# Goldney AR

## Software Product Engineering 2019/20

A portfolio by Jordan Taylor
James Elgar
Zhuyuqing
Frankie Woolf

## Overview

As part of our second-year module in Software Product Engineering, we have been tasked to develop a software solution for a live client in Bristol. This project requires us to adopt real-world software product engineering (SPE) methodologies, in our case agile modeling, partnered with progress and task managing software, namely Jira, to develop our solution.

Our client for this project is The University of Bristol Estates Department team who are seeking us to develop a mobile application to enhance the experience of visitors to Goldney Gardens. In an effort to encourage members of the public to enjoy the gardens without a human tour guide, we envision a virtual tour app that will guide visitors through the grounds. The project should include a robust application, friendly user interface, and location-aware technology.

Our vision for this project should be interactive, allowing users to access educational texts, stories, and videos via tagged QR codes around the garden via their phone. We hope to collaborate with a group of History students and allow then to create description of sections of the gardens they find interesting and through the combination of the points of interest create interactive and engaging tours.

# Requirements

## Interacting Stakeholders

- Goldney App Administrator (**John the Warden** – Goldney Groundskeeper)
- Goldney Visitors using the App
    - Children / School Children (***Little Timmy*** – *A pupil on a primary school history trip*).
    - Teenagers (***Chad*** – *An older teenager visiting the grounds as part of a college trip or with family who is technologically literate*)
    - Adults (***Jane*** – *A 30-year-old woman living in the local area*)
    - Elder Generation (***Giles*** – *elderly history and national trust enthusiast who is less technologically savvy*)
- Residents of Goldney (***Students***- *first year students at The University of Bristol, typically aged around 18-20 who fall between our Teenager and Adult stakeholders. This stakeholder typically spends a lot of time on this land however, is unaware of its History*)

User stories have been created after our first meeting with our client. The stakeholders have been identified.

## Non-interacting Stakeholders

- Goldney Visitors not using the App - evidently, a supplementary tour device is voluntary, therefore we must ensure visitors who choose not to use our product are not distracted or troubled by the implementation of it. (
- Bristol Estates Team
- Goldney Gardeners
- Legistrators (GDPR)

## User Stories:

| As... | I would like to... | So that... |
|---|---|---|
| John the Warden | Create my own points of interest | I can describe new features in the garden |
| John the Warden | Change the information in existing points of interest | I can update our information in the event that the garden changes. |
| John the Warden | Let my staff and some select students add educational material | We can encourage inter-disciplinary teams. |
| Old Giles | Be able to view the grotto from outside | He does not need to risk walking down the potentially dangerous steps. |

| | | |
|---|---|---|
| Old Giles | Follow a prescribed route around the gardens | I can minimize my time walking and getting lost. |
| Old Giles | Listen to an audio description of the garden's points of interest | I can experience the gardens without having to read the text on my phone screen. |
| Chad | Scan QR codes quickly with my phone | I can instantly access data without having to manually enter information. |
| Chad | Load and display the content in points of interest while not connected to the internet | I do not have to use my limited mobile data |
| Little Timmy | Play an educational game in the garden | I do not feel bored. |
| Little Timmy | Be allowed to run around the gardens with my friends | There is an element of competition in the game. |
| Little Timmy | Be rewarded for learning about the history of Goldney | I have an incentive to keep learning. |
| Jane | | |
| | | |

Stories of Importance:

## Scan QR Codes with my phone (Minimum Viable Product):

The core and underlying idea of our application is to be able to view content about points of interest through scanning QR codes. This is why it is the primary objective for deployment within the minimal viable product.

A user should be able to scan and retrieve data about pre-defined points of interest around goldney gardens. For the minimum viable product, this data needn't be accessed online.

*Basic Flow*

1. The user opens the app.
2. The user is greeted with a splash page prompting them to open the QR scanner activity.
3. The camera is opened and the user may now scan a QR code.
4. Once a QR code is scanned and successfully read, the app will halt camera operation.
5. A window displaying the information pertaining to the scanned tile is brought up.
6. Closing the window will bring the user back to the camera.

*Alternative Flow*

1. The user opens the app but does not have a functional camera.

2 The user is greeted with a splash page prompting them to open the QR scanner activity.

3 The user instead chooses to simply view a list of all point of interest tiles.

## Create my own points of interest (Beta Version)

The ability for John the Warden to create his own points of interest is of high importance. Without this ability, data within the app will be static and cannot be maintained as Goldney grows and changes.

To accomplish this, John should have access to a separate administrator app. To add a point of interest, he would have to walk to the location and tap 'generate new point'. This will take him to a new screen in which a text box allows for him to add a description to the point, an image of the point may be taken, a QR code is generated, and locational GPS data is gathered. Once filled in, the app will push this information to a cloud database.

Alternatively, this idea could be scaled back to simply utilise a web client interface to manually add data to the cloud, thus removing the need for a separate app.

In order to implement this feature, we will require:

- GPS location libraries
- In-app QR code generator
- Cloud database
- Web database interface (eg Spring boot)

*Basic Flow (mobile app)*

1. The user opens the administrator app.
2. The user is greeted with a prompt to create a new tile.
3. Clicking this allows for a text description to be entered, while also generating a unique QR code.
4. Once data has been entered, the app attempts to send the entered information to the cloud data server.
5. A success dialogue is displayed to the user.

*Basic Flow (web interface)*

6. The user logs into a secure webpage.
7. All tiles and tours are displayed to the user
8. Clicking add tile allows for a new point of interest to be created. Input boxes are displayed to the user
9. A unique record is added to the cloud database
10. A unique QR code is generated for the user

## View the grotto from the outside (Final Release):

The lack of disabled access to the Goldney grotto is an issue the estates team wish to resolve. Since the grounds are grade II listed, it is not possible to retrofit the grotto with wheelchair access ramps or large handrails. As a result, it has been suggested to make use of an array of 360 degree cameras to capture the grotto's interior. These images should be viewable in-app upon scanning a specialised QR code by the entrance, allowing the user to see the inside through their phone's screen by orienting it where they wish to look. The 360 degree images could be strung together in a path, allowing the user to 'walk' between multiple views.

In order to implement this feature, we will require:

- Gyroscope sensor libraries
- 360-degree image viewing libraries
- QR Code Scanning

FUNCTIONAL REQUIREMENTS:

Visitors to Goldney shall be able to use the app to display information about various points of interest in the grounds.

Visitors shall be able to view their current location in the grounds, as well as a brief description of their area.

Children may find a tour of a stately home boring. As a result, we could introduce an aspect of gamification (ie garden treasure hunt / hide and seek) to our app to keep them entertained.

Throughout our 24-week project space, we will deploy three versions of our product; a minimum viable product, beta version leading onto our final version. After each deployment, we will listen to client feedback and develop our product iteratively to best been our client's needs.

| Version Name | Date Deployed | Requirements |
|---|---|---|
| Minimum Viable Product | 13th December 2019 | Scan QR codes with my phone.<br><br>Load and display the content in points of interest while not connected to the internet |

| | | |
|---|---|---|
| Beta Version | 14th February 2020 | Create my own points of interest.<br>Change the information in existing points of interest<br>Listen to an audio description of the garden's points of interest |
| Final Version | 10th April 2020 | Be able to view the grotto from outside |

## Architecture

The architecture for our proposed application can be broken down into several key pillars:

- The Android Application Frontend
- The Cloud Database
- Web interface
- Administrator app

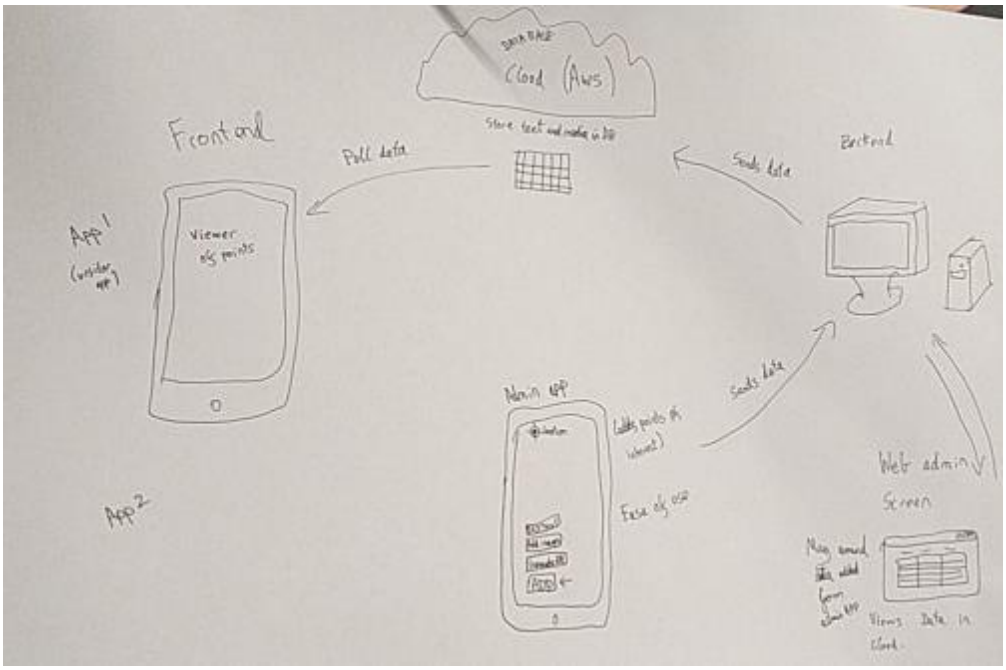
The application frontend is what will be presented to the end user. It is, in essence, our app. Through the frontend, visitors to Goldney will be able to scan QR codes to view the educational content of the tours. Since goldney's grounds do not have complete wi-fi coverage, constant and instant communication with a web based database is not a feasible solution to access our data. As a result, scanning a tile's QR code should return a unique key, which will in turn be used to query a locally cached datastore on the user's phone – thus allowing near instantaneous data access without an internet connection.

The frontend will regularly request updates for its locally cached datastore. These could be delivered in the form of a JSON object constructed by the database. The relatively small size of these updates makes it feasible to check for an update whenever the app has a stable internet connection. The database itself will utilise Spring Boot as the primary interface in the source code; hopefully reducing overall development time. The data will be stored in the cloud using Amazon Web Services on a postgres (SQL) datastore. This data stored in the AWS datastore will be accessed via the Spring database.

Our architecture plan also includes a web interface. From here, a system administrator should be able to view the app's entire cloud database, as well as have the ability to insert and amend data describing points of interest around the gardens. We hope to collaborate with history students and make possible for them to add their own points of interest though this panel.

A potential feature of our system's architecture is an admin mobile app. Providing there is an internet connection, this app should provide the end user with an intuitive way to add new points of interest to the cloud database. The user should be able to physically walk to an area in the garden, generate a QR code, then upload the appropriate information. In theory, the user should be able to make a new point of interest, print off the newly generated QR code, then plant it in the ground.

Although and important subject in software development, ensuring the security of our system should be fairly simple. The software should not be handling any sensitive or personal data, therefore a data breach would be inconsequential. Nevertheless, all net communication should take place over secure https channels.




## Development Testing

### Application Testing

Our aim for our application testing pipeline was to create a system that ran tests on every push of the code and a broader set of tests on every build.

Our initial plan for testing was to use Travis CI. Travis CI is an online CI provider which runs a docker instance on which to run the application. Within this docker instance tests can be run

on the application. The simplicity of Travis CI allowed us to get tests up and running very quickly. However there were a few limitation to this CI provider. In order to respond to push and versions on git Travis CI required the use of Github rather than Bitbucket.. It was also not designed directly for Android which meant the naïve setup we had, had to reinstall the android SDK on every push. Simliarly there were many features which were not designed for the android framework and a few features that were more difficult to implement which wesuch as an emulator for UI testing.

For this reason we looked for an alternative that had a greater focus on android development. The new option we found was Jenkins. This is an open source server which could easily be deployed in a docker container making starting a Jenkins server very easy. This also allows us to run the container locally or host it on a server with very little additional configuration. Jenkins offer the full pipeline we required including running tests in response to webhooks from git repositories, android (gradlew) testing and the option to run an android emulator for UI testing.

Currently we have a Jenkins docker image that contains the various packages we require to run the android application. This is both deployed locally for when we want to run tests locally without pushing and on a Digital Ocean server which in the future will link to our git repository. Unfortunately this also required the use of GitHub rather than Bitbucket as GitHub provide the required webhooks integrated with Jenkins. The future plan for Jenkins is to set up both unit and UI testing which will run on the digital ocean server after every push to Github. Moreover it will run a wider set of tests after a push to "release" branch which will be used for the latest version of the next release. We also hope to implement feed back to the git repository so the outcome of the tests is clear from the repository.

Spring Testing

We are going to use spring to build a cloud server that can update application's data. Our application should check and download the new data such as new tales and new tours automatically. And also some user with authority should be able to update the data in the cloud server (add or delete tales and tours).

Currently we decide to write testing in Juint. We will use CircleCI to run the test so we can run the test automatically. Also, we will use Java Code Coverage to make sure we test all the code we have written.

Here are the things we are going to test:

1. Test our application can connect to our cloud server.
2. Test the user with authority can modify the data in the cloud server.
3. Test our application can update its data from cloud server automatically.

## Release Testing

Currently our release testing will focus on the story of chad. The story of chad shows that our users should be able to get contents of Goldney by scanning our QR code. The core idea of our application is to be able to view content about points of interest through scanning QR codes. Therefore our release testing of Minimum Viable Product will focus on QR code and QR scanner. To test our application, we are going to use CircleCI to run some of the tests automatically and do the rest manually. The table below shows the release testing plan for our Minimum Viable Product, beta version and final version.

| Minimum Viable Product | | |
|---|---|---|
| Camera and QR scanner | Check our application can use the camera on mobile phone and the QR scanner working well in different environments ( eg the performance in different light level ). | Manually |
| QR code | Check all the QR code connect to the correct content. | Automatically |
| UI | Check all the buttons lead to correct pages. | Manually |
| Beta version | | |
| GPS | Make sure our application can get the correct location of users and shows it on the map. | Manually |
| QR generator | Check our in app QR code generator can make the QR code that contains the information we want. | Automatically |
| Cloud server | Check our application can connect to our cloud server successfully and users with authority can modify the data stored in the cloud server (eg add or delete tales and tours). | Automatically |

| | | |
|---|---|---|
| Automatically update | Make sure our application can connect to our cloud server and update at a specific time we choose. | Automatically |
| Web data base interface | Check our web data base interface accepts different kind of data(AR, video, text) | Automatically |
| Final release | | |
| AR | Our application should be able to show a 360 degree image of grotto with AR. | Manually |
| Gyroscope sensor | Make sure our application can use the Gyroscope sensor in the mobile phone. Check our application can show the correct view of grotto when users moving their phone | Manually |

# OOP and UMl Design + Diagrams

The architecture of our product is split into three main components.

We approached designing the solution to our product by exploring three different methodologies at varying levels of abstraction, where each encouraged us to consider the solution in greater level of detail.

The three methodologies were as follows:

- Firstly, we used storyboarding to explore all possible scenarios from the user's perspective which highlighted every functionality our minimum viable product would need.
- This then led onto the creation of Adobe XD prototypes, which were non-functional app mockups to visualize how objects would interact with each other and the human-computer interaction element to the product.
- Finally, we created static and dynamic UML diagrams allowing us to understand the objects and classes that would be required to form the foundations of the app and well as the user application interactions.

By adopting three different methods at three different levels of detail allowed us to gradually build up a rich profile of all functionalities required to create our minimum viable product, eventually considering all the way down to potential attributes and methods that would later be created.

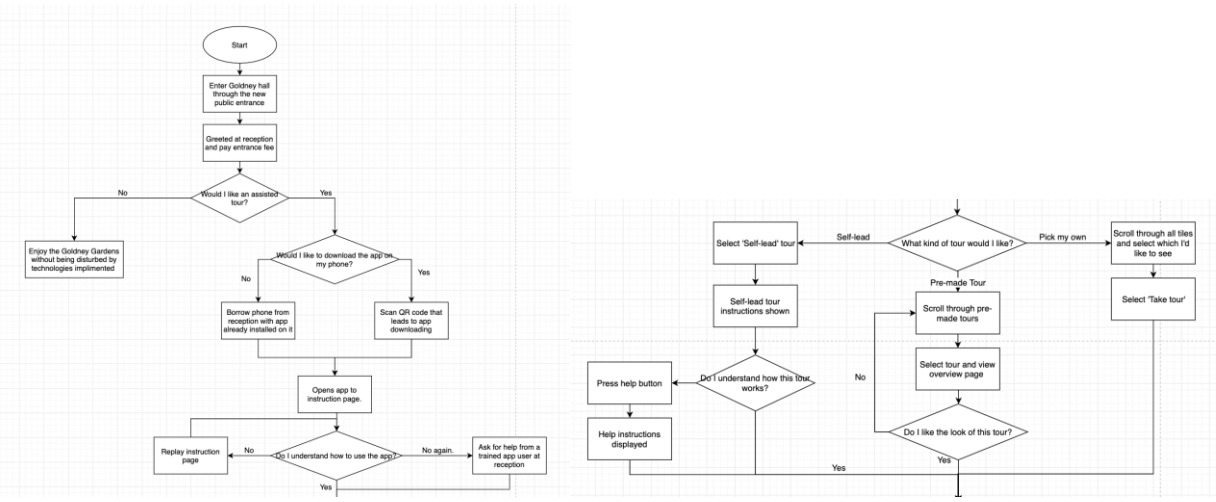
In this section we will cover what we gained by executing each of the three methodologies and the final outputs that will later be used to aid the implementation of our minimum viable product.

## Method 1: User Storyboarding

To begin, we considered how the application would be used from a user centered perspective. This means everything from the device to run the application, asking questions such as, *"if the user didn't have a suitable device or didn't want to download the app, what would their options be?"*. To questions like *"how would we teach users who had never scanner a QR code (fundamental to our minimum viable product) to do so?"*

The greatest benefit when creating the story board was that we were able to display it to our client and convey ideas and concept in a way that that doesn't require any technical literacy. This allowed our client to identify missing functionalities they required or remove any they felt weren't appropriate. It will also be a great tool to track our progress through development stages.

Similar to our user stories identified in the 'stakeholders' section of this report, our storyboard outlines the journey by a generic, non-specialized user of the app. This was then used as a basis for our next design methodology; Adobe XD prototyping.

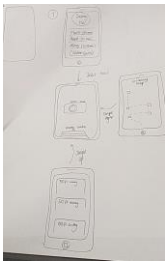



## Method 2: Adobe XD Prototyping

For our next stage of design development, we wanted a way of visualizing the ideas and features that had been discussed. After exploration, we settled on the software Adobe XD which is an application mock-up software. Note, this mock-up app has no functionality, it is purely to map out interactions between objects. This was advantageous as it allowed us to

clearly identify the objects and classes to later be identified in the UML diagram stage of the process.

Firstly, we started with paper prototyping, to establish a shared vision on what the application would roughly look like. This allowed us to freely experiment with different formatting options and discuss the benefits and disadvantages of each.

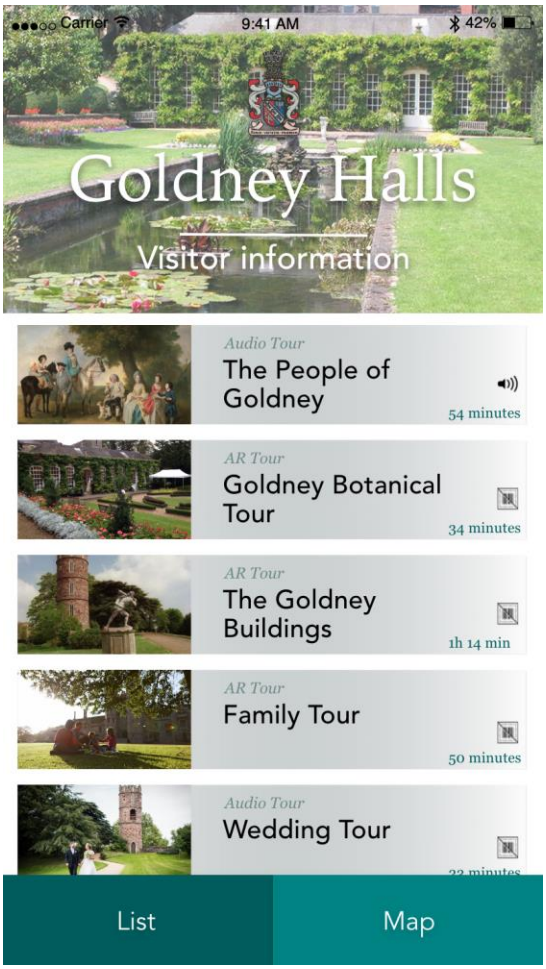


From these paper prototypes, we then agreed on creating 3 different prototypes for the client to trail and test out at our next meeting where they would be able to identify features they like and dislike

## Prototype A

Likes: That the tours have been categorized into clear content groups, lots of functionality

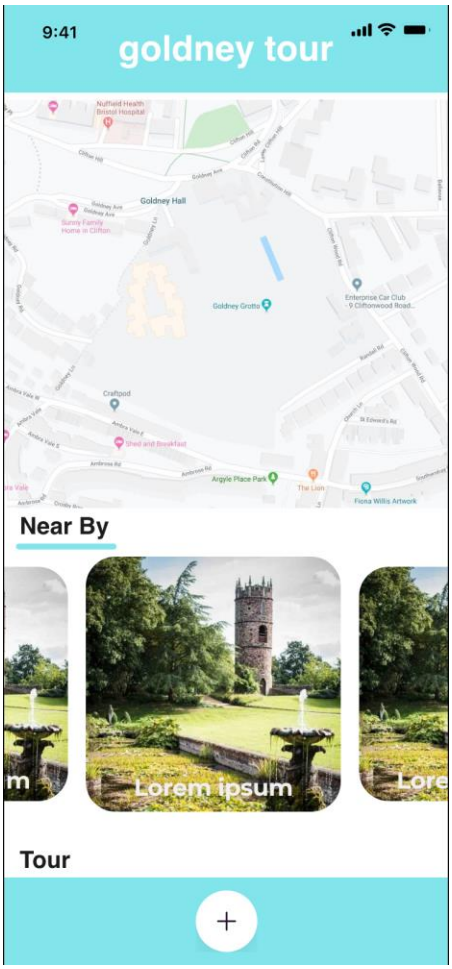
Dislikes: Symbols can be confusing as can't make out what they mean, looks a little cluttered (can be made slicker)




## Prototype B

Likes: The sliding feature to view the array of tiles (object) prior to selecting a tour, slick minimal design

Dislikes: Navigation through the app is not very intuitive, not much functionality




Prototype C

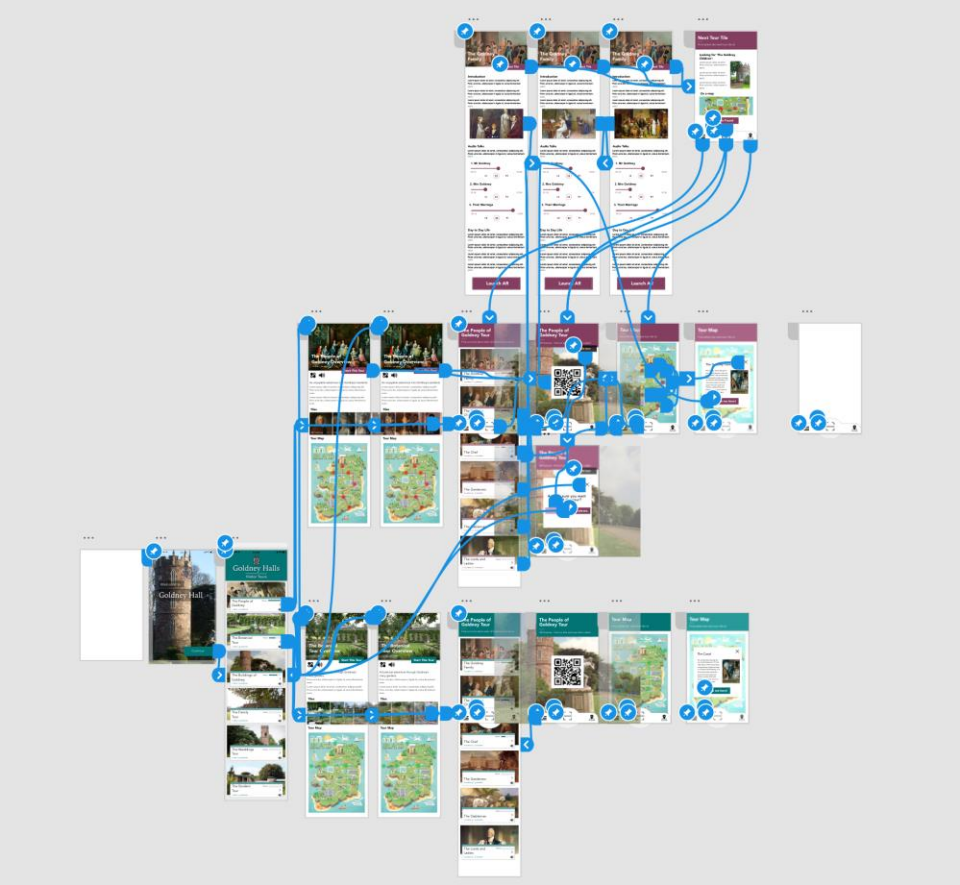
Likes: They liked the tiles page and how we displayed the information for each point of interest. They also like the navigation through the

Dislikes: They disliked the drag up menu and you could easily get lost in the application. For this reason we replaced this in the next version by merging the drag up menu and the tours page.


From this client feedback, we then complied all the features the client highlights as a positive and removed all disliked features into a final prototype.

Final Prototype:

https://youtu.be/ylYUDcpDG1E



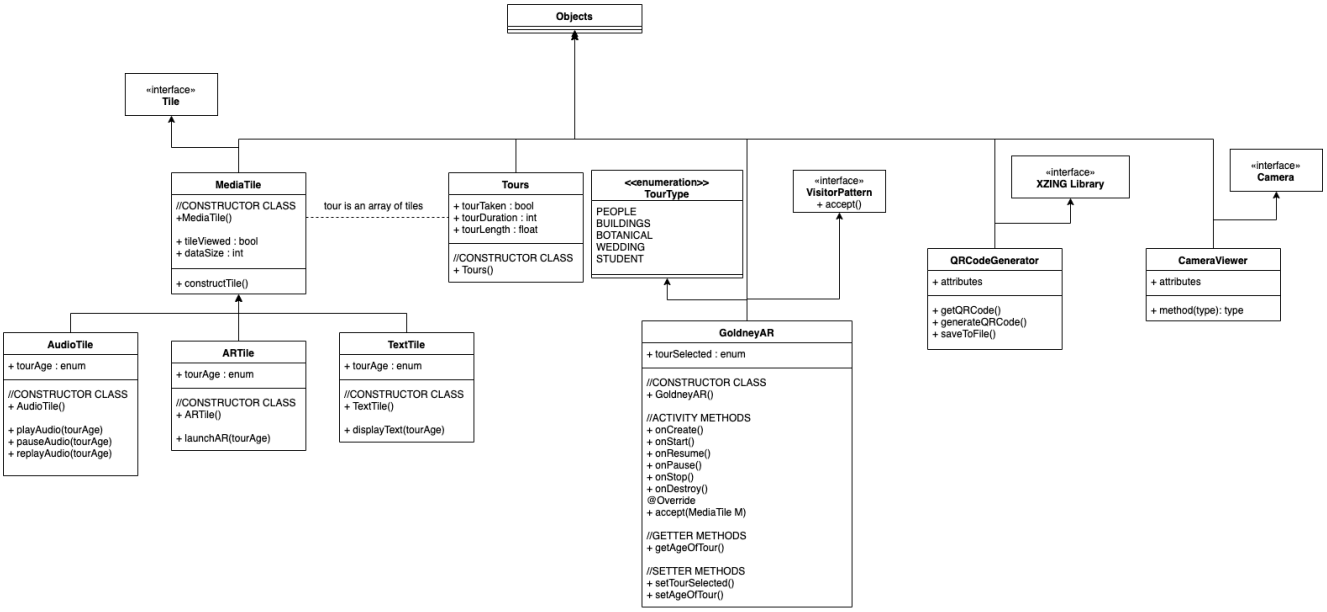
The image above displays all the interactions between application interfaces. This we aid up when coding these within Android studio as we have a self-made map of all widget communications.

## Method 3: UML Diagrams

Our final stage of designing the solution was to create static and dynamic UML diagrams, which forced us to consider the solution at the greatest level of detail. At this level of abstraction, the solution is considered at an object level therefore attribute and methods are starting to be considered.

The **motivation** behind the static UML diagram is to visualize a minimal representation of the object required as well as their relationship between each other. Within each class we have identified the attributes and methods required, but admittedly at this stage in the project, many of these qualities are yet to be identified since we are yet to code them. Therefore, we have highlighted the main, non-negotiable methods and attributes and as this project develops, this UML diagram will become richer in these methods and attributes.

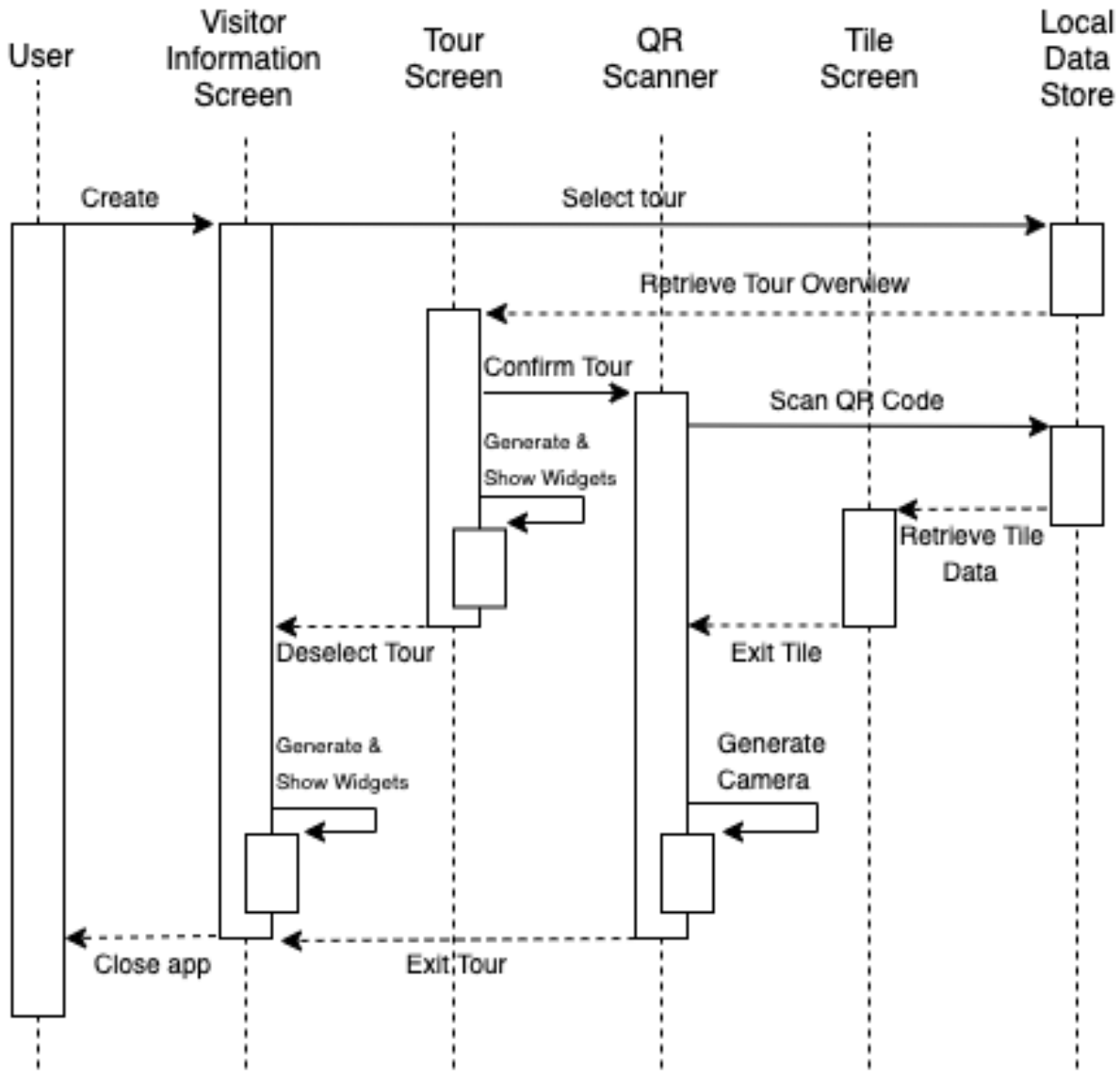
## Static UML




The Goldney AR application will run off a very simple concept. The grounds of Goldney will be split up into dozens of points of interest, in our terminology a **tile,** that will have a corresponding QR code to encode its location within the **local data store.** Each of these tiles will be of a different medium depending on the data stored in this tile, for example there could be a tile to launch a piece of text, audio or even display augmented reality through the users' camera. Each of these mediums have its own respective class, however all inherit the attributes and methods from its parent class **MediaTile**. All tiles also extend the interface 'tile' to ensure a uniform tile blueprint. Hierarchal attributes and methods include a Boolean attribute to flag whether this tile has been viewed and a method to construct the tile, while unique attributes and methods uniquely held by the child may include methods such as **launchAR**(). The user has the option of freely scanning any QR code within the grounds without having to follow a tour, however tours have been introduced to optionally direct and recommend arrays of tiles with a common theme, i.e.. The buildings within the grounds or the plants. The object 'Tours' is an array of **MediaTile** objects and is purely used to thread together tiles of a similar theme and display them in a user-friendly by directing them through this pre-determined route.

Finally, GoldneyAR is the class is one that interacts with all other objects and is where the programs main functionality will occur. This class therefore contains the key activity methods such as **onCreate()** and **onDestroy().** Additional classes required include cameraViewer or QRCodeGenerator that are using external libraries such as ZXing library which is a barcode imaging processing library – ideal for our minimum viable product.

## Dynamic UML: User Interactions




The above dynamic UML diagram described the interactivity between the user and the libraries.

1. The user opens the app on their phone, or a phone provided to them by Goldney Hall visitor information.
2. The user is faced with a scrolling list displaying the options of the different tours available.
3. The user selects a tour of interest which pull data from the local datastore to return the tour overview for this tour.
4. This information is then retrieved and sent back to the application on the tour screen. The tour screen iteratively generates and displays widgets for that screen.

5. The user then has the option to return to the visitor information if the tour is not wanted (from there the user can repeatedly fetch new overviews of other tours until the desired tour is selected) otherwise they are taken to a new page for that tour – the QR scanner.
6. The QR scanner requires the camera to be constantly operational, therefore for its duration it will be iteratively refreshing the camera.
7. From this page there are navigations to aid finding the next tile as well as displaying the tile information however this interactivity is not represented in this diagram.
8. The user is now required to find and scan a QR code correlating to information on a nearby place/thing of interest. Upon scanning the QR code, a unique id will be received which refers to the relevant tile (or tiles) for that point of interest. The user will therefore be redirected the Tile Screen with the relevant information pulled from the cashed datastore.
9. Once the user has consumed this data (the tour information) the tile is exited and steps 6-8 are iteratively repeated until all tiles on the chosen tour are complete or the user is satisfied with the information displayed.
10. The tour can then be exited with an option of selecting a new tour entirely (repeating steps 2-9) or the user can exit the app.

This diagram described the interaction between our two main components, the front end (the application) and the back end (the local data store), which is the solution for our minimal viable product. The next stage of the project that we intend to implement for out beta release is additional software that will allow the maintenance team to self-upload tiles and content on the tiles to the local data store. The benefit of this will be that the application will be sustainable and provides a way of maintaining the application once we have finished the project. This additional feature would require a third component, like a 'middle-man' between the two, however this will be deployed at a later release.

The motivation behind creating a dynamic UML diagram is to graphically map out the interactions between the key user facing interfaces. This also allowed us to identify the best times to communicate with the local data store and how data would flow between the UI elements within the app and the backend.

Upon reflection, by exploring the design of our solution through three methods allowed us to consider the solution at three different levels of abstraction. Each methodology encouraged us to consider the product in a greater level of detail ensuring aspects of the product such as how the user will download the app, all the way down to the attributes within an object class have been studied. By creating these UML diagrams allowed us to visualize and in turn critically think about how we can make our solution more efficient which in the long run will save programming time and processing power. Choices such as cashing all data onto the users' phone prior to starting a tour benefits the user experience as it allows them to utilize the app, even if they become offline during a point in the tour – a feasible problem given the task that is being carried out.