation controller being the initial view, with one of the tab views to be for creating and viewing notes. I used a table view controller to view the notes that were going to be created. I noticed I wasn’t able to add any bar button items on the navigation bar, I moved the navigation controller and tab bar controller around and was able to add the item then, this is why two navigation controllers have been used in the final solution as shown in Figure 1. A segue was created using the bar item to a new view controller that was to be used for creating new notes and storing them.

I decided to use Core Data as the storage solution it was covered in the IOS workshops and very easy to implement as it is built into XCode.

The implementation for creating a new note is straight forward as it is using techniques to store the data as learnt in the IOS workshops, the only difference is that later I would implement an edit note function that would be triggered if a *Note* variable was passed to the controller, if a *Note* wasn’t passed to the controller then a new note would be created.

When the notes were shown in the table view, I decided to create another table view controller that would be shown when a user clicks on the note itself. Clicking on the note would pass the details using a *prepare* function to the new controller. When creating the UI for the controller, I used a vertical stack and text views for the title and the table would be used to show the URL’s linked to the note.

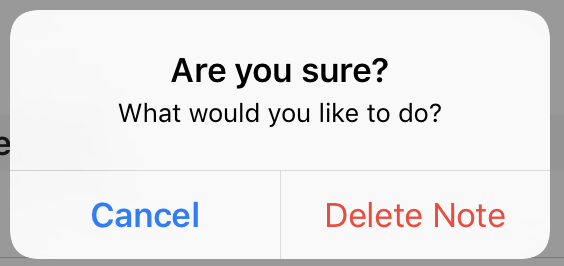


Figure 2 – UIAlertController to ensure user is selecting the correct option when deleting a note

Later into the assignment I added the ability to click and delete the URL’s listed. Clicking the URL would show the user the details of the Article linked to the note. Two buttons were added, one being *edit* which was mentioned earlier, this would use the create note UI and controller by passing a note to it. The other button being a delete note function. As the button was relatively close to the edit button, I added an *UIAlertController* following a tutorial [http://swiftdeveloperblog.com/code-examples/create-uialertcontroller-with-ok-and-cancel-buttons-in-swift/] online to make sure the user was selecting the right option.

One of the last requirements that was implemented was the search function. This was done by adapting the contents of an online tutorial [https://stackoverflow.com/a/41666125] to search through notes and titles. I found this easy to implement and worked very well.

Implementing the note functionality was fairly straight forward. The GitLab solutions helped massively and helped me have a better understanding of how Swift and XCode work.

## Search and Article Functionality

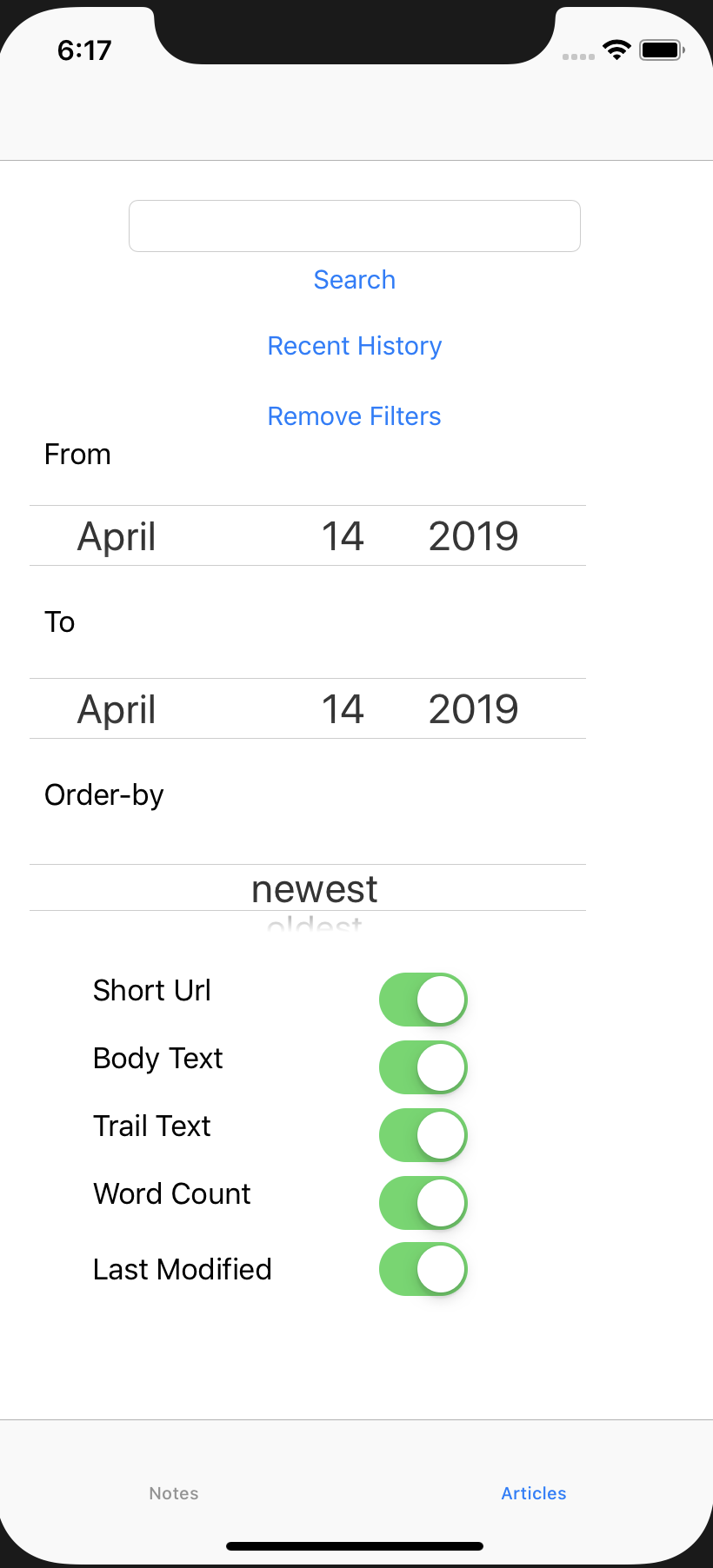


Figure 3 - Search UI

I used the code provided on Gitlab [] to communicate with the Guardian API. An example was shown on how to get a response from the API with a search term. I based the start of the search controller on this example. I created the UI, shown in Figure 3, with the additional filter controls as described in Appendix 1 FR-3a.

The UI used a spinning mechanism called a *Picker* which allows the user to pick an option from present values. To use the picker the values had to set programmatically, I used a tutorial online [https://codewithchris.com/uipickerview-example/] to help implement the necessary functions. The date picker was easy to extract data from as this didn’t require additional functions and setup.

Linking the filters to the search was easy as filter data types (*GuardianContentFilters*) just needed setting and passing to the provided Guardian client. The show-fields filter (triggered by using *UISwitches* shown in Figure 3) created a comma-separated list that would be passed to the client. The show-fields filter caused issues as when they were requested, an error would occur stating that the response was in the wrong format. This issue would soon be resolved when Neil Taylor released a bug fix for the provided code.

I created a class that would check for an internet connection and ping the Guardian API when the search button is pressed. A *Completion Handler* was used to decide what to do with the result. If no connection was present, it would alert the user to use the cached content, otherwise the API would be called, and the results would be cached and shown to the user. The search would also be stored in Core Data to be used for recent searches.

The Core Data *RecentSearch* entity takes the search term, the filters used, and the key used to store the results in the cache.

Caching the content saved from results was a difficult task for me. I began to look at methods of caching that didn’t use Core Data. I found *NSCache*, which I spent a considerable amount of time implementing, this worked but would throw errors when trying to retrieve data at certain times. It was later discovered that *NSCache* is only temporary, and the cache is removed once the application is closed. I then started to look at *NSKeyedArchiver* which can encode objects and store them into a file. I built a class that would take the *GuardianOpenPlatformResult* data type and the class used *NSCoder* to encode/decode the data. After a large amount of time was spent trying to get this to work, I resulted in editing the *Completion Handlers* in the code provided to pass the raw JSON and store this in the *UserDefaults.*

The recent searches would follow use the same functionality as the search function did. First checking if the phone has an internet connection and pinging the host. If the connection is up, the search will run again. If the connection is down, the user can click on recent searches that have the content cached. This was easy to implement.

The results of either a new search or of a cached search were then displayed on another table view controller. This was easy to implement as it had been done a number of times before. It allows users to click on the article and view the details about the article including the details that were requested from the *show-fields* filter. The detail page was implemented using an array containing the order in which the details about the article should be presented. This array is manipulated upon loading of the screen and the fields are displayed in their own sections. The detail page included the ability to click the URL section and open the link within a browser.

A button was added to the article detail page that would allow the user to assign an article to a selection of notes. I adapted the example code provided on GitLab that allowed for a multiple selection list on a table view, but the majority of the code remained the same.

When an article is assigned to a note, all the content is saved into Core Data and a Many-To-Many relationship is used between the article and note. Before creating a new article, it is checked to see whether the article already exist in Core Data.

As the information about articles can only be kept for 24 hours, as per requirement FR-5 shown in Appendix 1 it was necessary to create a background function that would check the *RecentSearch* and *Article* entities in Core Data to check for any content that needs to be removed. If a *RecentSearch* held the content for 24 hours, then the *cacheKey* attribute would be removed from the record and the cache would be removed from the *UserDefaults*. The same would happen to an *Article* record in that the attributes would be cleared except the *WebUrl* and *WebTitle*. To create this background function a *Timer* was used in the A*ppDelegate*.

# Evaluation

Overall, I believe that the project has be successful, the majority of functional requirements have been implemented, excluding FR-3b. The application is easy to use and is intuitive.

I enjoyed using XCode for the first time and found that people have shared similar problems and solutions to those that I have faced during the assignment.

I believe that my code implementation is repetitive at times and could be improved by creating classes to handle functions. An example of this would be to create a wrapper for the Core Data functions that would just use one declaration of the context, rather than have one in the majority of each view controllers.

This assignment has been beneficial to me as I this is my first native mobile application and has taught me:

1. How to develop native mobile applications
2. About the life cycles included in mobile applications (*viewdidload*, *viewdidappear*, etc.)
3. How data is managed in native applications (*Core Data*, *UserDefaults*)
4. How to create an intuitive mobile design.

Comparing the implementation, test results and what I’ve learnt with the learning objectives from the assignment as shown in Appendix 2, I can confirm that the assignment has been a success and I have learnt a lot.

As I think this was success and I am happy with the implementation to a degree, I would award myself a 2(i).

# Testing

See Appendix 1 for a detailed description of the requirements for the assignment.

These tests were completed using the simulator and iPhone provided by Aberystwyth Computer Science.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TestId | Testing Requirement | Test detail | Pass/Fail | Comments |
| 1-1 | FR-1 | A user can enter the title and detail for a note. | Pass. |  |
| 1-2 | FR-1 | A note has a date created and last modification date | Pass. | Printed on console when note was created. Also seen in Core data. |
| 1-3 | FR-1 | A user can edit an existing note. | Pass. |  |
| 1-4 | FR-1 | A notes last modified date gets updated when it is edited. | Pass. | Notes are displayed in last modified order in the table view |
| 1-4 | Additional functionality | A user can delete a note by accepting to delete it. | Pass. |  |
| 1-5 | Additional functionality | A user can cancel the deletion of a note. | Pass. |  |
| 1-6 | Additional functionality | A user can cancel creating a note. | Pass. |  |
| 1-7 | Additional functionality | A note gets added to the list of notes when saved. | Pass. |  |
| 2-1 | FR-2 | A user can search for content using the search term. | Pass. |  |
| 2-2 | FR-2 | When searching for articles it is possible to select a number of links and associate them with notes. | Pass/Fail. | While it is possible to link many articles to many notes, it is not possible to do so by selecting a number of links. It is done by selecting an article then selecting the notes you want to assign these to. |
| 2-3 | FR-2 | A user can assign an article to a note. | Pass. |  |
| 2-4 | FR-2 | A user can assign an article to many notes. | Pass. |  |
| 2-5 | FR-2 | The app stores the URL and the title. | Pass. |  |
| 2-6 | FR-2 | The app stores additional content about the article. | Pass. |  |
| 2-7 | FR-2 | Clicking “assign to note”, should bring up all the notes present. | Pass. |  |
| 2-8 | FR-2 | Clicking “assign to note”, then clicking cancel should stop the action. | Pass. |  |
| 3a-1 | FR-3a | As a minimum, the user can enter a free-text term as a search term and the user can specify the date-from, date-to, order-by and show-fields filters that are passed in the query to the API. | Pass. | The user can specify all this. |
| 3a-2 | FR-3a | Changing the “date from” and clicking search should change the search results. | Pass. | Verified by printing the results. |
| 3a-3 | FR-3a | Changing the “date to” and clicking search should change the search results. | Pass. | Verified by printing the results. |
| 3a-4 | FR-3a | Changing “Order-by” and clicking search should change the search results and order them in the specified order. | Pass. | Verified by printing the results. |
| 3a-5 | FR-3a | Selecting ShortUrl should include it on the Article detail page. | Pass. |  |
| 3a-6 | FR-3a | Selecting BodyText should include it on the Article detail page. | Pass. |  |
| 3a-7 | FR-3a | Selecting TrailText should include it on the Article detail page. | Pass. |  |
| 3a-8 | FR-3a | Selecting WordCount should include it on the Article detail page. | Pass. |  |
| 3a-9 | FR-3a | Selecting LastModified should include it on the Article detail page. | Pass. |  |
| 3a-10 | Additional functionality | Clicking apply filters allows user to apply filters, showing the filters view. | Pass. |  |
| 3a-11 | Additional functionality | Clicking apply filters allows user to remove filters, removing the filters view. | Pass. |  |
| 3b-1 | FR-3b | Optionally, for a more advanced project, the user can use Boolean operators, ( ) brackets and “ “ quotations to enter more advanced queries. This could be specified as a string, but a better implementation could provide a way to build that query using the UI in the app; this could help reduce the errors in more complex queries. Further, a more advanced project could support filters for production-office, section and tag. | Fail. | Not implemented. |
| 4-1 | FR-4 | When a user searches a term, it is added to recent searches. | Pass. |  |
| 4-2 | FR-4 | When a user searches a term with filters, the filters are saved and saved to recent history. | Pass. | Filters printed to console. |
| 4-3 | FR-4 | When a user clicks on a recent search, the search is run again. | Pass. |  |
| 4-4 | Additional functionality | The recent searches are shown in most recent order. | Pass. |  |
| 4-5 | Additional functionality | When using an old recent search, it gets moved to the top and not re-added. | Pass. |  |
| 5-1 | FR-5 | When a term is searched the result gets added to user defaults and the key is stored. | Pass. | Printed user defaults using the keys on console. |
| 5-2 | FR-5 | When an article is saved, the additional details get stored. | Pass. |  |
| 5-3 | FR-5 | When the system can’t access the API, cached content can be used through Recent searches. | Pass. | Changed API key to get an invalid response. |
| 5-4 | FR-5 | Searches without cached content can’t be searched for. | Pass. | User told to use cache content. Marked by a label on cells. |
| 5-5 | FR-5 | Cached search results older than 24 hours gets deleted. | Pass. | Shortened time to 5 minutes rather than use 24 hours. Search cache was removed for searches older than 5 minutes. |
| 5-6 | FR-5 | Article information except web-title and web-url are removed from the Core Data records if the record hasn’t been accessed in the part 24 hours | Pass. | Shortened time to 5 minutes rather than use 24 hours. Search cache was removed for searches older than 5 minutes. |
| 5-7 | FR-5 | **Check with no internet – not pinging**. |  |  |
| 6-1 | FR-6 | A user can view a note with its details. Details include title, notes and linked content (URL) | Pass. |  |
| 6-2 | FR-6 | A user can click one of the article URL’s linked to a note and the details will be shown. | Pass. |  |
| 6-3 | FR-6 | Clicking the URL of the article detail page will take the user to the site via a browser | Pass. |  |
| 7-1 | FR-7 | A search bar is available to the user | Pass. |  |
| 7-2 | FR-7 | If the contents of the search bar match the contents of a title, the note is displayed in the table. | Pass. |  |
| 7-3 | FR-7 | If the contents of the search bar match the contents of a note, the note is displayed in the table. | Pass. |  |
| 7-4 | FR-7 | If no contents match the search term nothing should be shown. | Pass. |  |
| 7-5 | FR-7 | Emptying the search bar will show all notes. | Pass. |  |
| 7-6 | FR-7 | The search will filter as the user types new characters into the search bar. | Pass. |  |
| 8-1 | Additional functionality | Navigation controls (back) work on each screen, leading back to either the search screen or table view showing notes. | Pass. |  |

# Appendix

Appendix 1- Functional Requirements

|  |  |
| --- | --- |
| Requirement No. | Requirement |
| FR-1 | A user can enter notes about a topic. A note has a title, a date it was created, a latest modification date and the text for the note. |
| FR-2 | A user can search for links to content for specified search terms the Guardian’s Open Platform API. From the returned results, it is possible to select a number of links and associate them with notes that are created in FR-1. The app should store the URL and the title for the links. |
| FR-3 | A user can enter search terms and apply filters to configure what is searched for and what is in the results that are returned.   1. As a minimum, the user can enter a free-text term as a search term and the user can specify the date-from, date-to, order-by and show-fields filters that are passed in the query to the API. 2. Optionally, for a more advanced project, the user can use Boolean operators, ( ) brackets and “ “ quotations to enter more advanced queries. This could be specified as a string, but a better implementation could provide a way to build that query using the UI in the app; this could help reduce the errors in more complex queries. Further, a more advanced project could support filters for production-office, section and tag. |
| FR-4 | A user can look through lists of recent searches for content in the API and run the search again. |
| FR-5 | To support offline access to information, the app can cache detailed article information, e.g. headline, body text, standFirst text, trailText and wordCount. These details should only be kept for up to 24 hours. The app needs to delete content older than 24 hours. |
| FR-6 | A user can view a note and its linked content from the API. By selecting one of the linked content items, the content is displayed in a web browser. |
| FR-7 | A user can search for notes that match specific text. |

Appendix - Learning Objectives

|  |
| --- |
| 1. Learn how to develop native mobile applications and running them on an emulator or simulator. 2. Understand and use the different choices for the top-level of a user interface and build a user interface. 3. Learn about data storage on mobile devices using SQLite, ROOM or Core Data and how to use these in an app. 4. Learn parts of the platform specific APIs. 5. Link the app to an API via the network. |