IoT Data Trading User Study Platform

Benjamin Thornton

Supervisor: Charith Perera

Moderator: Padraig Corcoran

CM3203: One Semester Individual Project | 40 Credits

Abstract

This project develops a web application which takes the user-centric approach of investigating the problem of the willingness of people to trade different types of IoT data to different organisations by creating a platform where people can create user studies, participate and analyse responses to them. It also involves conducting a user study to address the problem at hand and analyse how well the application is able to analyse the trends in responses from the user study.

Acknowledgements

I would like to thank my project supervisor Chairth Perera for his direction and suggestions which have been critical in the completion and success of this project. I would also like to thank those who have taken their time to participate in the user studies.

Table of Contents

Table of Figures

1. Introduction

1.1. Problem

The number of connected Internet of Things (IoT) devices is skyrocketing, with expectations of over 125 billion connected devices in 2030 (IHS Markit, 2017). Alongside the growth of physical devices comes an even larger amount of data produced by them. This vast amount of data leads to opportunities not only for data consumers to gain a greater amount of knowledge about their customers or area of interest but to give sensor owners greater control over their data and the ability to monetise it for these data consumers (Perera, et al., 2014).

Such a model which involves data exchange between sensor owners and data consumers is known as Sensing as a Service (S^2aaS). Implementing such a service has challenges in terms of its economic sustainability, requiring it to have a fair and transparent financial model (Perera, et al., 2014). The S^2aaS model is a market where there are many different types of data and many different types of data consumer. Sensor owners' attitudes towards data consumers differs, similarly their willingness towards selling different types of data is inconsistent. Therefore, there is a problem understanding what data a sensor owner would be willing to sell to different data consumers and the differences in prices required for the data. Such a problem needs to be investigated in order to address some of the economic challenges which could face such a service if it was implemented in the future.

1.2. Solution

Many companies attempt to strike parity between a product price and a customer's perceptions by utilising customer focus groups or surveys, etc (Cross & Dixit, 2005). The same user-centric approach will be applied here to analyse perception towards the sale of their data. In order to conduct such a survey, the design, development and testing of an application is proposed which allows the creation and participation of an interactive study. The study will allow its participants to rank items in one category against items in another and to give numerical values to their ranking. To get insights behind how participants are responding, the project also involves creating tools to analyse the data from the survey in a variety of ways.

User studies using the developed application were also conducted to attempt to address the problem of willingness to trade data to different organisations and at what price. The user studies were also performed to see how well the application could identify the kind of patterns in participant responses, verifying its potential to solve the data trading problem.

Beyond the use-case utilised in this project using such an abstract solution could be useful to researchers who are doing user studies to understand the relationships between pairs of categories over many different demographics. Alternatively, organisations wishing to implement such a marketplace of data in exchange for rewards in the future may want to quantify the rewards required for different data types over different demographics and this application may be a tool for them to achieve that.

1.3. Outcomes

The main outcomes resulting from this project are:

- To create an adaptable web application allowing:
    - Creation of studies where participants can respond by ranking and give numerical valuation to pairs of categories
    - Evaluation of those responses through various metrics.

- To conduct user studies with the application to investigate the willingness a group of people possess on trading different types of data to different organisations and the price required for the trade to take place. The user studies will also verify the applications potential to solve the data trading problem through its analysis tools.

2. Background

2.1. Internet of Things (IoT) and Sensing as a Service (S^2aaS)

The Internet of Things (IoT) describes the network of physical objects embedded with sensors and/or actuators, software and other technologies. These now have the ability to communicate over a network without needing human-to-human or human-to-computer interaction (Rouse, et al., 2020). IoT has a vast number of applications from consumer to health to military and solutions such as wearables, thermostats, trackers which have all come to the market in recent years. This has resulted in an explosion of data.

These IoT solutions all store data collected via sensors or otherwise independently. However, a significant amount of knowledge can be gained by aggregating these stores together hence improving both data consumers operations through increased consumer knowledge, reduced resource wastage, etc. Data owners (IoT device owners) may also receive the benefit of such data aggregated analysis through a greater personalisation of a service.

By giving a data owner the ability to trade data about themselves they could gain the benefit of having greater control over their data and rewards for sharing that data (i.e. monetary, loyalty points, gifts etc) to potentially recoup (or in part) the cost of their IoT device investment. In turn, data consumers (i.e. organisations) gain the benefit of an increased number of different data points so they can understand users better, have a better idea of where to invest and make more informed product or services decisions (Perera, 2017).

Figure 1 illustrates the steps behind the sensing as a service model. In steps 1-2 the figure shows information about the sensors being uploaded to the cloud so their potential to collect data can be recognised. Step 3 shows Mike giving permission for the sensor data to be uploaded to the cloud. After Mike has given permission, a data consumer requests the data held by the cloud service and provides an offer of value to Mike, be it monetary or otherwise. In step 5 he can choose to accept or reject the offer. If the offer is accepted, step 6 indicates the flow of data from the sensors to the data consumer. This scenario shows all of the interactions required in the model for background, however this project intends to focus on step 4 and 5 of the model and more specifically creating a tool to have

understanding between an data consumer and sensor owner about what kind of benefit a user would need to see for a certain type of data i.e. the sidedness of the trade.

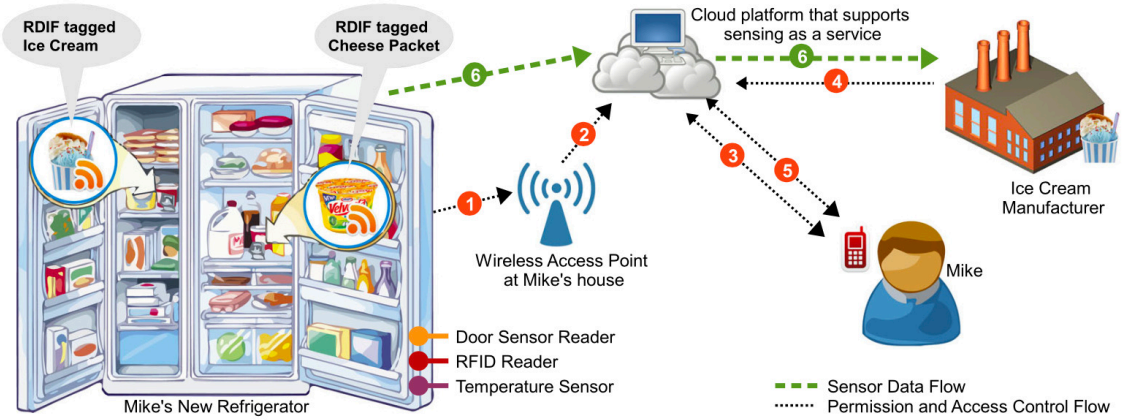


FIGURE 1: FUTURISTIC SCENARIO SHOWING THE INTERACTIONS IN THE SENSING AS A SERVICE MODEL (PERERA, ET AL., 2014)

2.2.Surveys

A survey is often used an examination of thoughts, opinions and behaviour through a set of predetermined questions given to a sample (Shaughnessy, et al., 2000). They are utilised to draw attitudes on a particular subject matter of a particular population, if that sample in the survey is representative of the larger population of interest. Surveys are conducted through many different physical mediums such as in person, paper based and on the web.

Over the past decade, online surveys have been increasingly popular over paper based or traditional surveys as they have shown to receive a greater number of responses due to their simplicity, a similar accuracy of response and ultimately are much more cost effective (Greenlaw & Brown-Welty, 2009; W3Counter, 2020). It is clear that by using a web-based solution, it would provide the best way to address the data trading problem at hand.

Many such online survey applications already exist for market researchers to take advantage of, they range from the more basic implementations such as Google Forms to more professional systems such as Typeform which allows more control over the UI and more advanced analysis.

I kept these solutions in mind when developing my application, however, these applications are general solutions and only utilise general HTML form components like text boxes, select boxes, radio buttons albeit with a few extras. In order to address the problem of the willingness to trade data I needed to create a more unique survey tool which could be able to draw attitudes of a representative population whilst still remaining relevant beyond the particular use case I am utilising it for.

2.3. Data Pricing Models

There have already been a number of research articles (Niyato, et al., 2015; Liang, et al., 2018; Mao, et al., 2019) published discussing and proposing new pricing models which should be implemented in order to maximise data consumers profitability. However,

research in this area tends to focus more on the economics from a data consumer side rather than taking a sensor owner centric approach in determining the sensitivity towards different types of data.

Consumer centric methods for determining price preferences such as Van Westendorp's Price Sensitivity Meter (Westendorp, 1976) have been used by a wide variety of researchers in the market research industry. However, this method focuses on a physical product or service that a consumer would want to buy rather than a piece of data they would want to sell.

3. Specification

3.1. Functional Requirements

The following functional requirements entail the specific functions that the application must implement in order to be useful to address the problem at hand, remaining within the scope of the project whilst still being useful beyond the problem that it is utilised for in this report.

The functional requirements left out detail to allow for implementation specific constraints and for the application to be adjusted or added to easily upon review.

3.1.1. Requirement 1

Title: An admin must be able to create "Card Sets"

Description: In order to represent items in a category that a participant can rank in a survey an administrator must be able to create a logical grouping of these items known as "Cards" into groupings known as a "Card Set"

Acceptance Criteria:

- An administrator must be able to create a group of cards where each card has a name and an optional description and image.
- An administrator must be able to specify a name for a card set and a measure by which the cards are ranked against in the study.

3.1.2. Requirement 2

Title: An administrator must be able to create "User Groups"

Description: In order to add any number of participants to a survey, a set of users must be created.

Acceptance Criteria:

- An administrator must be able to create a group of users where each user is specified by an email.
- An administrator must be able to specify a name for the user group.

3.1.3. Requirement 3

Title: An administrator must be able to create "Studies"

Description: In order to allow a User Group to participate in a study and allow the administrator to control what Card Sets the participants will be ranking and under what question(s) they will be providing a value for their reasoning.

Acceptance Criteria:

- Admin must be able to select 2 card sets for the study.
- Admin must be able to select 1 User Group For the study.
- Admin must be able to select how many numerical values the participant will be entering & their associated label.

3.1.4. Requirement 4

Title: A participant must be able to provide details about themselves.

Description: In order to draw an opinion of a given population, we need to identify the demographics of the participants taking part in the study.

Acceptance Criteria:

- Participant must be able to enter their age group, salary range, occupation, country of birth, current country, education level.

3.1.5. Requirement 5

Title: A participant must be able to participate in a study.

Description: In order to collect opinions on the ranking of different items in Card Sets against each other and the value behind their reasoning we must allow them to participate in such a study which makes this possible.

Acceptance Criteria:

- Participant must be able to rank the cards in each card set against each other
- Participant must be able to give numerical value(s) related to their ranking

3.1.6. Requirement 6

Title: An admin must be able to view the responses of the studies they have created

Description: In order for opinions to be identified from a particular study, processed responses and unprocessed responses must be able to be viewed.

Acceptance Criteria:

- Admin must be able to view the following for each study they have created:
  - Individual responses from participants
  - Average responses from participants
  - Heat maps showing a count of the position for every combination of cards from both Card Sets in the study.
  - Heat maps showing the average of normalised numerical values from each response for every combination of cards from both Card Sets in the study.
  - Charts showing the frequency of a rank given to each item over all responses in both Card Sets.

3.1.7. Requirement 7

Title: Access to the admin functionality should be secured

Description: To prevent unauthorised creation of responses and access to response data

Acceptance Criteria:

- Any attempt to access this functionality if the user is not authenticated and does not have the correct access rights should be rejected.

3.1.8. Requirement 8

Title: Access to participate in a Study should be secured

Description: In order to ensure the integrity of studies, nobody but those intended by the admin should be able to participate in a study.

Acceptance Criteria:

- Any attempt to access this functionality, if the user is not authenticated and does not have the correct access rights should be rejected.

3.1.9. Requirement 9

Title: Ensure an admin can retrieve data from the application using some common standard.

Description: In order to support analysis of data collected by the application further than what is provided with the application some method of retrieval of raw data is required.

Acceptance Criteria:

- Admin should be able to access data that they have produced themselves or by participants responding to their studies by other means than through the application.

3.1.10. Requirement 10

Title: Ensure any sensitive data about a participant cannot be traced back to the participant.

Description: Article 5(1)c of GDPR describes that personal data shall be “limited to what is necessary in relation to the purposes for which they are processed” (ICO, n.d.) as sensitive information is recorded about the demographics of the person who has participated and the information required to identify the person is no longer required, it should be removed.

Acceptance Criteria:

- By the time any sensitive information is stored in the database, no personally identifiable information shall be stored in the database.

3.1.11. Requirement 11

Title: The interface of the study should be modelled after the diagram shown in Figure 2.

Acceptance Criteria:

- The interface where participants rank items in one category against items in another category and give numerical values to their ranking should be similar to that of the representation in Figure 2.

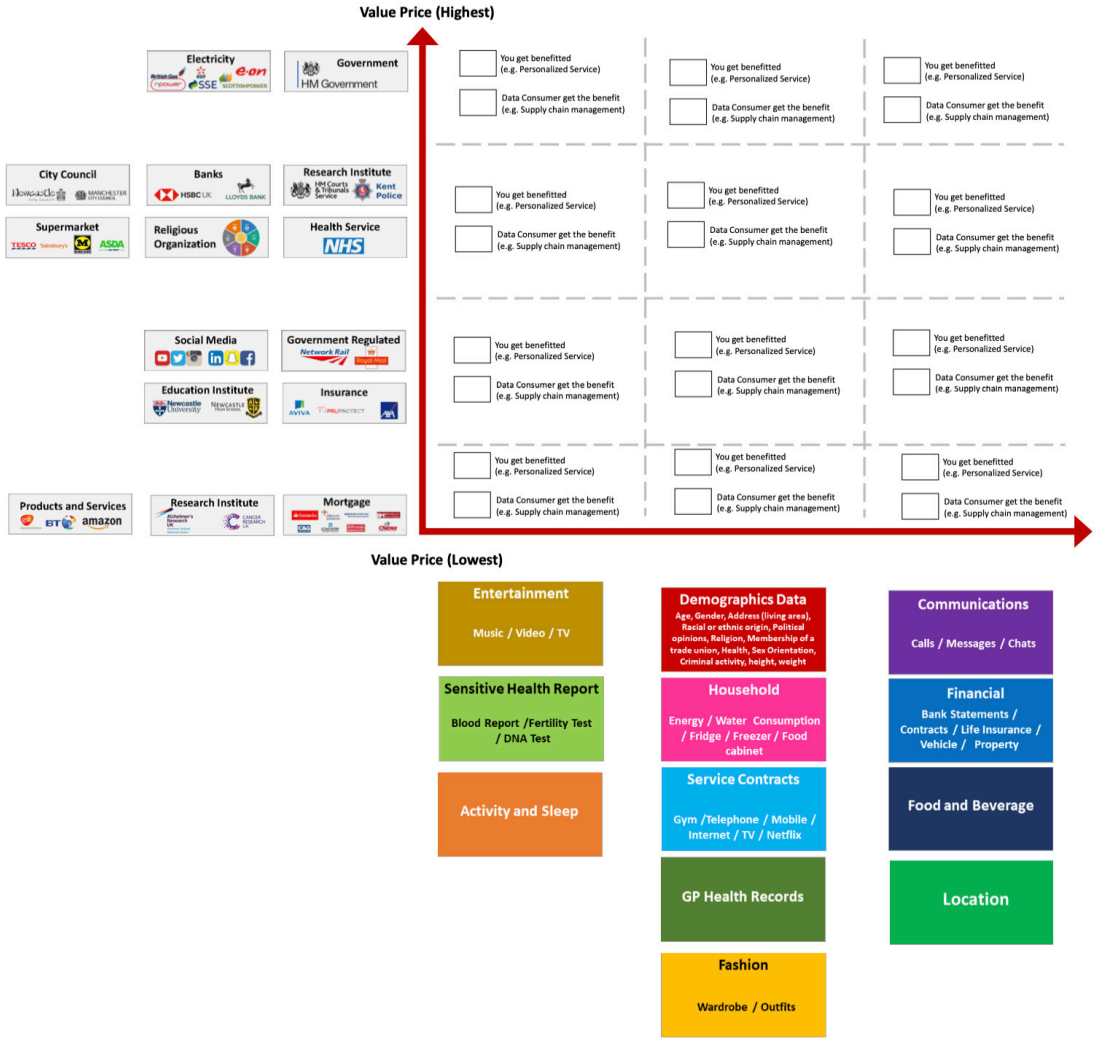


FIGURE 2: SURVEY INTERFACE REQUIREMENT

3.2. Non-Functional Requirements

The following non-functional requirements were identified to ensure the accessibility, security, reliability, performance, usability of the application. These non-functional requirements ensured that the despite the functionality the application was secure and those utilising the application could do so with ease.

3.2.1. Accessibility

1.1.1.1. Requirement 1

Title: The application should be supported across the main browsers

Description: In order to support the varying systems and settings of the users of the system it is important that there is cross-browser compatibility in order to ensure the experience is

Acceptance Criteria:

- The application must render the content in the same or very similar way across the top 5 Web browsers by market share which include Chrome, Safari, Edge, Firefox, Opera (W3Counter, 2020)

3.2.2. Security

1.1.1.2. Requirement 3

Title: All data shall be stored in a secure database

Description: As personal data is being collected, actions must be taken to minimise risk of any unauthorised access to the data

Acceptance Criteria:

- All data shall be stored within a database that is only accessible through the university network and is password secured.

1.1.1.3. Requirement 4

Title: The application should be hosted within the university network.

Description: As the application uses data from the database which stores personal information and is only intended for use within the university for this project. The risk of unauthorised access to personal data should be mitigated by limiting access to the application to only those who have access to the university network.

Acceptance Criteria:

- Application should not be accessible in any way to those who do not have access to the university network.

3.2.3. Reliability

1.1.1.4. Requirement 5

Title: The application must be able to operate as intended even under incorrect or unexpected use.

Description: In order to maintain the integrity of the system.

Acceptance Criteria:

- Errors must be handled appropriately as to not cause any harm to the operation of the system.

3.2.4. Performance

1.1.1.5. Requirement 6

Title: The application must respond quickly to any user action

Description: In order to support users on any system or network speed, it is essential that the system performs quickly even under load.

Acceptance Criteria:

- Application must load each view fully within 10 seconds

3.2.5. Usability

1.1.1.6. Requirement 7

Title: The application must be simple and intuitive to use.

Description: As there is no target demographic for this application, it is assumed people of all technological experience could potentially utilise the system hence it needs to be simple and attractive for anyone to use.

Acceptance Criteria:

- Where there are user interfaces, they should follow user interface design heuristics, common interface design patterns and any other guidelines or best practices.

3.3. Changes to Requirements

Only one notable change occurred to the functional requirements, namely functional Requirement 10 was incorporated to ease the ethics concerns of storing personal and sensitive data collected by the user study.

4. Design

4.1. Static Architecture

4.1.1. Modules

From the functional requirements, the application has been divided into a series of modules appropriate for each handling a subset of the requirements. Each module will also attempt to address every non-functional requirement where possible. Different viewpoints involved with the design of the application will then be applied to these modules.

1.1.1.7. Authentication

To authorise both participants and admins of the application as described in functional Requirement 7 and Requirement 8, we are required to create a module for authentication. Such a module will handle the login and logout flows for both parties.

1.1.1.8. Participation

To handle Requirement 4 and Requirement 5, allowing participant to enter details about themselves and to participate in a study we need a module to encapsulate such functionality.

1.1.1.9. Administration

To permit an admin to create studies, user groups and card sets as described in Requirement 1, Requirement 2 and Requirement 3 have been encapsulated into their own module.

1.1.1.10. Responses

In order to satisfy Requirement 6 another module is needed to handle the required methods in the ways desired.

1.1.1.11. API

In order to allow access to data for admins other than through the application as specified in Requirement 9, the API module is established to handle such functionality.

4.1.2. Application Architecture

For the general design of the of the application a web architecture was chosen in order to be able to support the majority of different hardware that the users of the application may have.

A web application architecture is a pattern of interaction between the web application components which consists of a client and server (Yaskevich, 2017). A client is typically a user's device which is capable of connecting to the internet and has web accessing software such as a web browser (Mozilla, n.d.). The client requests resources using a HTTP request from the server and is where the user interacts in some way with the requested resources. The server, on the other hand responds to requests for resources which it stores also using the HTTP protocol.

More specifically for the design of the proposed application, a multi-tier architecture was chosen which is slightly different from the single-tier architecture described above as the database is separate from the application server (Faircloth, 2017). Security is one reason of choosing such a multi-tiered design as it means that separating server code from data makes gaining both these entities more difficult as they are stored separately. Having a separate database server also lends itself to scalability benefits as multiple instances of a web server can be running simultaneously whilst all having the ability to access the same data. Consequently the application has the ability to have a greater amount of availability as if one instance of a web server fails, a new one can be created without losing any data (Bitnami, n.d.). Figure 3 shows the multi-tier architecture in use within the application.

FIGURE 3: MULTI-TIER ARCHITECTURE

There are two main approaches when designing the interaction between web server and client. One can either choose to utilise a single page application (SPA) or multi-page application (MPA). A SPA operates by retrieving all necessary resources such as HTML, JavaScript and CSS on initial request then page components are replaced by other components depending on user interaction. This means that the page does not need to be reloaded upon every user action (Madhuri, et al., 2015). One of the aims of SPAs is to make the user interface more fluid as a response is not always required from the web server in order to respond to a user action. Another big factor for larger companies such as Twitter and Facebook who use SPAs is that client-side loading reduces the pressure on servers handling requests. MPAs on the other hand require that resources are to be requested each time upon each user action (i.e. page change or data submission) and subsequently results in rendering a new page in the client (Lipski, 2017). In very large applications where large

resources are used across the application, MPAs can result in vast increases in average load times over SPAs.

Although there are clear benefits to SPAs in theory, many suggest that SPAs have greater amount of complexity involved in testing, error handling, etc (Łępicki, 2017; Navis, n.d.). Alongside the complexity gripes, I had little knowledge of JavaScript or TypeScript which are common languages amongst frameworks that utilise the SPA architecture and some of which have steep learning curves (Naumovski, 2017) making meeting deadlines more difficult provided the ambitious tasks of the project. Furthermore, given the size of the application and the number of users required to use the application would be small in respect to those companies who generate value from a SPA, It does not seem worth the additional complexities for minimal additional value.

Therefore, choosing a mixture of both an MPA for the majority of the application implementation whilst also incorporating features SPAs in some areas to meet some of the more interactive features required of the application that are challenging to be met with MPA's using a technology called AJAX which will be discussed later in the implementation section.

In order to focus on the main requirements of the application, and to not get too involved in areas outside of the scope of the project, a web framework was incorporated into the stack. A web framework commonly handles low level implementation overhead so that areas concerning web development such as HTTP specification, preventing common security flaws, etc. were not required to be developed alongside the project requirements and permit the project objectives to be achieved on time.

When designing the structure of the software I decided to use a Model-View-Controller (MVC) design pattern. The pattern is comprised of 3 modules; the model which defines what data is stored and is where data is stored, the view is what the user interacts with and finally the controller is the middleman between both view and model in that it updates the model when the user manipulates the view and visa-versa (Google, n.d.). When coding to such a pattern there are high amounts of cohesion amongst the elements in the modules. There is also loose coupling amongst the modules as they all serve a different purpose hence if a change in requirements occurred, large amounts of the application would not have to be altered as would be the case in a tightly coupled application. Both loose coupling and high cohesion are principles lead to a greater maintainability and reduced complexity of the module which is required as the application grows. Following such a pattern ensures that it will be manageable to both develop the application and to test the different components to see how far I have satisfied the requirements.

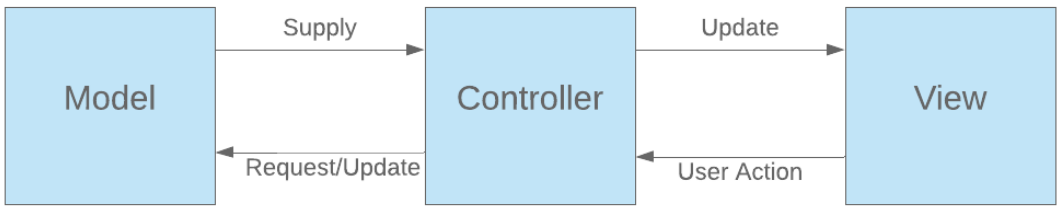


FIGURE 4: MODEL-VIEW-CONTROLLER (MVC) ARCHITECTURE

4.1.3. Entity Relationship Diagram

The following entity relationship diagram displays the final structure which the database has taken to store the data associated with the different modules described above.

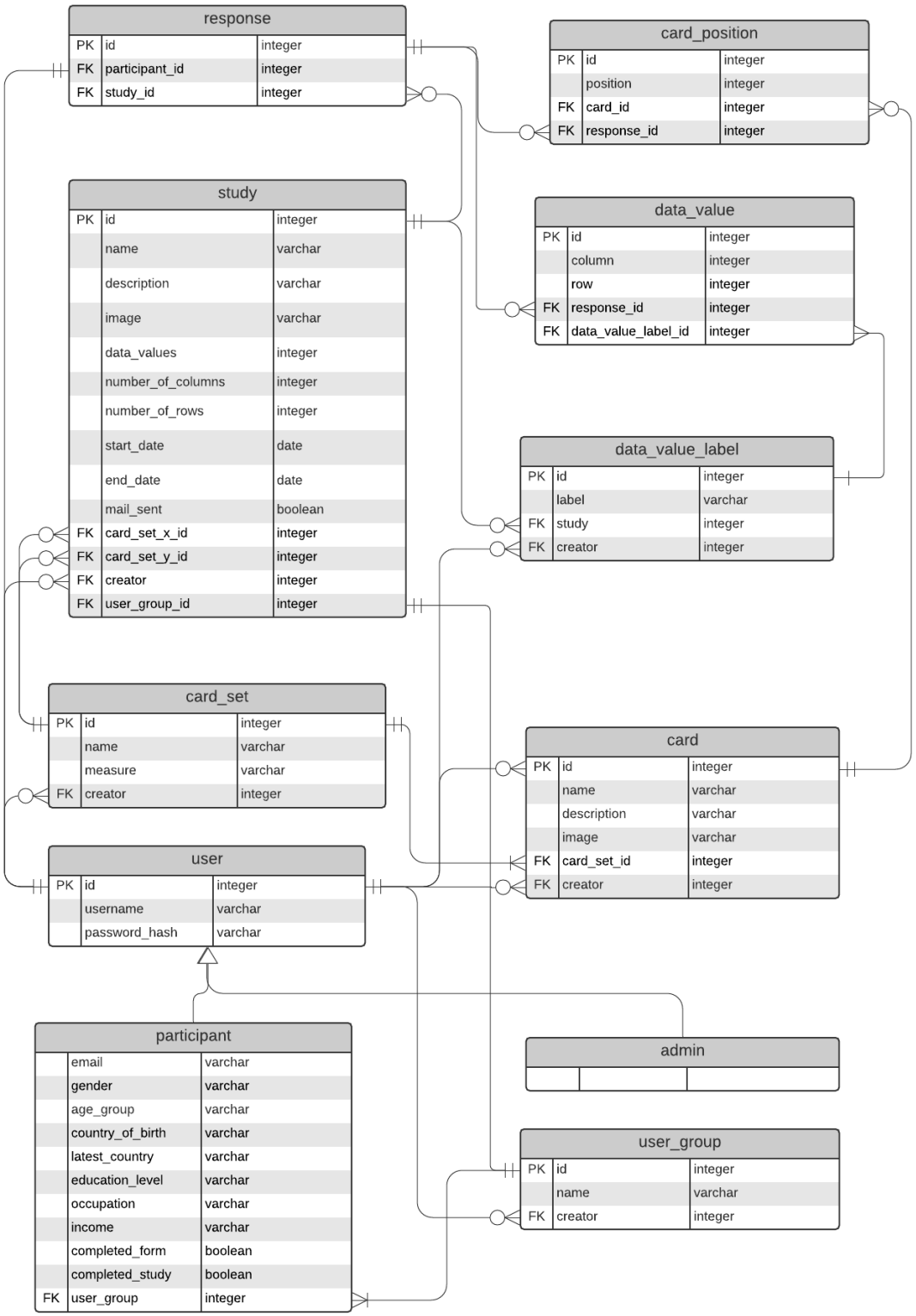


FIGURE 5: ENTITY RELATIONSHIP DIAGRAM

The ‘user’ table has two subclasses; admin and participant. These are used to store the two types of users of the application. Both subclasses are used to identify the user’s permissions i.e. what they can access within the application to uphold Requirement 7 and Requirement 8. The participant subclass stores the details about the participant for Requirement 4 and whether they have completed their study and details form. Most tables also contain a ‘creator’ attribute to identify which administrator created that tuple. This verifies that if there are multiple administrators in a single instance of the application, only data which they have created themselves or as a result of their studies can be retrieved. Such an attribute helps maintain the security requirements across the application.

The ‘user_group’ table is used to form logical groups of participants together so that they can then be assigned to a study, to enable those participants to have the permission to participate in that study.

The ‘card’ table represents an item which can be ranked, it stores related information such as a name, description to give the participant context behind what they are ranking. the ‘card_set’ table represents a category and constitutes a grouping of ‘cards’. The ‘card_set’ table also contains a ‘measure’ attribute which gives further context behind how the items (cards) should be ranked.

The ‘study’ table is then used to group two card sets and a user group together which make up the basic constituents of a study. two fields are used to explicitly connect a card set with an axis on the interface, because by only using a single foreign key we cannot ensure that some sort of order is enforced in the database. The model also stores other information for greater customisation of the study to make it more adaptable to the requirements of somebody conducting such a survey.

The data value label model is used to provide the label for the user entry in the study. Such a label helps to give context behind the numerical values participants are providing for their ranking of the cards within the study. Using a relationship between the study and data_value_label table for this feature means the user can define how many user entries they wish their study to have, allowing for a greater degree of freedom for the potential uses of the application.

The response table represents a link between a participant and how they responded to a study. In order to record how a participant has ranked items in a category, the ‘card_position’ table stores what position a participant has placed a card. Similarly, the numerical value placed upon a comparison of two items is represented using the ‘data_value’ table which stores the position of the value, value itself and the context behind the value with the relationship to the ‘data_value_label’ table. Both column and row attributes are used in the ‘data_value’ table in order to be able to locate the corresponding positions of cards in the ‘card_position’ table.

Previously, the ‘response’ table contained several attributes of type JSON which stored the ranking of items in a category and the values given to their ranking. This was not a good design, as by mixing the existing column-per-value schema that existed across every other table with JSON columns, it prevented the querying of individual items stored within that JSON structure.

4.2.User Interface

When designing the user interface (UI) for the application, I decided to create wireframes for the for each view in my application. Such an approach made me be able to try different layout rapidly the wireframes were able to be altered easily upon review.

When constructing the wireframes, I made sure to consider the usability non-functional requirements and went further by sticking to the Nielsen's 10 Usability Heuristics for User Interface Design (Nielsen, 1994). Heuristics are also called "mental shortcuts" they allow people to take off the cognitive load complex scenarios by consciously or unconsciously ignoring some information that is coming into the brain by applying these heuristics or "rules of thumb" (Lim, 2018). The heuristics made the prototyping of wireframes faster whilst still having a solid, factual usability reasoning behind the designs chosen.

For more specific, common problems, I applied UI design patterns which are tried and tested solutions that have proven to be effective at common usability requirements. Using such patterns reduced the time taken to create some elements of the interfaces and created a consistent look where these elements would persist across interfaces. As many other applications follow similar design patterns, they created a simple translation for the users from other applications, enhancing the intuitiveness of the application overall.

For the visual design of the wireframes and leading into the interface development I ensured I kept to the CARP principles (Williams, 1994). The name is an acronym for Contrast, Alignment, Repetition and Proximity. They aim to give the page visual interest through contrast, whilst giving a good page structure by organisation groups of related content together and repeating elements where needed to give a consistent look. This helps the application be more navigable and friendly for both types of user interacting with the system ensure that the usability requirements are met.

Beyond the visual elements I also thought about the information architecture on each page to ensure that different elements were easy to find through using one of location, alphabet, time, category and hierarchy (LATCH) to organise content where necessary (Wurman, 1989).

The figures below show some of the wireframes for the main requirements of the application and their justification relevant to the principles, design patterns and heuristics that have been discussed. All of the wireframes are available section 11.

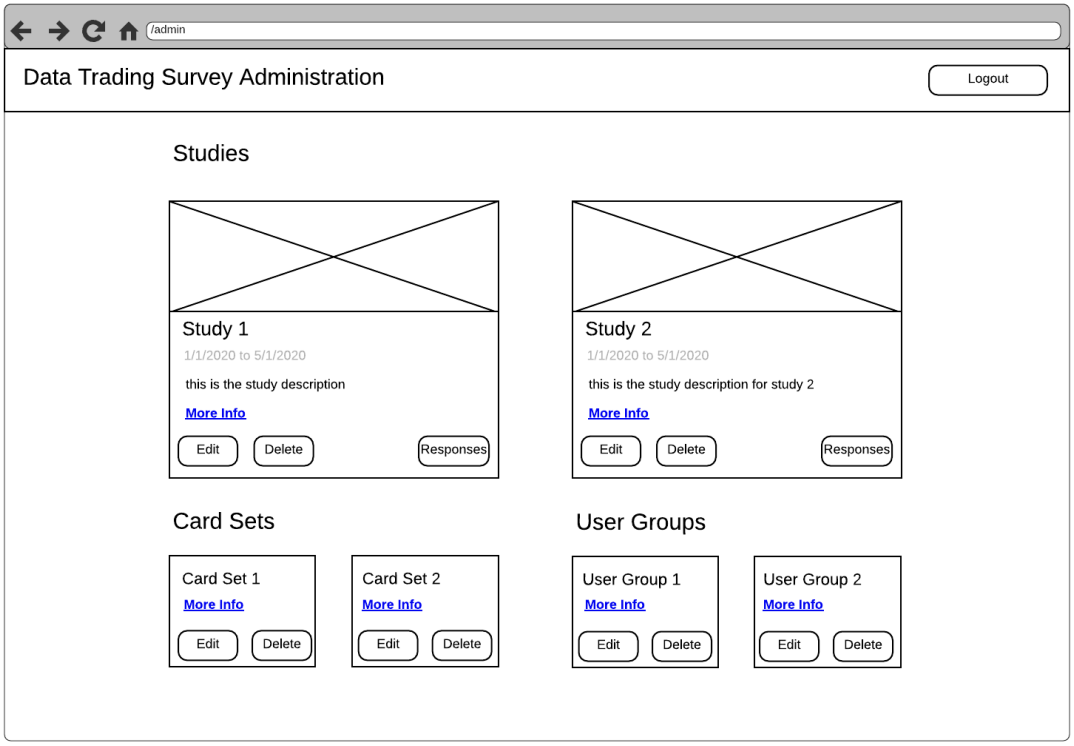


FIGURE 6: ADMIN HOME WIREFRAME

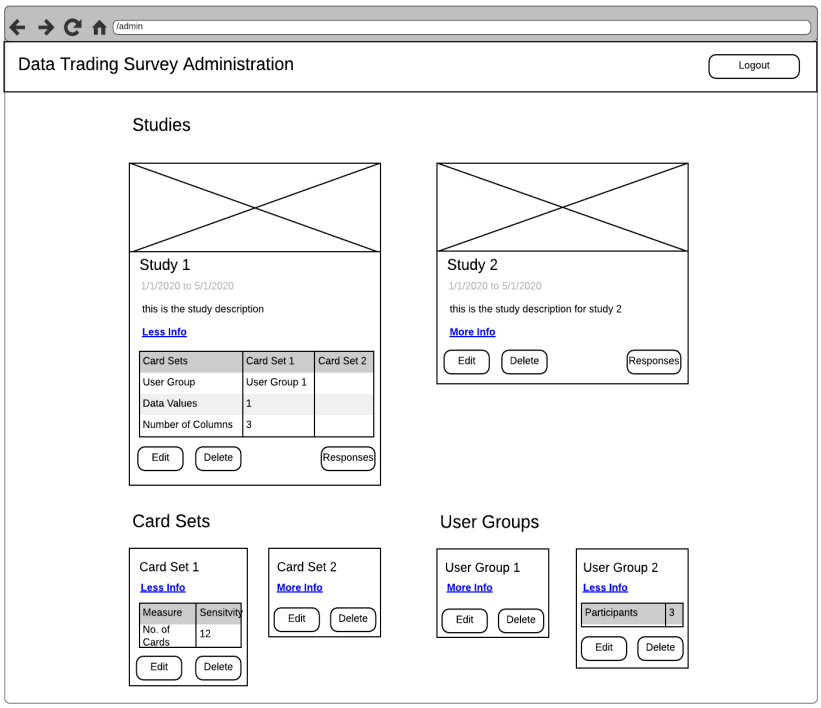


FIGURE 7: ADMIN HOME WIREFRAME 2

The administration home page shown in Figure 6 and Figure 7 gives an overview of all the studies, card sets or user groups the administrator has created. This gives them visibility about all of the different elements they have created.

Each object is represented as a ‘card’, Representing information like this gathers the attributes of the objects together into one coherent piece. Separating information in this way makes the information more digestible, hence the administrator can find information they are interested in easier (UI Patterns, n.d.).

Cards are also intuitive, in that they are used across many domains such as trading cards, business cards, etc. there is a match between the application and real world, following heuristic 8 of Nielsen’s usability heuristics (Nielsen, 1994). Such a design therefore makes the design more approachable for the administrator and hence enhance the application usability.

Comparing Figure 6 and Figure 7 you can see Figure 7 contains more information on each card. Such an approach follows the Progressive Disclosure design pattern which allows the user to focus on their task at hand by reducing cognitive load whilst still being able to retrieve more information about a given object if needed (UI Patterns, n.d.). Revealing information in such a manner also reduces what could have been a multi-step page loading process into one, following more of the SPA approach as discussed above, reducing the interaction time for the end user.

The information architecture orders the sets of cards into a hierarchy of importance (the studies having greater importance than card sets and user groups) with the most important at the top. The size of the card is also representative of it’s importance, giving a visual cue to the user that the card has high importance which is consistent with the ‘recognition rather than recall’ heuristic (Nielsen, 1994), making more important elements easier to find. Each group of cards is then ordered by the time they were added, giving them a familiar, logical ordering.

The mix of design patterns, “rules of thumb” heuristics and information architecture helps ensure that the administration home page is familiar, gives the user just enough information with the option to delve deeper and ensuring a logical information architecture so they can find what they are looking for to aid the usability of the application.

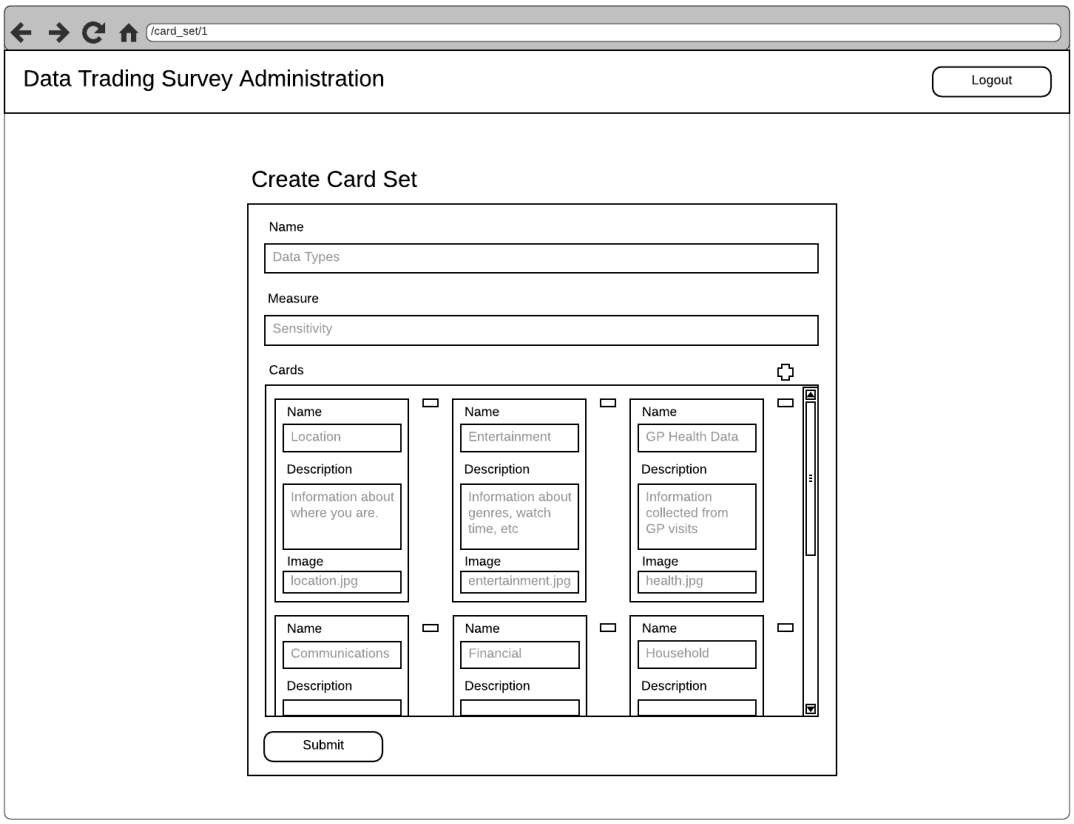


FIGURE 8: ADMIN CREATE CARD SET WIREFRAME

Figure 8 shows a wireframe for the form which allows an administrator to add cards and associated attributes to a card set.

The alignment of the user input's and their labels unifies the form as one set, leading to reduced cognitive load on the user as they don't have to link the label with its input, making the form easier to fill out.

The simplicity of the minus button next to each card makes it simple for the administrator to remove a card if it's added by accident without having to go through extra dialogue, in line with heuristic 3; user control and freedom (Nielsen, 1994).

The use of cards design pattern has its benefits as mentioned above, however there is also consistency between how they are created and how they are represented within the admin form and the actual survey as shown later. This parity gives greater understanding between the construction of a card in the administration interface and its actual representation to the participant in the study.

The aim of this UI was to make it as easy as possible for an admin to fill out the form and to give a link between what they are defining and the actual representation in the study, enabling the admin to create a study that they wish to create easier.

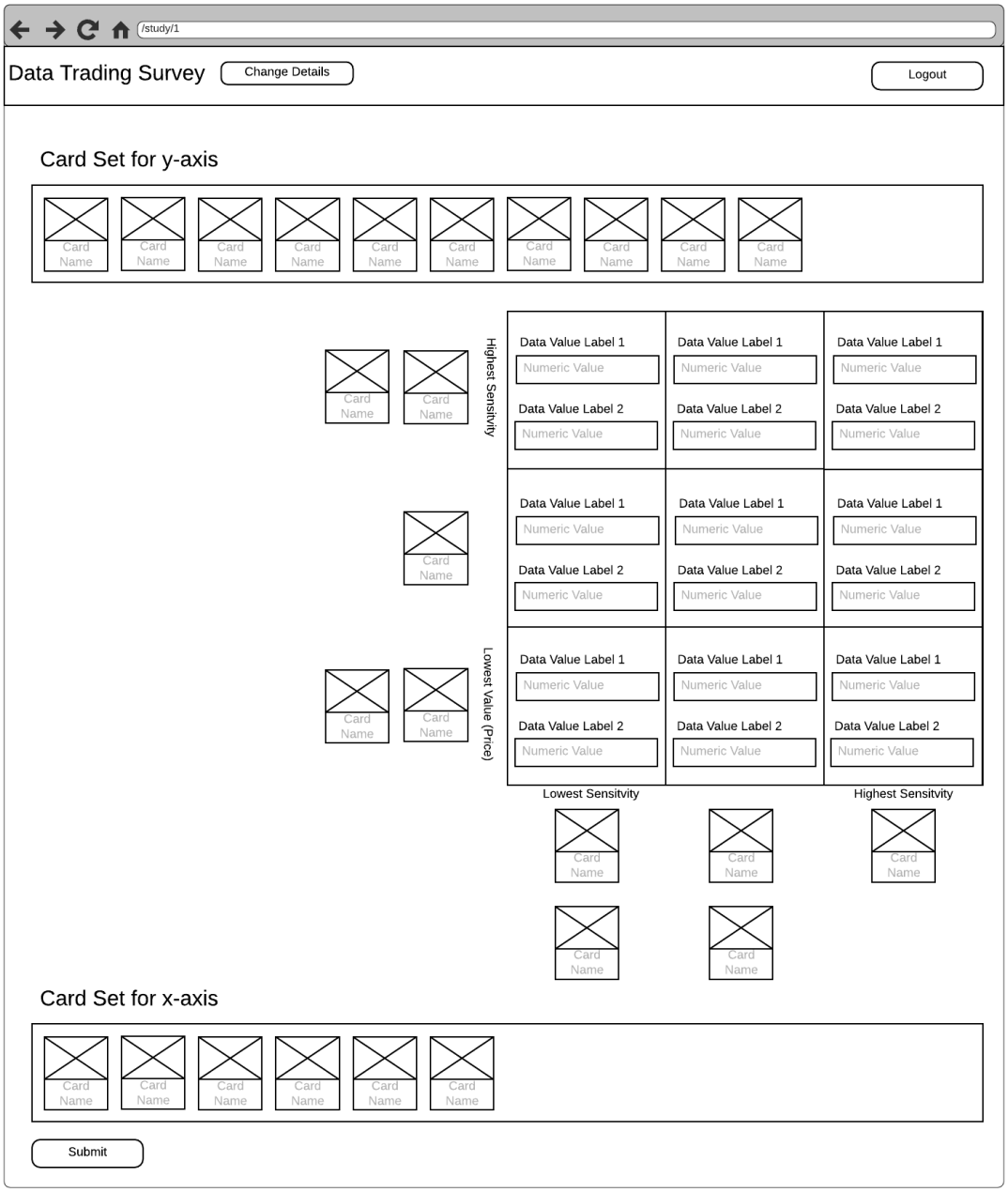


FIGURE 9: PARTICIPANT SURVEY WIREFRAME

The wireframe shown in Figure 9 is for the actual survey that the participant completes which is based upon functional Requirement 11 of the application. The wireframe has two axes where the participant can rank the cards in one card set on one axis against cards in another card set on the other axis. The ranking of a card in its card set is given context by a measure, ensuring the participant understands how to rank the cards. The input boxes also

provide a place for participants to give numerical values to their ranking of the cards they have placed on corresponding positions of both axes.

Each card set in Figure 9 is shown to be close to its respective axes. The proximity of these two elements is used to show these two items are related, decreasing the cognitive load of associating one element with another.

This UI also uses the 'drag and drop' design pattern. This allows participants to drag and drop cards onto the grid. Such a pattern has a match between the system and real world as in Nielsen's heuristics as participants are familiar with organising items by manually dragging them around into their desired position (Nielsen, 1994). This pattern makes it simpler and faster for a participant to organise data rather than going through an extended, confusing dialogue.

5. Implementation

5.1. Back-end

To implement the backend business-logic of the web application which was run on the server and something which enabled the use of the model-view-controller pattern as discussed in the design section, I decided to use the lightweight Python web application microframework called Flask.

A microframework is a framework which keeps the core simple but extensible, it does not enforce any dependencies or project layout, it is up to you, the developer, what tools and libraries you use. Unlike other generic web application frameworks, Flask does not include a database abstraction layer, form validation or anything else which can be provided by a 3rd party package. Rather, Flask gives you the ability to use extensions as if it was part of Flask itself (Flask, n.d.). Giving the choice and the barebones of a web application enables the developer to be more informed about how the application works and keeps the project more lightweight as there aren't any unused dependencies which is great for portability as discussed in the DevOps section below.

Flask does, however, provide you with a template engine which allows you create dynamic HTML. It also provides you with a WSGI web application library for creating request, response objects, a routing system for matching URLs and endpoints and testing features such as a development WSGI server, test client and debugger. Such features allow for a greater focus on requirements of the project and rapid development rather than on the lower level operation of a web application. Furthermore, these features also handle some of the common security flaws in web applications taking the burden of this knowledge and implementation away whilst still meeting security requirements.

The following details the implementation and its associated justification for each module as defined in the design section above. Only code that is critical to the application or is particularly interesting is has been described.

5.1.1. Database

When developing the application, I decided to use separate databases for each environment the application would be used in; development, testing and production. Using separate

databases meant that any modifications to the schema made in the development environment did not affect the production application and hence it could operate without disruption as a result of mismatch between the database and application code or error in the schema definition. Using a 3rd database for testing meant that the testing environment could set up and tear down a database needed to test the isolated functionality of the application without removing or altering vital data from the production or development database.

For development and testing I used 2 instances of a SQLite database which were hosted locally on the development machine. 'Lite' does not refer to its capability, rather, it being lightweight in respect to the setup time, administration and resource usage (Kreibich, 2010). SQLite is a server-less architecture unlike many database products it packages the database into a file and the database engine is implemented into the application which requires access to the database (Kreibich, 2010). SQLite was perfect for use in both development and test environments as the file can easily be deleted and reconstructed when needed.

My choice of production database to implement the structure shown Figure 5 was constrained by non-functional security requirements of the project stating that the database had to be hosted from within the university. Due to this the relational database, MySQL, which the school hosts was chosen. The school also hosts a database (MongoDB) which does not follow the relational model- often known as NoSQL. I chose to follow the relational model as in my opinion relational databases are better for use in an application which has a well-defined structure, such as the one being developed for this project, while NoSQL databases are better for those applications with less of a defined model structure. Each environment and their associated database are shown in Figure 10.

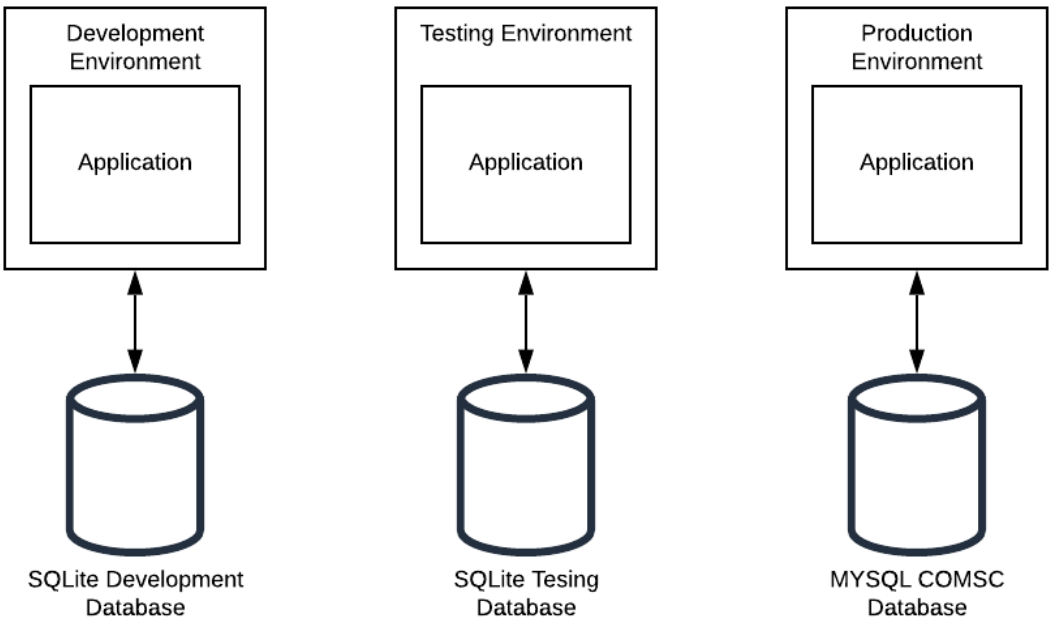


FIGURE 10: SEPARATE ENVIRONMENT DATABASES

To make working with the databases simpler an object-relational mapper (ORM) called SQLAlchemy was brought into the development dependencies. An ORM implements the

object-relational mapping technique which involves the conversion from complex SQL queries to an object-oriented paradigm, bridging the impedance mismatch between tabular data and appropriate objects (O'Neil, 2008). Writing updates, deletions, creations in an object-oriented manner in Python is much simpler than writing arduous SQL code that can take up many lines (see Figure 11). SQLAlchemy went further by taking away the burden of writing application code to interface with the database you are using making switching between different databases simple (SQLAlchemy, n.d.), so the application code could remain the same across all environments even if they used different databases. Queries are also optimised with an ORM so you can focus on feature implementation as queries are often more performant than someone who is not a SQL expert (SQLAlchemy, n.d.).

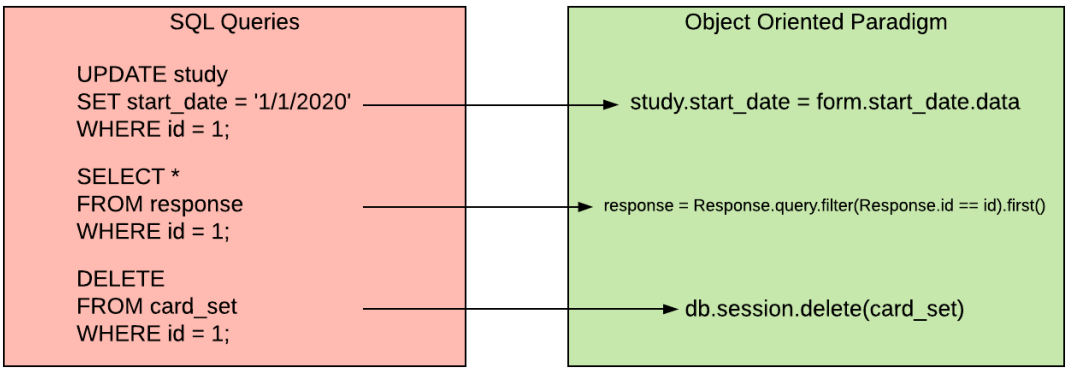


FIGURE 11: ASSIGNMENT OF VALUE IN A TUPLE IN THE OBJECT-ORIENTED PARADIGM

5.1.2. Authentication

For authentication the Flask-Login package was used. This provides; user session management, login and logout methods. Similar to the microframework approach, Flask-Login does not impose any restrictions on the method of authentication, user permissions or the user model. The package benefits are similar to that of Flask as focus could be put on requirements rather than user session management.

To handle the two different users of the application; administrators and participants I inherited from a base user class as discussed in the design section. Although relational databases don't support inheritance, the ORM provided me with the ability to map class inheritance to relational schema. The ORM only creates a single table from the base user class and using a discriminator attribute in the base class indicates the type of object represented within that tuple. This means that both administrators and participants can be queried as though they are separate tables although their data is actually contained within the same table. Using inheritance for defining relational schema has benefits similar to that of any class inheritance, such as code reuse, maintainability, however, it allowed me to differentiate between the two actors for authentication purposes.

In order to prevent a participant from accessing administrator endpoints or visa-versa, I used decorators, which wrap an endpoint function to modify its behaviour. The decorators for the authentication module check the discriminator attribute of the currently logged in user to see if they had permission to access a particular endpoint and redirect them if they did not. The wrapper function for checking for admin permissions is shown in Figure 12 and how an endpoint is decorated with @admin_required is illustrated in Figure 13. The use of

decorators in this scenario enabled me to reuse code rather than checking for correct permissions manually at each endpoint which decreases the size of the application, making it more readable and in turn it increases the application's maintainability as only one method has to be updated or modified if there was any change required in the process of checking for permissions.

```
def admin_required(function):
    @wraps(function)
    def wrapper(*args, **kwargs):
        if current_user.type == 'admin':
            return function(*args, **kwargs)
        else:
            return redirect(url_for('auth.login'))
    return wrapper
```

FIGURE 12: ADMIN PROTECTED ENDPOINT WRAPPER FUNCTION

```
@bp.route("/new_study")
@login_required
@admin_required
def new_study():
    try:
        study = Study()
        study.creator = current_user
        db.session.add(study)
        db.session.commit()
        return redirect(url_for("admin.study", id=study.id))
    except:
        flash("There was a problem creating a new Study.")
        return redirect(url_for("admin.index"))
```

FIGURE 13: EXAMPLE OF DECORATOR USAGE ON ENDPOINT FUNCTION

5.1.3. Participation

The main sections of the participation module consisted of forms both for the user information entry and the actual study itself.

For the user information entry, the Flask-WTF package was used which is an integration of Flask and WTForms. Flask-WTF was chosen for its simplicity in the rendering of generic HTML5 forms and their validation which are prevalent throughout the application. The package also came with security features to prevent against Cross-Site Request Forgery (CSRF) which is an attack where an "adversary causes a victims browser to perform an unwanted action on a trusted website via a malicious link or other content" (Lin, et al., 2009) further supporting the security requirements of the application. Flask-WTF was also able to handle file uploads which was helpful where uploading images was required in functional Requirement 1.

An example form definition for user information entry using Flask-WTF is shown in Figure 14. The UserInfoForm inherits FlaskForm which itself inherits the WTForms Form class, the fields are then specified declaratively as class attributes. Here, each field is a SelectField with choices that are a list of tuples generated from JSON files, making the updating of the potential choices simple. WTForms also ships with built-in validators for common use cases such as DataRequired validator shown Figure 14 which, along with the FlaskForm method

validate_on_submit (see Figure 15) allows the automatic validation of forms that have been submitted as and a corresponding error message to be displayed alongside the invalid field.

```python
class UserInfoForm(FlaskForm):
    gender = SelectField(
        "Gender", choices=data_genders, validators=[DataRequired()]
    )
    age_group = SelectField(
        "Age Group", choices=data_age_groups, validators=[DataRequired()]
    )
    nationality = SelectField(
        "Country of Birth", choices=data_countries, validators=[DataRequired()]
    )
    latest_country = SelectField(
        "Latest Country", choices=data_countries, validators=[DataRequired()]
    )
    education_level = SelectField(
        "Education Level", choices=data_educations, validators=[DataRequired()]
    )
    occupation = SelectField(
        "Occupation", choices=data_occupations, validators=[DataRequired()]
    )
    income = SelectField(
        "Income", choices=data_incomes, validators=[DataRequired()]
    )
    submit = SubmitField("Submit")
```

FIGURE 14: EXAMPLE FORM DEFINITION IN FLASK-WTF

```python
@bp.route("/user_info", methods=["POST", "GET"])
@login_required
@participant_required
@check_complete_study
def user_info():
    form = UserInfoForm()
    if form.validate_on_submit():
        current_user.gender = form.gender.data
        current_user.age_group = form.age_group.data
        current_user.country_of_birth = form.nationality.data
        current_user.education_level = form.education_level.data
        current_user.occupation = form.occupation.data
        current_user.latest_country = form.latest_country.data
        current_user.income = form.income.data
        current_user.completed_form = True
        try:
            db.session.commit()
            return redirect(url_for("study.index"))
        except:
            flash(
                "There was a problem submitting your information. Please try again."
            )
            db.session.rollback()
```

FIGURE 15: EXAMPLE USAGE OF FORM VALIDATION METHOD

For the actual participation in the study, generic HTML5 form elements provided by WTForms would not suffice for the UI that was designed, therefore a different method was required to capture the participant response to the study which will be discussed later in the front-end section of the implementation.

Once a response has been submitted, it is turned into an instance of the Response class defined in the database section. The data that is used to analyse the responses themselves also need to be updated as a new response would alter any existing heat map or average response.

5.1.4. Administration

The administration implementation was one of the more difficult sections to complete for the backend of the application. There was a lot of business logic required for this module

which limited the ways in which objects could be created and modified by the administrator which impacted the functionality of the system.

Some of the difficulties stemmed from the requirements of the application, such as functional Requirement 10 which created the problem that a participant's email had to be deleted after an invite was sent to ensure that sensitive information collected from a participant was not stored alongside personally identifiable data. This meant that modifying a user group after it had been assigned to a study was impossible as no user could be identified. The same user group could not be applied to multiple studies, because they would not be able to receive an invite to a later study due to their email having been removed.

Other difficulties were concerned with the modification of card sets which had or were being used in existing studies if a study's card sets were to be altered, existing response data for the old card sets would be invalidated, making analysis of the data very difficult. Alongside this problem, when choosing to edit a card set from the interface, pre-populating the value of the HTML file input value is prevented due to security reasons. This made it difficult to understand if an image was desired to be removed from a card or not.

There was significant time dedicated to attempting to resolve these difficulties, however due to time constraints of the project these difficulties eventually materialised into limitations of the project. Although these are limitations of the project, they do not fail to meet the requirements of the project, rather, they slightly limit the flexibility of the application.

In order to address these limitations on modification and deletion of studies, card sets and user groups in the implementation of the applications, decorators were used to ensure that the object does not have any restrictions on modification, or deletion. These decorators operate and have benefits similar to how they were described in the Authentication implementation section.

For the administration controller, there are 3 sets of 3 URL endpoints, each endpoint managing the creation, deletion and modification of each of the study, card set and user group objects. Although object instantiation and deletion are simple through code, separate endpoints were needed to handle the functionality which enabled the deletion and creation of objects through the administration page. To be clear, when talking about modification I mean both the assigning attributes from null and reassignment of attributes or, in simpler terms filling out a form from blank or changing the values on a form you have previously submitted.

The majority of the implementation for the modification of each of the objects constituted handling the submission of forms or pre-populating forms with existing data.

To enforce the business logic, custom queries for forms had to be created to ensure only user groups not assigned to a study could be chosen. Custom validators ensured that cases such as the same card set for both axes being chosen, or incorrect date combinations being chosen were not possible and gave the user satisfactory feedback when such criteria were not met.

To allow both card sets and user groups to contain any number of cards or users respectively, FieldLists from Flask-Forms were used. A FieldList is a wrapper for a group of fields, this allows us to embed any number of sub-forms into a form, warranting the ability to create multiple users or cards within one form rather than submitting multiple forms for each user or card. How such forms were defined are shown in Figure 16 and Figure 17.

```
class UserGroupForm(FlaskForm):
    name = StringField("User Group Name", validators=[DataRequired()])
    users = FieldList(
        label="Users", unbound_field=FormField(UserForm), min_entries=1
    )
    submit = SubmitField("Create")
```

FIGURE 16: USER GROUP FORM DEFINITION CONTAINING A FIELDLIST

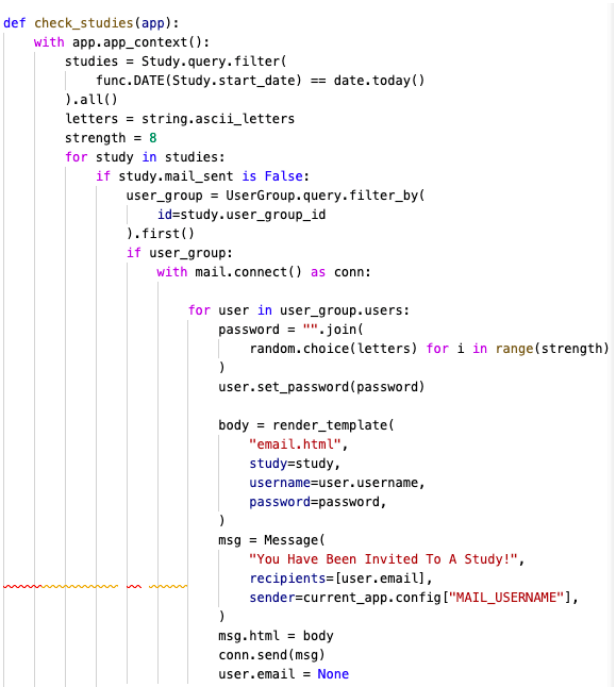
```
class UserForm(Form):
    email = StringField("Email", validators=[DataRequired(), Email()])
```

FIGURE 17: USER SUB FORM DEFINITION

To invite participants to participate in a study at the correct time, I used both Flask-APScheduler and Flask-Mail packages. Flask-APScheduler adds support for Flask from the regular APScheduler library. This library allowed me to schedule a Python function to be executed periodically, every hour to check if there were any current studies which needed an invite to be sent. Flask-Mail was used inside the function to send emails via the SMTP protocol to invite participant to the studies, providing them with a username, password and instructions for accessing the application from their machine. The combination of the two packages and functionality is shown in

```
scheduler.api_enabled = True
scheduler.init_app(app)
scheduler.add_job(
    id="check_studies_job",
    trigger="cron",
    func=check_studies,
    hour="*",
    minute=5,
    args=[app],
)
scheduler.start()
atexit.register(lambda: scheduler.shutdown(wait=False))
```

FIGURE 18: SCHEDULE JOB CODE



```
def check_studies(app):
    with app.app_context():
        studies = Study.query.filter(
            func.DATE(Study.start_date) == date.today()
        ).all()
        letters = string.ascii_letters
        strength = 8
        for study in studies:
            if study.mail_sent is False:
                user_group = UserGroup.query.filter_by(
                    id=study.user_group_id
                ).first()
                if user_group:
                    with mail.connect() as conn:

                        for user in user_group.users:
                            password = "".join(
                                random.choice(letters) for i in range(strength)
                            )
                            user.set_password(password)

                            body = render_template(
                                "email.html",
                                study=study,
                                username=user.username,
                                password=password,
                            )
                            msg = Message(
                                "You Have Been Invited To A Study!",
                                recipients=[user.email],
                                sender=current_app.config["MAIL_USERNAME"],
                            )
                            msg.html = body
                            conn.send(msg)
                            user.email = None
```

FIGURE 19: SCHEDULER FUNCTION TO SEND INVITES

5.1.5. Responses

Along with the Administration module implementation, the Responses module also took up a large amount of time in the development stage. Unfamiliarity around working with data analytics and how to transform data into something meaningful proved difficult. A lot of data was being collected by the application, demographic data paired with response data leads to potentially thousands of ways of interpreting the data. Time limitations and challenges of implementing the project meant I focused on a few key indicators of how a particular user group was responding to a study so some conclusions could be drawn. In the end, no analytics based upon responses over different demographics within a study was developed.

The main crux of the Responses module back-end revolved around functions which parsed data from the ‘response’, ‘card_position’ and ‘data_value’ tables. The functions transformed data into different forms so each view in the module could show a different representation of the data that the other views may miss.

The parsing function which created an average position of the cards for a study involved going through all responses for a study, totalling the position for each card. Each total was then divided by the number of participants who have responded to get an average position for each card. A similar process was also involved in the function that calculated the average numeric values for the study, by totalling the value entered for each data value position in the grid over all responses. Each total was then divided by the number of participants who entered a value in that position. Results from both functions were then combined to produce the overall average response.

Once a participant had submitted their response, the position of each card, as placed by the participant is stored in the 'card_position' table. Each value along with its position and the data value label which gives the description of the value is stored in the 'data_value' table.

To create a heat map showing the frequency of placement of a pair of cards, the position of each card is queried for each response and the counter is incremented at the intersection of the positions of each card (see Figure 20 for a diagram of the conversion).

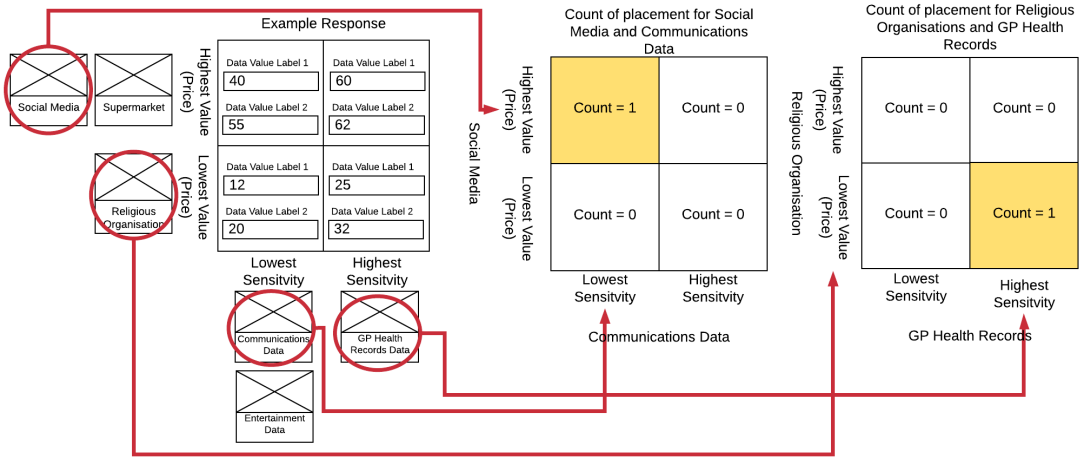


FIGURE 20: CONVERSION FOR THE HEAT MAP SHOWING THE FREQUENCY OF PLACEMENT OF A PAIR OF CARDS.

To create a heat map showing the average normalised value given to a pair of cards and a data value label, first, for each response the position of each card is queried. The value for the data value label that intersects the positions of the cards is then queried and normalised based upon the normalisation across all data values for that participant's response and specific data value label. The normalised value is then added to a running total for the position of the pair of cards. After going through all responses in this process, totals for the pair of cards in each position on the grid are then divided by the number of participants who have placed the pair of cards which intersect with the position on the grid (see Figure **21** for a diagram of the conversion)

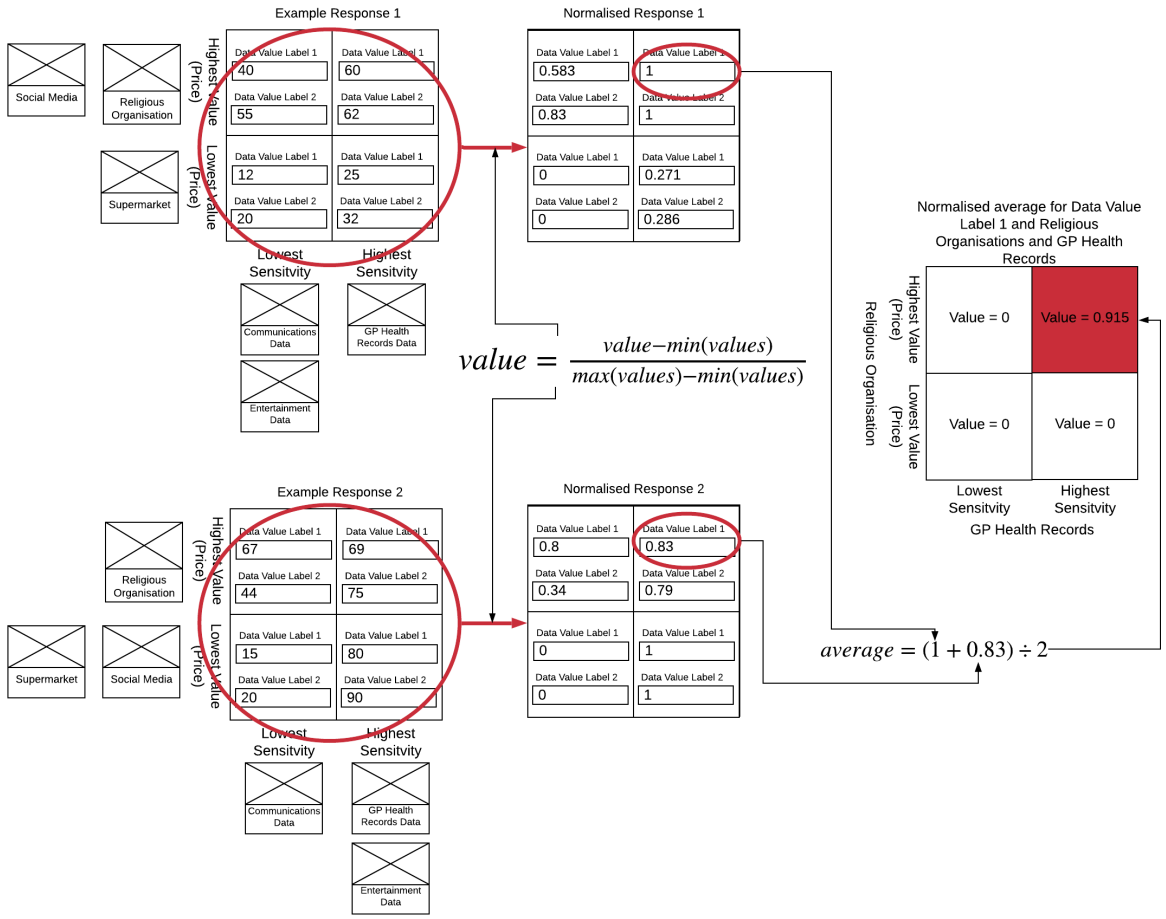


FIGURE 21: CONVERSION FOR THE HEAT MAP SHOWING THE AVERAGE NORMALISED VALUE GIVEN TO A PAIR OF CARDS.

Comparisons of responses for a study was also enabled in this module by reproducing individual responses from the data stored in the Responses table. This functionality was made simple by using a slightly modified version of the HTML structure coded for the actual survey. The HTML structure was also reused further by giving the administrator the ability to reproduce a participant's response in a PDF form, rather than a webpage. To create this feature I used a package called pdfkit which was a python wrapper for wkhtmltopdf that handled the conversion of the responses represented in HTML to a PDF, however this feature is only partially implemented due to the difficulty in incorporating images into the pdf and some CSS features present not being supported in the package.

5.1.6. API

The API was one of the easiest modules to implement in the whole system as it largely involved building endpoints for each of the tables shown in the entity relationship diagram then using the URL arguments for that endpoint to query the table and return the data in a JSON format.

The main challenge came with how to authenticate those who were requesting data through the API, for this, I used the package called Flask-JWT-Extended. This package gave me the ability to create access tokens upon validation of an existing administrator's

username and password. The access tokens returned are JSON Web Tokens (JWT) and importantly, they are signed using a private secret so they cannot be manipulated to retrieve another user's data.

When requesting data from an endpoint in the API module, the administrator would send this access token in the authorisation header of their HTTP request. A decorator provided by the package would then validate the access token before allowing the endpoint to be called. The package also enabled the storage of an identity within the token, this is used to identify the specific user requesting data and ensured only the data created would be returned. Figure 22 shows a section of the authentication of users for an access token and Figure 23 shows an API endpoint which provides data about a participant in JSON format and is protected by a decorator which checks for the validity of the access token.

```
user = User.query.filter_by(username=username).first()
if user is None or not user.check_password(password):
    return jsonify({"msg": "Incorrect username or password"}), 401
if not user.type == "admin":
    return jsonify({"error": "Unauthorised"}), 401

access_token = create_access_token(identity=user.id)
return jsonify(access_token=access_token), 200
```

FIGURE 22: API AUTHENTICATION AND JWT ACCESS TOKEN CREATION

```
@bp.route("/get/participant/<int:id>")
@jwt_required
def get_participant(id):
    participant = Participant.query.filter_by(id=id).first()
    if participant is None:
        return jsonify(error_not_found)
    else:
        if participant.user_group.creator_id == get_jwt_identity():
            return jsonify(create_participant_json(participant))
        else:
            return jsonify(error_not_found)
```

FIGURE 23: API GET PARTICIPANT ENDPOINT

5.2.Front-end

For the front-end implementation I used a range of software to help me create an interface to meet the design requirements for the user interface, which was responsive to different types of browsers and viewport sizes. To meet some of the more rich, interactive interface designs I needed scripting software which allowed the manipulation of the HTML DOM tree after the page has rendered.

For the structure of the content on the webpages returned from the webserver I used Hyper Text Mark-up Language (HTML) this gave me the means to represent the forms defined in the backend, denote structural semantics for headings, lists, etc (W3C, [no date]). As mentioned in the back-end section Flask wraps a templating engine called Jinja. This templating language allows you to create dynamic HTML using loops, template inheritance, etc and has added security features such as "automatic HTML escaping system for XSS prevention" to supplement the security requirements of the application. This engine supports the requirements of creating user interfaces that are derivatives of user input whilst having the added benefits of template inheritance to create consistency needed across user interfaces for the application as shown in the UI design.

I assisted the HTML structure with Cascading Style Sheets (CSS) for the presentation of the HTML elements on screen which include colours, layouts and fonts (W3C, [no date]). Alongside the use of pure CSS, I also used the CSS framework called Bootstrap. This framework shipped with predefined style sheets to create visually appealing styles across all of the interfaces. Bootstrap comes with rich abilities with its content and components features. These features enabled the application to easily apply the heuristics, principles and patterns that were discussed in the UI design section. The layout feature of Bootstrap this allowed the application's interfaces to be responsive and adapt to the size of the browser viewport so a greater amount of hardware could also be supported.

Interactive capabilities of the interface were enabled through the use of JavaScript a client-side scripting language. I decided to use a JavaScript library called jQuery as it took common tasks that require many lines of pure JavaScript and condenses them into a single method. This overall made the codebase smaller for the front-end and made manipulating the DOM and CSS simpler. Alongside this functionality, jQuery also made it easier to send AJAX requests to a web server. AJAX allows data to be exchanged with a web server and update parts of the webpage without it being reloaded, similar to how a SPA works, which was desired for the more interactive elements of the application.

The templating engine, Jinja, provides the ability to template inheritance as mentioned. This was used widely throughout the HTML files that constituted the structure for each of the user interfaces implemented in the application. A large proportion of wireframes include a navigation bar at the top of the interface to allow the user to easily access different views of the application from any other view. By using inheritance where there was commonality between different interfaces it means that HTML would not have to be repeated and resulted in only one file needing to be updated to alter the common elements among many views, making the code more maintainable, saving time and reducing the codebase.

Similarly, to the backend, the following sections detail the implementation and its associated justification for each module. Only code that is critical to the application or is particularly interesting is has been described. Note that the API module has been omitted as it does not require any front-end implementation and Authentication has been omitted as its front-end implementation is relatively standard and uninteresting.

5.2.1. Participation

This module proved to be the trickiest for implementing the interface that those participating in the study would use to rank different elements of different categories against each other and give numerical reasoning behind those rankings. The drag and drop design pattern mentioned tied in with the tabular representation of the study created difficulties in effectively styling the interface so it was appropriate for a range of viewports and different studies.

In order to represent the cards shown on the wireframe in Figure 9: Participant Survey Wireframe the card component was used from Bootstrap. It offered a vast amount of flexibility in terms of their content and sizing and met the design that was desired for the interface. In order to rank these cards on the axes as shown, jQuery's sortable interaction was used to enable the card element to be clicked and dragged to a desired position on the axes and have all other cards adjust to fit.

To represent the area where the participant ranks the cards and fills in the numerical values, a HTML table was used as it matched the design of the interface in Figure 9: Participant Survey Wireframe. The actual developed interface for participation in the survey is shown in Figure 24.

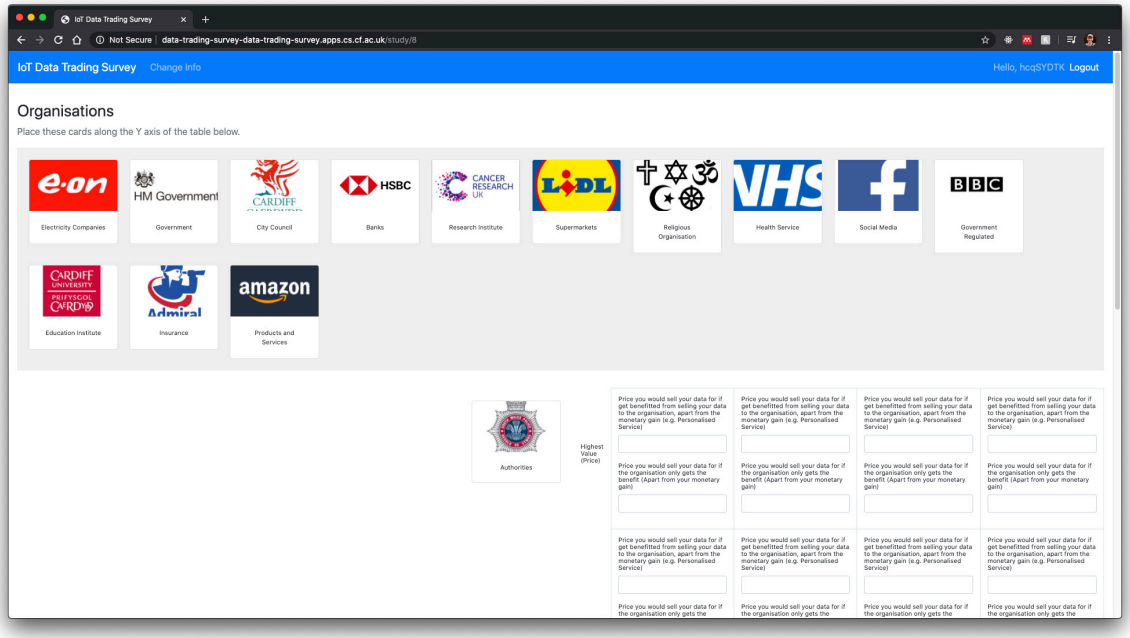


FIGURE 24: PARTICIPANT SURVEY INTERFACE

Difficulty was also encountered regarding how to send the response data back to the server to be processed and stored. A standard HTML form could not be utilised for this interface as the positions of the cards on the axes cannot be obtained using generic HTML5 form elements.

In order to overcome this difficulty, jQuery was used to scrape and transform the relevant sections of the DOM and AJAX was used to post this data back to the server to be processed and stored. Once the study had been submitted, the jQuery code traverses the table and adding each card to its correct position in a JavaScript object, according to where the participant has placed them. Similarly, during the traversal, the numerical values entered by the user are added to an object depending on their position in the table. The jQuery code also validates the user input to check they have added all cards to the table and the relevant user input values are also filled out. Validation on the client-side means that the page is not reloaded and hence any progress is not lost as a result of an incomplete study. Once the form has been validated, the processed data is sent back to the web server using AJAX where it can be processed for the analytics of the responses.

A section of code which processes the cards placed on the y-axis so they can be sent back to the server is shown in Figure 25 and the method used to send the response back to the server asynchronously using AJAX can be seen in Figure 26.

```
$(".cards-row").each(function(){
    var row = $(this).attr("row")
    cards_y[row] = []
    $(".card", $(this)).each(function(){
        var card_id = parseInt($(this).attr("card_id"));
        var card_name = $(this).find(".title").text();
        var card_img = ""
        var card_desc = ""
        if($(this).find(".img") != ""){
            card_path = $(this).find(".img").attr("src")
            if(card_path !== "undefined" ){
                card_path_split = card_path.split('/')
                card_img = card_path_split[card_path_split.length-1]
            }
        }
        if ($(this).find(".description").attr("title")){
            card_desc = $(this).find(".title").attr("title");
        }
        card = {'id':card_id, 'name':card_name, 'image':card_img, 'description':card_desc}
        cards_y[row].push(card)
    })
});
```

FIGURE 25: JQUERY TO PROCESS PARTICIPANT RESPONSE

```
$.ajax({
    type : 'POST',
    url : url,
    contentType: 'application/json;charset=UTF-8',
    data: JSON.stringify(data),
    dataType: "json",
    success: function(response){
        window.location.href = response['url'];
    },
    error : function (response) {
        console.log(response);
    }

});
```

FIGURE 26: AJAX CODE TO SEND DATA BACK TO SERVER

5.2.2. Administration

The card component from Bootstrap was also used in the administration home page to represent the user groups, card sets and studies that the administrator had created. They were structured in the desired format as shown in Figure 6 by using the grid system built into Bootstrap's layout feature. It uses a series of containers, rows and columns to align and organise content. The feature is built using flexbox – the CSS layout module so is fully responsive different to viewport sizes ensuring it can be used across different hardware which was specified as a requirement.

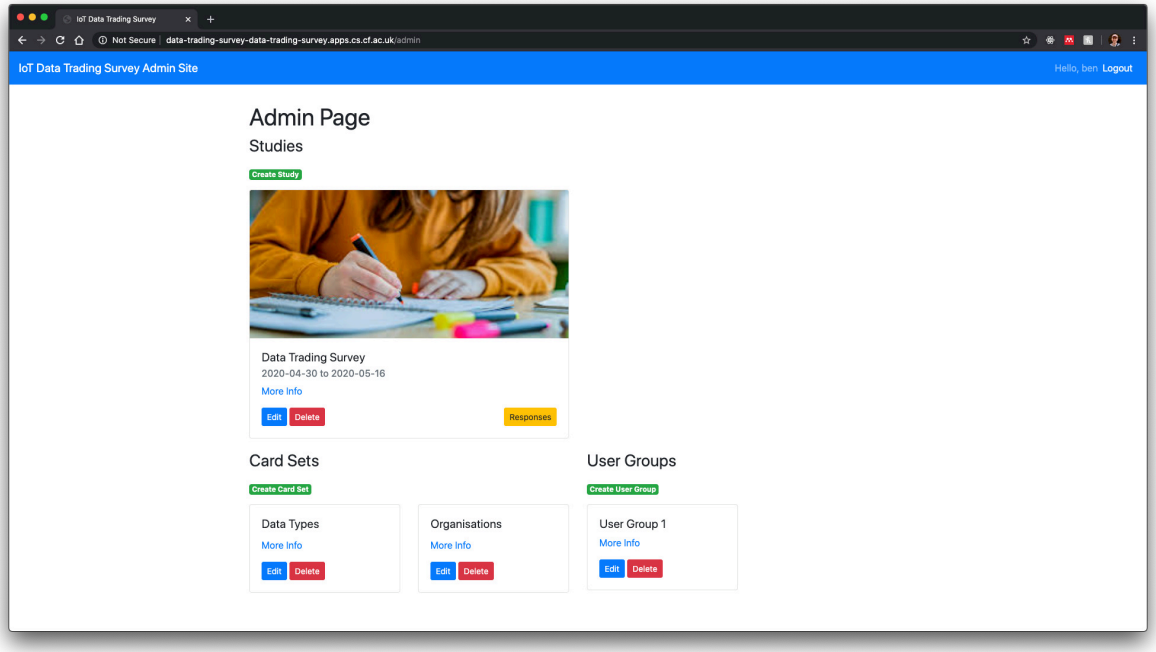


FIGURE 27: ADMINISTRATION HOMEPAGE INTERFACE

The interfaces for creating user groups, card sets and studies were largely structured using standard HTML form elements and styled using Bootstraps form component which simplified alignment of the form elements and their tags. The component also enhanced the visual appeal of the form elements more than their standard HTML counterparts.

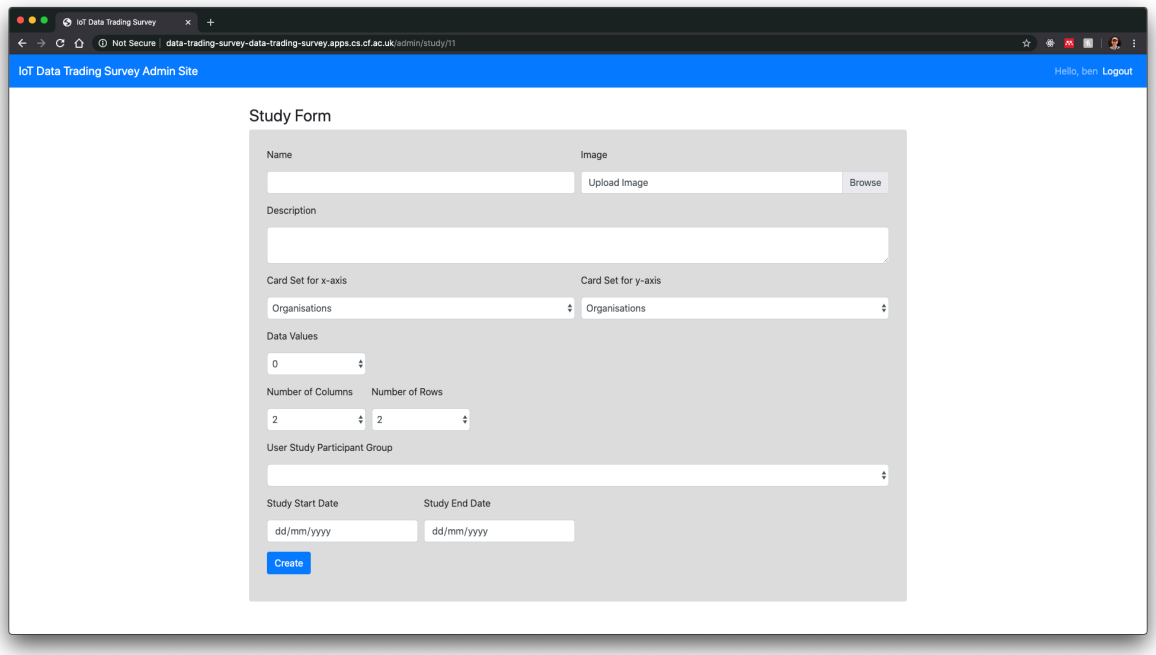


FIGURE 28: CREATE STUDY INTERFACE

To allow the administrator to define any number of participants or cards from one single view the application was required to have the functionality on the client side to duplicate a group of form elements or remove a specific group of form elements. As mentioned

previously, the FieldList form field from the Flask-Forms package was used to provide this extensibility in the backend, however there was a challenge at the front-end to implement this flexibility. The problem was Flask-Forms renders each group of fields in a Field List using a unique identifier- this identifies the position of the fields within the group, therefore when duplicating the fields, you need to modify the identifier and when removing a specific field the trailing identifiers in the group need to be modified too. In order to duplicate DOM and modify the identifier, I used a jQuery plugin called czMore.

Figure 29 shows the interface for grouping users together and utilises the czMore plugin to add or remove email fields. Further 'nice to have' features were implemented in this interface including uploading a csv file of emails to make it simpler to add multiple emails to the user group form. This functionality used a file reader and DOM manipulation to make this possible.

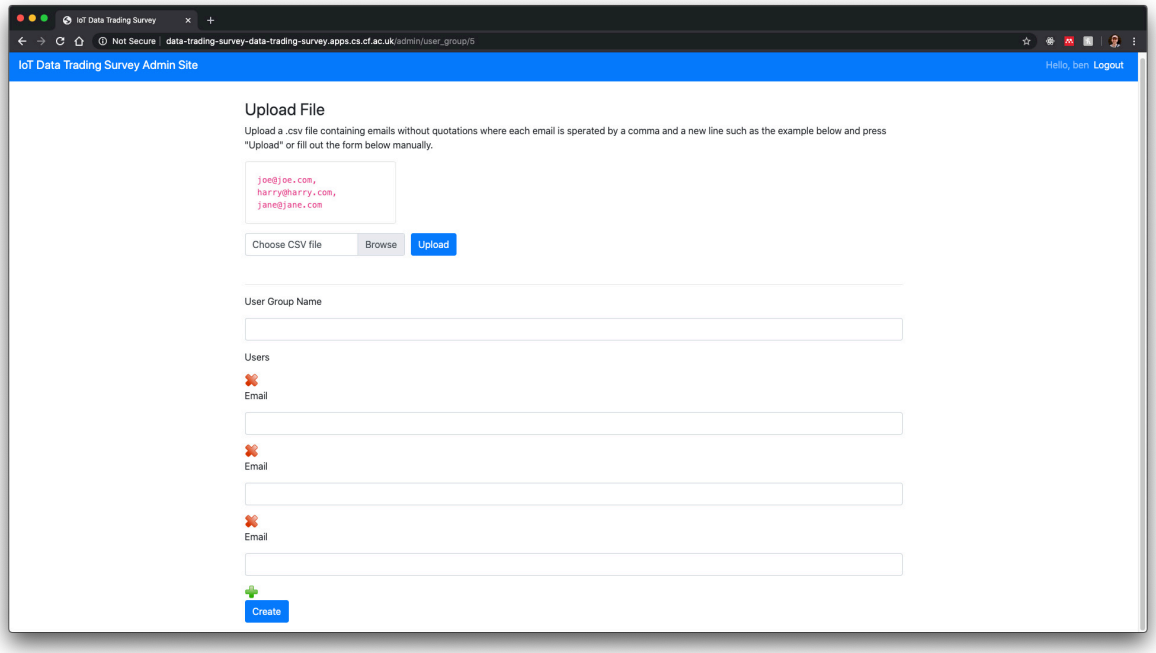


FIGURE 29: CREATE USER GROUP INTERFACE

5.2.3. Responses

The Responses module was split across many different interfaces, each to represent the response data in a different way that may be meaningful for the administrator.

Some of the interfaces in the response's module were simple reconstructions of the responses submitted by the participants or the average of all responses. These reconstructions used a HTML structure similar to that from the study view from the participation module discussed earlier.

Some other interfaces required for the application represented the data in such a way beyond the capabilities of HTML and CSS. For this, a visualisation library called Bokeh was used. Bokeh allowed me to create interactive plots using Python quickly and easily. This library would generate JavaScript representing plots derived from the response data. This

JavaScript would then be embedded into the HTML. Figure 30 shows the 'general' responses interface, using Bokeh to produce bar graphs to show frequency of the placement of cards at different positions on the grid for each card set in the study.

Figure 31 shows the options to add heat maps to the page. AJAX is used to send the form options back asynchronously to the server which then produces the JavaScript for the heat map based upon those options and is subsequently added to the page without reloading. I also used jQuery's draggable to allow the heat maps to be positioned to the users liking, making comparisons easier.

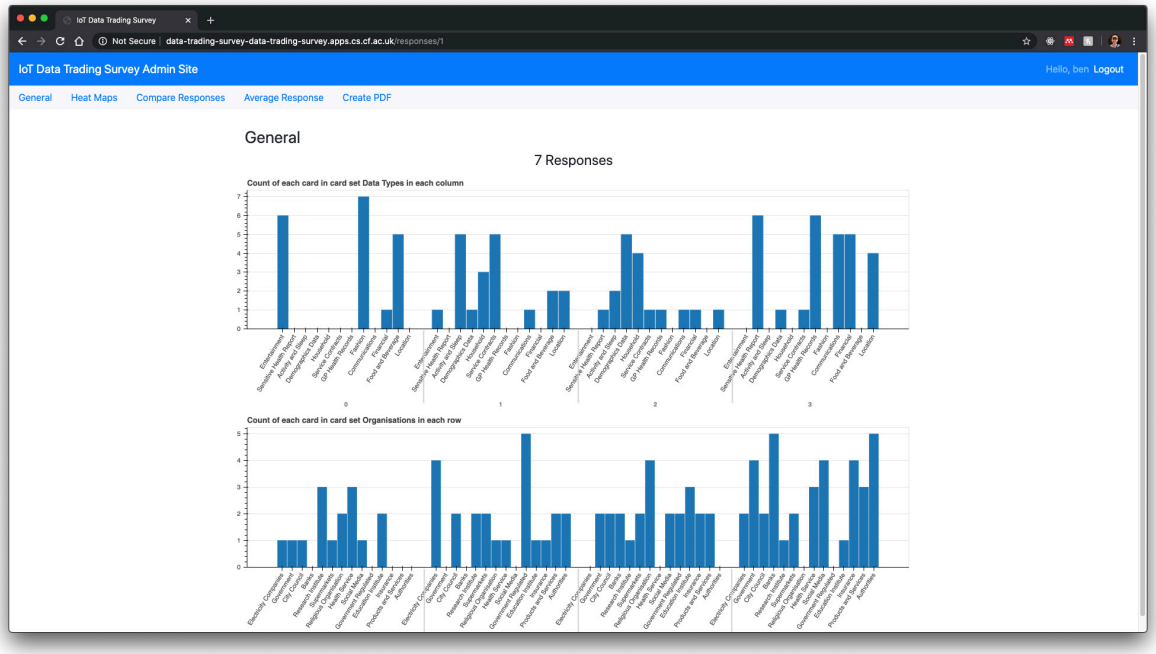


FIGURE 30: OVERVIEW OF RESPONSES INTERFACE

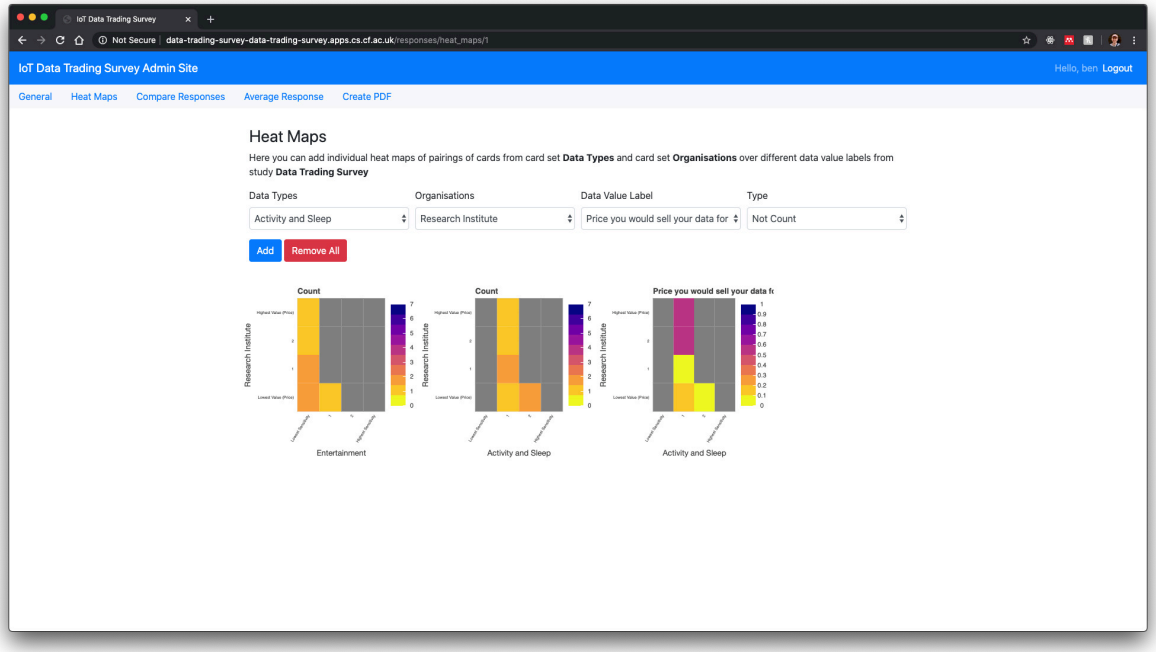


FIGURE 31: HEATMAPS OF RESPONSES INTERFACE

5.3. Dev Ops

DevOps is about integrating development and operations, it facilitates the connections between these traditionally separated workflows. Using automated tools across these workflows DevOps can help to deliver code to production faster and accelerate problem resolution (Ebert, et al., 2016).

5.3.1. Version Control

For the application a version control platform called GitHub was used. Using this software would help address risk concerns around data loss as mentioned in the Initial Plan as GitHub stores a remote repository of all code committed to it. Therefore, any code committed to GitHub would not be lost as a result of data loss on a local, development device. GitHub also gave me the freedom to revert changes where certain implementations or features did not work which came in useful when encountering some of the more difficult implementation features of the application.

GitHub also ships with the ability to connect other tools that were used in the DevOps pipeline for the project. This allowed for the automated processes that DevOps embodies to occur seamlessly between developer environment to production deployment of the application.

5.3.2. Deployment

As there was a requirement to host the application from within the university network for security reasons, there was limited options for hosting the site. There was a project web server which the school maintains which can only serve static files or PHP which was unsuitable for my applications ecosystem and there was also the option of using OpenShift which is a container-based deployment and management platform for applications which was much more suitable for the needs of the application.

In order to fully support the portability of the application and automation when it came to move the code from the development environment to OpenShift, the production environment I enlisted the OS-level virtualisation tool called Docker. This allows for the creation of an 'image' which represents a packaged application with all its dependencies in one file. A 'Dockerfile' included with the source code describes how the image should be created. Each command in a 'Dockerfile' creates a new layer in the image, where a layer is a change in an image or an intermediary image.

The 'Dockerfile' for creating the image for the application in Figure 32 displays the first line as using the Linux Alpine image hosted on Docker Hub as the base image. Alpine was used as it is a very resource efficient, by stripping out the unnecessary Linux features as the image is only 5mb in size (Alpine, n.d.) it also provided the apk package manager to easily install dependencies needed for the application (Alpine, n.d.) shown with the 'RUN apk' command in the Dockerfile below. The Dockerfile also shows the use of the COPY command which copies the required folders/ files for the application from the current directory on local machine to the image. The ENTRYPOINT command points to the boot.sh file (seen in Figure 33) which will be run once the container for this image is started.

```
FROM python:3.6-alpine

RUN adduser -D userstudy

WORKDIR /home/userstudy

COPY requirements.txt requirements.txt
RUN python -m venv venv
RUN apk update && apk add mariadb-connector-c-dev
RUN apk add jpeg-dev
RUN apk add wkhtmltopdf
RUN apk add make automake gcc g++ subversion python3-dev
RUN venv/bin/pip install -r requirements.txt
RUN venv/bin/pip install gunicorn pymysql

COPY app app
COPY migrations_prod migrations_prod
COPY tests tests
COPY userstudy.py config.py boot.sh ./
RUN chmod a+x boot.sh

ENV FLASK_APP userstudy.py

RUN chown -R userstudy:userstudy ./
USER userstudy

EXPOSE 5000
ENTRYPOINT ["./boot.sh"]
```

FIGURE 32: DOCKERFILE FOR CREATING DOCKER IMAGE

For the building and storage of images the Docker Hub registry was employed for its simplicity and integration with GitHub and OpenShift to link the two pieces of software making the automatic workflow desired possible. A webhook alerted the Docker Hub once new code had been pushed to the GitHub repository, Docker Hub would then begin building the image from the Dockerfile. Once a new build was finished, OpenShift would pull this new image and spawn a new container of this image, running the shell script presented in Figure 33 below. The script which would check for any new migrations which needed applying to the production database and start then web server. The workflow from the development machine to actual production deployment can be seen in Figure 34.

```
#!/bin/sh
# From The Flask Mega-Tutorial Part XIX: Deployment on Docker Containers
source venv/bin/activate
while true; do
    flask db migrate --directory=migrations_prod
    flask db upgrade --directory=migrations_prod
    if [[ "$?" == "0" ]]; then
        break
    fi
    echo Deploy command failed, retrying in 5 secs...
    sleep 5
done
exec gunicorn -b :5000 --access-logfile - --error-logfile - userstudy:app
```

FIGURE 33: SHELL SCRIPT RUN BY OPENSHIFT CONTAINER

Instead of running the development web-server gateway interface (WSGI) which ships with Flask and is poorly optimised towards a production environment where speed and robustness is desired, I used Gunicorn which is a python production WSGI. Gunicorn allows you to configure a number of workers which are each able to serve requests concurrently to handle a greater amount of users it is very simple to set up but is used across many production environments as it's light on server resources but fairly speedy (Gunicorn, n.d.).

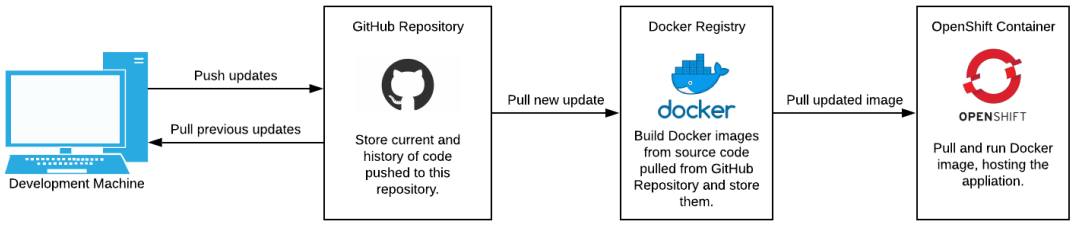


FIGURE 34: DEVELOPMENT WORKFLOW

6. User Study

6.1. Procedure

I originally planned and organised to carry out the user studies with a group of people in person in late March 2020 however, as a result of the COVID-19 pandemic and the subsequent UK Government lockdown meant that meeting in person was infeasible. This meant that some content and procedures that would have been carried out in the meeting would have to be fully remote. Extra time was allocated for this transition and as a result, an informational video on how to complete the survey was created and participants were asked to send a digital consent form via email.

6.1.1. Participants

As there was a requirement to host the application internally, only those people who had access to the university network could be invited to participate. The participants were a range of students from varying countries, ages, some had a job alongside studying whilst others were working on placement years. Table 1 shows the participants and their information as entered by them into the application. As the information was not entered under supervision as originally intended when meeting in person, the information is only as good as how the participant perceived the question. Although the questions are common throughout online forms, hastiness or reluctancy to ask questions over a remote medium could lead them to not wishing to ask if they had a simple query about what was required of them. Such a problem could have been reduced by conducting the study with participants in person.

ID	Age Group	Salary Range	Country of Birth	Current Country	Education Level
1	40-49	£10000-£20000	Sudan	United Kingdom	7
2	30-39	£20000-£30000	Saudi Arabia	Saudi Arabia	7
3	20-29	£0-£10000	Turkey	Turkey	8
4	20-29	£20000-£30000	United Kingdom	United Kingdom	6
5	20-29	£0	United Kingdom	United Kingdom	3
6	20-29	£10000-£20000	Switzerland	United Kingdom	5
7	20-29	£0	United Kingdom	United Kingdom	6

TABLE 1: USER STUDY PARTICIPANTS

The ‘current country’ column indicates the country that participant has lived In most for the past 10 years and the ‘education level’ indicates the highest level of qualification achieved by the participant as defined by the UK Government (UK Govenrment, n.d.).

6.1.2. Invitations

As mentioned in Section 5.1.4, the application sent out emails to prospective participants which was the reason behind the administrator to enter participant emails as a part of creating a user group. The email contained information on how to access the university network using a VPN as well as the credentials for the application.

6.1.3. How-To

A screen-capture video of how to fill out the survey was created in order to replace the in-person demonstration of how to respond to the survey. A link to the video can be found in Section 12.

6.1.4. Study Parameters

Two different types of card set were created; the first representing different types of data and the second representing different types of organisations. These separate card sets allow us to be able to rank both different types of data against different types of organisation, as required by the problem that is being investigated. The following comprises the cards used within the data types card set along with a description of the data type:

- Entertainment – Information about what TV programmes, films and genres you watch.
- Sensitive health report – sensitive health information including fertility tests, blood report and DNA test.
- Activity and sleep – Exercise and sleep data from exercise apps or smart watches.
- Demographics data - data about your Age, Gender, Address, Religion, Political opinions, height, weight etc.
- Household data - data about your home i.e. energy/ water consumption, fridge contents etc.
- Service contracts – information about recurring bills such as gym membership, broadband contract, phone contract, etc.
- GP Health Records – data collected from general practitioner (GP) visits.
- Fashion – Information about clothes stored in wardrobe.
- Communications – Data about specific messages, call history, etc.
- Financial - data about bank statements/ contracts/ insurance/ stocks.
- Food and Beverage - data about food and beverage consumption through smart ovens, coffee machines etc.
- Location - data about places you visit.

The following list comprises the cards used within the organisations card set and any associated examples or description:

- Electricity Companies – companies that supply electricity to domestic households.
- Government.
- City Council – i.e. Cardiff City Council who provide services such as waste collection, street cleaning and community centres.

- Banks.
- Research Institute – i.e. Medical research, Science and technology research, Natural world research.
- Supermarkets.
- Religious Groups.
- Health Service – National Health Service (NHS).
- Social Media – i.e. Facebook, Instagram, Snapchat, Twitter.
- Government Regulated – i.e. Network Rail, BBC.
- Education Institute – i.e. Universities, Colleges, Sixth Forms, Secondary Schools.
- Insurance – i.e. Car, Home, Life insurance.
- Products and Services – i.e. Amazon, BT, Google, Spotify.
- Authorities - i.e. Police and Courts.

These cards were chosen as they represented a wide variety of data collected by IoT devices. Similarly, these organisations were chosen as they represented the large sectors of industry both public and private. The cards within each card set was ranked according to a measure which ranged from low to high for that given measure. The measure for the data types card set was defined as 'Sensitivity', meaning how sensitive you are towards selling that type of data. The measure for the organisations card set is defined as 'Value (Price)', meaning how willing you are to sell any type of data to that organisation.

Two different numeric values were also associated with the study, each of which represented the consequence of selling the data to an organisation under different circumstances. The first numeric fields label described the price you would sell the corresponding data type(s) to the corresponding organisation(s) if you, the sensor owner, apart from the monetary gain, saw the benefit of the data being sold, such as a personalised service. The second numeric value label allowed the participant to define the price they would be willing to sell the corresponding data type(s) to the corresponding organisation(s) if, apart from the monetary gain, only the organisation saw the benefit, such as a better supply chain management. These two labels allow us to identify price preferences for a group under two common business cases for companies collecting such IoT data.

Each participant ranked the cards within each card set according to their respective measure and filled out the numeric value inputs in the grid according to the intersecting data type(s) and organisation(s). Finally, they submitted their final ordering and values to be analysed by the application.

7. Results and Evaluation

7.1. Functionality Results and Appraisal

In order to check the backend of the application, tests needed to be performed which addressed to what extent the requirements set out in section 3.1. had been achieved. Functional testing of the application was carried out to instil confidence in the code about meeting those requirements and the wider operation of the application under adverse user input. A Python testing framework called pytest was used to aid the testing of the backend code. Each functional test constituted of an instance of the application, and when needed, an instance of the testing database. To ensure the integrity of each test was not

undermined by previous tests, the testing database was torn down after each test and reconstructed for the next.

Each module was subject to testing on the correct rendering of its views based upon data its dependent on that was created within the test. Where a module contained forms, the submission of correct and invalid data from a test checked whether the data was processed correctly, or the correct error message was shown respectively. Decorators described in section 5.1 which provide access control to endpoints and enforce some of the limitations on the flexibility of the application were also tested across every decorated endpoint to prove those endpoints were properly protected. The API module was tested validating that each endpoint required a valid access token and the correct JSON data was returned.

Alongside functional testing, unit testing of the singular functions, outside of the endpoints was also employed. The unit tests covered testing the parsing functionality of the response's module covered in section 5.1.5..

Out of the 85 automated tests for the application, 82 passed and 3 failed, I expected a 100% pass rate as I was developing the application to pass the tests to ensure I was meeting the requirements of the project. The 3 failed tests were due difficulties encountered in developing tests for forms which were defined with the FieldList functionality of the WTForms package. Despite this failure using pytest, the functionality was shown to be working upon manual testing, by manually submitting valid and invalid forms and checking for a valid response. A summary of all the tests performed can be found in section 13.

Despite a good test coverage of the backend of the implementation, automated testing of the front-end implementation of the application was not completed. Despite this, Chrome Developer Tools feature came in handy when performing manual testing of the application, giving you the ability to manipulate viewport sizes to simulate the interface across different viewport sizes and to set breakpoints in JavaScript to validate the correctness of the code. Manual testing through such methods only gave a slight confidence about the robustness of the interface, for full confidence automated testing frameworks would need to be used to ensure a coverage of the front-end that is similar to that of the backend.

7.2. User Study Results

After the study responses had been submitted, the analysis tools that had been developed for the application were used to identify trends and anomalies in participant responses. These tools illustrated the willingness to trade different types of data to different organisations and the prices required for such a trade to take place. The following discusses some of the results identified from the study using these tools.

7.2.1. Not just special category data is highly sensitive

Under the GDPR, special category data is data which may create significant risks to an individual's fundamental rights and freedoms (ICO, n.d.). Figure 35 shows the special category data which includes sensitive health report data and GP health records being placed exclusively by participants in the highest two columns for sensitivity. This correlates with the ICO's view that this data is very sensitive to the data subject.

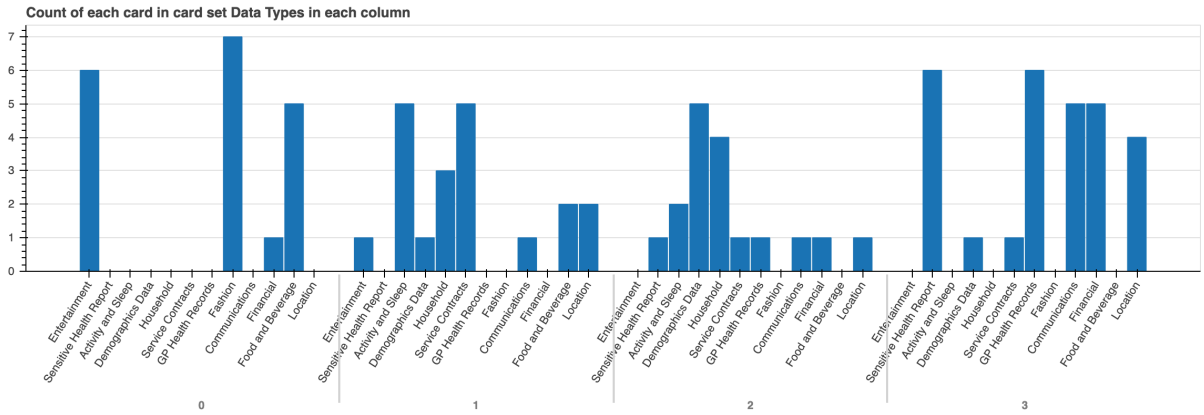


FIGURE 35: PARTICIPANT SENSITIVITY OF DIFFERENT DATA TYPES

Interestingly, non-special category data such as communication data also featured mostly in the highest column for sensitivity, as shown in Figure 35. This is interesting as nearly 2.5 billion people use messaging apps such as Facebook Messenger and WeChat (Clement, 2020) which do not enable end-to-end encryption by default or support it at all (Morse, 2019; Chiu, 2019). This essentially allows these companies to read all communications through these apps as if they had bought the communication data through the Sensing as a Service model itself. With Facebook and WeChat both being social media companies, it could be inferred that with such high market share within the communication market that their users trust their communications data with them. However, data collected from the study that was conducted shows that social media companies would have some of the highest costs for purchasing communication data relative to all types of data (see Figure 36), suggesting that participants have mistrust in social media handling their communication data, a direct contradiction with the previous remark.

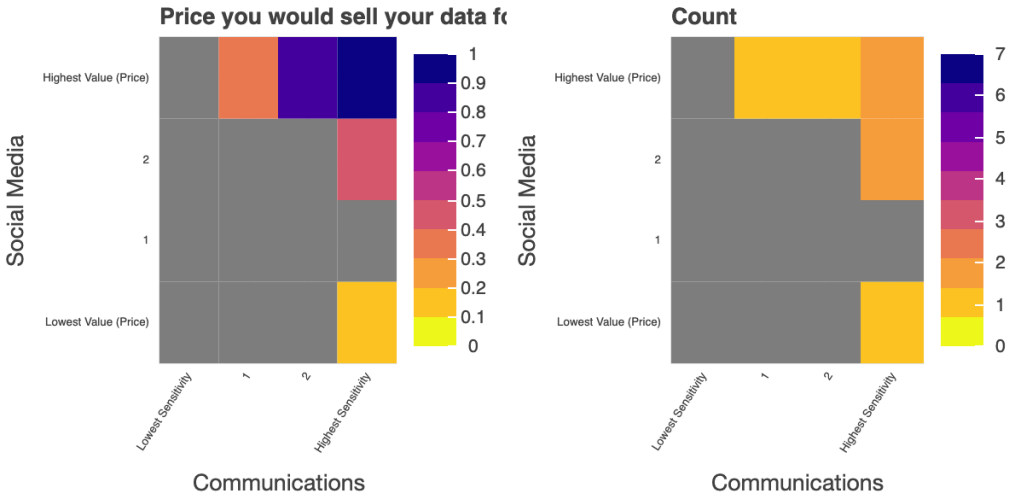


FIGURE 36: HEAT MAPS SHOWING PRICE AND ATTITUDE OF TRADING COMMUNICATION DATA WITH SOCIAL MEDIA COMPANIES.

7.2.2. Mistrust in trading data with organisations

On average, banks appeared highest on the ranking in terms of the price required to obtain data above that of the government (see Figure 37), suggesting that they are not as trustworthy. This is not what I expected as some research suggests that up to 70% of the public trust financial institutions accessing their personal data, greater than the percentage for the central government (Black, et al., 2018). This could be due to the large proportion of the participants in the study being younger and study conducted by Facebook has shown that only 8% of millennial's have trust in financial institutions (Facebook, 2016) leading them to charge banks the most for their data.

Carrying on with this idea of mistrust amongst organisations, Figure 37 shows that on average, there were no cards in the lowest position for value. This highlights the fact that many of the participants may not be very willing to trade their data with any organisation. This is further backed up in Figure 38 where 37% of all the organisations were placed in the highest value position. Such placement of cards could be related to distrust about the organisations ability to handle IoT data responsibly once the trade has occurred, with one report suggesting that only 25% of respondents believing that most companies handle their sensitive data responsibly (PwC, 2017).

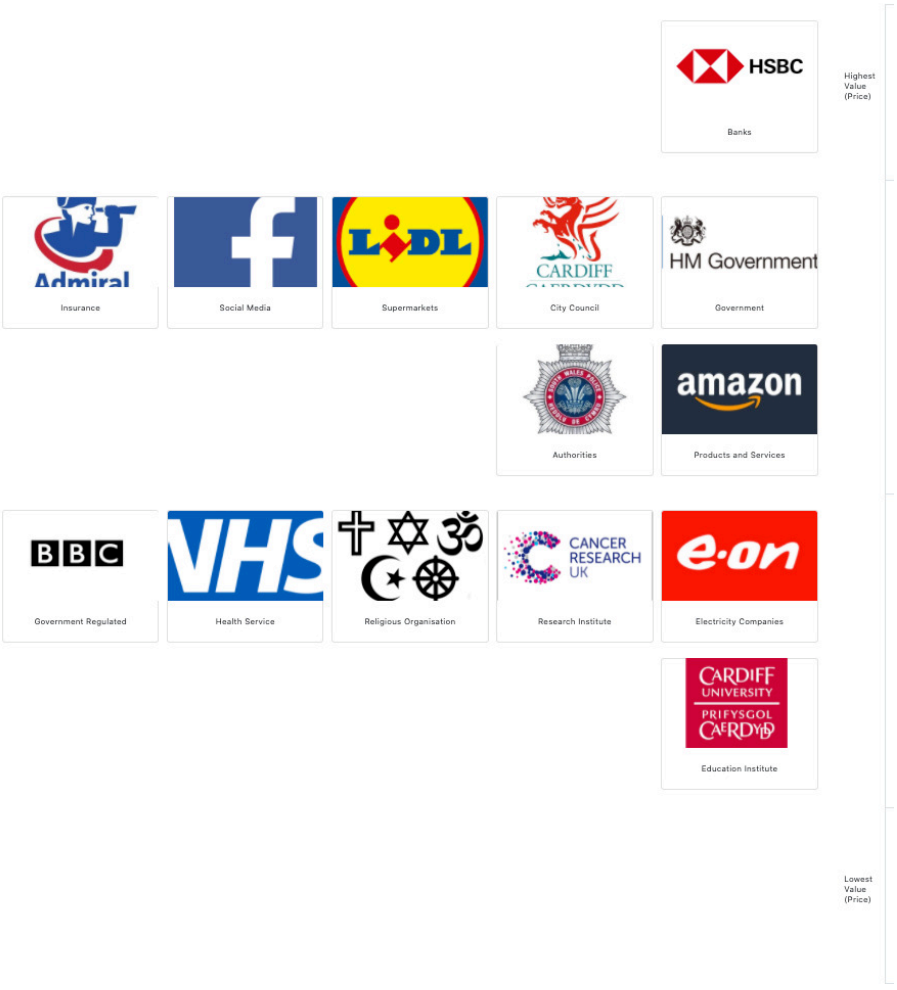


FIGURE 37: AVERAGE TRUST IN DIFFERENT ORGANISATIONS

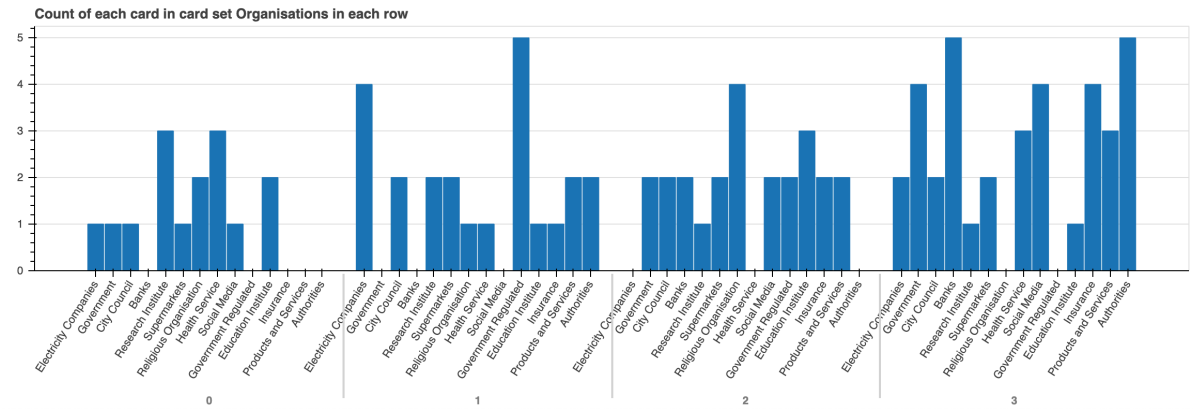


FIGURE 38: PARTICIPANT TRUST IN DIFFERENT ORGANISATIONS.

7.2.3. Participants don't care about the benefits of selling data beyond the monetary gain.

The study allowed participants to fill out 2 values for each combination of positions for an organisation and data type. It was thought that the price for which the participant was willing to sell their data to an organisation and receive some kind of extra benefit (such as a personalised service) would be less than that if they did not receive any extra benefit from selling their data. However, in many instances this was not the case as the price to sell data and receive extra benefit was greater than receiving no extra benefit at all (see Figure 39 for the average normalised values across the whole grid). In one extreme case participant ID 7, would actually give away some data for free if they received no extra benefit, but charge for selling data where they got extra benefit. Such a trend could be due to people not believing that companies will use their data to improve their lives, with one study showing that only 15% of people that participated in a survey believed they could (PwC, 2017). However, this fact does not offer a reasoning behind why people would want to charge more for selling

their data if they have the chance of receiving extra benefits, rather, it could be due to a misunderstanding of what the question means.

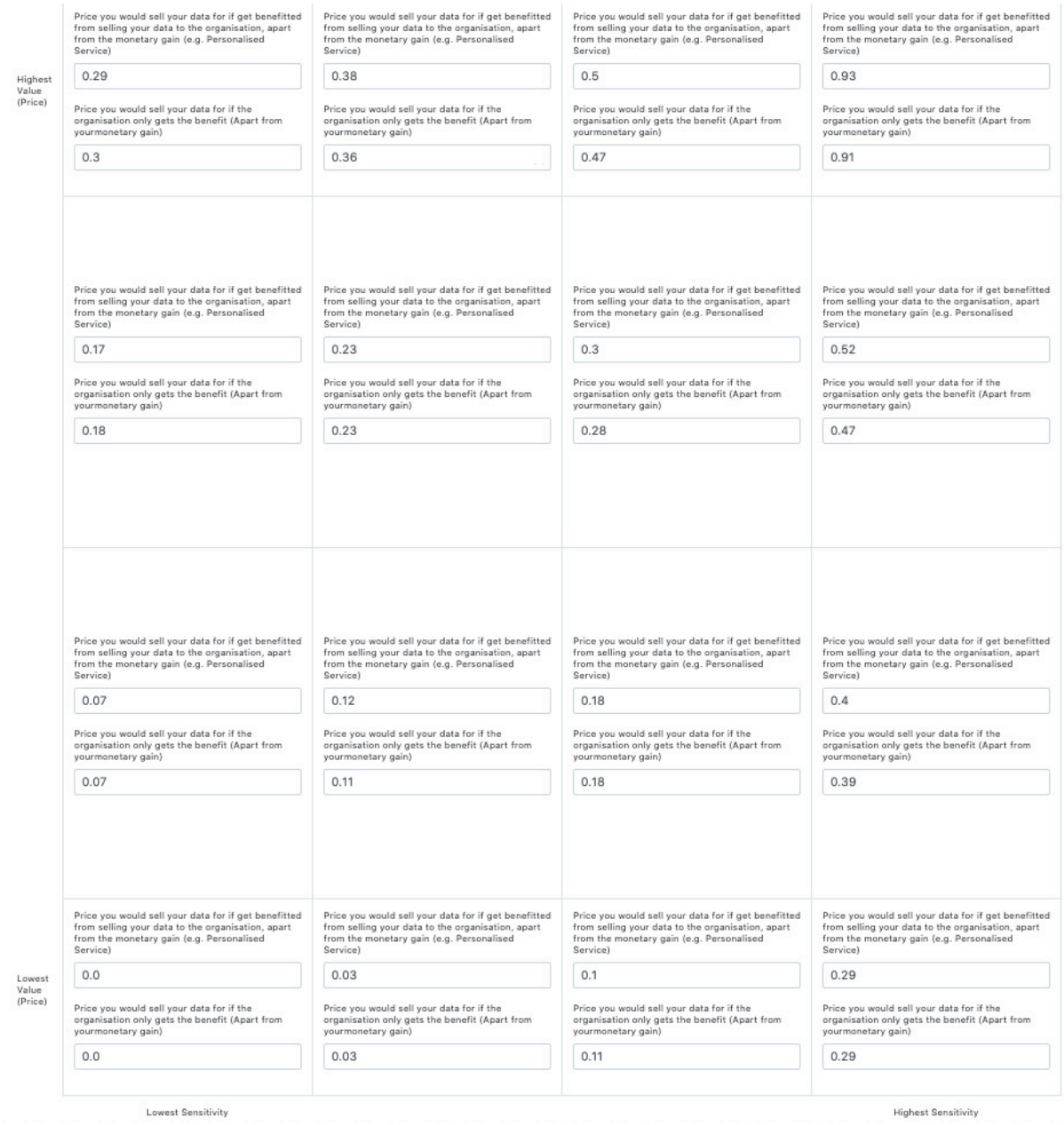


FIGURE 39: AVERAGE NORMALISED DATA VALUES ACROSS ALL RESPONSES FOR EACH POSITION IN THE GRID.

7.2.4. Free data for 'public good'

Despite the sensing as a service model being geared towards providing sensor owners control over their data and the ability to make a profit from it, 2 participants were willing to give data away to some organisations. Participant ID 7 was willing to sell basic, non-sensitive, fashion and entertainment data to public organisations and research institutes for free and participant ID 4 was willing to give away all their data to the same organisations (see Figure 40). This opinion is shared among other research, which showed that 47% of

respondents people are willing to share their medical data if it contributed to some ‘public good’, such as it helping develop new medicines and treatments (Black, et al., 2018).

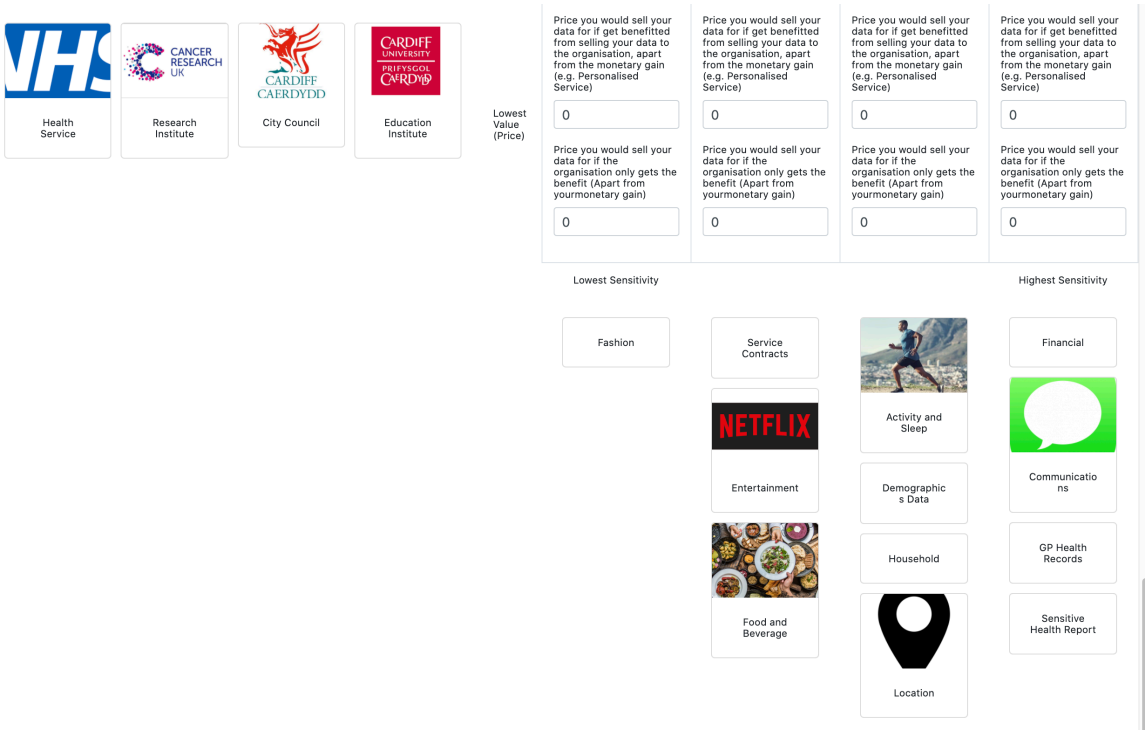


FIGURE 40: GIVING AWAY DATA FOR FREE TO PUBLIC AND RESEARCH ORGANISATIONS

7.3. Evaluation of Application

In light of some of the results that were identified using the application developed, it is clear that the tools used to analyse the responses of the willingness and price required to sell data to different organisations can be investigated to some extent.

It is very clear to see trends amongst where participants positioned organisations and data types respective of their measure, telling the story of how participants value different types of data and organisations independently.

Knowledge of the relationship between both data type and organisation was also able to be investigated through the heat maps. One type of heat map showed themes about the intersection of the placement of both data type and organisation, giving value towards the question of attitudes of a group of users towards any given organisation for any data type.

The other type of heat map was useful in showing the price relative to all other prices required for an organisation to purchase a given data type by using an average of normalised values across all responses. Normalising a participants values by scaling them from 0 to 1 meant that the results were not adversely affected by the units of measurement that the participant used to give a price, for example, a participant in Nigeria may enter ‘470’ as the price in Nigerian Naira whereas somebody in the UK may enter ‘1’ as the price in Pound sterling. These two values are equivalent as 1 GPB ≈ 470 NGN, however the application cannot understand context of the users unit of measurement so by normalising we can remove any potential bias in the results for differences in units of measurements

across all participants. By averaging the normalised values for each response, we can see the trend of pricing across all positions for an organisation and data type.

Despite the benefits that the tools bring in terms of representing the results of the study, there are still some shortcomings of the tools, preventing some conclusions to be made. For instance, you cannot analyse responses to see how opinions change over different demographics. This along with other omissions in the representation of the data could lead to a failure to satisfy all questions about people's willingness to trade different types of IoT data with different organisations.

In terms of the findings of the user study, although there were inklings towards a representative opinion for the problem investigated in the study, not enough responses were collected to form any valid conclusion. Thousands of participants were involved in other reports of a similar nature (PwC, 2017; Black, et al., 2018), meaning a similar number would be required to gain a representative population to address the problem fully. Due to practical constraints, limiting the study to those within the university, reluctancy of people to participate and the advent of the pandemic, it was hard to get a large participant group together. As a result, the study served as more of a use case to show that it was effective at representing results from a study rather than being scientifically sound research of the problem.

Although there is no correct way to fill in the survey, the way that some participants filled in the two different prices, showing that they would charge more for receiving a greater benefit from selling their data leads to the belief that they did not understand the question posed. This could be explained by the interface that was required to be developed was not very self-explanatory. Although it is not possible to determine if the participant was confused by the question due to the way the studies were required to be carried out and the anonymity behind the responses, any survey should seek to be as self-explanatory as possible to mitigate any misunderstanding of the question.

8. Future Work

Although largely the application meets the requirements of the project and satisfied its main objectives, there are always areas that can be expanded and improved on for the development of the application and research of the problem.

The following future work for the development of the application encapsulates extra features which enhance the functionality of the application as well as improvements that could be made to the design and implementation of the application.

- Allow comparisons of responses over the different demographics collected by the application to aid the ability to make conclusions about a particular survey over a representative population.
- Make the survey interface more intuitive, potentially by splitting it across multiple interfaces, making it more obvious how different parts of the survey link together.
- Addressing the problems encountered in Section 5.1.4 by making the administration module more flexible, giving the ability to update and delete different card sets, studies, and user groups under more conditions than are currently supported whilst

still maintaining the integrity of other modules that rely on a previous state of these entities.

The future work possible in terms of researching the problem further are:

- Collecting a greater number of responses from a wide variety of different demographics so more reliable conclusions can be drawn across those different demographics.

9. Conclusions

I have fulfilled my objective of creating an adaptable web application which supports the ability to create studies for user group. I met all of the requirements of the project both functional and non-functional. In particular, the implementation of the survey itself was particularly good, being close to the design provided and using a combination of different technologies to achieve both the interactivity required in the interface and a backend which supported being able to analyse the data produced by the survey responses.

The design of the database supported the requirements of the project whilst ensuring the simplicity of the implementation of the back-end of the codebase. The use of external software throughout the stack such as Flask, SQLAlchemy and Bootstrap aided the ability to deliver the project on time whilst taking out some of the pains of developing a web application. The pytest package helped create confidence behind the code that was developed and showed that the requirements of the project had been met. Despite the painless benefits that much of the external software provided, others, such as Bokeh came with their challenges to use and integrate with the existing code.

The DevOps workflow that helped streamline the process of moving from a development environment to a production environment, making it painless when the time came for people other than myself to use the application to participate in the user studies.

The user study conducted found some insights into understanding the willingness to trade different types of IoT data to different organisations, however with a larger sample set and greater functionality in the analysis of the data more reliable conclusions could be inferred. Despite this, the user studies did show the applications potential to solve the data trading problem through the analysis tools that were developed.

Overall, I believe the project has been a large success in meeting the aims that were set out.

10. Reflection

Throughout the whole lifecycle of the project, there has been a lot learnt and many decisions made about what was believed to be the best options for the project. The following is a reflection on the impact of the various decisions or approach taken for different characteristics of the project.

The design of the various constituents of the application proved difficult, in particular many revisions had to be made with the entity relationship diagram (see Figure 5). As the approach of iterative development was followed; designing, developing and then testing

different sections of the application, often there was an unequivocal rush to implement a feature and not spend much time thinking about design. Such a rushed approach often materialised themselves into problems when the application was larger, resulting in multiple revisions to the codebase. In the future, a greater consideration for the design of the components should be undertaken to prevent wasting time on implementations that wouldn't work.

The discussion of a MPA or SPA approach to creating a web application and the decision to use a MPA was justified within section 4.1.2. This decision was based upon various opinions about SPAs being more complex and worries a lack of knowledge in front-end development frameworks and languages. However, as the project matured and a greater understanding of SPAs grew through my own personal interests, there was vague doubts surrounding whether the MPA approach outweighed the negatives of a SPA. In the future, rather than rushing into using a technology that one may be comfortable with, much more careful consideration needs to be taken in terms of the options available for development. This ensures that the application is being developed with the most appropriate technologies available to it.

Beyond the design and implementation of the project, A great deal behind producing a quality product was learnt. By deciding to implement automated tests from the early stages of developing a feature of the application, I could prove consistently that I was meeting those requirements which made my confidence in the application greater than that of an untested one. In light of the testing procedure that was undergone, I felt there was still more which could be achieved. By practicing test-driven development (TDD), where tests are written before implementation (Janzen & Saiedian, 2005), you can use tests to drive design processes and software development, only producing enough code to make the tests pass, then have it refactored. By manipulating the order in which these tasks occur we can influence the projects maintainability, reusability and overall increase the quality of the software (Janzen & Saiedian, 2005).

Hofstadter's Law "It always takes longer than you expect even when you take into account Hofstadters Law" (Hofstadter, 1999) rang true for the course of the development of the project. Often, during the latter stages of each module, there was time devoted to extra features such as uploading a CSV file of emails or a display picture for a study. Whilst having these features was nice, they were never a requirement of the application and meant that time pressure would be placed on the next module. Amounting time pressures through these extra features, problems encountered with implementation and extra learning required lead to time pressures on some of the later parts of the project such as the user studies and the report writing. By sticking to a strict list of what needed to be implemented rather than going beyond could help me save some more time to focus on other, important areas of the project.

My decisions for using the DevOps workflow described in section 5.3 helped the purveyance of a quality piece of software, allowing for continuous updates and simplified the move from a development environment to a production one. Where there were problems encountered in production, the workflow ensured that fixes could be brought forward with speed and little human intervention. This kind of time and consideration to something often forgot about in development is definitely something that will be taken forward into future projects.

11. Appendix I: User Interface Designs

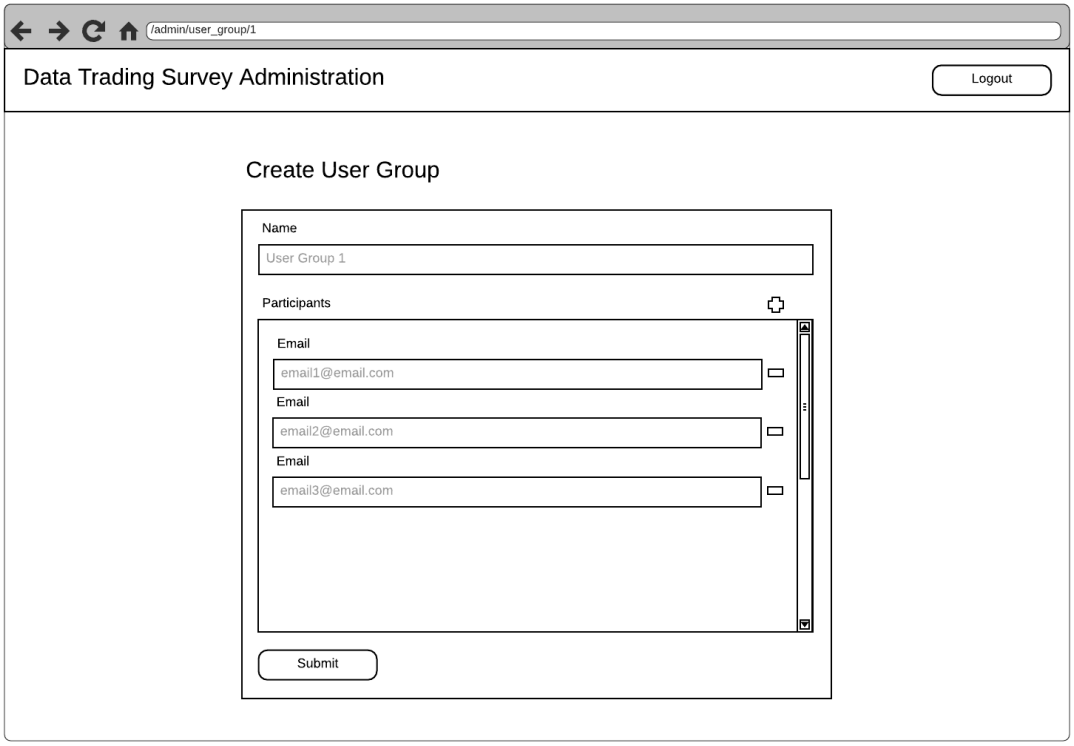


FIGURE 41: ADMIN CREATE USER GROUP WIREFRAME

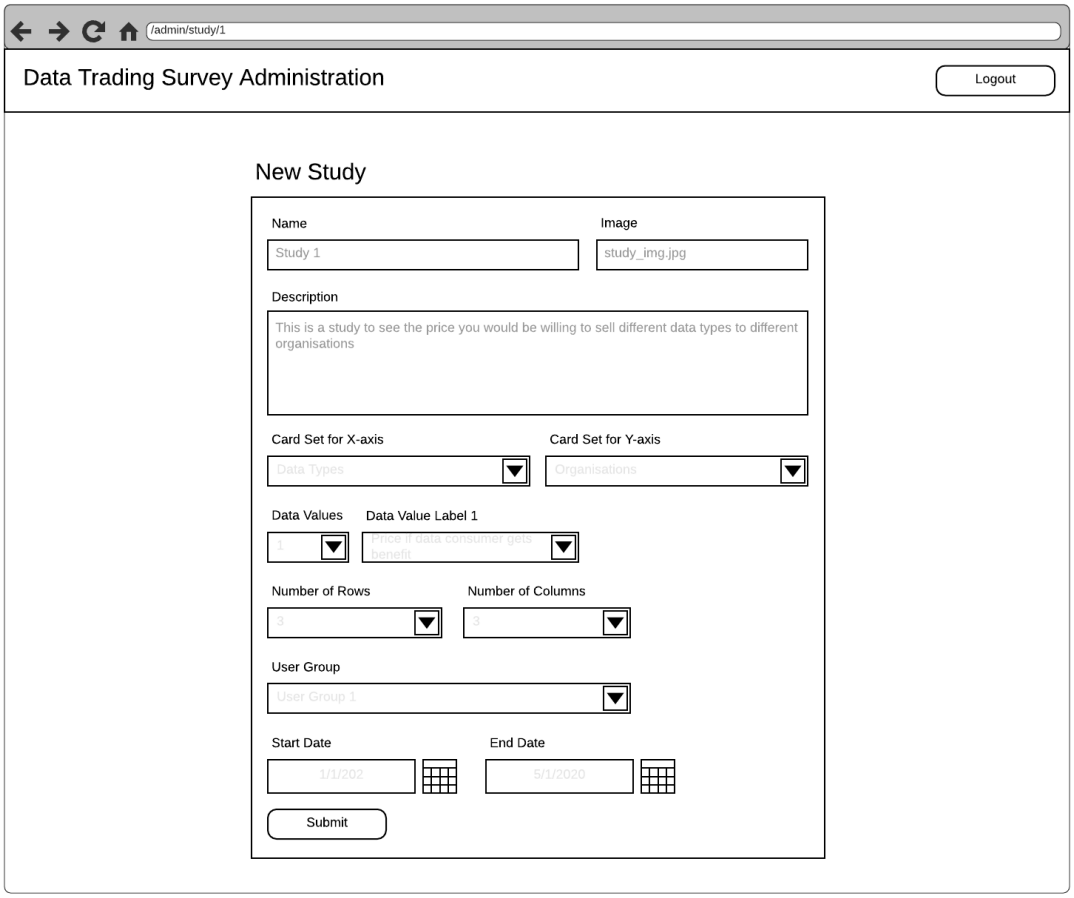


FIGURE 42: ADMIN CREATE STUDY WIREFRAME

Figure 42 shows the form for dealing with the creation of studies bringing together the other card set and user group forms.

The alignment of the user input's and their labels unifies the form as one set, leading to reduced cognitive load on the user as they don't have to link the label with its input.

To choose the start and end date of the study a calendar picker design pattern (UI Patterns, n.d.) was used as it simplifies the input of a date. There are many ways to input a date so manual entry can be fraught with invalid data therefore by using a date picker these potential errors are mitigated whilst minimising user frustration.

The use of select boxes where there are only finite options to choose also helps minimise potential errors further.

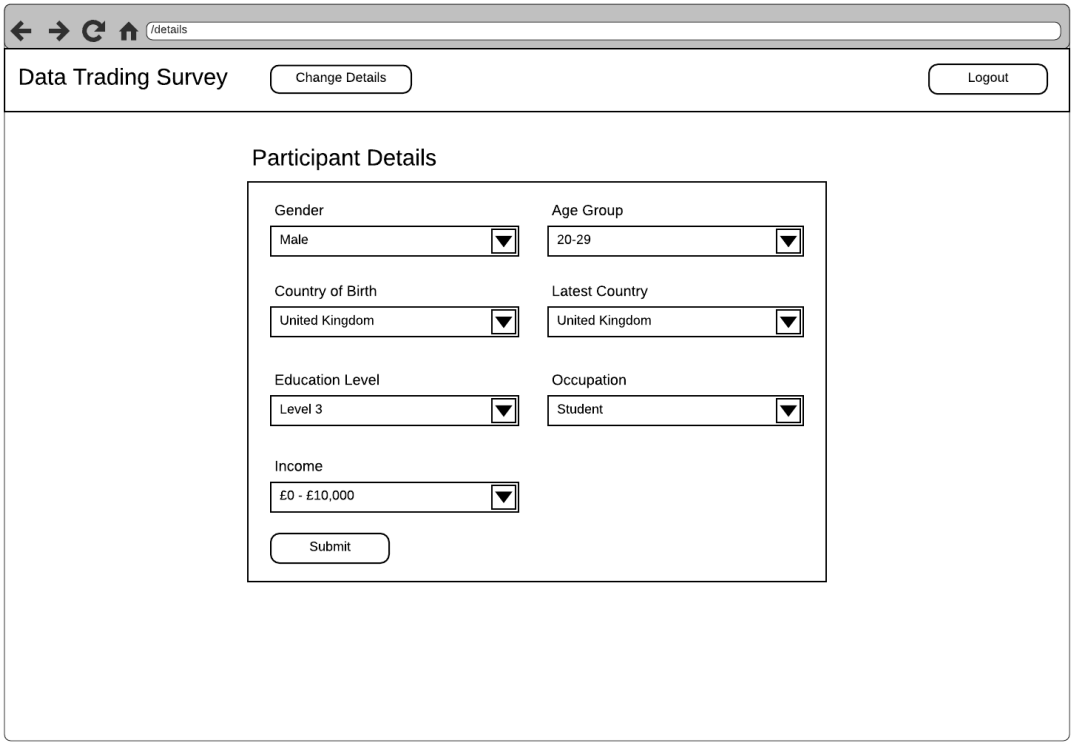


FIGURE 43: PARTICIPANT DETAILS FORM WIREFRAME

Figure 43 shows the UI for the participant details form, this includes all the details as mentioned in Requirement 4.

Figure 42 also shows select box menus, similarly this UI is entirely made up of select boxes minimising the potential space for an invalid form. Such an approach is also useful for narrowing down the range of values that users can select thereby making comparisons against different participants easier.

The alignment of select boxes and their labels unifies the form as one set, leading to reduced cognitive load on the user as they don't have to link the label with its input.

The information organisation of the form is based upon categories as in the LATCH principles. Those form elements which have a similar category are placed on the same row to give a logical connection, making the form simpler to fill out whilst reducing the amount of real-estate required for the form.

12. Appendix II: Supporting Video

- How-To style video used to aid the correct participation in the survey: Link

13. Appendix III: Test Summary Report

The test summary can be found within the application here (Cardiff University VPN is required to view the resource) alternatively, view the Appendix III Test Summary Report file.

14. Bibliography

Łępicki, H., 2017. *Why you should not build your start-up as Single-Page Application?.* [Online]
Available at: https://www.amberbit.com/blog/2017/9/20/why-you-should-not-build-your-startup-as-spa/
[Accessed 5 5 2020].

Alpine, n.d. *About.* [Online]
Available at: https://alpinelinux.org/about/
[Accessed 4 5 2020].

Alpine, n.d. *Alpine Docker Images.* [Online]
Available at: https://hub.docker.com/_/alpine
[Accessed 2 5 2020].

Bitnami, n.d. *Single-Tier Vs. Multi-Tier Architecture: Choosing The Right Bitnami Package.* [Online]
Available at: https://docs.bitnami.com/google-templates/singletier-vs-multitier/
[Accessed 2 5 2020].

Black, C., Setterfield, L. & Warren, R., 2018. *Online Data Privacy from Attitudes to Action: an evidence review,* s.l.: s.n.

Chiu, K., 2019. *Is WeChat too big to escape from even amid privacy concerns?.* [Online]
Available at: https://www.techinasia.com/wechat-big-escape-privacy-concerns
[Accessed 14 May 2020].

Clement, J., 2020. *Most popular mobile messaging apps worldwide as of October 2019, based on number of monthly active users.* [Online]
Available at: https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/
[Accessed 14 May 2020].

Cross, R. G. & Dixit, A., 2005. Customer-centric pricing: The surprising secret for profitability. *Business Horizons,* 48(6), pp. 483-491.

Ebert, C., Gallardo, G., Hernantes, J. & Serrano, N., 2016. DevOps. *IEEE Software,* 33(3), pp. 94-100.

Facebook, 2016. *Millennials + money: The unfiltered journey,* s.l.: s.n.

Faircloth, J., 2017. Testing enterprise applications. In: *Penetration Tester's Open Source Toolkit.* s.l.:Elsevier, pp. 243-271.

Flask, n.d. *Foreword.* [Online]
Available at: https://flask.palletsprojects.com/en/1.1.x/foreword/
[Accessed 2 5 2020].

Google, n.d. *MVC Architecture.* [Online]
Available at: https://developer.chrome.com/apps/app_frameworks
[Accessed 2 5 2020].

Greenlaw, C. & Brown-Welty, S., 2009. A Comparison of Web-Based and Paper-Based Survey Methods Testing Assumptions of Survey Mode and Response Cost. *Evaluation Review,* 33(5), pp. 464-480.

Gunicorn, n.d. *Gunicorn - Python WSGI HTTP Server for UNIX.* [Online]
Available at: https://gunicorn.org/
[Accessed 2 5 2020].

Hofstadter, D. R., 1999. *Gödel, Escher, Bach.* Anniversary Edition: An Eternal Golden Braid ed. s.l.:Basic Books.

ICO, n.d. *Guide to the General Data Protection Regulation (GDPR).* [Online]
Available at: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/principles/
[Accessed 2 5 2020].

ICO, n.d. *What is special category data?.* [Online]
Available at: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/special-category-data/what-is-special-category-data/
[Accessed 14 May 2020].

IHS Markit, 2017. *Number of Connected IoT Devices Will Surge to 125 Billion by 2030, IHS Markit Says.* [Online]
Available at: https://news.ihsmarkit.com/prviewer/release_only/slug/number-connected-iot-devices-will-surge-125-billion-2030-ihs-markit-says
[Accessed 2 5 2020].

Janzen, D. & Saiedian, H., 2005. Test-driven development: Concepts, taxonomy, and future direction. *Computer,* 38(9), pp. 43-50.

Kreibich, J. A., 2010. *Using SQLite.* s.l.:O'Reilly Media, Inc.

Liang, F. et al., 2018. A Survey on Big Data Market: Pricing, Trading and Protection. *IEEE Access,* Volume 6, pp. 15132-15154.

Lim, A., 2018. *Heuristics: The Psychology of Mental Shortcuts.* [Online]
Available at: https://www.thoughtco.com/heuristics-psychology-4171769
[Accessed 2 5 2020].

Lin, X., Zavarsky, P., Ruhl, R. & Lindskog, D., 2009. *Threat modeling for CSRF attacks.* s.l., s.n., pp. 486-491.

Lipski, R., 2017. *Single-page applications vs. multiple-page applications: pros, cons, pitfalls.* [Online]
Available at: https://ozitag.com/blog/spa-advantages/
[Accessed 2 5 2020].

Madhuri, J. A., Balkrishna, S. R. & Deshmukh, A., 2015. Single Page Application using AngularJS. *(IJCSIT) International Journal of Computer Science and Information Technologies,* 6(3), pp. 2876-2879.

Mao, W., Zheng, Z. & Wu, F., 2019. *Pricing for Revenue Maximization in IoT Data Markets: An Information Design Perspective.* s.l., Institute of Electrical and Electronics Engineers Inc., pp. 1837-1845.

Morse, J., 2019. *Just a reminder: Facebook Messenger isn't end-to-end encrypted by default.* [Online]
Available at: https://mashable.com/article/facebook-messenger-not-encrypted-by-default/?europe=true
[Accessed 14 May 2020].

Mozilla, n.d. *How the Web works.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/How_the_Web_works
[Accessed 2 5 2020].

Naumovski, A., 2017. *Straightening out the React/Redux learning curve part 1 - Intro to React.* [Online]
Available at: https://dev.to/andrejnaumovski/straightening-out-the-reactredux-learning-curve-part-1---intro-to-react-18b
[Accessed 2 5 2020].

Navis, G., n.d. *The Architecture No One Needs.* [Online]
Available at: https://www.gregnavis.com/articles/the-architecture-no-one-needs.html
[Accessed 5 5 2020].

Nielsen, J., 1994. *Enhancing the explanatory power of usability heuristics.* Boston Massachusetts USA, Association for Computing Machinery, New York, NY, United States, pp. 152-158.

Niyato, D. et al., 2015. Smart Data Pricing Models for Internet-of-Things (IoT): A Bundling Strategy Approach.

O'Neil, E., 2008. *Object/Relational mapping 2008: Hibernate and the entity data model (EDM).* New York, New York, USA, ACM Press, pp. 1351-1356.

Perera, C., 2017. Sensing as a Service (S2aaS): Buying and Selling IoT Data. *CoRR,* p. c.

Perera, C., Zaslavsky, A., Christen, P. & Georgakopoulos, D., 2014. Sensing as a service model for smart cities supported by Internet of Things. *Transactions on Emerging Telecommunications Technologies,* 25(1), pp. 81-93.

PwC, 2017. *Consumer Intelligence Series: Protect.me,* s.l.: s.n.

Rouse, M. et al., 2020. *internet of things (IoT).* [Online]
Available at: https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT
[Accessed 3 5 2020].

Shaughnessy, J. J., Zechmeister, E. B. & Zechmeister, J. S., 2000. *Research methods in psychology (5th ed.).* s.l., McGraw-Hill.

SQLAlchemy, n.d. *Key Features of SQLAlchemy.* [Online]
Available at: https://www.sqlalchemy.org/features.html
[Accessed 6 5 2020].

UI Patterns, n.d. *Calendart Picker Design Pattern.* [Online]
Available at: https://ui-patterns.com/patterns/CalendarPicker
[Accessed 2 5 2020].

UI Patterns, n.d. *Cards Design Pattern.* [Online]
Available at: https://ui-patterns.com/patterns/cards
[Accessed 2 5 2020].

UI Patterns, n.d. *Progressive Disclosure Design Pattern.* [Online]
Available at: https://ui-patterns.com/patterns/ProgressiveDisclosure
[Accessed 2 5 2020].

UK Govenrment, n.d. *What qualification levels mean.* [Online]
Available at: https://www.gov.uk/what-different-qualification-levels-mean/list-of-qualification-levels
[Accessed 5 11 2020].

W3Counter, 2020. *Browser & Platform Share.* [Online]
Available at: http://www.w3counter.com/globalstats.php?year=2020&month=3
[Accessed 2 5 2020].

Westendorp, P. v., 1976. *NSS Price Sensitivity Meter (PSM)—A New Approach to study Consumer-Perception of Prices.* Venice, s.n., pp. 139-167.

Williams, R., 1994. *The Non-Designer's Design Book.* 1 ed. s.l.:Peachpit Press.

Wurman, R. S., 1989. *Information anxiety.* s.l.:Doubleday.

Yaskevich, A., 2017. *Web application architecture: Components, models and types.* [Online]
Available at: https://www.scnsoft.com/blog/web-application-architecture
[Accessed 2 5 2020].