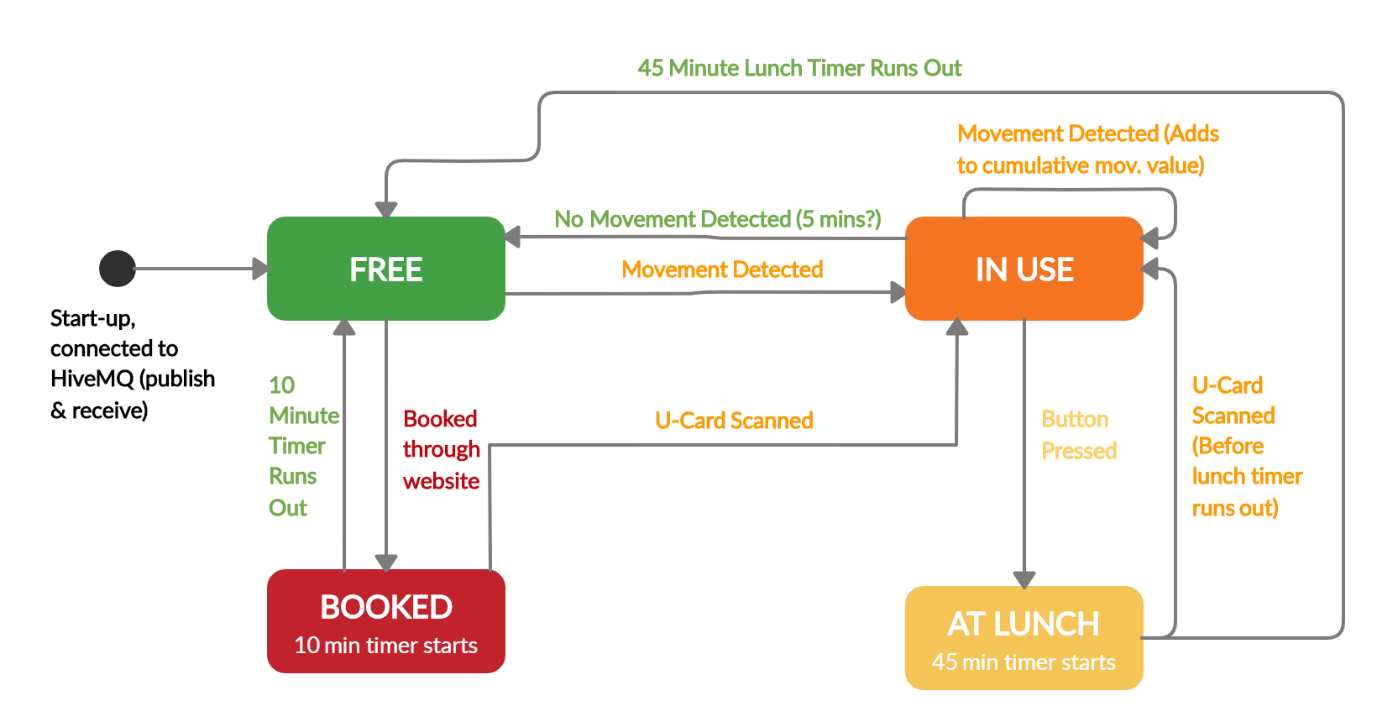
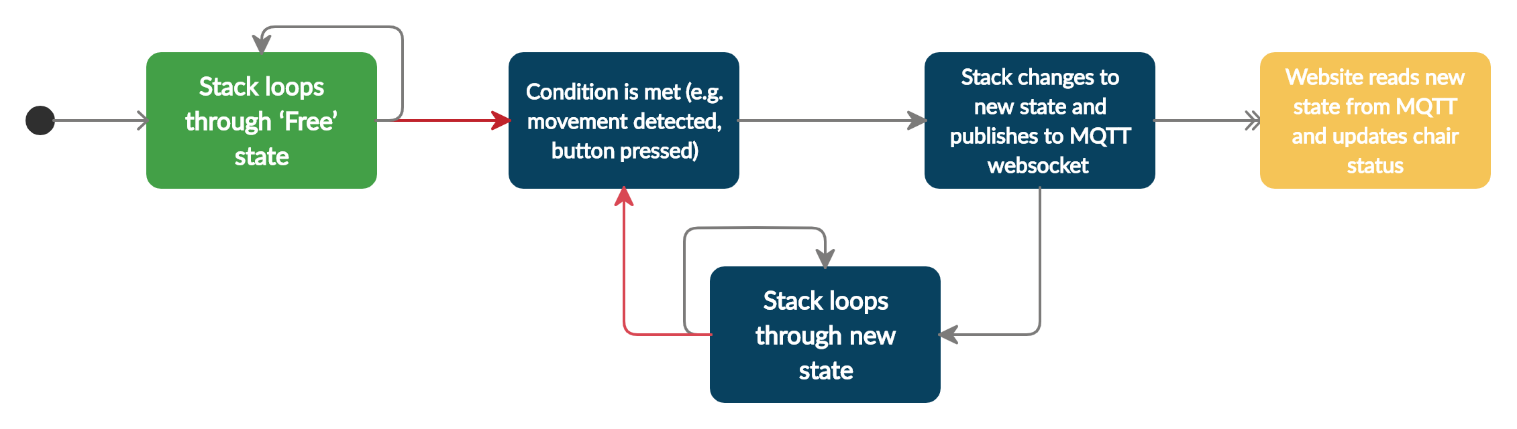
**M5 Stack/Stick Code**

****Both devices are programmed to follow the below state machine:

**Design Process:**

Initially had the stack set to free, then a lot of ‘if’ statements testing for the conditions above all in one loop. Once conditions were met, the appropriate message was published to MQTT (‘booked’, ‘use’, etc) which triggered the callback function when the stack read its own messages. The callback function then printed the appropriate screen to the stack and changed the state.

* This caused a significant delay between conditions changing (i.e. detecting movement) and the screen changing because it had to publish then read back its own message.
* Code was very messy with no clear functions, and trying to set a timer was very difficult.
* The ‘In Use’ mode only changed based on current movement.

Revised code design compartmentalised the states in to functions which run while loops until conditions are met to change them. Code now follows below process:

This refactoring removes the delay from publishing and reading, while still publishing the chair state for the website to update and keep track of.

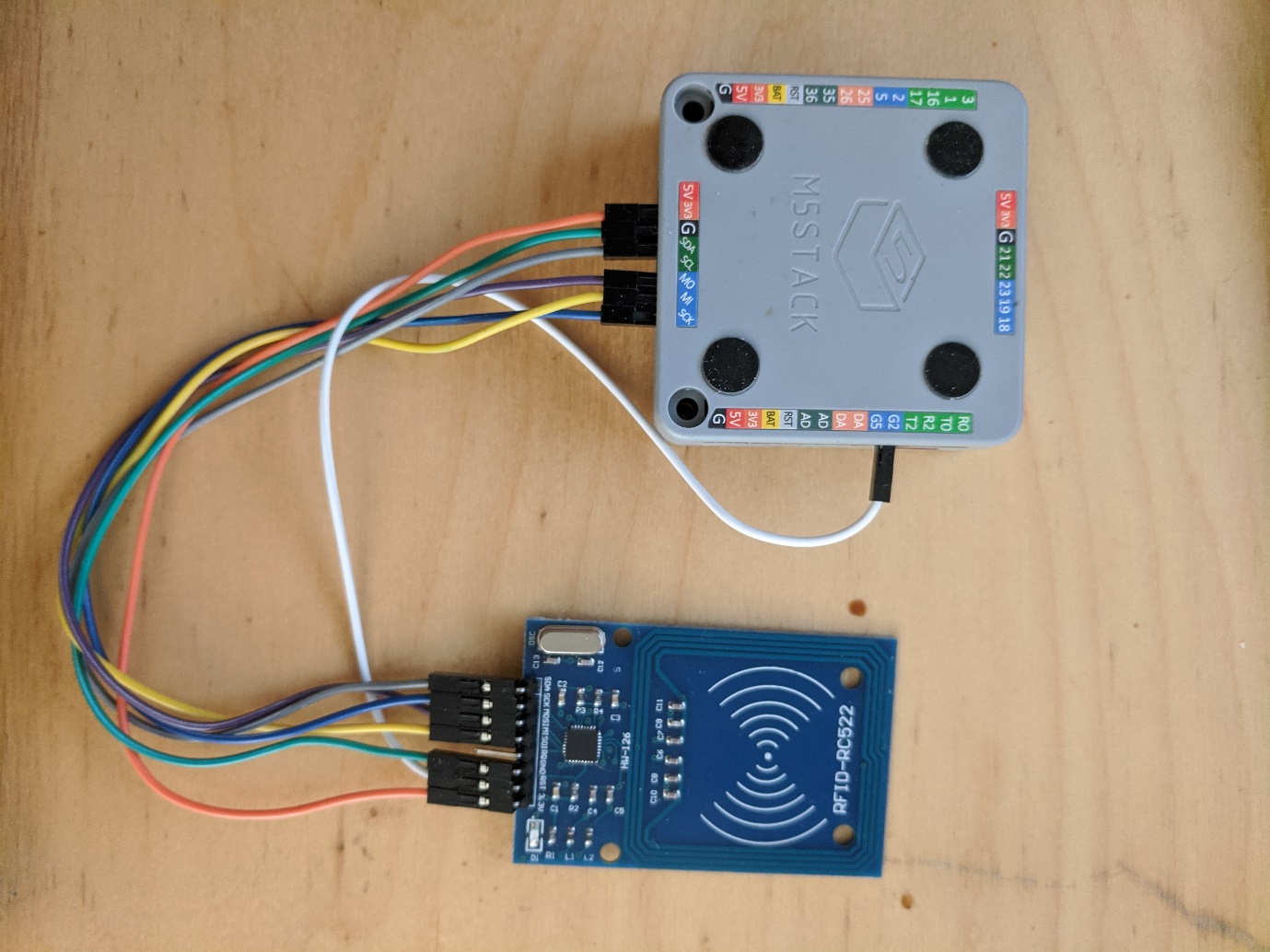
‘Use’ mode now tracks cumulative movement and has an initial starting value which decreases with time and increases when it detects movement (proportional to the amount of movement detected). This prevents the state from reverting back to ‘Free’ as soon as no movement is detected, and also stops the chair from staying in ‘Use’ when it is knocked by somebody walking past.

‘Lunch’ mode on the stick differs from the stack because of the RFID capabilities. Stack reverts to ‘Use’ from ‘Lunch’ when a card is scanned on the stack, but when it detects continued movement on the stick. ‘Booked’ is also disabled by pressing the button on the stick, but scanning a card on the stack.

**Implementing UCard (RFID) scanning functionality:**

In order to confirm the correct individual is using the chair, the student will need to scan their UCard upon arrival. This must be completed within 10 minutes of booking or the chair will be re-released into the pool of available chairs. The UCard acts as an RFID (Radio Frequency Identification Device) and its unique identifier is compared with the ID sent from the web app. If the ID sent from the web app matches the ID scanned, then the student has been successfully authenticated and can begin using the desk.

Reading the ID within the UCard would be achieved using an external RFID reader. The module chosen to achieve this was the MFRC522. The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56 MHz. This board is manufactured by NXP but can be purchased from the M5Stack store in an enclosure that connects to the stack’s 4-pin ‘GROVE’ connector. As the delivery times from the online store were large, the standalone MFRC522 board was purchased from eBay, this unit did not include a GROVE interface however. This module was therefore connected to the stack’s I/O bus via jump cables:



The MFRC522 supports two communication protocols, I2C (Inter-Integrated Circuit) and SPO (Serial Peripheral Interface). I2C is sometimes preferred over SPI because it allows for multiple ‘slave’ devices to communicate with multiple ‘masters’ (the stack in this case) using only two lines. One to transmit the data (SDA) and one to carry a synchronising clock signal (SCL). As there are only two lines, I2C requires each slave to be addressed and this is reflected in the MFRC522 object within the source code where the constructor requires an address to be provided when I2C mode is utilised:

*MFRC522(byte chipAddress);*

SPI offers faster data rates compared to I2C however needs at least 4 lines to be utilised, a Clock (SCK), a line to transfer data from the CPU to the peripheral MOSI, a line to transfer data from the peripheral back to the CPU (MISO) and a chip select line (CS) to select the peripheral in question (hence many more lines may be needed for multiple devices).

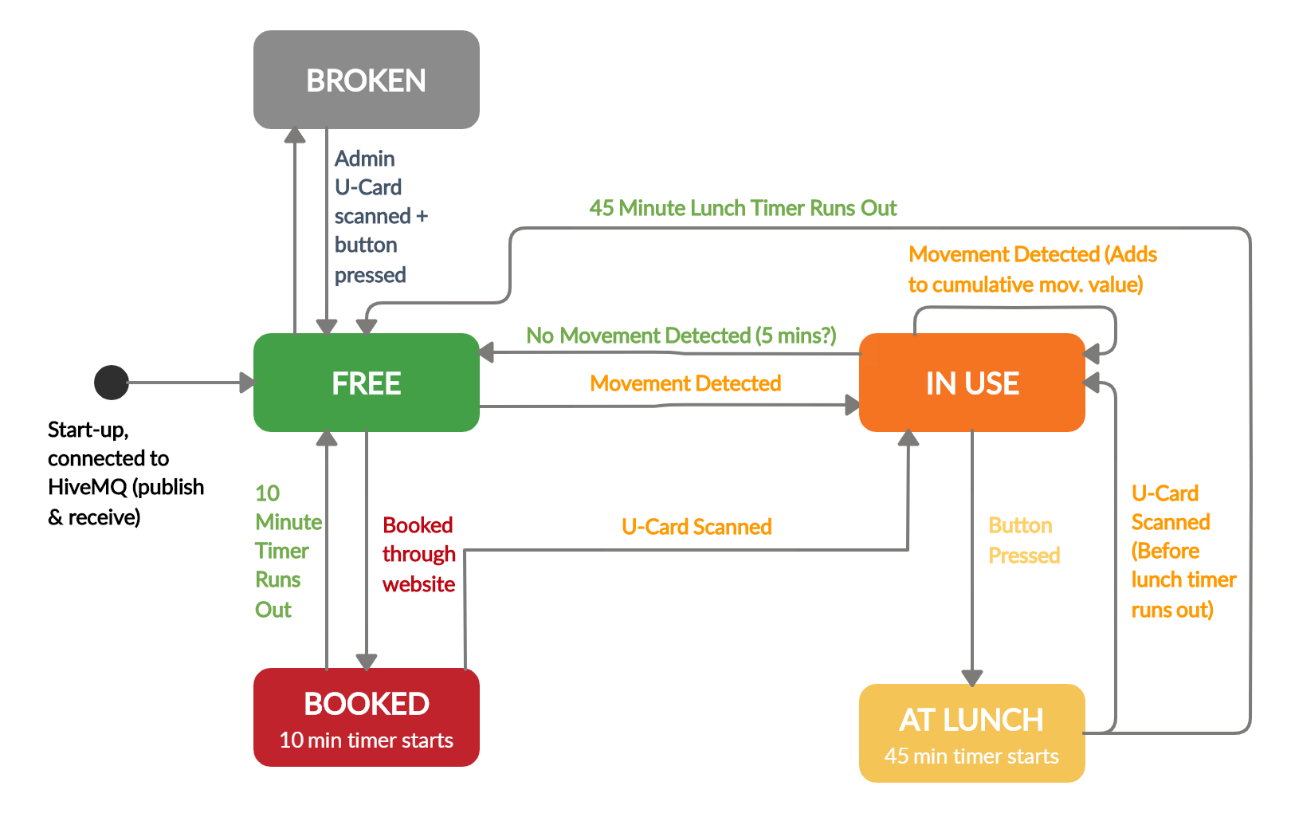
The constructor for the MFRC object in the source code utilising SPI therefore requires a requires the pin value of the peripheral to be selected as a parameter:

*MFRC522(byte chipSelectPin, byte resetPowerDownPin);*

Unfortunately, we were unable to get the MFRC522 to interface with the stack, this using either of these connection protocols. We were not sure whether the This may have been due to the fact that eBay bought MFRC522’s can often be temperamental.

**Portfolio Points:**

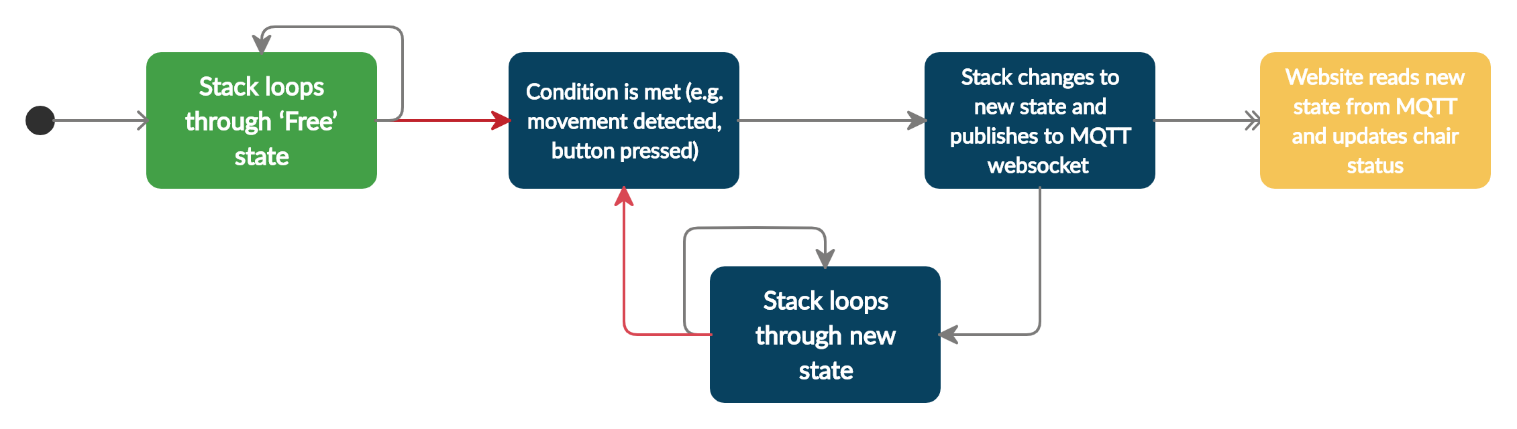
**1a) Architecture of the entire system**

**1b) Object-Oriented design of key sub-systems (e.g. Desktop Application, Web Application etc.)**

*M5Stack State Machine Diagram*

The M5 Stack subsystem is responsible for detecting the current state of the chair and displaying it, which is determined by following the above state machine diagram. The stack initialises at the black circle and moves to free, then stays in free until one of the stated conditions is met. It can only then move between the states which are connected by the arrows shown above (e.g. it cannot move from ‘booked’ to ‘at lunch’ directly). The stacks onboard accelerometer is used to measure vibration from the chair, and if it is above a certain threshold the state then moves to ‘in use’. While in use, the leftmost button can be used to start the chair user’s lunch break, initialising a 45-minute timer in which the chair is reserved and cannot leave this state. To exit this state, either the user must scan their U-Card, returning the chair to the ‘in use’ state, or the timer can run out, returning to the ‘free’ state (because evidently the user hasn’t made it back in time from lunch, so their seat is now available).

From ‘free’, the chair can also change to the ‘booked’ state, which means that a user has reserved that seat through the website application. This initialises a 10-minute timer, allowing the user to get to the seat and scan their U-Card, changing the state of the chair to ‘in use’. If they fail to scan their U-Card within 10 minutes, the chair state reverts to ‘free’. A fifth administrator state was added to the state machine, allowing the chair to be declared ‘broken’ if an administrator U-Card is scanned and the leftmost button is pressed. Changing to this state then notifies the processing application that a chair is broken so it can be repaired – the only way out of this state is for an admin to once again scan their U-Card and press a button, thereby preventing the chair from being used when it is broken.



*M5Stack MQTT Logic Diagram*

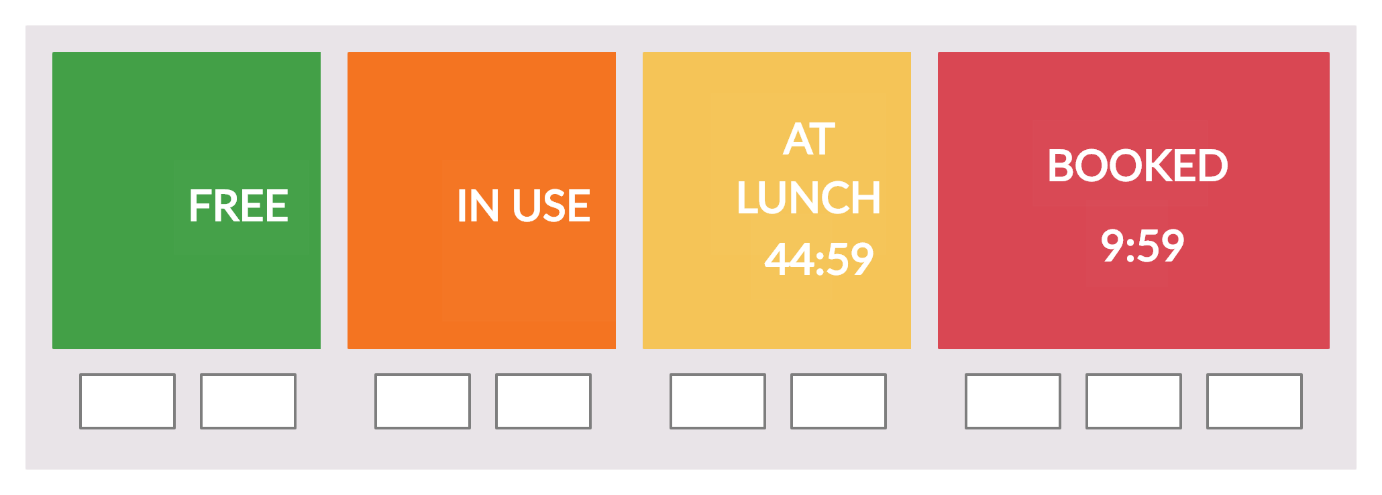
When the stack changes between two states, it publishes its new state to the MQTT WebSocket (which the web client receives and updates on the website and within the database). This can be seen in the diagram above; where a condition is met, the state is changed and published to MQTT, and then the stack loops through its new state until another condition is met and the state changes again.

The stack can also receive information from the website and send information to the processing application. Both of these are compartmentalised and it is only able to receive the ‘booked’ command from the website when it is in ‘free’ mode (as dictated by the state machine), and it will only send out information to the processing application when it enters the ‘broken’ state (this is also achieved through the use of MQTT).

In terms of Object-Oriented design, the code for the stack itself is written in C/C++ in a functional way, but in terms of the entire system the stack in itself is very much a Stack object, which has an id and a state which can be passed around and stored by the processing and web applications.

**1c) Requirements of key sub-systems (in the form of selected user stories)**

**1d) The evolution of UI wireframes for key sub-systems**



*M5Stack Initial User Interface Designs*

The initial user interface designs for the M5 Stack were basic, simply indicating the current state with text and a unique colour so that the user could see the state of the chair from a distance. The ‘lunch’ and ‘booked’ states also had timers counting down to indicate the time left in that state – both reverted back to free if the timers expired without any user input. As is included in the state machine, the lunch and booked states are designed for the user to be able to scan their U-Card to access that chair but as a temporary measure for the initial design, the user simply had to press the left-most button to access it.

**Add real pictures of latest designs – with date/time, chair id/ lunch prompt**

**(Also insert pictures from video demonstration?)**

*Later versions of the UI include a time/date in the corner, along with each chair id which allows administrators to know the chairs intended location within a room* (by checking against the web/processing app). *A prompt on the ‘in use’ screen also tells the user that they can start their lunch break by pressing the leftmost button.* Implementing an RFID scanner was attempted to add the ability to scan a U-Card so it could be accessed from the ‘lunch’ and ‘booked’ states, however this proved exceptionally difficult. *As a more suitable temporary measure, a unique combination of button presses, which is sent to their account on the web app, is now required by the user to disable these states.*

**Add picture of ‘broken’ state**

*As a later design change, we also added in the fifth state of ‘broken’, which allows an administrator to scan their U-Card, press a button and report the chair as broken to the processing application (so they can arrange for it to be repaired).* **USER STORY**

**Add picture of ‘setup’ state (if we decide to do it)**

*Finally, so that the stacks could be more ‘plug and play’ and the id isn’t hard-coded, we created a sixth ‘setup’ mode which the stack initialises in to, allowing an administrator to set the 8 digit chair id. This screen will only show on first use, so every other time the stack starts it will go straight to the ‘free’ state.* **USER STORY**

**2a) Breakdown of project into sprints (showing the users stories implemented in each).**

**SPRINT 1:**

* Deciding on the stack functionality, the different states and how it could move between them.
* Drawing up wireframes + creating a paper prototype for the stack on the back of a chair
* Wrote initial code to display different chair statuses and publish/subscribe to MQTT so the web app could initialise ‘booked’ state, with button functionality for lunch
* Issues to improve for next SPRINT
  + Long delay between publishing and changing states, change so that MQTT status takes precedence and is only published to when state is changed (details in m5 documentation document)
  + Add booking, is use and at lunch timers which display on screen
  + Design state machine for M5 Stack system

**SPRINT 2:**

* Drew a formal version of the M5 Stack state machine so that its functionality was clear to the rest of the team (some confusion over the exact functionality)
* Refactored code to follow the state machine (meant that stack didn’t have to publish state, read in the new state, and then change state which fixed the delay – now follows the M5Stack MQTT logic diagram)
* Added the BOOKED and LUNCH timers to the screen – user interface now looked like the UI initial diagrams
* Issues to improve for next SPRINT:
  + ‘in use’ mode only changes based on instantaneous movement – want it to factor in previous movement as well so it stays in use while the user sits still
  + Decide on JSON format for communication protocols

**SPRINT 3:**

* Updated the M5 Stack state machine – user now scans U-card to return from booked and lunch
* Refactored the code so that the ‘in use’ state depends on a cumulative movement value (details in m5 documentation)
* Change MQTT publishing format to match the JSON format used by the web and processing applications

**SPRINT 4:**

* Added the ‘broken’ state to the state machine *and wrote the code for the ‘broken’ state*
* Started writing documentation for stack design process and current (final) design

**SPRINT 5:**

* *Added a ‘setup’ mode to the state machine and the code?*

**Future Work**

* Studies will need to be conducted to determine the correct level of vibration detection so that the chair stays in the ‘use’ state for the entire duration that a person is sitting in the chair. This is to avoid the undesirable situation of the chair reverting back to free while they are sitting still, or if they get up for a minute or two (which is partly solved using the cumulative movement values in the ‘use’ loop, but the numbers used are only estimates). The vibration detection limits could also vary with stack placement and from stack-to-stack, so this will also need to be investigated before a large scale installation is carried out.
* *(IF WE DO THE SETUP MODE) Add the functionality to change the chair id by scanning an administrator U-card, entering setup mode and changing it (this may require the addition of a seventh ‘admin’ state.*
* Change the ‘in use’ to ‘lunch’ state so that the leftmost button must be pressed twice to go to lunch (to prevent accidental pressing) – **CHANGED BECAUSE OF USER FEEDBACK IN A LATER SPRINT?**