TheatreCapture

Health Data Insight 2021 Intern Project

A Prototype For Patient, Implant, and Anaesthetic Record Data Collection in Operating Theatres

Katy Higton
Jaskirat Kaur
Alex King
Laith Marsden
Tomas Pickford
Noo Rashbass
Jia Xiu Sai
Louise Treacy

Contents

1 Introduction

TheatreCapture was conceived in response to the IMMDSReview report, and provides a solution for the collection of patient, location, anaesthetic record and implant information at the time of operation. Its two main components are a Raspberry Pi computer which extracts data from the anaesthetic machine, and a smartphone app for the collection of patient, implant, and location data via QR code and data matrix scanning. TheatreCapture has a number of benefits. It enables the identification of faulty or expired implants, and by linking to a central database of faulty implants, could prevent the use of these implants in the first place. By automating data collection from the anaesthetic machine, anaesthetists and nurses need no longer record vital stats manually, allowing them to focus on patient care. This also means data can be recorded at more frequent intervals, creating a large data set for future analysis. By using existing technology, we offer a cheap, robust solution that will ease the transition towards digital technology in healthcare.

1.1 Project Overview

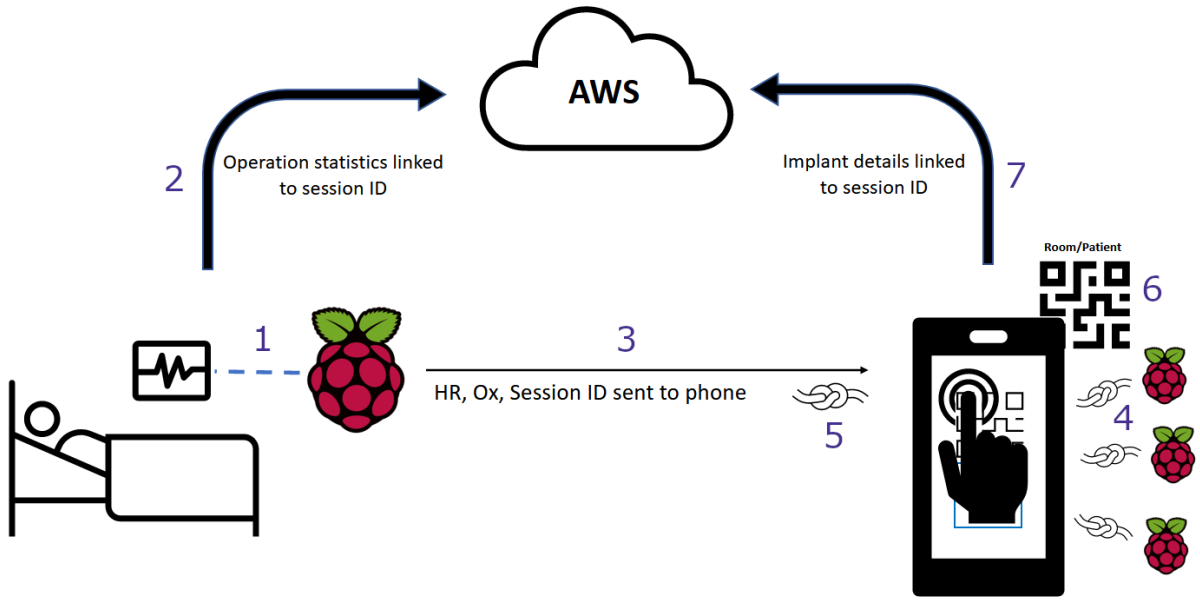


Figure 1: A diagrammatic overview of an operation using TheatreCapture

The following procedure summary outlines the use of TheatreCapture over the course of an operation:

1. Data is extracted from the anaesthetic machine using a Raspberry Pi.
2. The Pi sends this data in real-time to the cloud along with a session ID.
3. It also sends this data via Bluetooth to any devices that connect to it.
4. The smartphone app initially displays all the Pis (Anaesthetic machines) it can hear.
5. The user selects (or scans the barcode on) the correct anaesthetic machine, and the app connects to that Pi and disconnects from the other Pis. The app records the session ID received from the selected machine so that the information scanned by the app can be linked with the anaesthetic data.
6. The app is then used to scan:
   - The room code
   - The patient wristband
   - Any implants being used
7. The app sends this information to the cloud as it is scanned along with the session ID it received.
8. The session ID links app and Pi data in the cloud.

1.2 In This Document

The TheatreCapture project can be broken down into three main parts:

- The anaesthetic machine
- The app
- Marketing

Sections 2 - 4 will go in-depth about each of these parts respectively and possible future progressions. The GitHub repositories containing different components of the project are linked in section 5.

2 Data From The Anaesthetic Machine

Currently, a Raspberry Pi Zero W is used to extract data from the machine, then share it via Bluetooth and store it in the cloud. The Pi Zero W is the cheapest Raspberry Pi model with WiFi and Bluetooth capabilities.

2.1 Extracting Data From The Machine

The Raspberry Pi code that reads data from the anaesthetic machine is written in C# and was refactored from this repository. The Pi is connected to the machine using a USB to RS232 converter and a null modem.

2.2 Sending Data Via Bluetooth

Data is shared from the Raspberry Pi to the phone using the Bluetooth GATT protocol. The Pi is the GATT server and the phone is the client - the phone connects to the Pi and reads data from it. The server contains one service, which has three characteristics - session ID, heart rate (HR), and oxygen saturation (SpO2). Heart rate and SpO2 are notifying characteristics, which means they update any connected devices with new values. The server does this every five seconds. The session ID is read-only, so connected devices have to read again if they want to check their session ID is up to date. Ideally, the server would notify of session ID change. The code for the GATT server is written in python, and is available on GitHub.

For details of how the phone reads data from the Pi GATT server, see section 3.2.

2.3 Sending Data To The Cloud

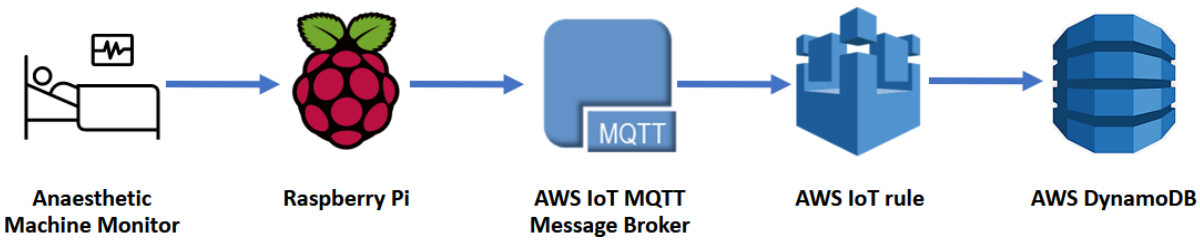


Figure 2: Data flow from the anaesthetic machine to the database in the cloud

The Raspberry Pi code that reads data from the anaesthetic machine also sends it to the cloud - specifically Amazon Web Services (AWS). The process of sending data to the cloud is structured to eventually use AWS GreenGrass.

The following documentation explains the final three steps shown in Figure 2:

- AWS IoT MQTT
- AWS IoT Rules
- AWS DynamoDB

The AWS DynamoDB database on the EU West server has four tables:

- GTINLookup - A demonstration table of Implant GS1 codes for GTIN-Item name lookup
- ScanRecord - Stores data of codes scanned by the TheatreCapture app
- TheatreCapture-PiData - Stores data collected from the anaesthetic machine
- TheatreCapturePatientStore - Currently not used

2.4 Anaesthetic Machine - Additional Notes and Future Progression

Currently, anyone with a bluetooth scanner can connect to and read data from the Raspberry Pi GATT servers, provided they know the unique ID. This would provide access to anaesthetic parameters (heart rate and oxygen sats) and a session id. Ideally, we would encrypt the GATT server broadcast, and potentially only allow specific devices to connect (e.g. a device with the TheatreCapture app and a valid login).

The code that is extracting the readings from the anaesthetic machine is refactored from a public repository. We plan to re-write this in Python to make the code more maintainable and skip the extra step of piping the readings to stdout and reading from that to broadcast it.

In the future, the data collected should be output to the National Theatre Data Set automatically. A custom dashboard can be developed so that the tables can be displayed graphically and queried using a GUI.

2.4.1 Indoor location

As part of the project, an experimental method for determining the location of an anaesthetic machine using Bluetooth Beacons was developed. A beacon would be placed in each operating theatre, and the Raspberry Pi on the back of the anaesthetic machine would use the signal strengths from the beacons it could hear to determine which room it is in. The code for this is found in the `beacon-fingerprinting` folder of the code-snips repo.

The signal strength is measured using the Received Signal Strength Indication (RSSI), and is expected to decrease with distance from the Raspberry Pi measuring device[1][2][3]. RSSI values tend to fluctuate a lot, so averages taken over longer periods of time will give more accurate estimates of the distance of a beacon from a machine. This is not a problem for the anaesthetic machine since it is likely to stay stationary for long periods of time.

Currently, our solution records RSSI values from surrounding beacons until it has a specified number of data points, typically around 5000, which takes roughly 15 minutes. Analysis is then carried out on this data post recording. It was found that RSSI measurements did not typically follow a standard statistical distribution, so Central Limit Theorem was used to obtain normally distributed data for each of the beacons. Many means were calculated for samples of size 50, and these means followed a normal distribution. A z-test could then be carried out on these distributions. With the result from the z-test, one can work out the probability of having got that result if the two beacons actually had the same signal intensity. If this value is low enough, one can reject this hypothesis and infer that the louder beacon does indeed have a higher signal intensity, suggesting that it is closer to the Raspberry Pi. Since this beacon is associated with a specific operating theatre, the Pi can infer which room it is in. This would enable the app and machine to be automatically connected, since the app knows which room it is in from the QR code on the wall.

In order to use this method in the future, it would need to be adapted to perform hypothesis tests as the data comes in. Then, the Pi would have to perform the analysis continuously. In order to account for the face that the machine might move rooms, the Pi would have to flush out old data as new data came in.

3 Mobile Application

The mobile application is written using JavaScript and React Native. Additionally, React Native is open-sourced, allows for cross-platform development which reduces development time, and is widely used, ensuring support for the library into the future. The code for the app can be found in the intern2021-react-native repository. Figure 3 describes the process of using the app.

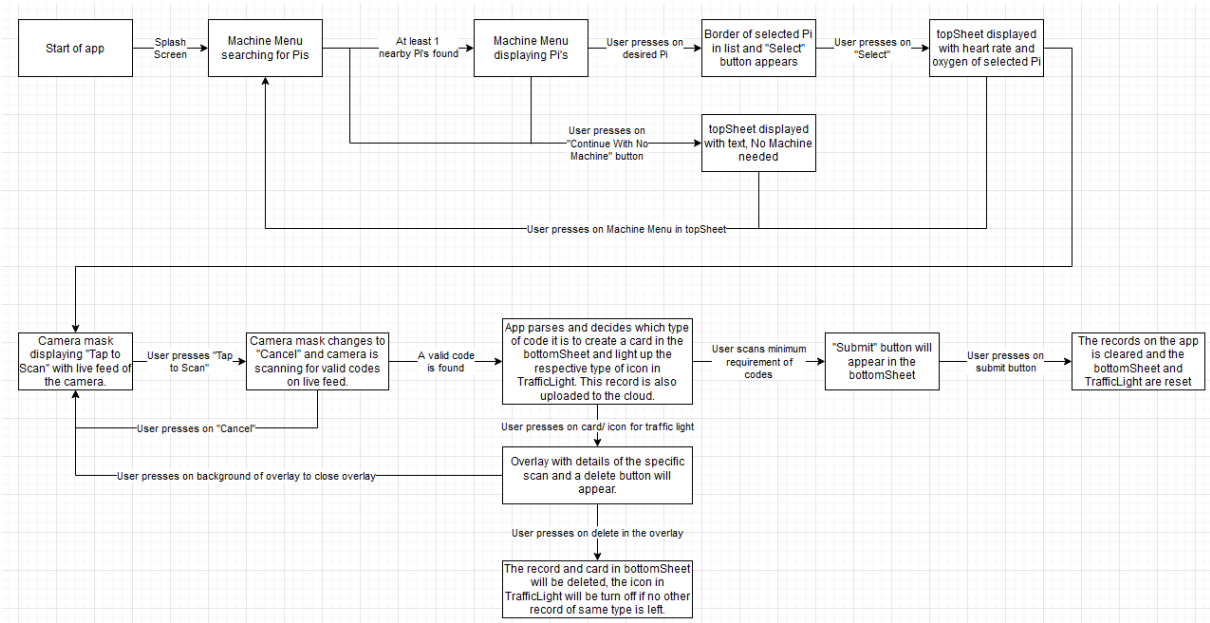


Figure 3: Flowchart diagram of the app.

3.1 Code scanning

The app acts as a scanner that records information about the patient, operating theatre and implants used in the cloud. It uses the `expo-camera` module which has built-in code detection.

3.1.1 GS1 Data Matrix

Global Standard 1 (GS1) is a not-for-profit organisation that develops and maintains global standards for business communication. GS1 standards in codes like bar codes, QR codes and data matrix codes are the most widely used system of standards in the world. GS1 Application Identifiers (AIs) are 2, 3 or 4 digit numbers which define the meaning and the format of the data that follows. This is important because implants will come with GS1 data matrix codes that encode their serial number, batch number, expiry date, etc. which can be used to flag implants that are expired or from a bad batch.

3.1.2 NHS Patient Wristbands

NHS Patient Wristbands also use GS1 Data Matrix codes, but they only encode 13 possible fields, with 4 of them specifically for newborn infants. These codes have a built-in checksum to ensure that the NHS number is valid.

NHS number checksum

The method to check if an NHS number is valid is as follows. An NHS number is 10 numbers in 3-3-4 format (xxx-xxx-xxxx), for example: 943 476 5919.

1. Multiply each of the first nine digits by 11 minus its position.
   - 9 is the first digit: $9 \times (11 - 1) = 90$
   - 4 is the second digit: $4 \times (11 - 2) = 36$
   - 3 is the third digit: $3 \times (11 - 3) = 21$

     $\vdots$
   - 1 is the ninth digit: $1 \times (11 - 9) = 2$
2. Sum the results,

$$90 + 36 + 24 + 28 + 42 + 30 + 20 + 27 + 2 = 299.$$

3. Get the remainder of the sum divided by 11,

$$299 \pmod{11} = 2.$$

4. Subtract the remainder from 11 to get the checksum,

$$11 - 2 = 9.$$

5. If this is not 10 and matches the last digit of the NHS number, it is valid.

3.1.3 Room Codes

The room codes are QR codes containing the ANANA code of the organisation followed by the operating theatre number.

3.1.4 Pi Codes

The Pi codes are QR codes containing the name of the Pi connected to each anaesthetic machine. Each of these codes encoded start with "piName-" followed by the name of the Pi.

3.2 Bluetooth Communication With The Anaesthetic Machine

Bluetooth connection between the Raspberry Pi GATT server and the phone is handled in `BleList.js` component of the app, making use of the `react-native-ble-plx` package. Details of the Bluetooth communication from the Pi side are in section 2.2.

The app connects to all the Pi GATT servers it can hear and displays them in a list ordered by proximity. Also in the list are heart rate and oxygen saturation readings from the machine - other vital signs could also be displayed if desired. The user can then make a comparison between the machine in front of them and the readings on the app to select the correct machine. Alternatively, the user can scan a QR code linked to the Raspberry Pi to connect them to the machine.

Proximity to a machine is calculated using the Bluetooth signal intensity (RSSI) received from it. Signal intensity values fluctuate quite significantly [3], so multiple measurements are made and an average is taken over a period of 5 seconds. A longer averaging period would lead to more reliable results but would make the app slower to update if the user moves around. Therefore, a balance needs to be found between reliability and speed. More testing needs to be done on this.

Before the app can start scanning for Bluetooth devices, a new instance of the BleManager class from `react-native-ble-plx` must be created. Then it begins scanning and connects to devices with the UUID of our GATT server. Once connected to a device, the app reads its session ID, and begins monitoring the HR and SpO2 characteristics. In addition, a loop starts that measures the RSSI value from the device every 0.5 seconds, and averages these readings every 5 seconds. These loops run asynchronously for each device that connects, and the order of the list updates every time a new average is calculated for a device.

The user has two methods of connecting to a machine, by selecting a machine in the machine menu or by scanning a QR code that encodes the name of the Pi connected to the machine. Once a machine is chosen, the app disconnects from all the other devices. The RSSI averaging calculations are stopped for all devices. If alternatively the user chooses to proceed without a machine, all devices are disconnected. Whether a machine is chosen or not, the user has the option to search again for Bluetooth devices. When a rescan is initiated, the old BleManager is “destroyed”, and a new instance of the class created. This may not be necessary, and the code could be changed to use the same instance the whole time the app is running.

3.2.1 Known Bugs

There is currently a bug where the app stops measuring RSSI values from the devices it is connected to after scanning for a period of time on the order of 30 seconds to a minute. The `react-native-ble-plx` package makes heavy use of promises, and the function to measure the RSSI of a connected device, `readRSSI()`, returns a promise. After some time and for an unknown reason, the app stops resolving these promises, causing a large number of pending promises to build up, which eventually crashes the

app. This has been partially resolved by cancelling promises if they take too long to resolve. However, this still leaves the problem that RSSI values stop being measured after around a minute of scanning, which means the ordered list is no longer being updated. This is most likely a bug with the BLE package we are using, so the solution may be to deal with Bluetooth connection for Android and iOS natively.

For other bugs, see the Github issues page.

3.3 User Interface

The app uses React Native Elements UI toolkit. This section will go through each component of the app and explain its functionalities. The app can be simplified into three main parts:

Top Sheet
The top sheet displays the current characteristics of a Pi if it connected.

Figure 4: Top Sheet with Pi Connected

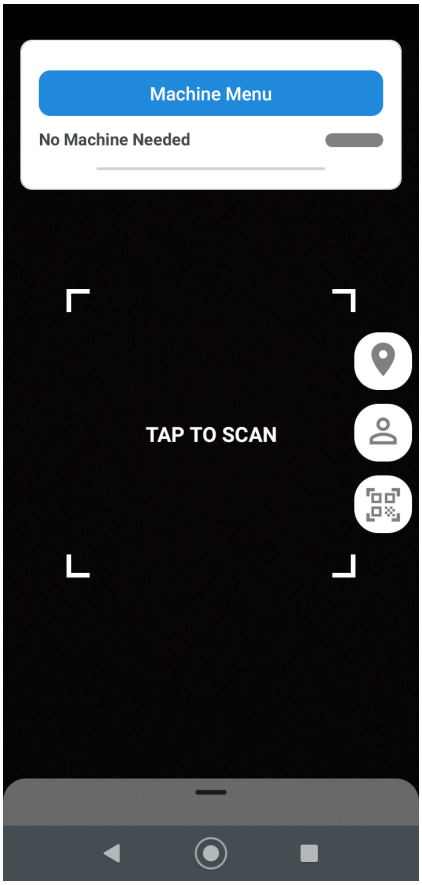


Figure 5: No Pi Selected

Once the user presses the "Select" button in the machine menu, the machine menu will be replaced by a top sheet, that is partially hidden. The user can reveal the top sheet by tapping on the handlebar and the top sheet will be fully revealed, displaying the live readings from the anaesthetic machine as shown in Figure 4. However, if the option "Continue with no machine" is selected, the machine menu will be replaced by a top sheet without any characteristics as shown in Figure 5.

Machine Menu
The machine menu handles the connection and selection of Pis. This component is stored in `MachineMenu.js`.

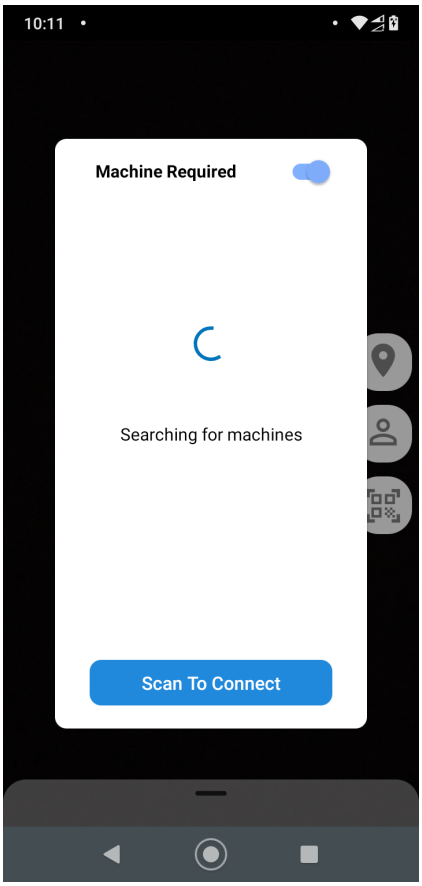


Figure 6: Machine Menu Searching

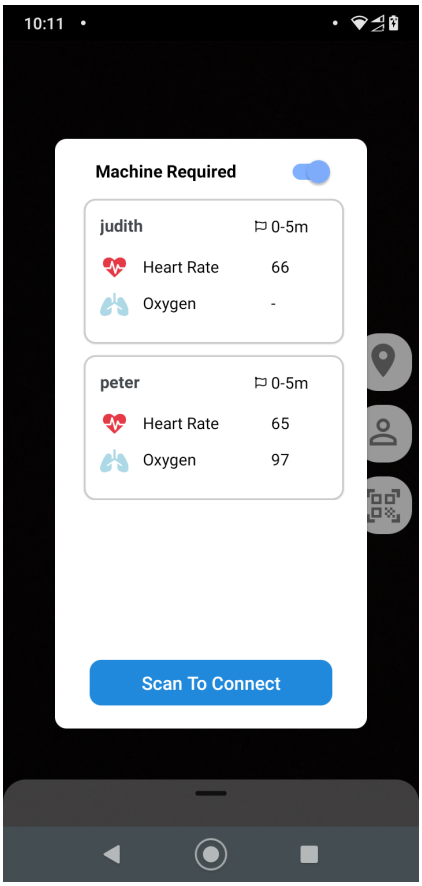


Figure 7: Machine Menu Options

Upon opening the app, the machine menu will initialise automatically, searching for any nearby Raspberry Pis broadcasting data from the anaesthetic machines as shown in Figure 6. Once one or more Pis are detected, the cards will appear displaying the characteristics of the Pis as shown in Figure 7.

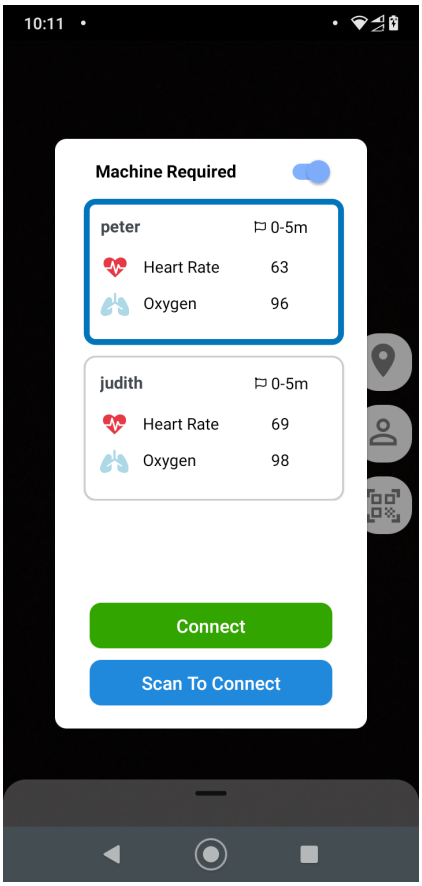


Figure 8: Machine Menu Selected

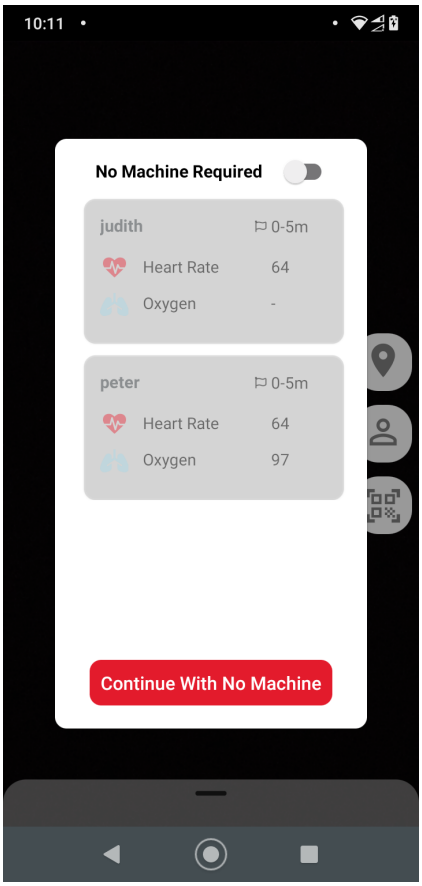


Figure 9: Machine Menu Disabled

The user can then press on the machine that needs to be connected to and the option of "Select" will appear as shown in Figure 8. If no machine is required, the toggle on the top of the machine menu can be pressed for the option of "Continue With No Machine", which will appear as shown in Figure 9.

Bottom Sheet

The bottom sheet displays the codes scanned. This component is built using the bottom sheet package (Version 4.15) and stored in `CustomBottomSheetScrollView.js`.

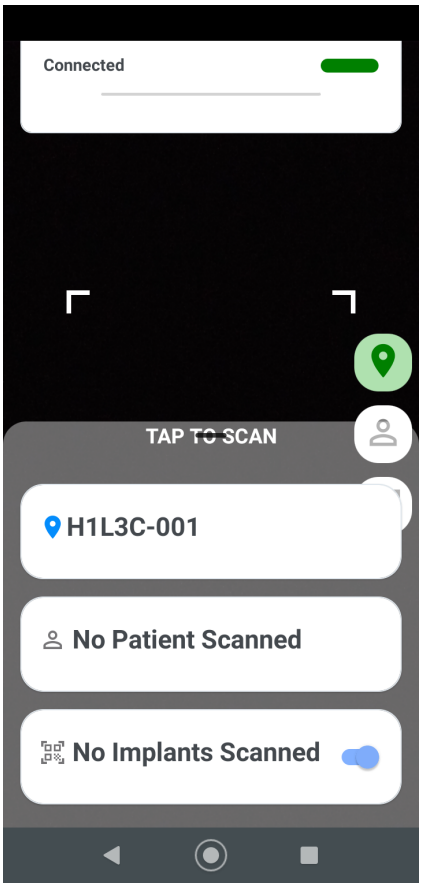


Figure 10: Bottom sheet with room scanned

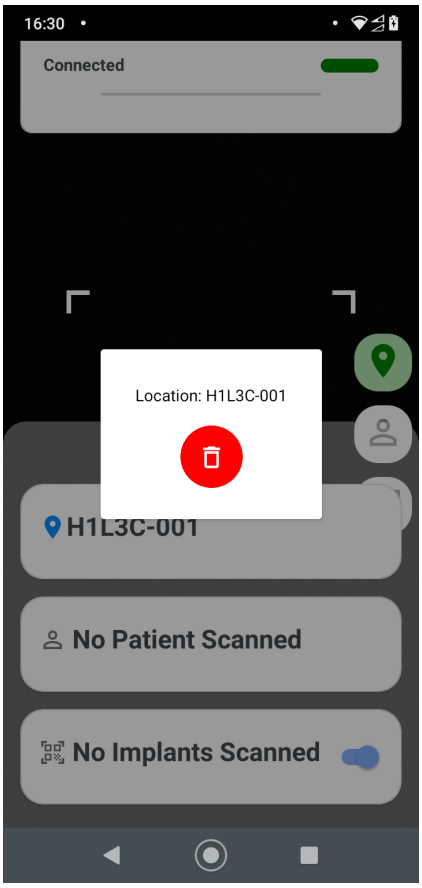


Figure 11: Overlay of scanned code displaying its details

Once a code is scanned successfully, the data will be stored on the phone and automatically uploaded to the cloud. The icons on the right represent each different type of code and are stored in `TrafficLights.js`. If a room code is successfully scanned, the room code will be recorded and the location icon will turn green as shown in Figure 10. Tapping on the item in the bottom sheet or the icons in the traffic lights will bring up an overlay with the details of the code scanned, with an option to delete from the record as shown in Figure 11.

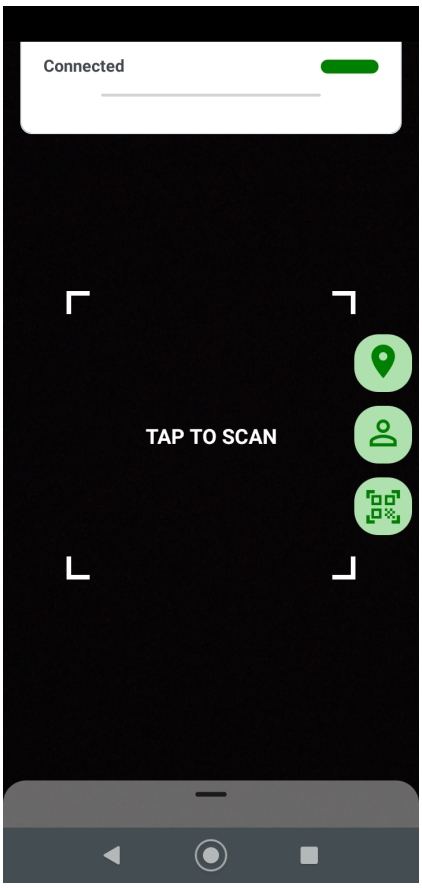


Figure 12: Successful icons

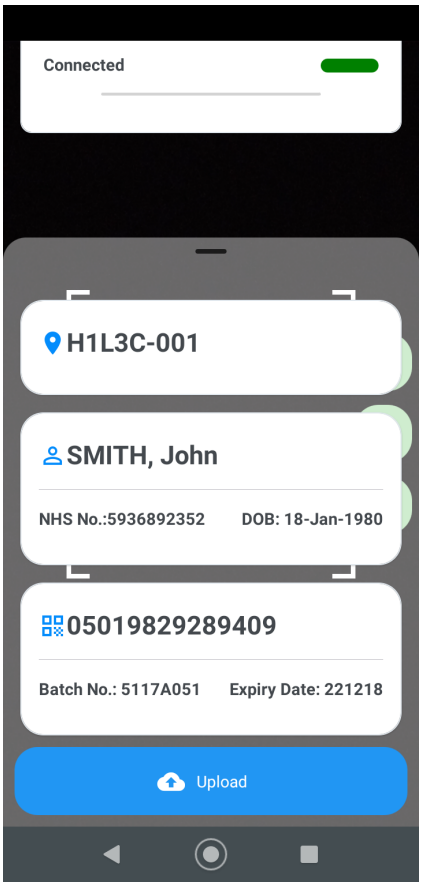


Figure 13: Bottom sheet with a code scanned for each type

If all three types of code are scanned, the icons on the right-hand side of the app will turn green as shown in Figure 12. The bottom sheet can also be expanded to show a summary of all the codes scanned in the session. An "Upload" button will appear if all the required codes have been scanned, as shown in Figure 13. This uploads all data and closes the session but if the "Upload" button is not presseed, the data is still uploaded to the cloud.

3.4 Communication With The Cloud

The phone uses the same protocol as the Pi to send data to the cloud as mentioned in 2.2.

3.5 App - Future Progression

There are some areas in which improvements to the app can be made. Currently, the app is written in a mix of functional and class components and although not required, it is recommended that functional components are used as the react ecosystem moves toward the use of hooks which means functional components are the standard.

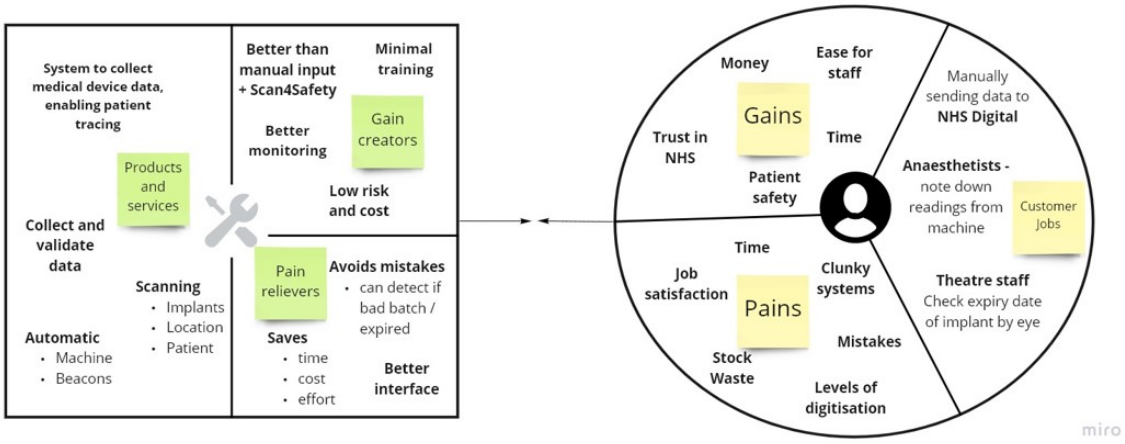
If the app were to be rewritten in React Native in the future, TypeScript and modules like Redux can be used to improve the maintainability and reliability of the app.

Although the app works on both Android and iOS, most of the manual testing and development was done in android. Therefore, further testing and optimisation on iOS are required. Adding onto that, only internal user testing has been carried out during the project. User feedback from those who will be using the app in the hospital environment is crucial.

Currently, the bad batch flagging system is a prototype as, at time of writing there are no databases storing faulty implants in the UK. However, NHS digital is developing a database, the dm+d database, which is currently in beta phase.

4 Marketing

4.1 Value Proposition



]

Figure 14: Value proposition canvas for the TheatreCapture product. The circle represents customers and the square is the company developing this system.

The key aims of the TheatreCapture project were the following:

- Time-saving
  - Medical professionals should spend most of their time looking after patients.
  - The system should minimise input required, automating as much as possible.
- Increase patient safety
  - Keep a record of patient implants to keep track of faulty implants.
  - Flag up expired implants or those from bad batches at the time of operation.
- Easy adoption
  - Our product should use existing technology as much as possible.
  - It should be easy to implement in all hospitals, regardless of digitisation levels.
  - It should be easy for staff to use, requiring minimal training.

4.2 Branding

When coming up with a name for the project, we had to take into account already existing names like MediScan, Scan4Safety and CareScan. Additionally, we found it hard to put "operating" smoothly into a name. Therefore we decided on TheatreCapture. Mock-ups of logos were made and voted upon by team members. The mock-up for logos can be found here. Some are derived from the HDI logo, others based on a certain key aspect of the project i.e. scanning codes, heart rate.

4.3 Costing

Once we were set on the technology needed for the system, we were able to calculate a cost estimate. Hospitals have eight operating theatres on average, so our estimate is the cost for eight operating theatres.

Item	Cost	Quantity	Total
Printing A2 QR codes for walls	£0.50	16	£8
ICT technician install	£500	1	£500
Smartphones*	£60	8	£480
Raspberry Pi Zero W	£10	8	£80
Accessories for Pi: - case - microSD card - adapter	£20	8	£160
USB host, regular & mini HDMI cables	£10	1	£10
AWS (Amazon Web Services)**	£55	1	£55
		TOTAL:	£1,293

* Only required if staff do not want to use their phones.
** This cost will recur yearly.

Our initial prototyping was done with a Raspberry Pi 4 Model B, but we realised that costs could be minimised by using a Raspberry Pi Zero W instead, costing under £10 each.

4.4 Unique Selling Proposition

We think our product holds advantages over current systems for the following reasons:

- Clinically valuable
  - Currently, information from anaesthetic machines are anonymised, limiting their use for study as you don't know who those vitals belong to. We are connecting patient vitals from anaesthetic machines to information that has been scanned which makes this data clinically valuable.
- Fast
  - Scanning to record information is faster than manual input, freeing up staff time.
- App
  - Current scanning schemes use handheld scanners that do not usually have a display. However, by developing a smartphone application, we have removed the need for these scanners, which reduces the cost of implementation. By using an app, we have been able to optimise the user interface to make it intuitive and easy to use, minimising the need for staff training.
- Reduced Cost
  - By using existing technology and devices the cost is significantly reduced. Most people will be able to use their own smartphones, and some hospitals even have tablets already.
- Bridges the digital gap in healthcare
  - Low cost and ease-of-use makes the integration of this service to all hospitals easy, regardless of funding or level of technology. Using a common system across UK hospitals is important to provide everyone with the same level of healthcare and measure of safety.

5 GitHub Repositories

These repositories are owned by Health Data Insight's GitHub account and some of them are private.

- React Native app: intern2021-react-native
- Extracting data from anaesthetic machine: VSCapture
- Broadcasting data from the anaesthesia machine: interns2021-greengrass-component
- Experimentation with geolocation and using smartphone microphone audio to connect machines and phones: intern2021-code-snips

References

[1] Received signal strength indication. https://en.wikipedia.org/wiki/Received_signal_strength_indication, 2021. Last accessed: 29/09/21.

[2] Log-distance path loss model. https://en.wikipedia.org/wiki/Log-distance_path_loss_model, 2021. Last accessed: 29/09/21.

[3] Eladio Martin, Oriol Vinyals, Gerald Friedland, and Ruzena Bajcsy. Precise indoor localization using smart phones. *MM'10 - Proceedings of the ACM Multimedia 2010 International Conference*, pages 787–790, 10 2010.

