System Design

# Abstract

A: Architecture of the entire system.

The architecture of the system centres around ‘chair’ objects, with 3 different sub-systems all interacting with the ‘chair’ objects in different ways. These 3 sub-systems also communicate with each other, sending information regarding the number of chairs, the state that any chairs are currently in and if there is a booking made for the chair. The communication is carried out by sending JSON objects through MQTT, a lightweight messaging protocol, with the messages automated by the programs in the applications. The 3 applications are designed for 3 different use cases.

1. The desktop application, built using Processing, is for administration staff at a university. It provides tools for adding rooms and library spaces to different buildings in the university and for specifying the number of chairs that are contained in each room. The admin can add and remove chairs freely (for example if a chair is damaged, or adding chairs during busy exam periods) as well as adding and removing rooms (if a room is under construction, or if it will be booked). This information is published to the web application using a JSON string sent through MQTT.
2. The web application, built using HTML, CSS and node.js, is designed for students who wish to see the current current states of the rooms (e.g. which libraries are the busiest or the number of free chairs in each room) and to book a chair. It is run using an express server, so it can continuously be checking for messages from the other devices. It subscribes to the desktop application through MQTT so that the website can be updated with the correct number of rooms in each building and the correct number of chairs within each room. This information is saved in a nosql database (MongoDB) so that information can be reloaded (rather than the desktop application continuously sending messages regarding the current room states). MongoDB was chosen because it is consistent with the JSON string messages received from the desktop application. A sql database was also considered and could have been an equally suitable solution, due to some of the entities having relational features (each building contains many rooms and each room contains many chairs). Originally with the first prototype, local storage was used. However this proved to be problematic. More specifically, the user had to be on the website whilst a message was received from the desktop application or the stack. This would rely on both the other applications having to send messages nearly continuously to ensure the web pages were concurrent with the current state of the rooms and chairs. Also if a user was to delete their local storage then they would again lose all the information and could potentially try to make a booking on a chair that was already occupied by someone else. The web application also subscribes to the m5stack so that it can update the web page with the current state of the different chairs (occupied, free, booked or at lunch). This information is stored in the database. Using this information, students are able to see the current occupancy of each room and to make a choice about where they would like to study. Further to this, a student is able to select a chair and make a booking. This will subsequently publish a message to the m5stack on that particular chair, reserving it for that user. When the chair becomes free again the m5stack will send a message back to the web application, informing it that it is now free.
3. STACK STUFF

Evolution of UI wireframes for website:

-

System Design

a)

b)

c) Requirements of key sub-systems:

The website is the front end of the system. It allows students or others to get an overview of which study spaces are busy, and to book free spaces. To be able to do this, it needs to accept setup input in the form of JSON objects from the admin processing application, and have a two way communication with the stack components again through JSON objects to set a chair to booked as well as receive a booked or free status from the stack. It then needs to be able to present this information in a user friendly UI on the website. needs to show the user confirmation that the seat in question has been booked, as well as data persistence that allows the student to verify which exact chair he has booked if leaving and then coming back to the website. Add more stuff here that we then say actually wasn’t a good idea. Add hours during which booked?

d)Evolution of wireframes:

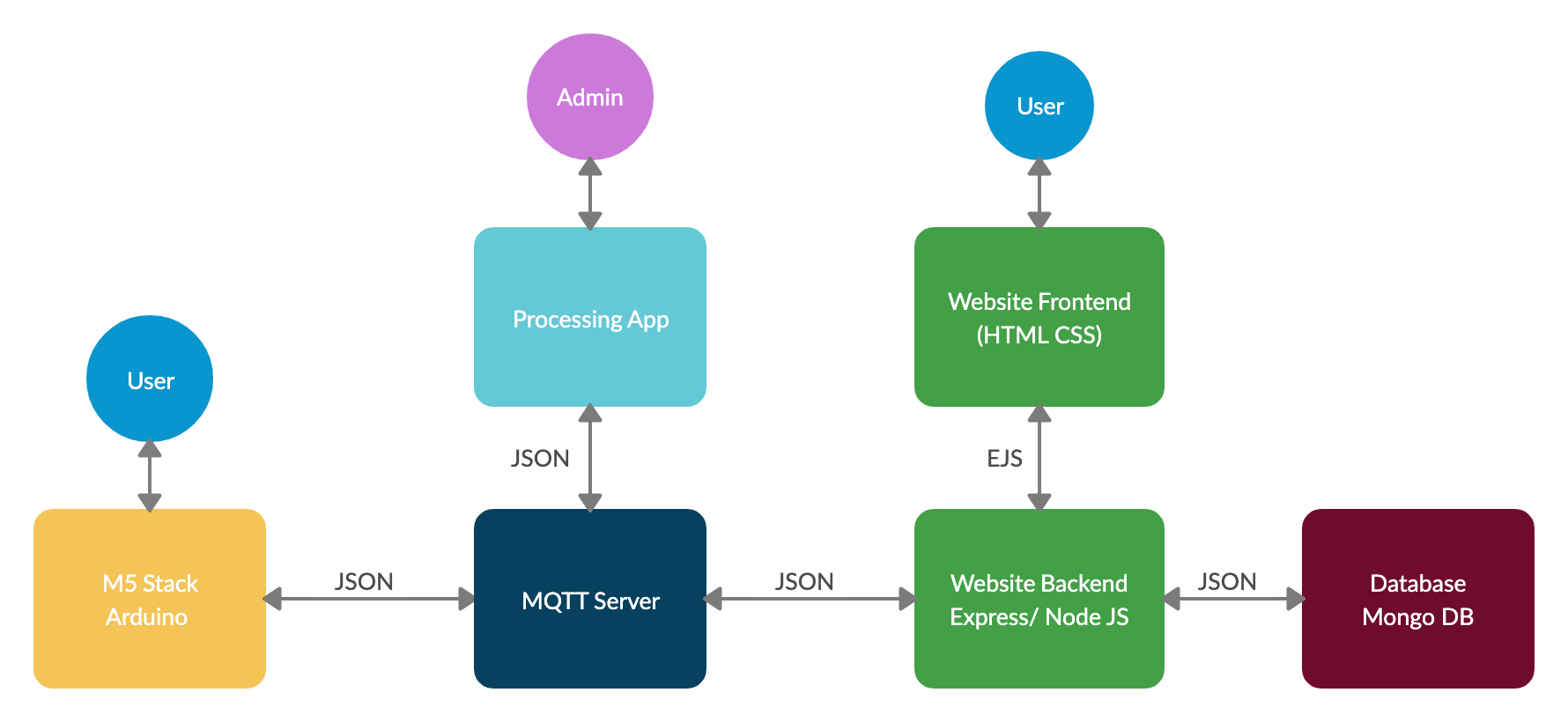
Paper wire frames – then static wireframes – finally screenshots of the real web application

Another round? To show evolution

e) MQTT

f) Mongo DB - JSON

g) HTML CSS NODE JS EXPRESS EJS – Make a diagram? Underneath justification for use?



# System Implementation

Broad Timeline diagram showing what was accomplished in each sprint over the entire 2 months?

1. Sprint 1: The first sprint focused on fleshing out the front end and back end design of the different components. We created various different diagrams and UI wireframes building on our initial paper prototypes. In the case of the website, this involved developing the UI and then creating a basic static website in HTML and CSS that implemented the parts of the UI that we could do without any Javascript.

Evaluation: To evaluate the work of this sprint we had a group meeting where each subsystem group gave their work to another to discuss.

Insights: We decided to not actually implement a login page/user accounts as it would take significant time to develop and for our purposes only a single user was necessary to demonstrate how the system would work. Limitations?

1. Sprint 2: The goal of the second sprint was the validate a working MVP of the entire system, having all of the individual components communicating via JSON files sent through MQTT. To achieve this for the website, we created a basic Javascript program to connect to MQTT. At this early stage we were saving the chair data in the browser’s local storage on reception and using that data to dynamically update the frontend to reflect booked/free chairs.

Evaluation: To evaluate this stage, as we had a working MVP we decided to do a bit of user testing with three of our friends to represent each of the three user stories, explaining the basic concept and getting them to try to use the system. Following that we also decided to attempt different ways of breaking the system. Limitations?

Insights: Local storage with simple JS script wouldn’t work as a data persistence mechanism as it requires the user to be on the specific webpage whilst a message was received from the desktop application or the stack. This would rely on both the other applications having to send messages nearly continuously to ensure the web pages were concurrent with the current state of the rooms and chairs. Also if a user was to delete their local storage then they would again lose all the information and could potentially try to make a booking on a chair that was already occupied by someone else.

1. Sprint 3: This sprint focused on implementing an Express server to run the website as well as a Mongo DB database to store the chair statuses. The Express server would have the advantage of being running continuously in the background to not miss out on any MQTT messages.

Evaluation: Team code review, how can we make things more efficient? Compared what we have achieved so far with what we originally set out to do? Limitations?

Insights: Decision was made to unify the JSON objects for simplicity: having one JSON object with building id, room id, array of chairs. Use EJS to simplify updating front end. Add confirmation page feature for website.

1. Sprint 4 focused on implementing these last features.

Evaluation:

Insights: chairs need to be numbered as you can’t find them

Evaluation techniques – SWOT at the end to evaluate the entire project.

# Project Evaluation

1. The project overall turned out to be a success? Talk about how the processing part felt like ticking a box and could have actually been done through the website to simplify things with an admin login? Not sure if we have really done OOP for most the project. We accomplished most of what we set out to do. Perhaps with some features not as developed as they could be: different user accounts, RFID scanner etc. A lack of testing?
2. Implementing above features, better UI. More robust system testing. More User testing