CMP3035M Cross-Platform Development Code Wallet

Contents

Concept

Overview

Code Wallet is a barcode storing app with geolocation-based suggestions. This is ideal for loyalty cards, as the app will recommend the correct loyalty card for you to use based on your location and proximity to the shop. The app also supports custom barcodes, as well as an effective scanning solution for adding your existing barcodes to the app.

The QR code to download the app can be seen in Figure 1.

Figure 1

Requirements

The main usage scenario for this app is to let users store barcodes such as loyalty cards in one place on their phone. Users should be able to add barcodes by either scanning one with the phone camera, or by entering details manually.

Each stored barcode should be able to be linked to a geolocation chosen by the user. By assigning geolocations to each barcode, the app will be able to sort stored barcodes by distance from the user. This means users can very easily find the barcode they are most likely to be looking for, as it will be at the top of the list most of the time, due to it being the nearest stored location to the user.

The target audience of the app would be anybody that needs to access barcodes such as loyalty cards based on locations. While it should function properly without setting locations for barcodes, the app would not be able to suggest the most suitable barcode to the user.

The requirements decided on for the app are as follows:

- Users can store barcodes and view them on request
- Users can set geolocations for each barcode
- The list of saved barcodes will be sorted by distance, with the nearest location showing at the top
-

Competitors

During my research, I could not find any barcode storage apps that would suggest barcodes to the user based on geolocation. This was partially what inspired the creation of this app, allowing it to fill a gap in the market while also improving on other features from other apps. However, there were plenty of other barcode storage apps that didn't implement geolocation functionality.

Stocard

One of the more popular barcode keeping apps on the iOS app store is Stocard. This app provides a few advanced features, such as passbook support and retrieval of images for popular supermarkets.

One of the features of the app is the advertisement of offers for supermarkets you have loyalty cards for. While this can be a useful feature, for users that do not want this feature, it can be intrusive with offers being suggested, when the user might just want to bring up their loyalty card.

The passbook support within the app can be a very useful feature, allowing access of Stocard saved barcodes without going into the app. However, this means the user must manually look through the stored barcodes to choose the correct one. The app being proposed would save users from having to do this by suggesting the correct barcode to use based on location, a feature which Stocard does not offer.

One problem with the passbook option in Stocard is the ability to only see 1 card at a time. If the user has a lot of cards, it would be very tedious to flick through all the stored cards to find the one needed. This can be seen in the screenshot in Figure 2.

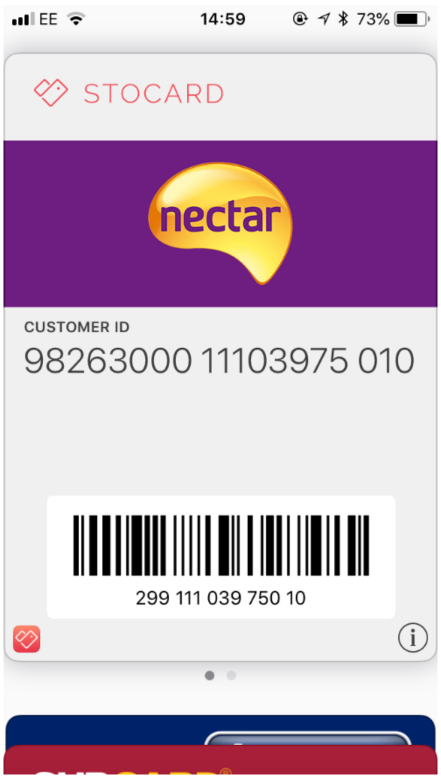


Figure 2

One of the downsides of using Stocard for saved barcodes is the method to bring up a full screen view of a stored barcode. Instead of a single click from the main menu like in the proposed app, users of Stocard need to click once to go into the barcode's menu, and then click again to view it in full screen. This slows down the user experience and makes bringing up loyalty cards a much less elegant experience.

Card Mate

Card Mate is another competing app and has passbook support like Stocard. This brings with it the disadvantage of not suggesting barcodes to users, resulting in users having to manually filter through their saved barcodes. This is even more frustrating as the user interface of the app could use a lot of improvement. Each saved barcode takes up most of the screen, resulting in a very limited number of barcodes being seen at a time. This issue can be seen in Figure 3, in which only 2 barcodes can be seen at once due to oversized tiles and adverts at the bottom.

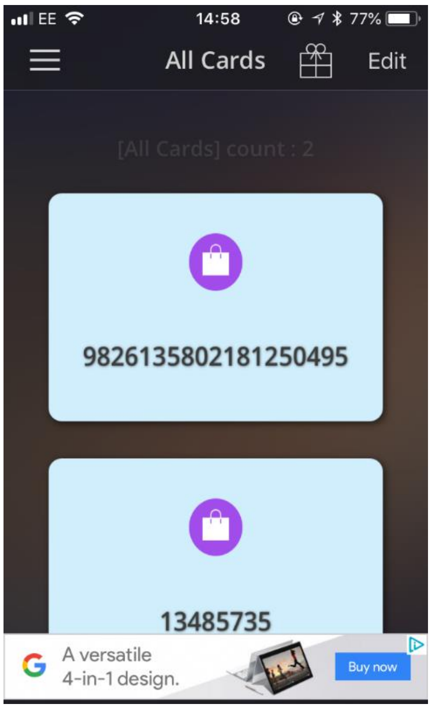


Figure 3

As with Stocard, this app also does not have a single press way to view the barcode in a full screen view. This would slow down the user experience during a mobile moment, making the app less convenient than using a physical loyalty card. With the proposed app solution, the mobile moment would be a very smooth and elegant action, opening the app and clicking the top item.

Another big issue with Card Mate is the use of advertisements that pop up and cover the entire screen, making the user experience very uncomfortable, as the user would need to close out of the advert to carry on using the app.

Prototyping

Stage 1

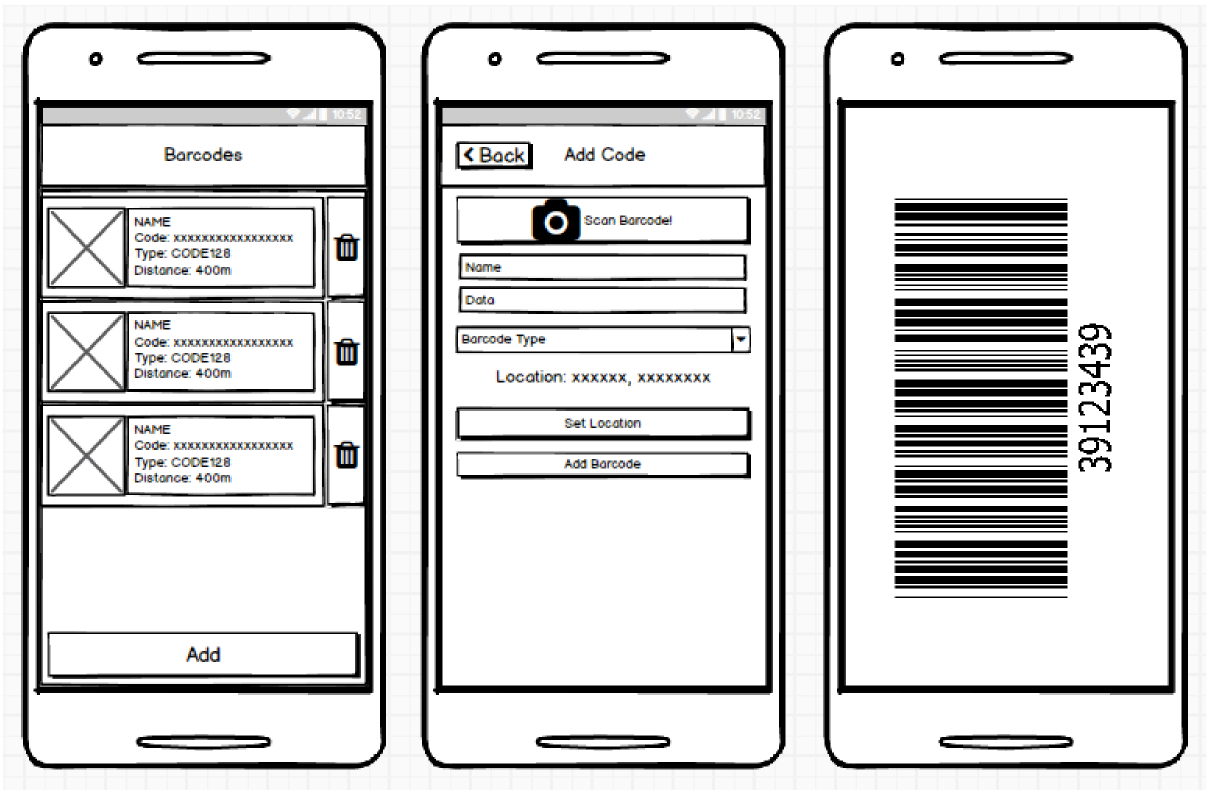


Figure 4

The initial prototype fulfilled all the requirements of the app and can be seen in Figure 4. The main screen would have a list of all the saved barcodes, with a button to let the user delete the saved code. When a user would click on the main section of the list item, it would open the screen with the full screen barcode. While on this screen, the app would turn the brightness to full to improve the chances of the barcode being read by scanners.

The main screen would also have a button at the bottom for adding barcodes. This button would open the add barcode screen. On the add barcode screen, the user would have the option to fill in the barcode details manually, such as name, data and barcode type. Alternatively, the user could press the scan button, opening the camera, and would autofill the data and type fields if a valid barcode was scanned. These 3 entry fields would be part of a form and would be required in order to add the barcode.

The set location button would get the user's current location and enter it into the location text area. If the barcode is added, the location data would be saved with it. This would not be required, as not all saved barcodes would need to be tied to a geolocation.

Stage 2

After creating the initial prototype, feedback was acquired from potential users about the interface design. One major concern raised was the fact that users would have to be at the location desired for the barcode to set it up correctly. To remedy this, the second iteration would use the Google Places API, allowing users to search for places within Google Maps, and select the appropriate location. The Google Places API would be used to display a suggestions dropdown while the user is typing their request. This system would be accompanied by a Google Maps widget, displaying the chosen location on the map, and letting users look around the map to ensure the correct place is chosen.

The next concern was that accidentally deleting stored barcodes was too easy, as the delete button was on the main page, very close to the touch area for displaying the barcode in a full screen view. To improve on this design flaw, the new location selection interface and the delete button were moved into a new “Edit” page. This page would be used to modify the location of a saved barcode or to delete existing barcodes if they were no longer required.By moving this task to its own page, it follows the recommendation to “Prioritize one primary action per one screen.” (Studio by UXPin, 2018). This page would be accessible from the main page, as the delete button would be replaced by a new settings button, directing the user to the new edit page.

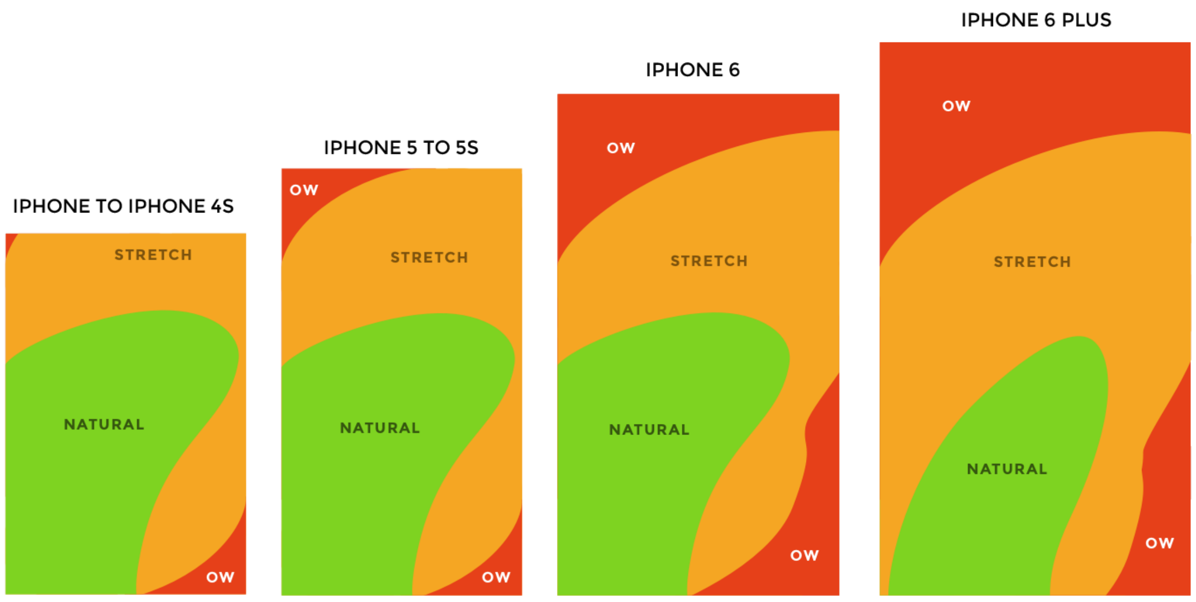


Figure 5 - (Scott Hurff, 2018)

While the button for adding codes was in a very usable position, as can be seen in Figure 5, it was using up valuable screen space on smaller devices, restricting the number of barcodes that could be seen in the list at any one time. The main suggestion was to put the button in the top toolbar of the app. While this would be in the region of the screen that requires stretching your thumb, the add code button would be rarely used, as most users would setup their loyalty cards and leave them configured for a long time. For this reason, the button was added to the top right corner of the screen, improving the amount of space available for list items being displayed.

The result of these proposed changes can be seen in Figure 6:

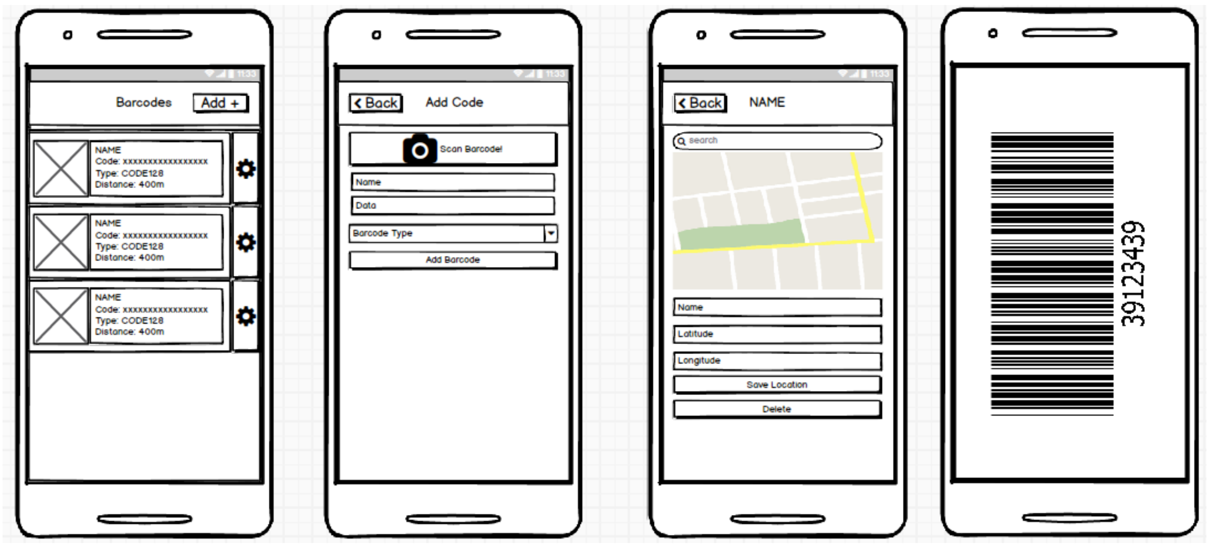


Figure 6

Stage 3

After producing a prototype for the second stage, more feedback was acquired. This time, the feedback was about very minor problems, as the major issues were resolved. One point brought up was the difficulty to look for a specific barcode if the user was not at the location. While the barcode could be found by scrolling through and looking for the name, this can be impractical when there are a lot of saved barcodes. To remedy this issue, a search bar was added, allowing users to search through the list of barcodes effectively.

Another concern raised was the proximity of the delete button to the save location button within the edit page. While moving the delete button to a different page helped, it was still easy to accidentally press it, removing the barcode permanently. This issue was improved upon by making the button red in colour, warning users of a potentially destructive action. As well as this, a confirmation dialog was added, ensuring users want to remove the barcode from storage.

The results of these changes can be seen in Figure 7:

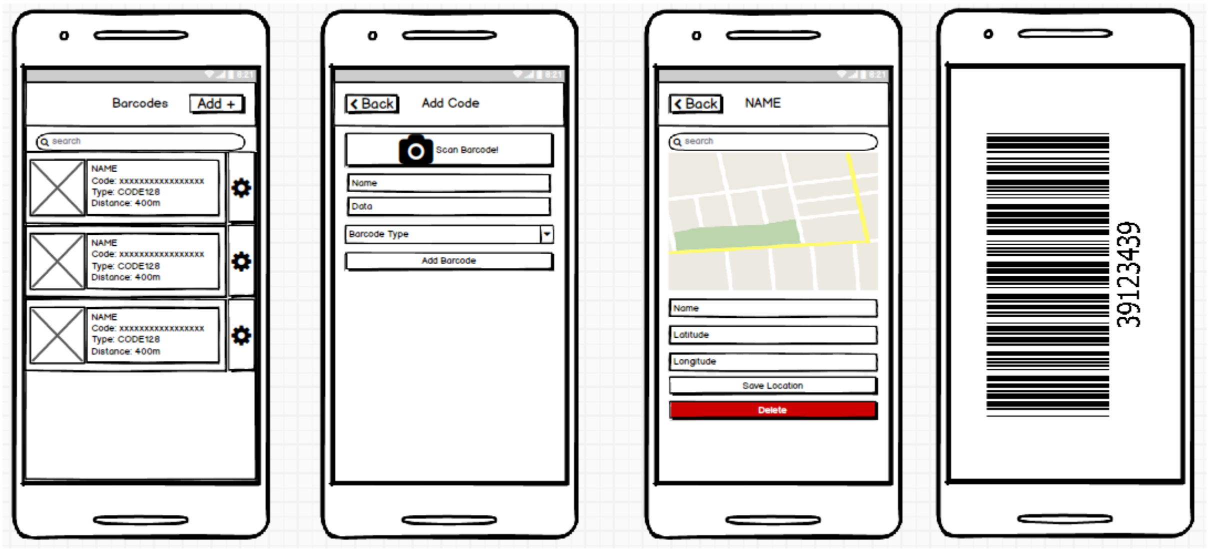


Figure 7

Final App

One important part of the app was storing the data of the barcodes. At first glance, local storage would be the obvious choice, as barcodes would not have to be saved as images, but as raw data values, due to the app generating the barcode image at runtime. However, "because of the non-persistent property of LocalStorage in the WebView of Android and iOS. In iOS stored data from LocalStorage can be removed by the OS, when running out of memory." (Cordova plugin NativeStorage, 2018). This resulted in the use of a plugin to store data locally instead, as data loss is very bad for user experience and can deter users from the app.

As one of the main recommendations for mobile app design is to "consistently refer to the native OS design guidelines first for maximum quality" (Designing for Mobile Apps, 2018), the app features a swipe to go back gesture on all pages except the home page. This is very important for iOS devices as most native iOS apps respect this design philosophy. Not having this feature could deter iOS customers due to the inconsistency between this app and most other iOS apps.

Geolocation Sorting

The geolocation feature lets users store a geolocation for each barcode. This allows the app to sort the saved barcodes by distance, showing the nearest barcode locations at the top of the list.

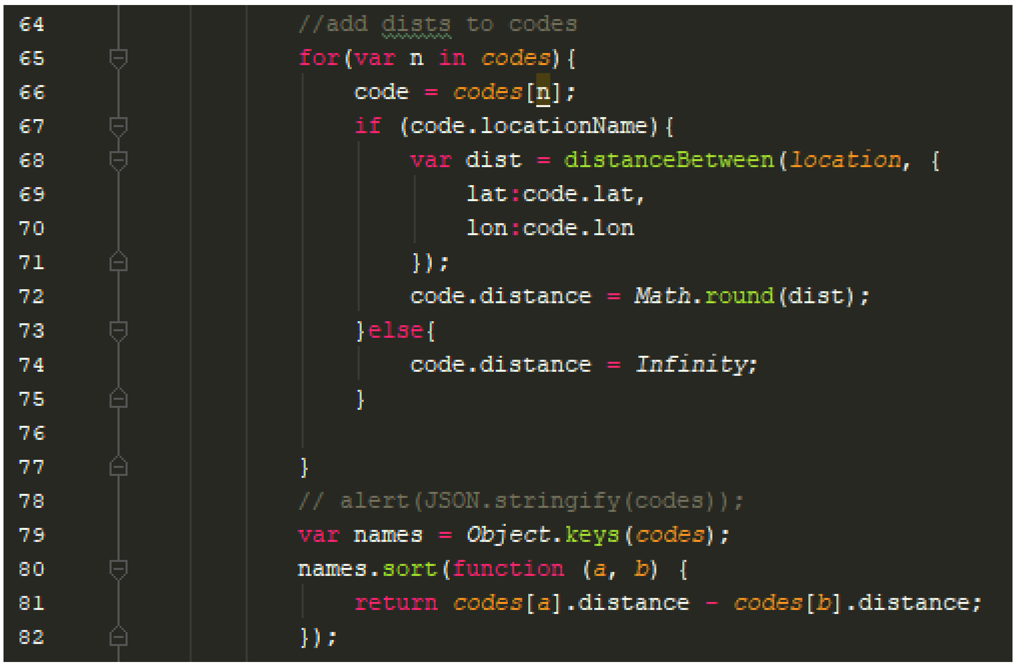
User Experience

"The less friction and confusion users have when interacting with an app (e.g. the cognitive load), the better the chance that app stays around." (Studio by UXPin, 2018) The sorting features aims to improve user experience by reducing work from the user's point of view. By automatically keeping the list sorted by distance whenever the user opens the app, the only interaction the user will have with this feature is selecting the top item in the list.

Because of this simplistic user interaction, most times when the app is used, the user will open the app, press the top item in the list, and will have the correct barcode displayed. This results in a very elegant user experience, as can be seen in the accompanying video, in which an example scenario took 10 seconds to unlock the phone, open the app, scan the correct barcode at the till, and close the app. This lets the app fit seamlessly into the mobile moment of paying for groceries and swiping a loyalty card at a checkout. The user retrieving their phone is usually not a concern as a lot of users would be getting their phone to pay for their shopping using Android Pay or Apple Pay.

Technical Perspective

To sort the list items by distance from the user, the app first acquires the user's current GPS coordinates using the Cordova geolocation plugin. By using the coordinates of the user and the barcode being considered, a mathematical formula can be applied to get the direct distance to the location in meters. The distance calculated will be the direct distance, not the walking distance. This is not an issue as the app only needs to use this value for proximity calculations, not walking distance planning. For the use of the sorting algorithm, any barcodes with an undefined location will have their distance set to Infinity, this can be seen in the code snippet (Figure 8). This ensures that they will be at the bottom of the list, and not taking up space at the top where the recommended barcode will be positioned.



```
//add dists to codes
for(var n in codes){
    code = codes[n];
    if (code.locationName){
        var dist = distanceBetween(location, {
            lat:code.lat,
            lon:code.lon
        });
        code.distance = Math.round(dist);
    }else{
        code.distance = Infinity;
    }

}
// alert(JSON.stringify(codes));
var names = Object.keys(codes);
names.sort(function (a, b) {
    return codes[a].distance - codes[b].distance;
});
```

Figure 8

Google Places API

User Experience

The Google Places API is utilised in the edit menu, for letting users select places on the map through a search box with suggestions. This vastly improves the user experience of setting locations as they can search for the store name of the loyalty card and add the appropriate store from a drop-down suggestions box. While adding new barcodes and setting the location is not a frequent activity for this kind of app, it is still essential that it is an easy task to perform for the user.

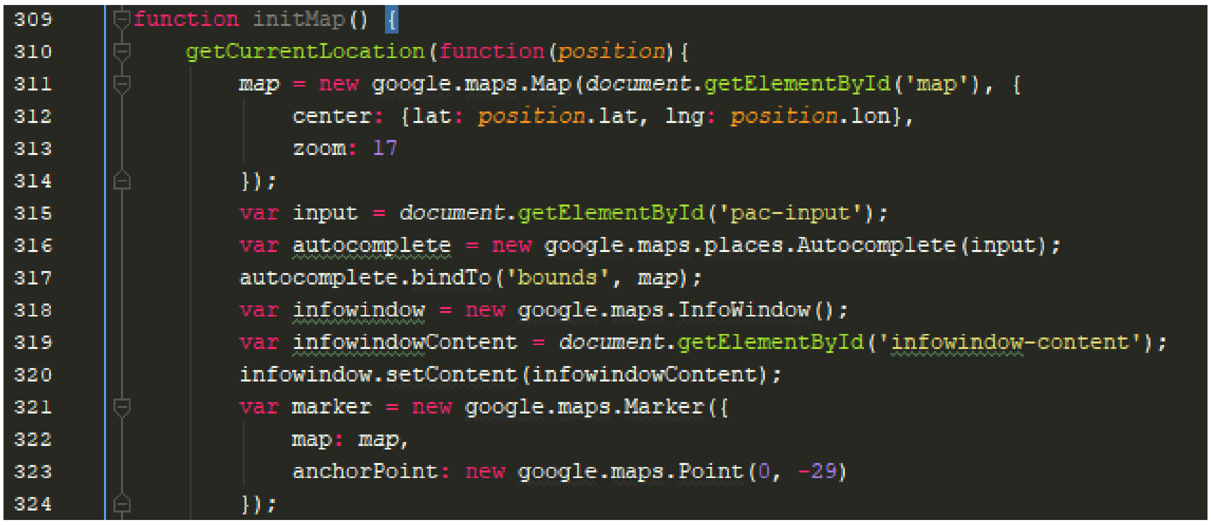
The Google Places API also presents a map for the user to look around in, as well as showing where their selected place is, helping the user ensure the location is setup correctly. This is important as a user not choosing the correct store would result in the barcode not being suggested when the user is at the store.

Technical Perspective

From a technical perspective, the app utilises the API whenever the user changes the text in the search box. When the value is changed due to the user typing, the app makes a request to Google with the contents of the search box and receives a list of suggestions for the user. These appear in a dropdown underneath the search box. When a user selects one of these, the name and location of the place are extracted and placed in the input boxes below.

Due to the dependency on an API, this operation requires an active internet connection and would not be functional without one. This is not a major issue as the user is not going to require this functionality very often, only when adding new cards to the storage. However, in the situation where a user doesn't have an internet connection and wants to add a location, the input boxes will accept custom inputs from the user. This allows the user to add location data without an active internet connection.

In the following code snippet (Figure 9), you can see the app initialising the map and centring it around the user's current location:



```
function initMap() {
    getCurrentLocation(function(position){
        map = new google.maps.Map(document.getElementById('map'), {
            center: {lat: position.lat, lng: position.lon},
            zoom: 17
        });
        var input = document.getElementById('pac-input');
        var autocomplete = new google.maps.places.Autocomplete(input);
        autocomplete.bindTo('bounds', map);
        var infowindow = new google.maps.InfoWindow();
        var infowindowContent = document.getElementById('infowindow-content');
        infowindow.setContent(infowindowContent);
        var marker = new google.maps.Marker({
            map: map,
            anchorPoint: new google.maps.Point(0, -29)
        });
```

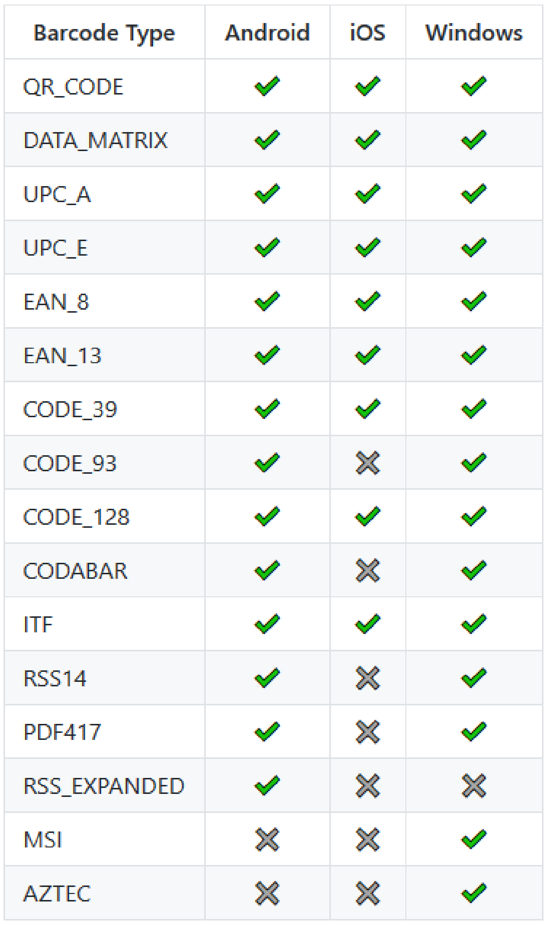
Figure 9

Reflection

One of the main issues noticed when taking a cross platform development approach was that plugins would not be the same across mobile operating systems. One example of this in this project was the barcode scanner plugin. The barcode types that were scannable varied depending on the operating system. The difference can be seen in Figure 10 to the right. This has resulted in a reduced amount of barcode types being supported by the app.

On the other hand, the availability of plugins that implement high level features for you is a great advantage of cross platform development. This project has shown that it can be easier to develop an app for a cross platform audience than it would be to develop the app natively for one OS.

The ease of implementation does come with disadvantages such as performance and access to operating system specific features. Initial loading times for cross platform apps can be significantly slower than native apps, resulting in a slightly disrupted user experience during the loading time.



Barcode Type	Android	iOS	Windows
QR_CODE	✓	✓	✓
DATA_MATRIX	✓	✓	✓
UPC_A	✓	✓	✓
UPC_E	✓	✓	✓
EAN_8	✓	✓	✓
EAN_13	✓	✓	✓
CODE_39	✓	✓	✓
CODE_93	✓	✗	✓
CODE_128	✓	✓	✓
CODABAR	✓	✗	✓
ITF	✓	✓	✓
RSS14	✓	✗	✓
PDF417	✓	✗	✓
RSS_EXPANDED	✓	✗	✗
MSI	✗	✗	✓
AZTEC	✗	✗	✓

Figure 10

References

Studio by UXPin. (2018). The Guide to Mobile App Design: Best Practices for 2018 and Beyond. [online] Available at: https://www.uxpin.com/studio/blog/guide-mobile-app-design-best-practices-2018-beyond/ [Accessed 3 May 2018].

Scott Hurff. (2018). Scott Hurff. [online] Available at: http://scotthurff.com/posts/how-to-design-for-thumbs-in-the-era-of-huge-screens [Accessed 3 May 2018].

Cordova plugin NativeStorage. (2018). TheCocoaProject/cordova-plugin-nativestorage. [online] Available at: https://github.com/TheCocoaProject/cordova-plugin-nativestorage [Accessed 3 May 2018].

Designing for Mobile Apps. (2018). Designing for Mobile Apps: Overall Principles, Common Patterns, and Interface Guidelines. [online] Available at: https://medium.com/blueprint-by-intuit/native-mobile-app-design-overall-principles-and-common-patterns-26edee8ced10 [Accessed 3 May 2018].