Title: Making Data Subjects better Data Controllers: Engineering a data manager for Data Subjects to track who holds what data.

Author: Harrison Morley

Course: Cyber Security and Forensic Computing

Project code: PJE40

Supervisor: David Williams

May 2025

Please tick

- ☐ I give permission for my project to be published in the University library and/or be made available to other students as examples of previous work. (optional).
- ☐ I confirm that I have read and understood the University Rules in respect of plagiarism and student misconduct.
- ☐ I declare that this work is entirely my own. Each quotation or contribution cited from other work is fully referenced.

Date: _____

[left intentionally blank]

Acknowledgements

I would like to thank my supervisor Dr David Williams for their invaluable feedback and guidance across the duration of the project. I am also grateful to Dr Rich Boakes for the initial inspiration for the project.

Abstract

This report explores the gap between the rights given to Data Subjects under the GDPR and the lack of means to practically enact those rights. An application was developed to acquire user data, inform Data Subjects on that data and enable them to enact their rights in a central place. While the application addresses most issues presented within the paper, future work needs to be conducted to make it a viable solution in the real world.

Table of Contents

Table of Figures

Table of Tables

Chapter 1: Introduction

1.1 Project aim and objectives

The aim of this project was to create a management system to enable users to make more informed privacy decisions and make it easier for them to enact those decisions. Three main objectives were set to achieve this:

- Acquire data from Data Controllers.
    - Create a way for Data Subjects to see their data in a central place.
- Inform Data Subjects about the extent of their data.
    - Create a way for Data Subjects to see statistics about their data.
    - Help Data Subjects make more informed decisions on their data
- Help Data Subjects enact data decisions.
    - Create a way for Data Subjects to manage their data from a central place.
    - Enable Data Subjects to make data decisions from a central place.

1.2 Project Deliverables

1. Data Manager Application with Testing Client. (https://github.com/H4RMLY/FYP-Data-Manager)
2. Development sprint reports (Appendix G to J)
3. Software Requirements Specification (Appendix F)
4. Literature Review (Chapter 2)
5. Design diagrams throughout the project.
6. User Survey (https://docs.google.com/forms/d/1tCBUklec7ZORoEOTWheSCx0Va5Nr_2Z67dtXDamSy4c/edit)
7. Raw Survey Results (https://docs.google.com/spreadsheets/d/1zAYiAdQ9vbvBkiWijzttiG0Z9FvBEF53-XAjUzmWvnU/edit?gid=1672741365#gid=1672741365)

1.3 Project constraints

During the project there were four constraints. Firstly, one person was undertaking management, research, requirements capture, design, implementation, testing and evaluation.

The project assumed a moderate level of knowledge in all aspects of the project lifecycle so there was a steep learning curve for the individual undertaking it.

Not only were there a lack of people but a strict time frame for the project. From start to finish the project was undertaken in under 8 months in tandem with other projects. This meant carefully planning throughout the project we needed and a minimum viable product had to be defined to keep within scope.

Lastly, there was no external funding for this project, all tools used throughout the project were free and any costs for developing the application had to be kept at a minimum.

1.4 Project Approach

The project approach loosely followed the PRINCE2 project management framework as the tasks and timelines were clearly defined throughout (PRINCE2, 2017). Agile methodologies were also used as the project was split into three-week sprints. Weekly meetings were

scheduled with the project supervisor to assess progress with sprint retrospectives held in the final week to help plan the following sprint.

During the four development sprints a software development life cycle was used to create a minimum viable product at the end of each sprint, mitigating the risk of having no software deliverable at project completion.

JIRA (Atlassian, 2024) was used throughout to strengthen project management ensuring task durations were accurately defined with story points on a backlog. Sprint tracking using Kanban boards meant deadlines were met consistently, and automatically created burndown charts were used to assess progress and help improve story point estimates. Backlog by Nulab (2025) was explored, however, due to the lack of functionality in the free version compared to JIRA it was reconsidered.

1.4.1 Research

A strict research methodology was used to find relevant material for discussion. Keywords such as, ‘Personal Data Control’, ‘Data Subjects as Data Controllers’, ‘Personal Data Ownership’, ‘Personal Data Store’, ‘Personal Data Vaults’, ‘Personal Data Manager’ and ‘Decentralised Data Storage’ were used in Google Scholar. Papers discussing context were discarded if they did not reference GDPR or were published before 2017. Papers discussing implementations we discarded if they did not include technical documentation or were published before 2009.

1.5 Legal and ethical considerations

This project concerns Data Subjects, Data Controllers and Personal Data and includes real participants for requirements elicitation. It adheres to the University of Portsmouth's Ethical Guidelines (University of Portsmouth, 2025) and received favorable ethical opinion from the ethics department (Appendix B).

Legally, the project strengthens General Data Protection Regulation (GDPR) by helping Data Subjects exercise their rights. All actions comply with EU regulation, and the authors do not hold and will not hold data collected by the application.

1.6 The report

1. Literature Review - Outlines the context for the project and why it is being undertaken.
2. Requirements Capture - Describes the requirements created for the software and the methods used to obtain them.
3. Design - Describes the design and features for the application.
4. Development - Discusses the implementation process and describes each stage of the software development life cycle within each development sprint.
5. Evaluation - Assesses whether the application meets the requirements, objectives and aim of the project.
6. Conclusion - Evaluates the effectiveness of the project management and discusses the value of the project and what can be taken from it.

Chapter 2: Literature Review

2.1 Introduction

Since 2016 there has been a push to give Data Subjects more rights to control their data (van Ooijen & Vrabec, 2018; Sim et al., 2022), however there is a question of how these methods benefit the Data Subject substantially (Hummel et al., 2020). This literature review considers the shift in Data Subjects becoming Data Controllers and the methods that have been implemented assist them.

The European Commission (2023) considers a Data Subject to be an identifiable individual who produces or has data relating to them, and a Data Controller to be the body that determines how and why personal data is processed. This definition is extended in this paper to a body that determines where the personal data resides and is processed.

2.2 Data Subjects and Data Controllers

Currently Data Subjects (DSs) and Data Controllers (DCs) are distinct. DSs will generate and send their data to be held and processed by DCs. DCs give access to processed data and services to DSs in return. In an environment where 123 Zetabytes of data was generated in one year (Taylor, 2024) with 360 million households owning at least one smart home device worldwide as of 2023 (Statista, 2023), DSs are more reliant on the effectiveness of DCs to store and protect their personal data than ever before.

Wilson et al. (2024) identify two forms of regulation in conversation across the world to protect DSs. Firstly The Fiduciary Model, based on traditional fiduciary relationships where there is an imbalance of power (Balkin, 2016). The DC acts as the individual with power over the DS as they hold their data, therefore legislation is based around the DC acting in good faith to the DS. Secondly The Rights and Obligations Model, giving rights to DSs and obligations to DCs. This model is predominant in most active legislation across the world such as GDPR.

Within GDPR DSs and controllers are written as separate entities; however the rights given to individuals enables them to become more like controllers. An example of this is Article 20 of GDPR (2018) ‘Right to data portability’ which gives the individual the right to receive their data in a way that can be given to a different data consumer. By allowing this the user can control where their information is held making them in some ways their own DC.

2.3 Data Subjects as Data Controllers

Governments within Brazil, China and the EU have implemented data protection laws that share common themes of user consent, consent withdrawal and DC transparency all of which can be viewed as giving the individual more control (GDPR, 2018; LGPD, 2020; PIPL, 2021).

GDPR gives the clearest example of control by outlining nine rights individuals can exercise. One of which as mentioned before is the ‘Right to portability’ which gives the DS the ability to control where their data resides, another is the ‘Right to erasure’ which allows the individual to request their data be deleted by a DC (GDPR, 2018).

Lazaro and Le Metayer (2015) identify that despite being given the control to make decisions about their data an individual will find it difficult to comprehend the amount of data they have. This is supported by Prince (2018) who adds that even when making decisions about their data individuals are unaware of what each decision entails. The root of this issue is summarised by Wolters (2018) and supported by Ausloos & Dewitte (2018) where they suggest that the rights afforded by GDPR will not benefit users because they cannot monitor how or where their data is being processed.

Collectively these papers outline three key issues:

1. DSs cannot comprehend the amount of personal data they have.
2. DSs are not fully aware of the impact of their data decisions.
3. DSs cannot coherently monitor who has their data and how it is being processed.

All of which impede the DSs ability to enact their rights under any of the Data Protection Laws around the world. Due to this it is important that the next step in data protection is to give DSs a tool to become better DCs.

2.4 Data Subjects as Better Data Controllers

As mentioned before there has been a great push to make DSs more like DCs however this cannot be done by simply allowing DSs to have complete control over their data.

One solution for making DSs better DCs is blockchain data management, which has seen a few implementations (Al-Abdullah et al., 2020; Khalid et al., 2023; Mishra & Haim Levkowitz, 2021). Another potential solution for this is the personal data vault (PDV). Also known as a virtual individual server (VIS) or personal data server (PDS). These applications allow for a user to store their data on a medium they own and control before it is sent to a DC.

There have been a several implementations of this idea (Cáceres et al., 2009; Mun et al., 2010; de Montjoye et al., 2014; HAT Project Research Team, 2015; Sambra et al., 2016); three notable examples include the virtual individual server developed by Ramón Cáceres et al. (2009), the personal data vault developed by Mun et al. (2010) and SOLiD pods developed by Sambra et al. (2016).

One issue raised by all three implementations is that there is an inherent security risk with DCs holding thousands of peoples data under a single administrative domain (Janssen et al., 2020). A recent example of this risk being exploited is the Lastpass data breach in august of 2022 where threat actors gained access to the cloud data storage application and had retrieved possibly millions of user accounts and encrypted data (Gentles et al., 2022; Spadafora, 2024; Twingate, 2024).

The second is that in most privacy policies DSs forfeit their data to be used however the DC sees fit. By having a server that only holds one individual's data this decentralises it making it safer from large scale data breaches. Secondly giving the user administrative control over the server and the data leaving it, the DS in turn becomes the DC. Inherently this improves the control a DS has over their data however enables the user to become a DC on a similar level to GDPR so therefore it falls to the same three issues identified in section 2.

Mun et al.(2010) PDV attempts to directly address the monitoring problem whereas SOLiD pods does this indirectly and the VIS doesn't at all. The PDV allows users to receive logs on where their data has been sent and what data they are providing using a feature called "Trace-Audit". The log data is then visualised onto the PDV in a digestible manner. SOLiD pods focus solely on the decentralised storage of data and only touches briefly on how this data can be managed. Pods are provided via "pod providers" so it is up to the provider on how the pods are managed. Despite this it is evident that pods, as smaller collections of data, can be easier for a DS to compartmentalise and manage in groups (Sambra et al. 2016). This does provide a central place for the user to see and manage their data however the user must intend to manage their data in a certain way.

The PDV also directly addresses the awareness problem where it is not mentioned at all by our other examples. Mun et al.(2010) developed a feature called "Rule-Reccomender" which recommends access control rules for the user to apply based on previous decisions as well as informing them on the implications of these decisions. This is a large step in enabling users to become better DCs as they need the context for the decisions they are going to be making which is currently not provided anywhere else. Unfortunately the usage of this rule recommender is limited to location data as this is the only data type this prototype is concerned with.

These three examples do not address the comprehension problem. The PDV briefly mentions visualisation of the trace audit log data however the extent of this visualisation is unclear. As the other two examples only focus on data storage there is no grasp on how the interfaces of these applications allow for data monitoring. Data comprehension is a common issue faced by most PDV implementations (Fallatah et al., 2023).

Giving DSs physical ownership over their data using a personal server solves issues surrounding mass data storage and privacy rights. However, in order for this to be effective consideration of how the user can manage their data needs to be taken into account. Currently under GDPR a DS has most of the control that the PDV examples afford without the benefits of decentralised storage however as mentioned before there are clear issues in how the user can enact these rights. A hybrid approach of data storage and management is a good solution to all of the problems raised and will heighten the DSs ability to become a DC.

2.5 Conclusion

In the current data environment there are a number of issues surrounding user data control despite efforts to improve them. Two areas have been identified where a DS could have control; data storage and data monitoring, with a large gap in the area of data monitoring.

We see a need for a data manager that considers the following requirements.

- Assist the user in comprehending the extent of their data.
- Allow the user to monitor the location of their data and how it is being processed.
- Allow a user to control their data though GDPR rights in a central place.
- Inform the users on the impact of their data decisions.
- All data stored by the manager is on a decentralised user owned device.

Having an application that achieves all 5 of these requirements would directly address all of the problems that have been identified in this paper and allow DSs to become more effective DCs.

Chapter 3: Requirements Capture (Analysis)

3.1 Introduction

Five requirements were identified before requirements capture however none of these considered users. An online survey (Appendix E) was conducted to collect user requirements and aid in prioritising feature development.

3.2 Requirements

Three non-functional and seven functional requirements were elicited from the survey which were combined with the five requirements identified from the literature review.

- Acquire - A-01
- Inform - I-01
- Enact - E-01

Table 1
All Requirements Identified For The Software

Level 3 - Functional Requirements			
ID	Requirement	Priority	Acceptance Criteria
A-01	Must be able to submit an ID for their manager to a vendor for a data request.	HIGH	An ID or code can be submitted to a vendor for communication.
I-01	Must be able to see recent data requests and accept or deny them.	HIGH	All data requests from vendors must be seen on the interface.
I-02	Must be able to see a collection of data that is held within the manager.	HIGH	All data stored on the manager must be seen on the interface.
I-03	Must be able to see a collection of all vendors and their ongoing data access.	HIGH	All vendors with access to data must be seen on the interface.
I-04	Must be able to filter vendors by how long they've had access to data, data they are	MEDIUM	The user can filter vendors based on access time, data in use and time last used.

	using and when they last used it.		
I-05	Must be able to see information about the consequences of denying a vendor access to their data.	MEDIUM	Consequences of a decision must be visible on the interface before being enacted.
I-06	Must be able to see recommendations on what decisions they could make.	LOW	Recommendations of decisions and justifications must be visible on the interface.
Level 2 - Non-Functional Requirements			
I-07	All recommendations and information must be easy to understand.	HIGH	Information displayed on the manager must be labeled. Recommendations must have clear instructions.
I-09	Key information such as the amount of vendors using data should be clearly displayed.	HIGH	Information relating to vendors, data or data requests must be visible on the respective display.
E-01	Decision making must be simple and streamlined and cannot require too many inputs.	MEDIUM	When a user wants to make a decision it must not require too many inputs.
Level 1 - General Requirements			
I-10	Assist the user in comprehending the extent of their data.	HIGH	Every aspect of user data should be visible. This can be accepted in the form of detailed charts.
I-11	Allow the user to monitor the location of their data and how it is being processed.	HIGH	Every piece of data should have a marker for where it is being stored.

E-02	Allow a user to control their data though GDPR rights in a central place.	HIGH	Data erasure requests must be available at minimum.
I-12	Inform the users on the impact of their data decisions.	MEDIUM	Information on the impact of a decision must be visible before it is enacted.

3.3 User Story 4 (US4) - The Data Manager

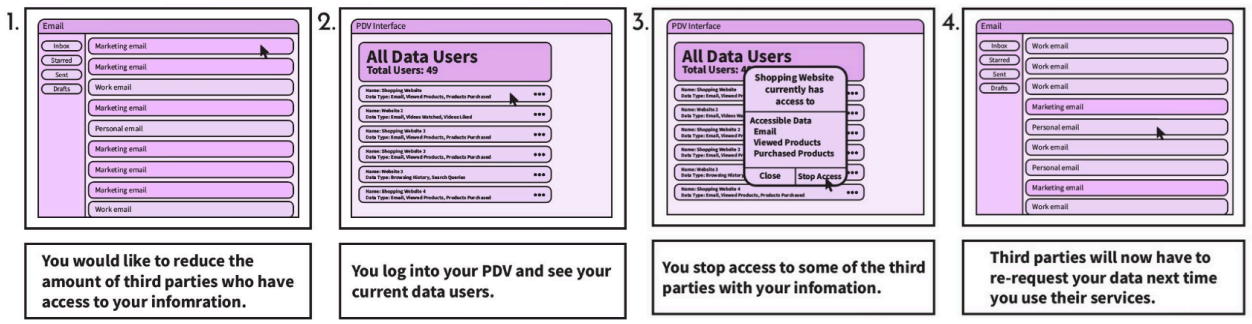


Fig 3.1 - Illustration outlining a user story for the data manager.

Figure 3.1 illustrates The Data Manager user story, showing how an individual can use the application to make data decisions for multiple vendors. As one of four user stories, The Data Manager was picked by participants as the one they'd use most, making it the focus of implementation and where the ten new requirements were derived from.

Participants were also asked to rank four features for each user story. For US4 they were asked to rank these features:

1. Quickly see how many vendors have ongoing access to your data.
2. Filter vendors based on how long they've had access to your data, what data they are using and when they last used your data.
3. You are informed of the possible consequences of certain decisions.
4. You can see recommendations on decisions you may want to make.

Requirements A-01, I-01 and I-02 were not ranked as they are essential to the core functionality of the user story. From the options given, Feature One received the highest average rating (4.3 out of 5) making I-03 the highest priority user requirement. This was followed by Features Three and Two (I-04, I-05), and lastly Feature Four (I-06).

The survey was posted on several forums on Reddit and Facebook as well as some student survey swap sites. Twenty-three people had completed the survey by the end of the one month running period.

Survey Results

The results were entered into a Google spreadsheet where a weighted average formula was applied to each score to identify the highest-ranked user story and the most valued features. This revealed the feature set that would be prioritised and the order of implementation.

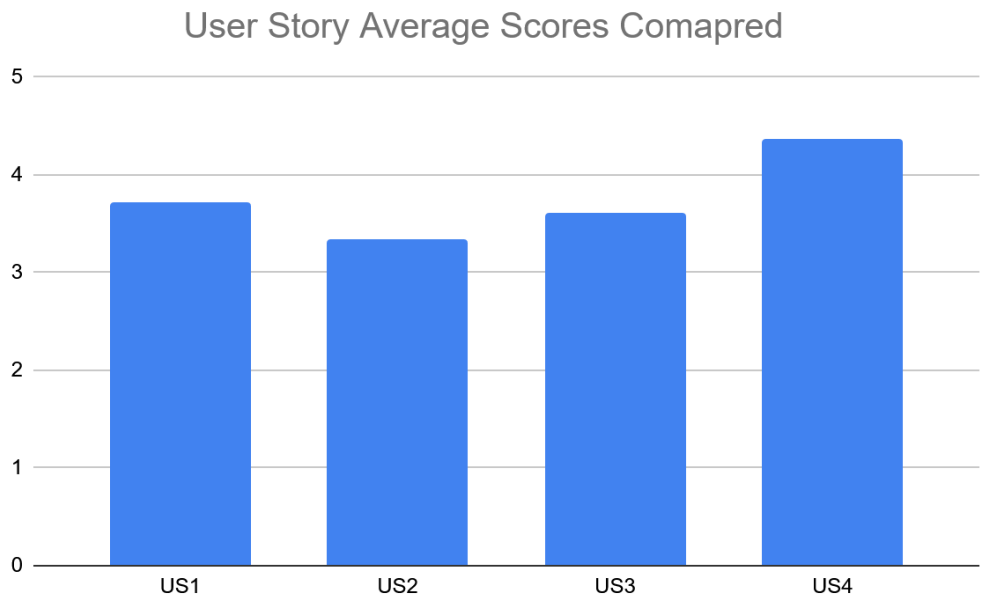


Fig 3.2 - Bar chart showing the average scores of each user story.

Figure 3.2 shows the user story ranking results, with US4 achieving an average score of 4.4 out of 5, followed by US1 far behind.

3.4 Summary

The survey added ten additional functional and non-functional requirements. Designing the application was now easier as a general framework for what it should do was already in place. The survey also steered the project toward a more useful endpoint as before the intention was to create an app closer to a PDV. The responses indicated that more value was seen in a manager than a storage application.

Chapter 4: Design

4.1 Introduction

From the requirements capture the below design was created (Fig 4.1). The system itself is composed of three core nodes with some components housed on the vendor side. Components on the vendor side were kept minimal to make integrating the system as easy as possible.

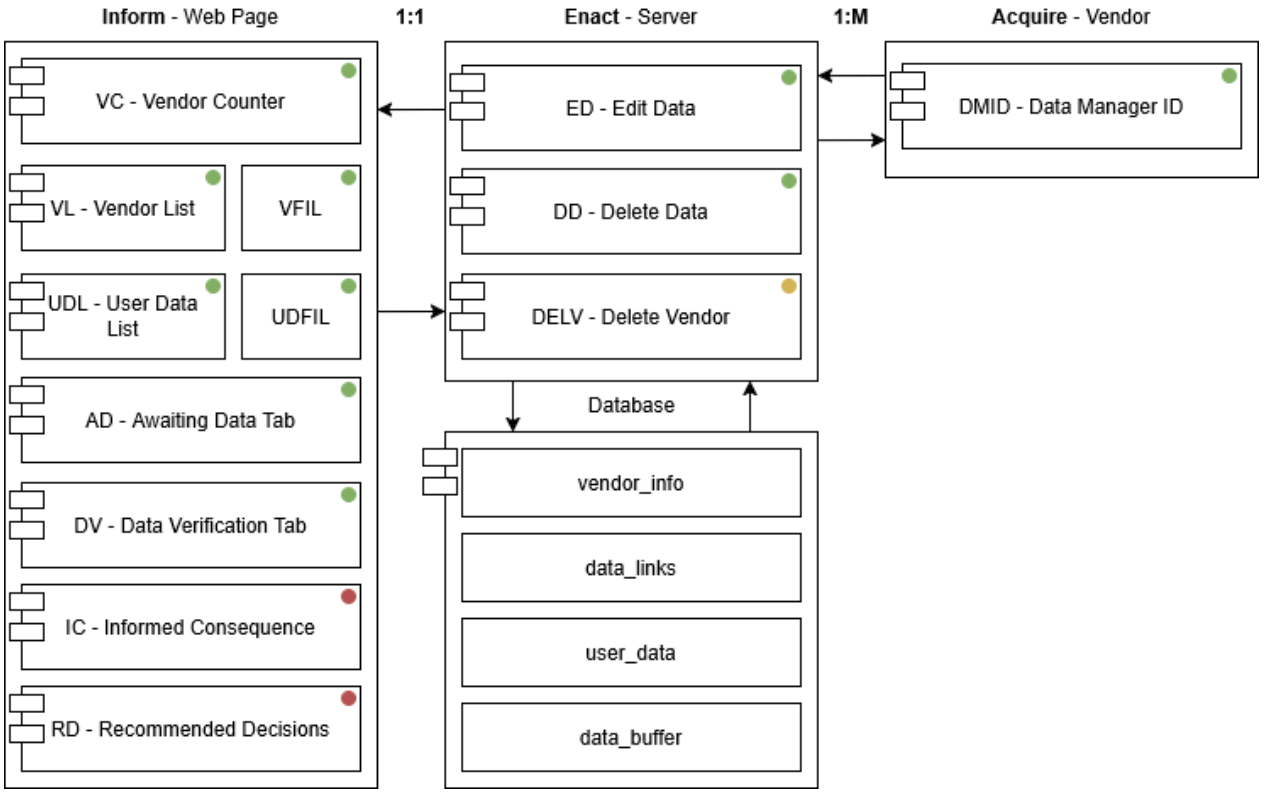


Fig 4.1 - Diagram Showing the Distribution of Functions For The Complete Application.

Features

Table 2
Table of Application Features

ID	Feature	Description	Requirements	Priority	Tests Cases
DMID	Data Manager ID	An ID system that allows individual managers to be identified and communicate d with.	A-01	HIGH	DMID-01, RD-01, RD-02, RD-03
VL	Vendor List	A list that displays all information about all vendors that have access to the users data.	I-03, I-09, I-10, I-11	HIGH	(VL-01 to VL-05), RD-02, RD-03, DV-03, UDL-03, (FIL-01 to FIL-06)
UDL	User Data List	A list of all data that is held by vendors with labels of which vendors have access to it.	I-02, I-09, I-10, I-11	HIGH	(UDL-01 to UDL-04), RD-03, DV-04 DV-05, ED-01, DD-01, D2UD-01, (MD-01 to MD-05), (FIL-01 to FIL-07)
DV	Data Verification Tab	A Tab used to verify new data entering the manager so the user can verify if it is theirs .	I-02	MEDIUM	(DV-01 to DV-06), D2B-01, D2B-02, RD-03
DD	Delete Data	A button on the user data list that allows the user to delete a piece of data from all vendors.	E-01, E-02, I-05	HIGH	DD-01, IV-01, IV-04, IV-07
ED	Edit Data	A button on the user data list that allows the user to	E-01, E-02	HIGH	ED-01, ED-02, IV-02, IV-03, IV-05, IV-06, IV-08, IV-09

		edit a piece of data for all vendors who store it.			
DELV	Delete Vendor	A button on the vendor list that deletes all data a vendor holds.	E-01, E-02	HIGH	UDL-02
AD	Awaiting data Tab	A tab that shows all data that is awaiting confirmation from a vendor that a decision has been fulfilled.	I-09, I-11	MEDIUM	AD-01, AD-02, IV-08, IV-09
VC	Vendor Counter	A counter at the top of the vendor list that displays how many vendors have access to the users data.	I-09, I-10	MEDIUM	VC-01
VFIL	Vendor Filter	A filter that allows the user to refine the information they see on the vendor list.	I-04, I-09, I-10	MEDIUM	FIL-01 to FIL-06
UDFIL	User Data Filter	A filter that allows the user to refine the information they see on the user data list.	I-09, I-10	MEDIUM	FIL-01 to FIL-07
IC	Informed Consequences	When a user goes to make a decision information is displayed about the	I-05, I-07, I-12	MEDIUM	Not Implemented

		consequences of that decision.			
DR	Decision Reccommender	A tab that recommends decisions for the user to make based on factors such as how long a vendor has had access to the users data	I-06, E-02	LOW	Not Implemented

The features in table 3 were outlined within the software requirements specification along with stimulus and response scenarios for each (See appendix F).

4.2 Acquire

Acquire is a small aspect of the system however it is an incredibly important one. DMID is the only example of a pure acquire feature as it solely facilitates the acquisition of data from the vendor.

There are two places data is acquired from a vendor. The first is when the manager logs user-submitted data from the vendor such as:

- The name of the vendor
- The data type of the data being sent
- The user-submitted data
- The origin url of the packet

Which is sent as a payload to the DMID at the point of submission. The second is confirmation of a completed decision from the vendor. This also uses the DMID to send a payload back to the manager once a vendor has enacted a requested decision.

4.3 Inform

Inform involves all aspects of the webpage and collecting data from the database to be displayed. Almost every feature directly or helps present information in some way.

There are four aspects to the webpage which all inform the user about different areas of their data. The first is the Vendor List (Fig 4.2). This is a list of all vendors that currently have access to any data logged on the manager. The user can get information such as vendor name and the data types they have access to from this tab.

The second is the User Data List (Fig 4.2). This is similar to the vendor list however it shows all the data that is stored by vendors. The user has access to information such as the vendors that have access to a piece of data, the data type and the data itself.

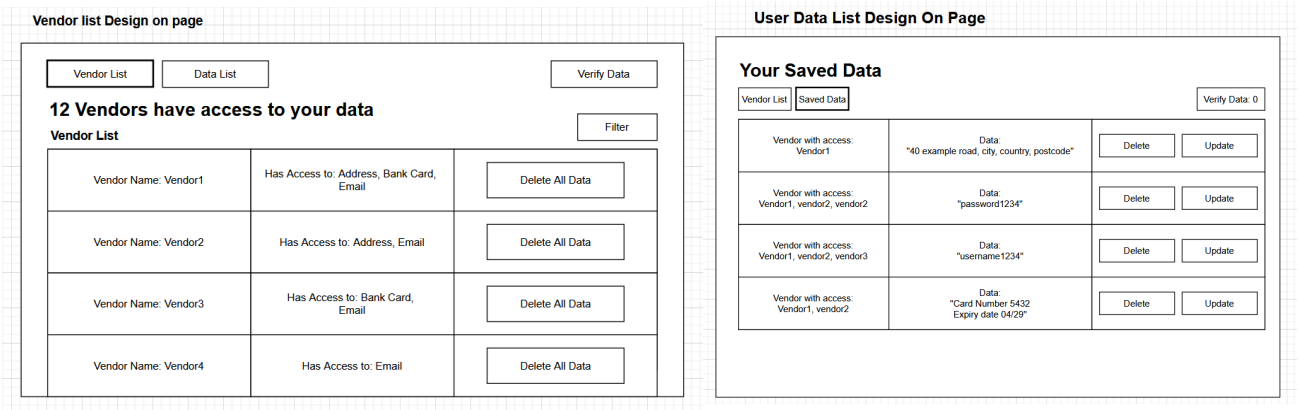


Fig 4.2 - Wireframes for vendor list (left) and user data list (right).

The third is the Data Verification Tab which shows all new data that doesn't exist in the manager already. Here the user has the option of accepting a piece of data which will then store the data permanently or they can reject it which will delete it from the manager all together.

On both the Vendor and User Data Lists the user can filter what they see based on attributes such as:

- Vendor
- Data Type
- Data

Last, is the Awaiting Data Tab. This allows the user to see what requests are currently being processed by a vendor. Within this list information such as the vendor that is processing the decision, the data that's being processed and the decision that is being processed can be viewed.

4.4 Enact

Enact involves the user making decisions on their data. Features such as Data Editing and Data Deletion form part of this aspect.

There are several places the user can enact a decision. The first is in the verification tab where the user decides whether the data will be stored in the manager or not. This decision is completely local.

The second is on the vendor list. The user can request to delete all data that a single vendor has access to. This removes the vendor from the vendor list and any link to the data the vendor once had in the data list. This is more complicated as it requires the user making the decision, that decision being sent to the vendor then enacting the decision locally once the vendor has confirmed it.

The last place the user can make a decision is on the data list. For each piece of data on the data list a user can either update or delete it for every vendor that is linked. This allows the user to rectify an error in data across multiple vendors in a single place or delete data for all vendors entirely.

2.1 Summary

At its smallest an application that can achieve all three aspects of the design by tracking where an individual's data is and allow them to make decisions from a central place will help DSs become better at controlling their data.

Taken further to the full design outlined in this section a data manager that can acquire a user's data with information about where that data is held; Can clearly inform the individual on statistics around their data such as how many third parties have access to their data or how many different types of their data there are; And can enable them to make decisions directly from the application itself will greatly empower the DS and will allow them to become an effective DC.

Chapter 5: Development

5.1 Introduction

The development of the application was split into four sprints of three weeks. Within each sprint four to five tasks were set and design, implementation, testing and evaluation were completed on each one. This meant that each sprint undertook a small software development lifecycle which helped the app grow steadily over time.

This section will be divided into the four development sprints that took place, with tasks and aspects of the life cycle detailed within each one.

5.2 DevSprint1 - DS1 (Appendix G)

Tasks

- Component - Virtual Machine with a suitable operating system.
    - Design
    - Implement component
    - Evaluate designs
- Component - Web Server to host a placeholder webpage.
    - Design
    - Implement component
        - Test component: WP-01
    - Evaluate designs
- Component - A means for the VM to store data.
    - Design
    - Implement component
        - Test component: RD-01
    - Evaluate designs
- Component - A means for the VM to receive data remotely.
    - Design
    - Implement component
        - Test component: RD-01
    - Evaluate designs

Design

There was very little visual design within this sprint. Requirements for the virtual machine operating system and initial ideas for how to remotely send data were the only plans that had been made in terms of implementation.

The operating system needed to be lightweight and low profile but still be able to run basic functions such as DNS, HTTPS and database hosting. Ideally this would have a very low storage requirement so that as much storage as possible could be used for the data rather than the application. This would also decrease the barrier to entry for users.

A custom developed server was chosen over a pre-built one as it could be completely customised to reduce redundant functionality. This also meant that HTTPS for remote data could be handled natively.

Implementation

One candidate for the operating system was Tiny Core Linux, a small distribution between 10 and 20 MB that despite its limited functionality still met the requirements of design (Shingledecker,

2013). Unfortunately when running inside VirtualBox booting issues were encountered. After testing multiple other operating systems of similar size the one that proved most reliable was Ubuntu Live Server 24.04.2 (Canonical Group Ltd, 2025).

This partially fulfills the requirements as it is lightweight due to the lack of GUI and has all the necessary functionality. Additionally some tools that would be required in the future were already installed. This installation came to around 2.4GB which is drastically bigger than the ideal implementation.

The server and webpage form the core of the application using Node.js (2024) as a runtime environment. This helps create an asynchronous web application that can handle many queries internal and external, while only requiring five lines of code to start the server (Fig 5.1). A MySQL database is also implemented at this stage containing one table to store all incoming data.

```
import express from 'express';
const app = express();
const webRoot = './';
app.use(express.static(webRoot));
app.listen(port);
```

Fig 5.1 - Server initiation code.

With a webpage now in place development shifted focus to remote data transmission. In accordance with the design, the server linked to the webpage and database could receive data using HTTPS requests. To test this a client was created that acts as a mock vendor to send JSON data to the server via HTTPS POST. Using Express.js (OpenJS Foundation, 2025) the server can process the remote data from the HTTPS request allowing data to be stored within the database (Fig 5.2).

```
app.post('/sendUserInfo', express.json(), recvInfo);

// Receives info from vendor and inserts data into the respective tables
function recvInfo(req, res){
	const vendorId = req.body.id;
	const vendorName = req.body.name;
	const type = req.body.datatype;
	const data = req.body.data;

	addVendor(vendorId, vendorName);
	let dataID = addData(type, data);
	updateLinkedData(vendorName, dataID);
}
```

Fig 5.2 - Express.js allows custom function calling when HTML requests are made to the server.

Testing

Across the four development sprints the testing strategy improved. At first only a few Gherkin test scenarios were created which would act as an end to end test for each feature. This meant that each feature was tested in whole to see if it achieved the task it was targeting; however the strategy did not help when identifying why a test may have failed.

Table 3
End To End Tests For DS1

ID	Test	Gherkin	Passed
WP-01	Placeholder webpage must show when the VM url is entered into a browser.	Given: The user has the correct url for the page. And: The VM is running. When: The user enters the url in any web browser on any network. Then: The webpage will be displayed in the user's browser.	Yes
RD-01	When a button is clicked on the client side a data entry must be added into the VM database.	Given: The client has the VM url. And: The VM is running. When: The add button is clicked on the client side. Then: The VM should receive the name and ID of the client and store it in its database.	Yes
VC-01	When the page is refreshed an updated count of all of the instances within the	Given: The VM is running.	Yes

	VMs database should be displayed on the screen.	And: The SQL database is running. When: The user refreshes the VM webpage. Then: The server should query the database and display the updated count on the webpage.	

Verification and Validation

All tasks within DS1 were components of future features therefore they do not target requirements.

Reflecting on the design requirements component one achieved a lightweight operating system using Ubuntu Live Server. Upon reflection the virtual machine may not be needed as it only serves as a hosting environment which can be achieved from the user's personal computer.

A custom Node Express server has been created which allows the creation of a custom server function such as figure 5.?. Because of this, component two has been successfully implemented.

Component four is achieved using HTTPS POST requests to the server and a mySQL database. This is vital for acquiring data from the vendor in the future for the Vendor and User Data Lists.

5.3 DevSprint2 - DS2 (Appendix H)

Tasks

- Feature - Data Manager ID - DMID
    - Design
    - Implement feature
        - Test feature: DMID-01
    - Evaluate requirements: A-01
- Feature - Vendor List - VL
    - Design
    - Implement feature
        - Test feature: VL-01, VL-02
    - Evaluate requirements: I-3, I-09, I-10, I-11
- Feature - Data Verification - DV
    - Design
    - Implement feature
        - Test feature: DV-01
    - Evaluate requirements: I-02
- Component - Acquire user-submitted data from the vendor using DMID.
    - Design
    - Implement component
        - Test component: RD-02
    - Evaluate requirements: A-01

Design

With a system foundation in place, visual design began with Feature DMID. There were two approaches to this, the first, a central domain with subdomains for each manager. The second, each manager hosts its own domain (Fig 5.3). The second method is better for the applications purposes as it allows the system to be completely decentralised.

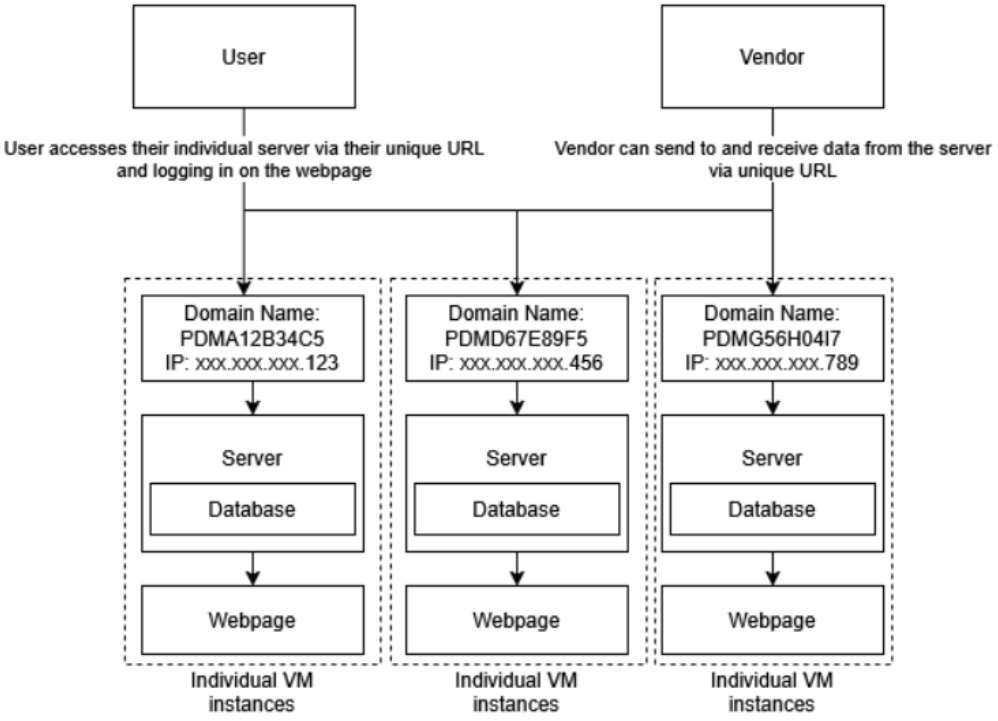


Fig 5.3 - DMID Design Two.

Upon installation the application would generate an 8 digit ID and create a domain. The domain can then be used to access that server from a browser. The ID would also serve as the method of delivery for data from vendors.

When considering the data acquisition component there were two possible designs. The first, has the vendor request data stored within the manager making the system closer to a PDV.

The second has the vendor send data to the manager when the user submits it to their site. This way the manager acts more as a ledger, logging where the user's data is stored rather than acting as the store itself.

The first design requires more interaction between the manager and the vendor as well as more operation from the user. This would become too cumbersome to use as the user is constantly switching between the manager and the vendor site.

Within design two, the only extra action the user is required to make is inputting the DMID. The manager then passively logs the data so the user only needs to interact with the manager when they want to monitor their data or make a decision.

Implementation

The system already allowed for remote data transmission using the VM IP so feature DMID was implemented last.

Feature VL was first to be implemented. Most lists within the application formats and collects content similar to the VL. The server sends a query to the database for all information stored in the respective table (Fig 5.4). The server sends the result to the browser, which formats it into a list (Fig 5.5).

```
// Returns a list of all vendors in the vendor_info database
async function getVendorList(req, res){
    con.query("SELECT * FROM vendor_info", async (err, result) => {
        if (err) throw err;
        res.json(result);
    });
}
```

Fig 5.4 - Server side component for the Vendor List.

```
// Fetches and displays all vendors from the database on the page
async function vendorList(){
      const response = await fetch('/vendorList');
      if (response.ok){
      let vendorList = await response.json();
      for (const vendor of vendorList){
            const template = document.querySelector('#vendorInfo-template');
            const listItem = template.content.cloneNode(true);

            const name = listItem.querySelector('.vendorNameText');
            name.textContent = vendor.vendor_name;

            const linkedData = listItem.querySelector('.linkedDataText');
            linkedData.textContent = vendor.linked_data;

            const vendorList = document.querySelector('#vendorList');

            const deleteButton = listItem.querySelector('.deleteButton');
            deleteButton.dataset.id = vendor.vendor_id;
            deleteButton.addEventListener('click', deleteVendor);

            vendorList.append(listItem);
      }
   }
}
```

Fig 5.5 - The first iteration of the Vendor List.

The first iteration of the list only showed the data IDs linked to the vendor which was bad for legibility (Fig 5.6).

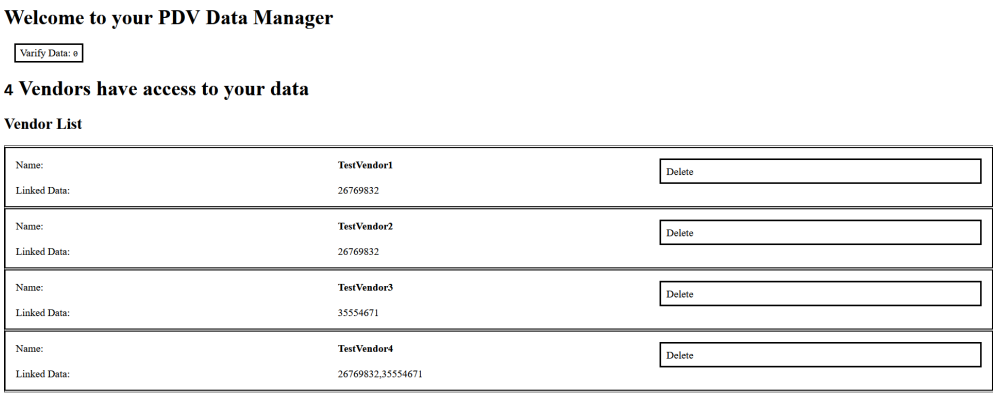


Fig 5.6 - A screenshot from the application of the first iteration of the vendor list.

While implementing data acquisition a user data table was added to the database, allowing the remote payload to include both user data and vendor information. User data is captured using basic input capture with JavaScript on the vendor side (Fig 5.7).

```
// Grabs values from address inputs and returns a formatted string
function grabAddress(){
    let inputs = document.querySelectorAll('.addr-input');
    let addrLine1 = inputs[0].value;
    let addrLine2 = inputs[1].value;
    let city = inputs[2].value;
    let county = inputs[3].value;
    let postcode = inputs[4].value;

    return `${addrLine1}, ${addrLine2}, ${city}, ${county}, ${postcode}`;
}
```

Fig 5.7 - Address input capture on the test vendor client.

A JSON payload can then be created and sent via HTTPS POST, structured as follows:

{vendorName, dataType, userData}

Additional information such as vendor and data IDs is generated on the server this way the burden for the vendor is minimal.

```
// Generates a unique 8 digit ID.
function generateID(){
      let idLength = 8;
      let id = '';
      const characters = '0123456789'; // Can be any characters for more IDs
      const charactersLength = characters.length;
      for (let i = 0; i < idLength; i++) {
         id += characters.charAt(Math.floor(Math.random() *
         charactersLength));
```

```
        }
        return id;
}
```

Fig 5.8 - ID generation code used for Vendor and Data IDs.

As data is being pushed to the manager rather than pulled, it is important to implement data verification since users could incorrectly input their ID, resulting in incorrect data being sent to a manager. Including a verification system ensures all data stored on the manager belongs to the user.

To achieve this a buffer table was created for incoming data. This way the user data and buffer tables can be compared for matches (Fig 5.9). If incoming data doesn't exist within the user data table it will remain inside the buffer until verified by the user.

```
// Checks if the data in the buffer table already exists in user data. If so
it creates a link with the existing data id and discards the buffer.
async function checkData(dataId, data, vendorName){
    con.query("SELECT * FROM user_data WHERE data = ?", [data], (err, result)
    => {
            if (err) throw err;
            if(result.length > 0){
                console.log(`Data already exists in user data, creating link
                to data for vendor ${vendorName}`);
                linkData(vendorName, result[0].data_id);
                deleteDataFromBuffer(dataId);
            } else {
                console.log(`Data does not exist in user data, adding to
                buffer for verification.`);
            }
        });
}
```

Fig 5.9 - checkData function for testing for new data.

The data stored inside the buffer table is presented using a list following the same method as the VL along with options to accept or deny the data.

If data is accepted, it is copied into the user data table, a link to the respective vendor is created and the buffer instance is removed. If the data is denied, it is removed from the buffer with no other changes. If the data already exists inside the user data table, it is automatically removed from the buffer, and a data link is created between the vendor and existing data.

The last feature to implement was the ID system. In the design this feature took the form of a domain name, with a unique ID generated by the application at install. However, during implementation this could not be achieved within the budget it required $2.82x10^{12}$ domains to be registered.

The first design, which uses a central domain with subdomains for each manager, is more cost-effective but compromises the decentralised aspect of the application. A system is already implemented that achieves the intended design however, as the IP of the VM has been used previously as if it were the DMID for testing.

One issue with keeping the ID in this form is that it involves submitting an unmasked IP address to an external party, which is a security risk. While this meets the acceptance criteria for requirement A-01, it is only suitable for a proof of concept until a more secure implementation is in place.

Testing

DS2 continued the same testing strategy with one end to end test per feature (Table 6). This time however extra tests were included that measured the speed of the VL function. The goal was to test how long it took for data to be displayed as the size of the table increases. Based on research stating a user may visit up to 200 websites per week (Kumar N 2024), the test started at 200 instances.

Table 4
Speed Testing For VL Function

Test ID: VL-02	
Number of Instances in database.	Time to query and display (millisecond average after 10 queries).
200	1.4 ms
500	1 ms
1000	1.5 ms
2000	1.6 ms

Table 5 confirmed the expected result: as the size of the database increases, the time to display the information also increases. The rate of increase is roughly 0.1ms per 1000 instances which is slow enough to not impact usability.

Table 5
End To End Tests For DS2

ID	Test	Gherkin	Passed
DMID-01	A vendor should be able to send unique data to two separate managers using different IDs.	Given: The vendor has a valid PDVid. When: The vendor sends the data to the url. Then: The manager with the	Yes

		corresponding ID should receive the JSON payload.	
DV-01	If the manager receives a new set of user data for the first time it should ask the user to confirm it is theirs.	Given: The vendor sends a payload to a valid url. When: The manager does not recognise an item of data. Then: The manager should ask the user to verify the data before it is stored in the database.	Yes
VL-01	A user should be able to see a list of all the vendors and their information on the managers webpage.	Given: The database has at least one data entry. When: The user views the manager webpage. Then: The page should show a formatted list of the data within the database.	Yes
RD-02	When the user submits data to the vendor that data should be sent to the manager.	Given: The user has entered a valid url. And: Submitted data to the vendor. When: The vendor sends the information payload. Then: The payload should contain the data the user has submitted.	Yes

Verification and Validation

Table 6 Validation Table for DMID		
Feature: DMID Component: Acquire user-submitted data from the vendor using DMID.		
Requirement	Acceptance criteria	Achieved
A-01	An ID or code can be submitted to a vendor for communication.	Yes

As seen in Table 7, DMID reaches the acceptance criteria of requirement A-01 despite not being achieved with the intended design. By using the IP of the VM each individual manager can be identified however this is insecure. In the future a way of masking the IP address needs to be developed if the application is to be used outside of a proof of concept. The feature is further supported by the data acquisition component which uses the feature to store user-submitted data sent from a vendor.

Table 7 Validation Table for VL Feature		
Feature: VL		
Requirement	Acceptance criteria	Achieved
I-03	All vendors with access to data must be seen on the interface.	Yes
I-09	Information relating to vendors, data or data requests must be visible on the respective display.	No
I-10	Every aspect of user data should be visible. This can be accepted in the form of detailed charts.	No
I-11	Every piece of data should have a marker for where it is being stored.	Yes

Table 8 confirms that the VL fulfills some of the acceptance criteria for the associated requirements. The feature does not meet requirements I-09 and I-10 at this stage as the data linked to a vendor is shown through an ID which to the user is meaningless.

Table 8 Validation Table for DV		
Feature: DV		
Requirement	Acceptance criteria	Achieved
I-02	Must be able to see a collection of data that is held within the manager.	Yes

DV allows the user to see all new data logged by the manager therefore it meets the acceptance criteria for I-02.

With the completion of DS2, the application could acquire information from the vendor and inform the user about which vendors have access to their data. The user can also enact decisions, such as data verification within the manager.

5.4 DevSprint3 - DS3 (Appendix I)

Tasks

- Component - Display data types for data instead of IDs on the VL.
    - Design
    - Implement component
        - Test component: VL-03, VL-05
    - Evaluate requirements: I-09, I-10
- Feature - User Data List - UDL
    - Design
    - Implement feature
        - Test feature: UDL-01, UDL-02, UDL-03, UDL-04, RD-03, DV-04, DV-05
    - Evaluate requirements: I-02, I-09, I-10, I-11
- Feature - Edit Data - ED (Local changes only)
    - Design
    - Implement feature
        - Test Feature: ED-01, ED-02, IV-02, IV-03, IV-05, IV-06
    - Evaluate requirements: E-01, E-02
- Feature - Delete Data - DD (Local changes only)
    - Design
    - Implement feature
        - Test feature: DD-01, IV-01, IV-04, IV-07
    - Evaluate requirements: E-01, E-02, I-05
- Component - Create a means for informing a vendor about a decision.
    - Design
    - Implement component
        - Test component: IV-01, IV-02, IV-03, IV-04, IV-05, IV-06, IV-07
    - Evaluate requirements: E-01, E-02

Design

The UDL was the first to be designed as it was more complicated than the previous lists implemented.

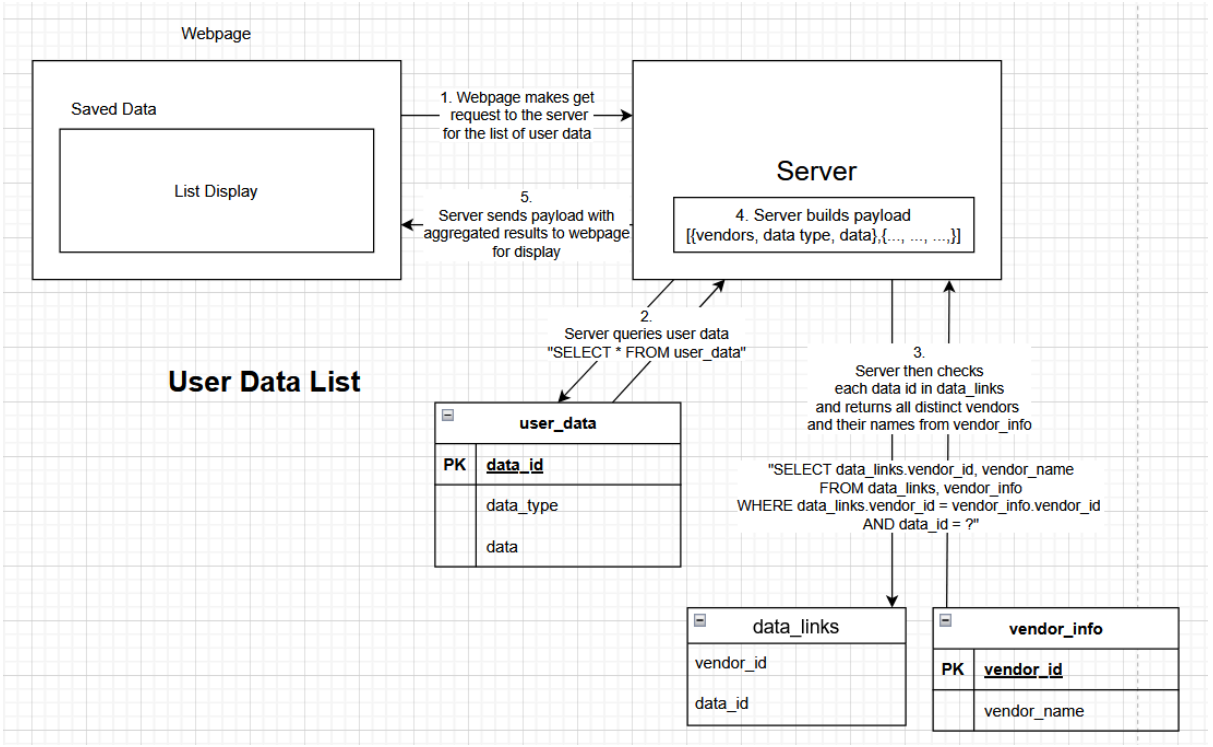


Fig 5.10 - Design diagram created for the user data list function.

Figure 5.10 is derived from the VL. However within this list, each time a list instance is created all vendors linked to it must also be displayed. This design includes a new table within the database which holds all links between data and vendors providing the ability to use more complicated queries and reducing the size of most functions.

Two designs were created for vendor informing. The first (Fig 5.11) prioritises enacting decisions locally before informing the DC. The second (Fig 5.12) prioritises confirming that the vendor had enacted a decision before making local updates.

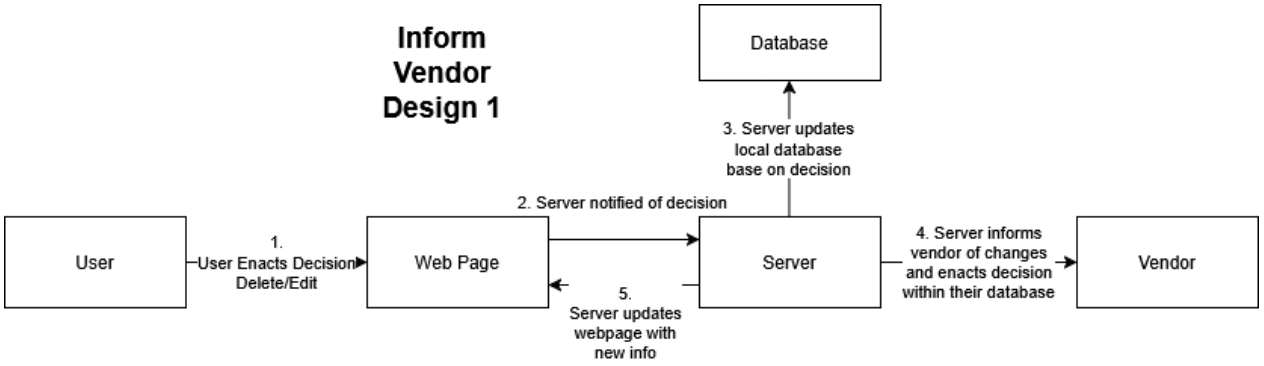


Fig 5.11 - The first inform vendor design diagram prioritising local changes.

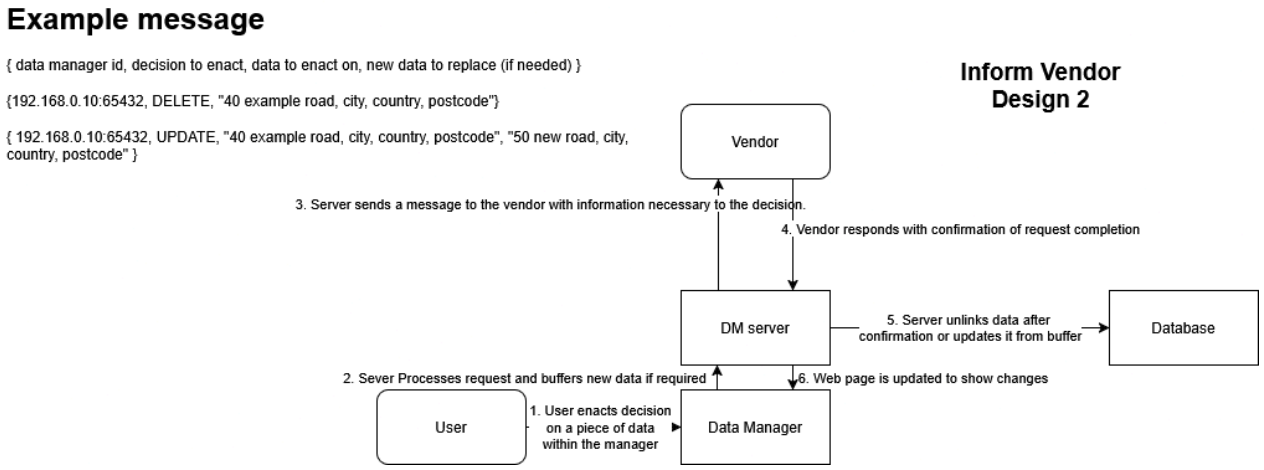


Fig 5.12 - The second inform vendor design prioritising vendor confirmation.

Design one is easier to implement however it runs the risk of the manager falling out of sync with what is stored on the vendor side. GDPR 2018 Art.12(3)(1) states that a DC has up to a month to reply to a data request meaning the manager could be displaying inaccurate information for up to a month.

Design two waits for confirmation from the vendor before local changes are made; this ensures the manager reflects what the DC is actually storing rather than what it should be storing.

Implementation

The first priority was updating the VL to display data types, as the logic would be similar for collecting the linked vendors of a data instance in the UDL. This also meant that the database would have been reworked by the time the UDL was being implemented.

The VL now works in two stages: First, the webpage requests all vendors inside the ‘vendor_info’ table and iterates over the result, each time requesting the data types linked to each vendor (Fig 5.13). This is a significant load for the browser and can be pushed server side however it is acceptable for the current state of the application.

```
// Fetches and displays all vendors from the database on the page.
async function vendorList(){
    const response = await fetch('/vendorList');
    if (response.ok){
        let vendorList = await response.json();
        for (const vendor of vendorList){
            const template = document.querySelector('#vendorInfo-template');
            const listItem = template.content.cloneNode(true);

            const instance = listItem.querySelector('.vendorInfo');
            instance.dataset.vendor = vendor.vendor_name;

            const name = listItem.querySelector('.vendorNameText');
            name.textContent = vendor.vendor_name;
```

```
            // Fetch data types
            const payload = { vendor_id: vendor.vendor_id };
            const response = await fetch('/getVendorDataTypes', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(payload),
            });
            let vendorDataTypes;
            if (response.ok){
                vendorDataTypes = await response.json();
                let linkedData = listItem.querySelector('.linkedDataText');
                linkedData.textContent = vendorDataTypes.join(', ');
                instance.dataset.dataType = vendorDataTypes;
            }

            const vendorList = document.querySelector('#vendorList');

            const deleteButton = listItem.querySelector('.deleteButton');
            deleteButton.dataset.id = vendor.vendor_id;
            deleteButton.addEventListener('click', deleteVendor);
            if(vendorDataTypes.length > 0){
                vendorList.append(listItem);
            }
        }
    }
}
```

Fig 5.13 - VendorList as it exists in the final software.

Fig 5.14 - A screenshot showing the multiple of the same data type being shown on the vendor list.

When applying the data types to the list if a vendor has access to multiple of the same data type it would appear twice in the list item (Fig 5.14). Although this functioned as expected this could be considered a bug therefore it was updated to only show one representation of each data type (Fig 5.15).

```
// Gets data types for a specific vendor to be displayed on the vendor list
async function getVendorDataTypes(req, res){
    const vendorId = req.body.vendor_id;
    con.query('SELECT vendor_id, data_type FROM data_links, user_data WHERE
    user_data.data_id = data_links.data_id AND vendor_id = ?;', [vendorId],
    (err, result) => {
```

```
        if (err) throw err;
        let dataTypes = [];
        for (const DT of result){
            if (!dataTypes.includes(DT.data_type)){
            dataTypes.push(DT.data_type);
            }
        }
        res.json(dataTypes);
    });
}
```

Fig 5.15 - GetVendorDataTypes function only.

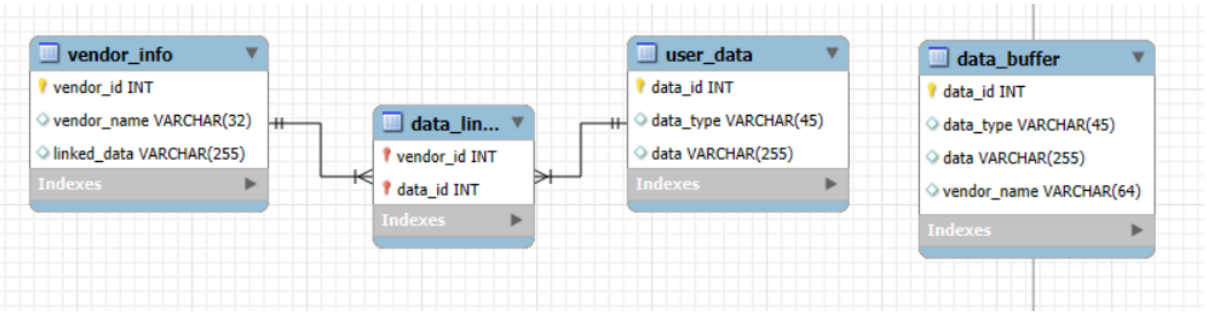


Fig 5.16 - Database diagram showing the final database structure.

With the new database (Fig 5.16) created and a way of iteratively collecting information from multiple sources developed, the UDL could be implemented. Similar to the VL, the webpage requests all instances within user_data. Instead of returning data immediately, the server aggregates a list of linked vendors for each data instance (Fig 5.17). Once completed the full data is sent to the webpage (Fig 5.18). Server side aggregation decreases the browser load and minimises the amount of fetch requests compared to the VL.

```
// Gets all user data then checks each data for vendors that are linked to it
async function getUserDataList(req, res){
    let userData;
    // Gets all user data
    con.query("SELECT * FROM user_data", async (err, result) => {
        if (err) throw err;
        userData = result;
        // Checks each data for vendors that are linked to it.
        for (const data of result){
            let result = new Promise((resolve, reject) => {
                getLinkedVendorName(data.data_id, resolve);
            });
            userData[userData.indexOf(data)].vendor_names = await result;
        }
        res.json(userData);
```

```
    });
}
```

Fig 5.17 - Server side function for User Data List.

```
// Fetches and displays all verified user data on the saved data list
async function userDataList(){
    const response = await fetch('/getUserDataList');
    if (response.ok){
        let userDataList = await response.json();
        for (const data of userDataList){
            const template = document.querySelector('#userData-template');
            const listItem = template.content.cloneNode(true);

            const instance = listItem.querySelector('.userDataItem');
            instance.dataset.vendor = data.vendor_names;
            instance.dataset.data = data.data;
            instance.dataset.dataType = data.data_type;

            const vendors = listItem.querySelector('.userDataVendors');
            vendors.textContent = data.vendor_names;

            const userData = listItem.querySelector('.userData');
            userData.value = data.data;

            const dataType = listItem.querySelector('.userDataType');
            dataType.textContent = data.data_type;

            const deleteButton = listItem.querySelector('.deleteButton');
            deleteButton.dataset.id = data.data_id;
            deleteButton.addEventListener('click', deleteDataEvent);

            const editButton = listItem.querySelector('.editButton');
            editButton.dataset.id = data.data_id;
            editButton.addEventListener('click', editDataEvent);

            const userDataListElem = document.querySelector('#userDataList');
            userDataListElem.append(listItem);
        }
    }
}
```

Fig 5.18 - Browser side function for User Data List.

Originally, edit and delete buttons were intended for the VL. However since there is no editable data the edit button was moved to the UDL, allowing the user to change individual data instances. The delete button was still added to the VL, deleting the vendor along with any links associated from the database. On the UDL, the delete button removes the respective data and associated links. The edit button updates the database to hold new data the user provides. Initially the buttons only enacted decisions locally.

Vendor informing uses the Fetch API (Mozilla Developer Network, 2025) to send data similar to how the server receives data. When a new vendor is added to the database, the server captures the origin IP of the data and stores it with the vendor's information for use in this function. When called, the function iterates over each vendor linked to a data instance and sends a payload to the corresponding origin IP with the following structure:

{vendorName, sourceURL, dataID, data, decision, ?(newData)}

While a decision is processed by a vendor, local updates must be held. To do this, an instance of the expected data is sent to the buffer table along with the decision. This method allows the manager to still reflect the data held by a vendor on the UDL (Fig 5.20).

After a decision has been processed by the vendor, a payload is sent back to the manager via the sourceURL. This contains the completed decision and the ID of the affected data. The server enacts the decision locally.

```
// Informs all vendors linked to a data item of a decision
async function informVendor(req, res){
    let sourceIp = req.connection.localAddress;
    const sourceURL = `${sourceIp.replace(/[:f]/g,
    '')}:${req.connection.localPort}`;

    con.query("SELECT vendor_url, vendor_name, data, data_type,
    user_data.data_id FROM vendor_info, user_data, data_links WHERE
    vendor_info.vendor_id = data_links.vendor_id AND user_data.data_id =
    data_links.data_id AND data_links.data_id = ?;", [req.body.dataId], async
    (err, result) => {
        if (err) throw err;
        // Iterate over vendors
        for (const vendor of result){
            const vendorURL = vendor.vendor_url;
            // Create payload
            const payload = {
                vendor_name: vendor.vendor_name,
                sourceURL: sourceURL,
                data_id: vendor.data_id,
                data: vendor.data,
                decision: req.body.decision
            };
            // Assign purpose and buffer local changes. If edit add new data
            to payload
            if (req.body.decision === "EDIT"){
```

```
                payload.newData = req.body.newData;
                addDataToBuffer(vendor.data_type, payload.newData,
                payload.vendor_name, 'awaiting edit', payload.data_id);
            } else {
                console.log(`adding ${vendor.data_id} to buffer for delete`);
                addDataToBuffer(vendor.data_type, payload.data,
                payload.vendor_name, 'awaiting delete', payload.data_id);
            }
            // Send payload to vendor
            console.log(`Sending payload: ${JSON.stringify(payload)} to
            vendor @ ${vendorURL}`);
            const response = await fetch(`http://${vendorURL}/decisionRecv`,
            {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(payload),
            });
            if (response.ok) {
                let msg = await response.json();
                console.log(`Message from vendor: "${msg}"`);
            } if (response.error){
                console.error(`Error sending payload to vendor:
                ${response.error}`);
            }
        }
        res.json("Vendor Informed");
    });
}
```

Fig 5.19 - Inform vendor function.

Testing

The testing strategy improved with the use of a testing table this sprint as most server functions needed retesting after the database update. Improving the testing strategy was helpful for tracking what didn't work for each test case and allowed errors to be corrected quicker than before. This sprint also included some tests of the previous style.

Table 10 shows the most significant tests for each feature. (For full testing table see Appendix I).

Table 9
Most Significant Tests From DS3

Feature - VL					
ID	Test	Input	Expected Output	Actual Output	Notes

VL-03	Vendor List should show data types each vendor has access to.	Payload { name: "ThisIsATest" , datatype: "address", data: "Test, Address, , , " }	Vendor list should display the data types the vendor has linked to them. In the case of vendor “ThisIsATest” the linked data should show “address, card”	Expected	
Feature - UDL					
RD-03	Vendor sends data to pdm.	Payload {name: “ThisIsATest, datatype: “card”, data: “12345,123,1234”}	Vendor name and id is inserted into vendor_info table and the data, datatype and generated data id is inserted into the data_buffer	Server crashed trying to split linked data. Correct data had been inserted into the database however.	This was due to the getVendorList function not being edited yet. Looking at the database however the data has been entered.
DV-05	When a new piece of data is verified in the manager it should appear in the user data list with the vendor it is linked to.	Verifying data { name: "Vendor1", datatype: "card", data: "NewCardDetails, 10/25, 092" }	When the verify button is clicked the data should appear within the user data list with one linked vendor named “Vendor1”	Expected	
UDL-03	When a piece of data is deleted from the user data list it should disappear from any vendor that it	Deleting { datatype: "card", data: "NewCardDetails, 10/25, 092" }	When the data is deleted “address” should be removed from “Vendor1”	Expected	When only one piece of data is linked to the vendor and is deleted the vendor still appears in the list.

	is linked to in the vendor list				
Feature - ED and DD					
ED-01	When enter is pressed after a piece of data has been edited the data id and new data should be sent to the server.	Payload {“id: 70918528, newData: editedCardDetails, 11/25, 092”}	Server should receive the correct id and newData	Expected	
DD-01	The user should be able to delete data from the manager using the delete button.	Given: The user has at least one item of data within the user data table. When: The user clicks the delete button on the manager. Then: The data should be deleted from the manager and user data table.		Test Passed	
Feature - Inform Vendor					
IV-01	When inform vendor function is called the vendor at the given url	Sending payload {source:”192.168.0.90:8080”, data “1234, 123, 12”,	The test vendor server should receive the info, log it in its own console then	Invalid url error from the fetch function within the inform vendor	This may be because the url is incomplete, I will add “http://” to the start of the URL.

	should receive the data then respond with a confirmation message	decision:“DELETE”} to url “192.168.0.90:65431”	send “Data Received” back to the manager server.	function	
IV-01	^	Sending payload {source:”192.168.0.90:8080”, data “1234, 123, 12”, decision:“DELETE”} to url “http://192.168.0.90:65431”	^	Vendor server is receiving undefined data	The vendorURL is created and stored within the server database so it can be automated to always include the “http://”. The undefined data issue was due to express.json function not being included in vendor recv function.
IV-04	Testing vendor confirmation for delete decision	Sending delete request	Vendor should respond after 3000 ms with the data that should be deleted.	Expected	I would like to change this to send and receive the data id as this will decrease the amount of data being sent
IV-05	Vendor should send data id instead of vendor name when confirming request	Sending edit request	Vendor should respond after 3000 ms with the data id to be edited	Expected	I am going to change the addToBuffer function to include an optional parameter for data id so it can be searched.

UDL-03 helped identify that when a vendor has no data linked to it, they will still appear inside the VL which was updated soon after. Without a test this specific this wouldn't have been found until much further in development.

DD-01 was conducted using the old testing method. Compared to test ED-01 which uses the improved one the detail in the input and output of the test is less than desired for troubleshooting.

IV-05 identified an update required for one of the server functions that allowed data with an existing ID to be inserted into the data buffer. This was essential to buffering decisions in progress and would have taken longer to identify without this test.

Verification and Validation

Table 10 Validation Table For VL Component		
Feature: VL Component - Display data types for data instead of IDs on the VL.		
Requirement	Acceptance criteria	Achieved
I-09	Information relating to vendors, data or data requests must be visible on the respective display.	Yes
I-10	Every aspect of user data should be visible. This can be accepted in the form of detailed charts.	Yes

With the updated component for the VL the feature now achieves all of the associated requirements. Data clarity is supported by displaying data types instead of IDs, making relationships between vendors and data more cohesive.

The implementation can be improved: each time the VL function triggers, it generates n+1 fetch requests (where n is the number of vendor instances) whereas the UDL, which is structured similarly, only generates one.

Table 11 Validation Table for UDL		
Feature: UDL		
Requirement	Acceptance criteria	Achieved
I-02	All data stored on the manager must be seen on the interface.	Yes
I-09	Information relating to vendors, data or data requests must be visible on the respective display.	Yes
I-10	Every aspect of user data should be visible. This can be accepted in the form of detailed charts.	Yes
I-11	Every piece of data should have a marker for where it is being stored.	Yes

The UDL has been achieved to design matching the wireframe that was created in the design stage (Fig 5.21) and has shown how to improve some of the previous functions of a similar type.

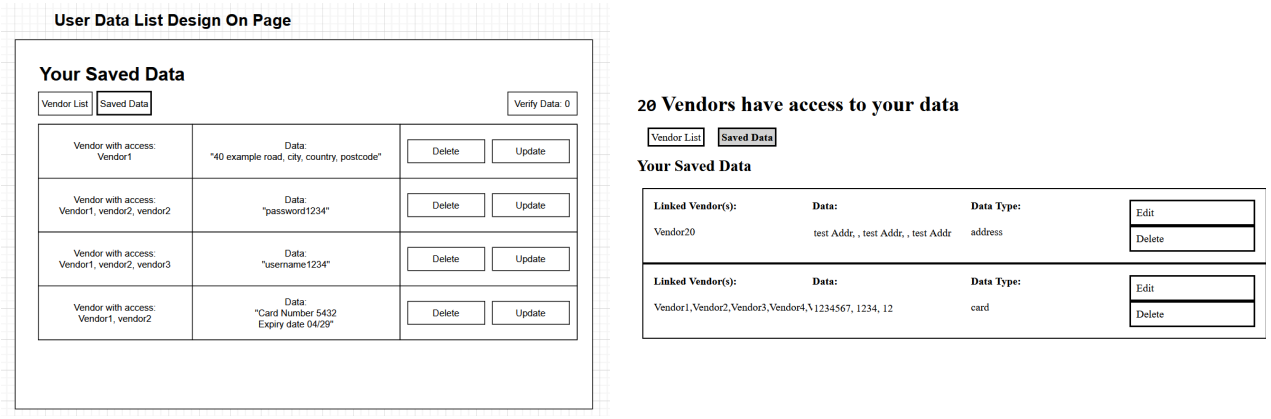


Fig 5.20 - Wireframe of the user data list (Left) compared to the implemented version (Right).

Table 12 Validation Table for ED and DD		
Feature: ED and DD Component - Create a means for informing a vendor about a decision.		
Requirement	Acceptance criteria	Achieved
E-01	When a user wants to make a decision it must not require too many inputs.	Yes
E-02	Data erasure requests must be available at minimum.	Yes
I-05	Consequences of a decision must be visible on the interface before being enacted.	Yes

E-01 was satisfied by both the Edit and Delete features, as users need no more than three actions to initiate a decision.

The Vendor Informing component was achieved using design two, ensuring the manager reflects data stored by the vendor, not the user's intent. This aligns with GDPR, which gives DCs up to a month to respond to a data request.

The application had grown to reveal bugs in the system that were not fatal to operation and needed to be logged.

1. When multiple instances of the same new data are verified, they are registered in the user_data table as separate IDs, even though they should be grouped under a single ID.

2. When a piece of data is edited the user may end up editing the data to one that already exists. This creates two instances of that data inside the user_data table when there should only be one.

5.5 DevSprint4 - DS4 (Appendix J)

Tasks

- Bug Fixing - Fix known bugs identified in DS3
  - Design
  - Implement solution
    - Test solution: D2UD-01, MD-01, MD-02, MD-03, MD-04, MD-05, MD-06, DV-06, ED-02
  - Evaluate designs
- Feature - Awaiting Data Tab - AD
  - Design
  - Implement feature
    - Test feature: AD-01, AD-02, IV-08, IV-09
  - Evaluate requirements: I-09, I-11
- Feature - Vendor Filter - VFIL
  - Design
  - Implement feature
    - Test feature: FIL-01,FIL-02,FIL03, FIL-04, FIL-05, FIL-06
  - Evaluate requirements: I-04, I-09, I-10
- Feature - User Data Filter - UDFIL
  - Design
  - Implement feature
    - Test feature: FIL-01,FIL-02,FIL03, FIL-04, FIL-05, FIL-06, FIL-07
  - Evaluate requirements: I-09, I-10

Design

The highest priority for DS4 was to fix the bugs identified in DS3. The first appeared when two pieces of the same data appeared in the buffer table. Once the user verifies each instance they exist inside the user data table under separate IDs.

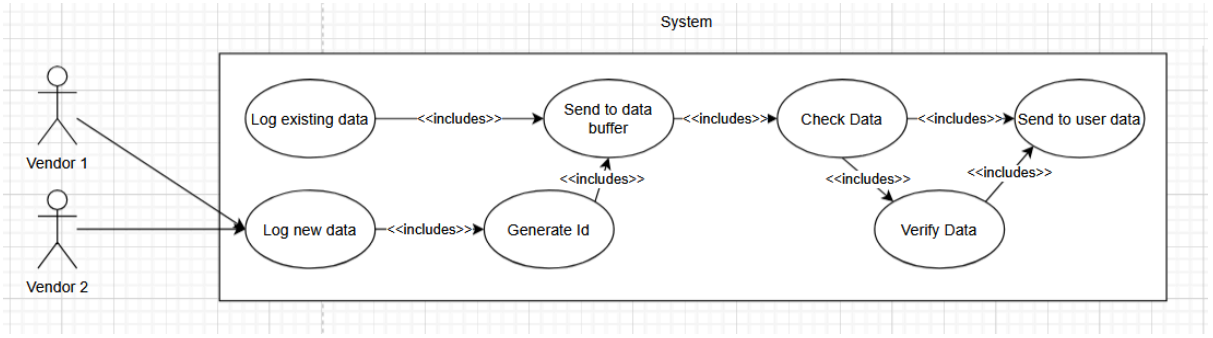


Fig 5.21 - Use case diagram showing the data verification flow.

The issue arises because the checkData function only checks the user data table not the buffer (Fig 5.21). One solution is to update the checkData function to check against both tables ensuring

any duplicates across the two are detected. Another would be to implement a merge function that leaves one instance of data (Fig 5.23).

The second bug emerges when a user edits an instance to match an existing one creating duplicate instances. As the second solution solves both issues it may be the better option.

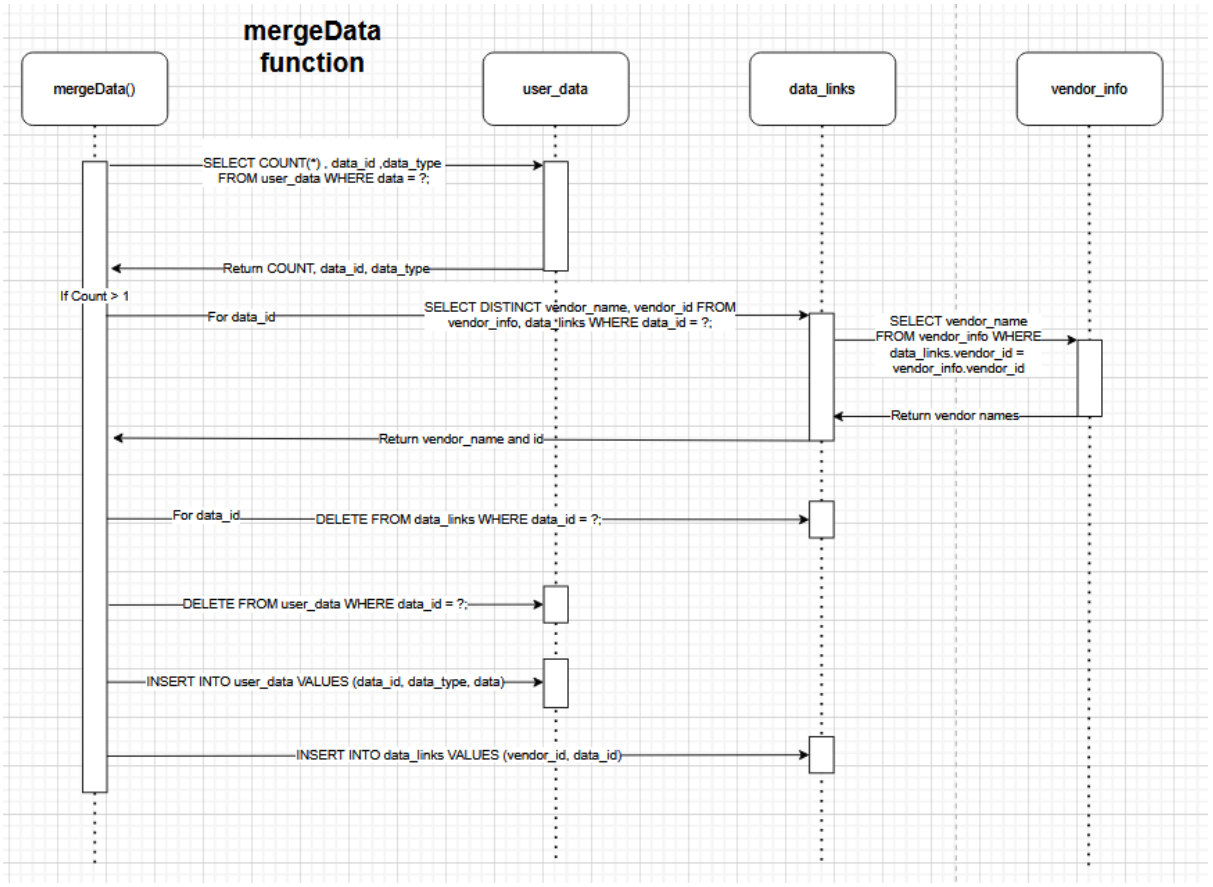


Fig 5.22 - Data flow diagram for the mergeData function.

The second priority was Feature AD which displays decisions being processed by vendors. As it is functionally similar to the Data Verification Tab it follows the same design.

Both filters function the same way but are applied to different lists therefore they share the same design. Two designs were considered.The first exists on the server and re-queries the database for data that matches the filter then rebuilds the list with new data. The second exists on the browser and hides list instances that don't match the filter. This design is quicker as it avoids server and database calls making it the preferred option.

Implementation

MergeData performs three functions. First, it gathers a count of all duplicate instances. If duplicates are found, it collects the IDs of each instance and collects their vendors into one array. Lastly, it deletes all instances and creates a new one with the aggregated data under one ID (Fig 5.24).

```
// Merges the linked vendors of two of the same data instance into one and
```

```
leaves only one instance of that data
async function mergeData(data){
    con.query('SELECT COUNT(*) as count FROM user_data WHERE data = ?;',
    [data], (err, result) => {
        if (err) throw err;
        console.log(result[0].count);
        if (result[0].count > 1){
            con.query('SELECT data_id, data_type FROM user_data WHERE data =
            ?;', [data], async (err, result) => {
                if (err) throw err;
                let vendors = [];
                for (const data of result){
                    let linkedVendors = await new Promise(async (resolve) =>
                    {
                       getLinkedVendorName(data.data_id, resolve);
                    });
                    for (const vendor of await linkedVendors){
                        vendors.push(vendor)
                    }
                    deleteData(data.data_id);
                };
                addDataToUserData(result[0].data_id, result[0].data_type,
                data);
                for (const vendor of vendors){
                    console.log(`Linking vendor ${vendor} to data ID
                    ${result[0].data_id}`);
                    linkData(vendor, result[0].data_id);
                }
            });
        }
    });
}
```

Fig 5.23 - mergeData function.

This is then applied in two places. At the end of the moveVerifiedData function, fixing verification bug, and at the end of the editData function fixing editing bug (Fig 5.25).

```
async function moveVerifiedData(req, res){
   . . .
        // create link to vendor and delete from buffer
        linkData(vendorName, dataID);
        deleteDataFromBuffer(dataID);
        mergeData(data);
   . . .
```

```
}

async function editData(dataId, newData){
      con.query('UPDATE user_data SET data = ? WHERE data_id = ?;', [newData,
      dataId], (err, result) => {
        if (err) throw err;
        console.log(`Data ID ${dataId} updated to ${newData}`);
      });
      mergeData(newData);
      deleteDataFromBuffer(dataId);
}
```

Fig 5.24 - Sections of code showing the mergeData function call locations.

Feature AD follows the same framework as the other lists within the application. A new field was added to the data buffer that specifies a purpose for the data. Data being processed will have “Awaiting Deletion” or “Awaiting Edit” in this field. Data awaiting verification will contain “Pending Verification”.

Both filters appear under one tab on the page. When each list item is built its attributes are stored within a dataset tag on the HTML element. These attributes include vendor, data type and data (Fig 5.26).

```
async function userDataList(){
  . . .
     const instance = listItem.querySelector('.userDataItem');
            instance.dataset.vendor = data.vendor_names;
            instance.dataset.data = data.data;
            instance.dataset.dataType = data.data_type;
  . . .
}
```

Fig 5.25 - Section of User Data List browser side function for assigning attributes to list instance.

The user can select an attribute and enter a filter string. On submission, the function checks each instance and hides ones that don't contain the filter string by adding a hidden class (Fig 5.27).

```
function filter(){
    removeFilters();
    const filterBy = document.querySelector('#filterBy').value;
    if (filterBy ==  'vendor'){
        const filterValue = document.querySelector('#filterInput').value;
        filterByVendor(filterValue);
    } else if (filterBy == 'dataType'){
        const filterValue = document.querySelector('#filterInput').value;
        filterByDataType(filterValue);
```

```
    } else if (filterBy == 'data'){
        const filterValue = document.querySelector('#filterInput').value;
        filterByData(filterValue);
    } else {
        removeFilters();
    }
}

function filterByVendor(vendor){
      const list = getCurrentList();
      const allInstances = list.querySelectorAll('.listItem')
      let count = allInstances.length;
      for (const instance of allInstances){
        if (!instance.dataset.vendor.includes(vendor)){
            instance.classList.add('hidden');
            count--;
        }
      }
      if (count === 0){
        removeFilters();
        alert('No items found for the given filter.');
      }
}
```

Fig 5.26 - Filter for User Data and Vendor List.

Testing

Throughout DS4 the improved testing strategy continued, this time splitting the table by each feature making it easier to see what tests applied to what feature. Tests were also split into unit and integration tests, which helped identify errors when combining feature components.

Table 13
Most Significant Tests From DS4

Bug Fix - Merge Data					
ID	Test	Input	Expected Output	Actual Output	Notes
D2UD-01	(Unit) addDataToUserData() should create an instance of the given data directly to the user_data	addDataToUserData(1234, test, testData)	An instance of (1234, test, testData) should appear inside the user_data table	Expected	

	table				
MD-05	(Integration) mergeData() should create a data link for every vendor using linkData()	mergeData(testingData, ,)	Two data links should be created for the two vendors to the same id.	SQL error too many connections	Adding connection end calls to functions that don't have them.
DV-06	(Integration) When a piece of data is verified from the data buffer that already exists in user data the two pieces should be merged	Verifying (testingData, ,)	Only one instance of the data should appear with a new link	MergeData() is called before data is put into user_data so data is not merged	Placed mergeData() lower down in the function which worked.
ED-02	(Integration) When a piece of data is edited to equal another they should be merged	Editing (dataToEdit, ,) to (editedData, ,)	Vendor1 who is linked to dataToEdit should appear next to editedData after it has been edited. The dataToEdit instance should disappear from the list.	Expected	
Feature - AD					
AD-02	(Unit) getAwaitingData() should log a count off all of the data awaiting confirmation	Calling getAwaitingData()	The console should return a list of two items the first is a count of 1 and the second is the data item awaiting edit	Expected	
IV-08	(Integration) When a decision is requested by	Requesting edit on data (testData, ,)	The awaiting confirmation count should show 1 on	Expected	This doesnt update automatically when the decision has been confirmed.

	the user the getAwaitingData function should be called after the data has been put into the buffer		the webpage		This only happens after refresh.
IV-09	(Integration) When a decision is made a list item of the awaiting data should be shown on the webpage	Requesting edit on data (testDataEdited, ,)	A list item should appear under the awaiting data button showing the information of the data awaiting confirmation including what decision it's waiting for.	No list item appears	The template hadn't been defined
Feature - VFIL and UDFIL					
FIL-01	(Unit) When building the list instances on both lists each instance should contain a dataset tag with the vendor name(s), data type and data associated with it.	Calling vendorList() and userDataList( )	All instances should show a data tags with vendors, data type and data within the element viewer on browser	Expected	
FIL-04	(Unit) When filter by vendor is selected and applied all instances that don't contain the name inputted on the text field	Appling vendor 1 when vendor is selected	All vendor instances except for vendor 1 should be hidden	Uncaught TypeError: instance.dataset.vendor.contains is not a function	Changed contains to includes. Hidden class has been added to the correct instances however there has been no effect. Hidden class is overwritten by

	should be hidden				listItem class. Added !important to hidden css to fix
FIL-05	(Unit) When a filter value is entered that doesn't exist the user should be alerted	Applying vendor4 when vendor is selected	A pop up saying there are no matches to the filter should appear	Expected	

The inclusion of integration tests allowed more complex issues to be identified. Test MD-05 revealed an error with excessive SQL connections due to each function creating a new one unnecessarily. If not conducted the cause of this would not have been identified.
Similarly, test IV-08 identified inconsistent application behaviour as some functions require a page refresh to see changes where others don't. Overall this testing method allowed a more iterative approach to developing features which in turn made them more reliable.

Verification and Validation

Table 14
Validation Table For AD

Feature: AD		
Requirement	Acceptance criteria	Achieved
I-09	Information relating to vendors, data or data requests must be visible on the respective display.	Yes
I-11	Every piece of data should have a marker for where it is being stored.	Yes

Feature AD was necessary for ensuring data clarity while enacting decisions. The requirement emerged when Design Two for Feature IV was implemented, as users needed a way to see decisions in progress.

Table 15
Validation Table For VFIL and UDFIL

Feature: VFIL and UDFIL		
Requirement	Acceptance criteria	Achieved
I-04	The user can filter vendors based on access time, data in	No

	use and time last used.	
I-09	Information relating to vendors, data or data requests must be visible on the respective display.	Yes
I-10	Every aspect of user data should be visible. This can be accepted in the form of detailed charts.	Yes

The addition of filters to the Vendor and User Data lists allows the user to refine the displayed information, assisting data comprehension; however, requirement I-04 has not been achieved as the user can not filter based on the attributes stated in the acceptance criteria.

All tasks within this sprint were achieved to design. Fixing the bugs identified in DS3 has allowed the application to become more robust which will make further development easier. Although robustness is not part of the requirements it does benefit the application as a whole.

Summary

With the completion of DS4 the application has reached its final form within the scope of this project. Each sprint the program evolved from basic foundations to a functional data manager that can acquire user-submitted data from a DC, inform users on where their data is stored, and help them enact data decisions from a central place.

The Vendor List allows users to see all vendors with access to their data, with options to filter vendor and data type. The UDL displays all data held by each vendor, where they can make rectification or erasure requests as well as filter by vendor, data type and specific data. To ensure data integrity, any new data must be verified by the user from the Data Verification Tab before it is stored.

Chapter 6: Evaluation

Introduction

To evaluate if the software makes DSs better DCs a requirements traceability matrix (RTM) (Appendix K) was created. This maps features to their associated requirements and tests. Features that meet the associated requirement are marked in green, ones that only partially meet the requirement, in yellow and ones that failed, in red. A requirement is considered achieved If over half of associated implementations achieve the acceptance criteria. This evaluation is limited due to lack of user testing. As a result, whether the features meet the user-specified requirements is based on the developers opinion not the actual users.

Discussion Of Evaluation

In total nine out of the fourteen requirements were met. The failed requirements include: I-01, I-05, I-06, I-07 and I-12.

Requirement I-01 failed as no features were developed to target it due to a shift in the application's focus. I-01 was defined in the early design stage while the survey was being created. During this time the intended design had the DC pull data from the application. When the survey had concluded the participants, supported by the literature review, indicated that management of data was more important than the storage. This changed the design to have the DC push data to the manager leading to the requirement falling out of focus.

The remaining failed requirements were due to features IC and DR not being implemented. During the design stage these features were medium and low priority as the participants of the survey ranked them low on the list of features. Due to lack of development time and low priority these features had to be cut from implementation.

Despite this the developed software achieved a 64% pass rate due to the implemented features and extensive testing. The three project objectives: Acquire, Inform and Enact have been met. The software can Acquire data from a DC allowing DSs to see their data in a central place using Features DMID, VL and UDL. The DS is informed on statistics about their data (Feature VC) and assisted in informed decision-making using Features VL and UDL. Finally, the DS can manage their data in a central place by enacting their rights to erasure and rectification directly from the application.

Lastly, although not a specific requirement, throughout the development of the application a conscious effort was made to keep the software, specifically the storage, decentralised. This means that for every instance of the manager that is installed the storage for the application is completely local to that instance. This supports future developments for expanding the application but also protects against wide scale data breaches for this software.

Looking back to the existing solutions discussed in chapter 2, PDV by Mun et al. (2010) was the closest to addressing all three issues with data comprehension, awareness of impact and monitoring. However its implementation lacked explanation in data comprehension. In comparison, the application developed in this project addresses data comprehension and monitoring but does not help the user with awareness of impact, despite it being designed. With the inclusion of a fifth DevSprint, the application could address all three issues becoming the superior solution by these metrics.

Despite this, all discussed implementations in chapter 2 are considered to make Data Subject better Data Controllers. By matching the number of addressed issues the developed application can be considered similarly.

Chapter 7: Conclusion

6.1 Introduction

As the software has been evaluated as successful in achieving the project aim it is important to consider how it can be taken further. Within this section, reflection on the project itself will also be discussed such as the risks, mitigation and effectiveness of project management.

6.2 Risks

Most of the risks identified at the start of the project (Appendix L) never materialised which meant the mitigation for them was successful. One risk that was realised however was lack of contact with the supervisor specifically during the mid-December to late January period. The mitigation for this risk was to plan regular meetings with the supervisor and develop in smaller sections so that if progress needed to be reset there were no major setbacks. The project had already been planned in three week sprints which meant the mitigation was already in place and because of this the project progress did slow down slightly however not to the degree it could have.

6.3 Project Management

Throughout the course of the project tasks and sprints have been managed using an agile methodology with the help of JIRA. This was crucial for submitting deliverables on time and allowed for the steady and consistent growth of the software.

During the project initiation stage a gantt chart was created to plan key stages and tasks of the project with the expectation of this changing as the project became more complex (See Appendix B). To monitor the progress of this another gantt chart was created at the midpoint of the project (See Appendix D) which logged the actual time frame of each task and the projected time to complete the remaining tasks.

When the midpoint chart was completed, the projected task completion dates closely matched the actual dates they were submitted. This was due to the strict and consistent three week sprint rotation that was in place since the start of the project. During each sprint the use of JIRAs kanban board and backlog meant that not only were the intentions of each sprint fulfilled, each task was completed in a timely manner.

For each task a story point estimate was made for how long it may take. The following scale was used.

Table 16
Story Point Scale

Story Points	Estimated Time
0.5	4 Hours
1	8 Hours / One Day
2	16 Hours / Two Days

Any tasks estimated to take more than two days were split into smaller tasks. Overall a total of 43.5 story points across 67 tasks were delivered.

Looking at the burndown charts produced at the end of the first sprint and the end of the penultimate sprint the tasks were completed in a more controlled and timely manner with more tasks being delivered overall as the team got used to the project management strategy (Fig 6.1).

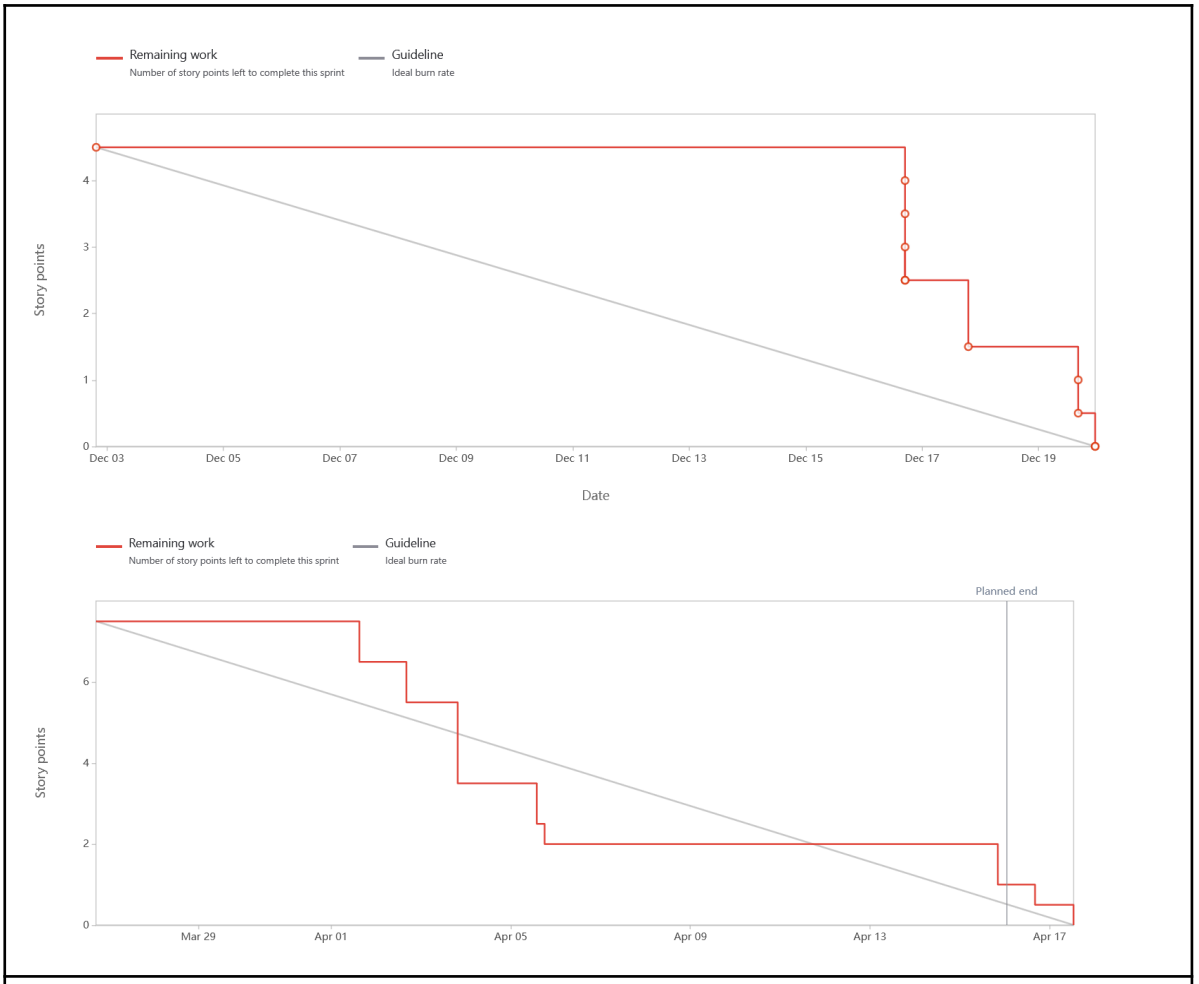


Fig 6.1 - Sprint burndown chart for Sprint 1 (Top) compared to Sprint 6 (Bottom).

Despite this success, some improvements to the management strategy are necessary in order to more accurately track tasks. One crucial problem was the team's failure in updating the Kanban board on task completion. This meant some tasks were reported complete after the sprint completion even if they were achieved on time causing sprint timeframes to become misaligned even if they were completed on time. Future projects could benefit from an automated logging system or an appointed auditor responsible for progress monitoring.

6.4 Recommendations

Recommendations based on the software

Although the aim of the project was achieved there is room for improvement. One recommendation is that the ID system for acquiring data must be made more secure either through masking the host IP or designing a different system altogether. This would make the software more viable for real-word use.

Second, there should be more mediums for informing the user about their data. This could be in the form of graphs or detailed reports however the information should be clear and easy to understand for any user.

Third, more decisions should be available via the application. The current decisions are powerful in themselves however the application can reach its full potential when it extends to rights such as restriction of processing, restriction of automated processing and for more nuanced decisions than erasure or rectification.

Finally, the data acquired from a vendor should be extended to include information about how the data will be processed and used. This can be used to facilitate more decisions which were discussed previously.

Recommendations for future projects

Future projects of user centric applications should allow time for user testing. During the evaluation of requirements within this project it was difficult to assess if some of the user elicited requirements were met as it was not assessed by the users themselves.

Making Data Subjects better Data Controllers should be approached with consideration of varying solutions. Although this paper briefly mentions implementations in the blockchain ecosystem such as the solution proposed by Mishra and Levkowitz (2021) these are not explored in-depth. Consideration of the solutions discussed in this paper, alongside other approaches may create a superior implementation than any single solution.

Lastly, data control has two main aspects. The first is control over where data is stored, including the individual's ability to manipulate and access the medium. The second is data monitoring, where the user can interpret how much data they have and who is using it. A successful application to giving individuals control will have full consideration of both aspects. The developed solution heavily focuses on the data monitoring; however, in the future focus on the storage aspect can be given.

6.5 Conclusion

This paper concludes that only giving Data Subjects the rights of GDPR but not a method to productively enact those rights is ineffective if data privacy is taking the path of individual control. The developed application creates the groundwork for an effective solution to this problem through acquiring data, informing on that data and enacting decisions, but should be taken further for it to be fully realised.

For the application to meet its full potential it should inform the Data Subject on how their data is being used as well as the consequences of making specific decisions. In future developments more consideration into the storage of data should be made as this is the second major aspect of data control.

References

1. Balkin, J. M. (2016). Information fiduciaries and the First Amendment. *SSRN*. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2675270
2. European Commission. (2023). *Data protection explained*. European Commission. https://commission.europa.eu/law/law-topic/data-protection/data-protection-explained_en
3. Gentles, J., Fields, M., Goodman, G., & Bhunia, S. (2022). Breaking the vault: A case study of the 2022 LastPass data breach. *ArXiv.org*. https://arxiv.org/abs/2502.04287
4. General Data Protection Regulation (GDPR). (2018). *General Data Protection Regulation (GDPR)*. https://gdpr-info.eu/
5. Lázaro, C., & Le Métayer, D. (2015). Control over personal data: True remedy or fairy tale? *HeinOnline*. https://heinonline.org/HOL/Page?collection=journals&handle=hein.journals/scripted12&id=3&men_tab=srchresults#
6. LGPD. (2018). *LGPD Brazil - General Personal Data Protection Act*. https://lgpd-brazil.info/
7. Mun, M., Hao, S., Mishra, N. K., Shilton, K., Burke, J., Estrin, D., Hansen, M., & Govindan, R. (2010). Personal data vaults: A locus of control for personal data streams. https://doi.org/10.1145/1921168.1921191
8. Prince, C. (2018). Do consumers want to control their personal data? Empirical evidence. *International Journal of Human-Computer Studies, 110*, 21–32. https://doi.org/10.1016/j.ijhcs.2017.10.003
9. Ramón Cáceres, Cox, L. P., Lim, H., Shakimov, A., & Varshavsky, A. (2009). Virtual individual servers as privacy-preserving proxies for mobile devices. https://doi.org/10.1145/1592606.1592616
10. Sambra, A., Mansour, E., Hawke, S., Zereba, M., Greco, N., Ghanem, A., Zagidulin, D., Aboulnaga, A., & Berners-Lee, T. (2016). *Solid: A platform for decentralized social applications based on linked data*. http://emansour.com/research/meccano/solid_protocols.pdf
11. Spadafora, A. (2024, December 17). Millions stolen from LastPass users in massive attack — what you need to know. *Tom's Guide*. https://www.tomsguide.com/computing/password-managers/millions-stolen-from-lastpass-users-in-massive-hack-attack-what-you-need-to-know
12. Statista. (2023). *Smart home - number of households in the segment smart home in the world 2025*. Statista. https://www.statista.com/forecasts/887613/number-of-smart-homes-in-the-smart-home-market-in-the-world

13. Taylor, P. (2024). *Data created worldwide 2010-2025*. Statista. https://www.statista.com/statistics/871513/worldwide-data-created/
14. Twingate. (2024, May 24). *What happened in the LastPass data breach?* Twingate. https://www.twingate.com/blog/tips/lastpass-data-breach
15. Wilson, N., & Reid, A. (2024). Data Controllers as data fiduciaries: Theory, definitions & burdens of proof. *University of Colorado Law Review, 95*(1), 175–218. https://heinonline.org/HOL/Page?collection=journals&handle=hein.journals/ucollr95&id=189&men_tab=srchresults
16. Wolters, P. (2018). The control by and rights of the Data Subject under the GDPR. *Journal of Internet Law, 22*(1), 7–18. https://repository.ubn.ru.nl/bitstream/handle/2066/194516/194516pub.pdf
17. Kumar, N. (2024, October 31). *27 Website Statistics (2025) — Traffic Data & Trends*.
18. DemandSage. https://www.demandsage.com/website-statistics/
19. Shingledecker, R. (2013, March 3). Tiny Core Linux, Micro Core Linux, 12MB Linux GUI Desktop, Live, Frugal, Extendable. Tinycorelinux.net. http://tinycorelinux.net/
20. Canonical Group Ltd. (2025, April 29). Ubuntu Server tutorial. Ubuntu Server. https://documentation.ubuntu.com/server/tutorial/
21. Node.js. (2024). About. Node.js. https://nodejs.org/en/about
22. OpenJS Foundation. (2025, March 31). Express - Node.js web application framework. Expressjs.com. https://expressjs.com/
23. Mozilla Developer Network. (2025, April 28). Using Fetch. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
24. Atlassian. (2024). Jira. Atlassian. https://www.atlassian.com/software/jira
25. de Montjoye, Y.-A., Shmueli, E., Wang, S. S., & Pentland, A. S. (2014). openPDS: Protecting the Privacy of Metadata through SafeAnswers. PLoS ONE, 9(7), e98790. https://doi.org/10.1371/journal.pone.0098790
26. HAT Project Research Team. (2015). HAT Briefing Paper 2 : The Hub-of-all-Things (HAT) economic model of the multi-sided market platform and ecosystem [PDF].
27. Hummel, P., Braun, M., & Dabrock, P. (2020). Own Data? Ethical Reflections on Data Ownership. Philosophy & Technology, 34, 545–572. https://doi.org/10.1007/s13347-020-00404-9
28. Nulab. (2025). Backlog | Project Management Software for Virtual Teams. Nulab. https://nulab.com/backlog/
29. PRINCE2. (2017). What Is PRINCE2? The Definition, History & Benefits | UK. Prince2.com. https://www.prince2.com/uk/prince2-methodology

30. University of Portsmouth. (2025). Research ethics. University of Portsmouth. https://www.port.ac.uk/research/our-research-and-innovation-culture/research-governance-and-ethics/research-ethics
31. van Ooijen, I., & Vrabec, H. U. (2018). Does the GDPR Enhance Consumers' Control over Personal Data? An Analysis from a Behavioural Perspective. Journal of Consumer Policy, 42(1), 91–107. https://doi.org/10.1007/s10603-018-9399-7
32. Sim, J., Kim, B., Jeon, K., Joo, M., Lim, J., Lee, J., & Choo, K.-K. R. (2022). Technical Requirements and Approaches in Personal Data Control. ACM Computing Surveys. https://doi.org/10.1145/3558766
33. Ausloos, J., & Dewitte, P. (2018). Shattering one-way mirrors – data subject access rights in practice . International Data Privacy Law, 8(1), 4–28. https://doi.org/10.1093/idpl/ipy001
34. Fallatah, K. U., Barhamgi, M., & Perera, C. (2023). Personal Data Stores (PDS): A Review. Sensors, 23(3), 1477. https://doi.org/10.3390/s23031477
35. Janssen, H., Cobbe, J., Norval , C., & Singh , J. (2020). Decentralized data processing: personal data stores and the GDPR . International Data Privacy Law, 10(4). https://doi.org/10.1093/idpl/ipaa016
36. Al-Abdullah, M., Alsmadi, I., AlAbdullah, R., & Farkas, B. (2020). Designing privacy-friendly data repositories: a framework for a blockchain that follows the GDPR. Digital Policy, Regulation and Governance, 22(5/6), 389–411. https://doi.org/10.1108/dprg-04-2020-0050
37. Khalid, M. I., Ahmed, M., Helfert, M., & Kim, J. (2023). Privacy-First Paradigm for Dynamic Consent Management Systems: Empowering Data Subjects through Decentralized Data Controllers and Privacy-Preserving Techniques. Electronics, 12(24), 4973. https://doi.org/10.3390/electronics12244973
38. Mishra, N., & Haim Levkowitz. (2021). PDV: Permissioned Blockchain based Personal Data Vault using Predictive Prefetching. BIOTC '21: Proceedings of the 2021 3rd Blockchain and Internet of Things Conference. https://doi.org/10.1145/3475992.3476001
39. Oracle. "MySQL :: MySQL 8.4 Release Notes :: Changes in MySQL 8.4.5 (2025-04-15, LTS Release)." Mysql.com, 2025, dev.mysql.com/doc/relnotes/mysql/8.4/en/news-8-4-5.html.

Appendices

Appendix A: Project Initiation Document

Project Initiation Document

Project Details:

Project Title:	*How users can take back control of their data using PDV*
Student:	*Harrison Robert Morley*
Course:	*Cyber Security and Forensic Computing*
Project code:	*PJE40*
Supervisor:	*David Williams*
Date:	*10/24*

Declarations (Please tick)

✓	I give permission for this document to be made available to other students as examples of previous work. (optional).
✓	I confirm that I have read and understood the University Rules in respect of plagiarism and student misconduct.
✓	I declare that this work is entirely my own. Each quotation or contribution cited from other work is fully referenced.

1. Client / Target Audience

This project is aimed at the wider public as it is an effort to create a more decentralised storage medium for personal data.

2. Degree suitability

This project will require knowledge of cryptography, programming, data security and some knowledge of the file system all of which are taught across Cyber Security and Forensic

Computing. Modules like Programming, Web Programming and Cryptography all help with the software engineering and cryptography side of things and modules such as forensics fundamentals and ethical hacking will help with data security and the file system.

3. The project environment and problem to be solved

Currently the vast majority of organisations store their data in huge data warehouses or databases which allow for a single point of failure for data breaches and creates a large target for hackers. Even worse if these do fail to keep threat actors out masses of data is compromised and hundreds possibly thousands of users are affected. This project aims to create a decentralised medium for storage of personal data where users store their data themselves rather than trusting an organisation to do it for them. This also creates multiple points of failure instead of just one where if a single point is breached only a small amount of data is revealed rather than the data of potentially thousands of people.

There are some approaches to this already proposed however this project focuses on just one called personal data vaults which is the idea that users store their data in mediums that they themselves own either in their home or on their personal devices that only authorised users or services have access to.

4. Project aim and objectives

Aim:
Create a decentralised storage medium for users so they can control their data and make better privacy decisions.

Create a management system to enable users to make more informed privacy decisions and make it easier for them to enact those decisions.
Objectives:

- Educate the public on how their data is currently handled and how this can be improved
- Make clear the implications of certain data privacy decisions
- Create a medium that users can use to store their personal data
- Allow users to make privacy decisions on data leaving their personal data vault
- Make a step toward a widely accepted decentralised storage medium for users

5. Project constraints

- I am the only person developing this application which means I will have to think of and implement everything myself.
- There is no external funding for this project so any frameworks/tools I use will have to be free.
- This is quite a large project to undertake in the timeframe we have as well as having to work around my other modules and deadlines.
- The level of technical knowledge I need to undertake this project effectively which I am currently not at.

6. Facilities and resources

- Laptop/PC - Both are available to me at all times.
- GitHub - This will allow me to store my work in the cloud allowing me to develop seamlessly across my PC and my laptop.
- My personal supervisor - My supervisor is incredibly knowledgeable on the software and cryptography aspect of my project however he has limited availability as he is a lecturer within the university.
- Google Drive and Google Facilities

7. Log of risks

No	Description	Likelihood (high, medium, low)	Impact	Mitigation/Avoidance
1	Laptop Failure	Low	Will cause possible data loss and slow down the development process.	Back up work to physical media and Github.
2	PC Failure	Low	Will cause possible data loss and a large slow down to the development process.	Back up work to physical media and Github.
3	Github goes down	Low	Will remove access to the cloud version of the project.	Keep a physical version of the project at all times.
4	Scope of planned project becomes too large	Medium	Will hider progress significantly or may cause a project reset	Plan everything thoroughly before attempting the project itself.
5	Ill health	Medium	Depending on the severity this could cause a minor delay in work flow or pause the project all together for a period.	Make sure to look after my body by getting regular sleep and nutrition.

6	Unable to contact my supervisor	Medium	Lack of feedback may mean my work strays off track.	Make sure when planning to make plan short term sprints rather than long periods of work so it is easier to reverse or change something if it goes off track.
7	Functionality at the end of the programming stage is not testable or minimum viable product	Medium	This means I wont have as many deliverables as planned.	Make sure to plan a reasonable MVP that can be completed relatively easily and meet some of my aims.
8				

8. Project deliverables

- A Personal Data Vault Application
- Evaluation report from each dev sprint evaluation
- List of technical requirements/features
- Final Report
- Literature review

9. Project approach

I plan to largely use the PRINCE2 agile project management scheme to manage my project with some minor changes to fit my workflow and project a bit better. For the project planning and initiation stages I will be using the regular PRINCE2 methodology as these parts in my opinion do not need breaking down into smaller tasks and sprints which will also help me keep my planning consistent and continuous. When moving into the delivery management stages I will then be using the agile method by splitting my application into small deliverable features which will come together as a minimum viable product at the end of each sprint. Within each sprint I will be applying a software development cycle as if each one is an application itself therefore at the end of it having some form of deliverable. The next sprint therefore is adding to the previous application so it slowly grows throughout the lifecycle of the project. Essentially this project will be treated as a series of smaller projects within the Deliverable management stages applying a waterfall software development cycle at each sprint. As each sprint follows an SDLC there will be some form of evaluation at the end of each stage which will be kept in a report for logging.

Before I did anything on this project I collected a table of research papers related to my research problem which I can refer to in my literature review and when starting my project. This has allowed me to see what solutions have been proposed to similar issues and how they have been implemented as well as what the general conversation around this topic is. These were found using google scholar then forward and backward searching through the papers that I had found which allowed me to get a collection of 20 papers relating to my project.

10. Project tasks and timescales

No	Stage	Dates	Main Tasks
1	PID Document	19/10/24 - 25/10/24	Finish PID document.
2	Initiation Plan	28/10/24 - 06/12/24	Review Existing Technical Papers, Create List of Features, Create Outline for MVP, Plan Dev Sprints, Complete Ethics Form.
3	Lit Review	02/12/24 - 31/12/24	Write Lit Review.
4	Dev Sprint 1	06/01/25 - 31/01/25	Basic Storage Implementation, Basic UI Implementation, Evaluate Current Features and Alter Next Sprint With New Context.
5	Dev Sprint 2	03/02/25 - 28/02/25	Develop Encryption Diagram, Implement Encryption into Storage, Continue UI Development, Evaluate Current Features.
6	Dev Sprint 3	03/03/25 - 04/04/25	Develop Privacy Features, Finish Developments Towards MVP, Evaluate MVP Against Original MVP Spec.
7	Report Write Up For MVP	07/04/25 - 02/05/25	Collect Planned Technical Spec, Collect Evaluations From Dev Stages, Write Up Technical Documentation for MVP, Assess How MVP Solution Satisfies Problem, Suggest How Project Can Be Taken Further, Create Priority List for Extra Features.
8	Dev Sprint 4	05/05/25 - 16/05/25	Develop Priority Feature 1, Evaluate Feature On How It Improves MVP.
9	Dev Sprint 5	19/05/25 - 30/05/25	Develop Priority Feature 2, Evaluate Feature On How It Improves MVP.
10	Conclude Project Report	01/06/25 - 18/06/25	Add Extra Feature Evaluations To MVP Report, Re-Evaluate How The Application Satisfies Problem, Collect Deliverables, Submit.

11. Supervisor meetings

My supervisor and I have agreed on a 3 week schedule for regular meetings across the year. First is a 1-1 meeting to discuss specific issues relating to my project. The second is a group meeting with the other students that my tutor supervises where we can all discuss aspect of our project and

get general advice from our supervisor and the third is a group drop in where we can ask any questions we like about our project and listen to the answers of other people questions incase they can also help us.

12. Legal, ethical, professional, social issues

Legal:

- As this project is dealing with users personal information it must be handled with care under GDPR.

Ethical:

- Because this is possibly disruptive this could cause people to lose jobs with data storage departments within companies

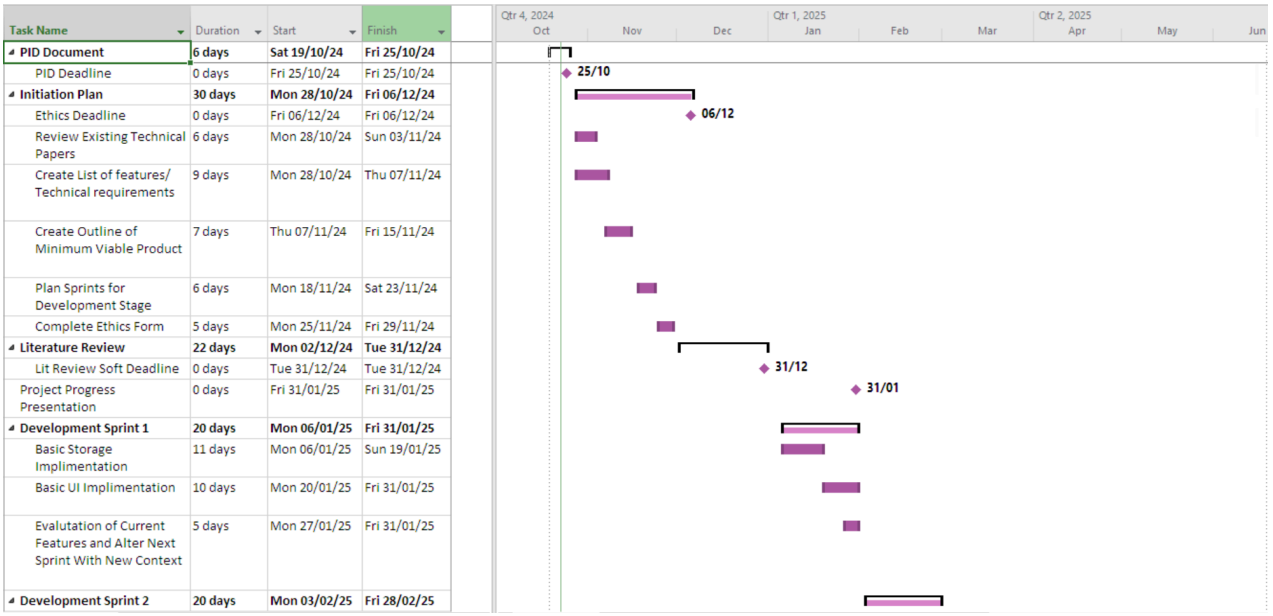
Professional:

- This has the potential to cause a major shift in the way companies operate and handle personal data as well as the general public which could cause whole departments and jobs to shrink.

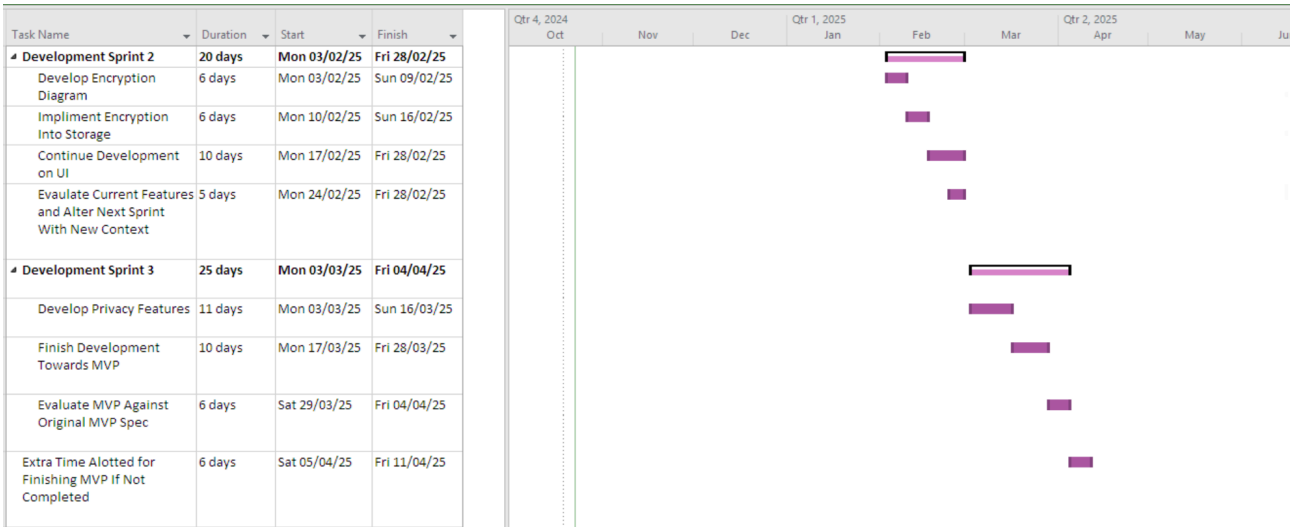
Social (if any):

- Shifting perception to users in control of their data rather than companies in charge of lots of data can shift blame when data breaches happen and can cause less sympathy on an individual level to victims when not as many people are affected.

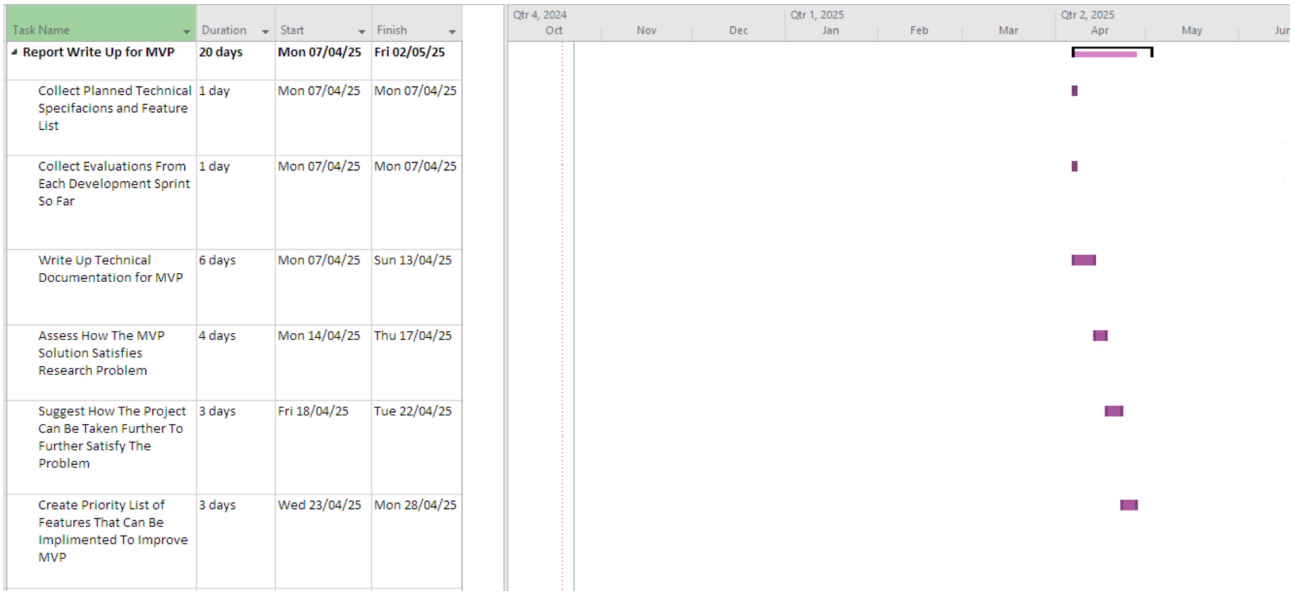
Appendix B: Start Of Project Gantt chart



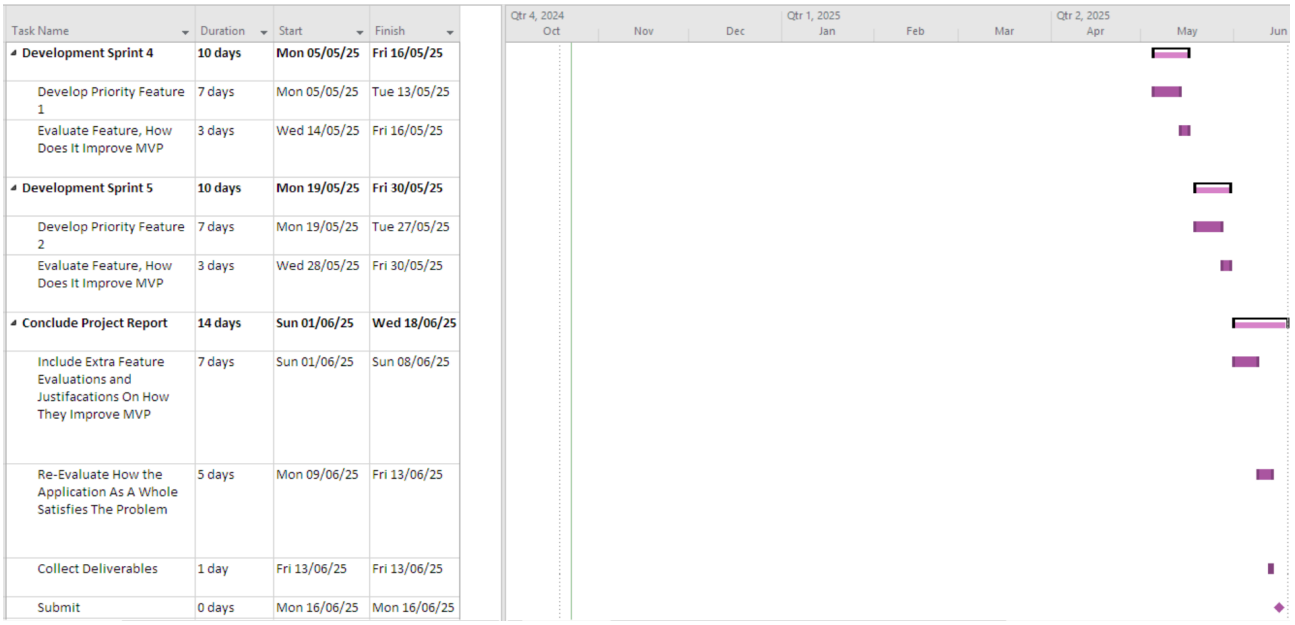
Task Name	Duration	Start	Finish
PID Document	**6 days**	**Sat 19/10/24**	**Fri 25/10/24**
PID Deadline	0 days	Fri 25/10/24	Fri 25/10/24
Initiation Plan	**30 days**	**Mon 28/10/24**	**Fri 06/12/24**
Ethics Deadline	0 days	Fri 06/12/24	Fri 06/12/24
Review Existing Technical Papers	6 days	Mon 28/10/24	Sun 03/11/24
Create List of features/ Technical requirements	9 days	Mon 28/10/24	Thu 07/11/24
Create Outline of Minimum Viable Product	7 days	Thu 07/11/24	Fri 15/11/24
Plan Sprints for Development Stage	6 days	Mon 18/11/24	Sat 23/11/24
Complete Ethics Form	5 days	Mon 25/11/24	Fri 29/11/24
Literature Review	**22 days**	**Mon 02/12/24**	**Tue 31/12/24**
Lit Review Soft Deadline	0 days	Tue 31/12/24	Tue 31/12/24
Project Progress Presentation	0 days	Fri 31/01/25	Fri 31/01/25
Development Sprint 1	**20 days**	**Mon 06/01/25**	**Fri 31/01/25**
Basic Storage Implimentation	11 days	Mon 06/01/25	Sun 19/01/25
Basic UI Implimentation	10 days	Mon 20/01/25	Fri 31/01/25
Evalutation of Current Features and Alter Next Sprint With New Context	5 days	Mon 27/01/25	Fri 31/01/25
Development Sprint 2	**20 days**	**Mon 03/02/25**	**Fri 28/02/25**



Task Name	Duration	Start	Finish
Development Sprint 2	**20 days**	**Mon 03/02/25**	**Fri 28/02/25**
Develop Encryption Diagram	6 days	Mon 03/02/25	Sun 09/02/25
Impliment Encryption Into Storage	6 days	Mon 10/02/25	Sun 16/02/25
Continue Development on UI	10 days	Mon 17/02/25	Fri 28/02/25
Evaulate Current Features and Alter Next Sprint With New Context	5 days	Mon 24/02/25	Fri 28/02/25
Development Sprint 3	**25 days**	**Mon 03/03/25**	**Fri 04/04/25**
Develop Privacy Features	11 days	Mon 03/03/25	Sun 16/03/25
Finish Development Towards MVP	10 days	Mon 17/03/25	Fri 28/03/25
Evaluate MVP Against Original MVP Spec	6 days	Sat 29/03/25	Fri 04/04/25
Extra Time Alotted for Finishing MVP If Not Completed	6 days	Sat 05/04/25	Fri 11/04/25



Task Name	Duration	Start	Finish
Report Write Up for MVP	**20 days**	**Mon 07/04/25**	**Fri 02/05/25**
Collect Planned Technical Specifacions and Feature List	1 day	Mon 07/04/25	Mon 07/04/25
Collect Evaluations From Each Development Sprint So Far	1 day	Mon 07/04/25	Mon 07/04/25
Write Up Technical Documentation for MVP	6 days	Mon 07/04/25	Sun 13/04/25
Assess How The MVP Solution Satisfies Research Problem	4 days	Mon 14/04/25	Thu 17/04/25
Suggest How The Project Can Be Taken Further To Further Satisfy The Problem	3 days	Fri 18/04/25	Tue 22/04/25
Create Priority List of Features That Can Be Implimented To Improve MVP	3 days	Wed 23/04/25	Mon 28/04/25



Task Name	Duration	Start	Finish
Development Sprint 4	**10 days**	**Mon 05/05/25**	**Fri 16/05/25**
Develop Priority Feature 1	7 days	Mon 05/05/25	Tue 13/05/25
Evaluate Feature, How Does It Improve MVP	3 days	Wed 14/05/25	Fri 16/05/25
Development Sprint 5	**10 days**	**Mon 19/05/25**	**Fri 30/05/25**
Develop Priority Feature 2	7 days	Mon 19/05/25	Tue 27/05/25
Evaluate Feature, How Does It Improve MVP	3 days	Wed 28/05/25	Fri 30/05/25
Conclude Project Report	**14 days**	**Sun 01/06/25**	**Wed 18/06/25**
Include Extra Feature Evaluations and Justifacations On How They Improve MVP	7 days	Sun 01/06/25	Sun 08/06/25
Re-Evaluate How the Application As A Whole Satisfies The Problem	5 days	Mon 09/06/25	Fri 13/06/25
Collect Deliverables	1 day	Fri 13/06/25	Fri 13/06/25
Submit	0 days	Mon 16/06/25	Mon 16/06/25

Appendix C : Ethics Certificate

Certificate of Ethics Review

Project title: How users can take control of their personal data using PDV.

Name:	Harrison Morley	**User ID:**	up2097531	**Application date:**	07/11/2024 13:34:54	**ER Number:**	TETHIC-2024-109660

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the **School of Computing** is/are **Elisavet Andrikopoulou, Kirsten Smith**

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

Which school/department do you belong to?: **School of Computing**
What is your primary role at the University?: **Undergraduate Student**
What is the name of the member of staff who is responsible for supervising your project?: **David Williams**
Is the study likely to involve human subjects (observation) or participants?: Yes
Will you gather data about people (e.g. socio-economic, clinical, psychological, biological)?: No
Will your data collection be strictly limited to gathering anonymous insights about a particular artefact or research question (e.g. opinions, feedback)?: Yes
Confirm whether and explain how you will use participant information sheets and apply informed consent.: I will be sending out a google form that requires the user to give consent before advancing to collect data, they will have access to the participant information sheet however most content of the sheet will be on the google form.
Confirm whether and explain how you will maintain participant anonymity and confidentiality of data collected: I am not collecting any identifying information as I will only be collecting desired requirements for my application.
Will the study involve National Health Service patients or staff?: No
Do human participants/subjects take part in studies without their knowledge/consent at the time, or will deception of any sort be involved? (e.g. covert observation of people, especially if in a non-public place): No
Will you collect or analyse personally identifiable information about anyone or monitor their communications or on-line activities without their explicit consent?: No
Does the study involve participants who are unable to give informed consent or are in a dependent position (e.g. children, people with learning disabilities, unconscious patients, Portsmouth University students)?: No
Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants?: No
Will blood or tissue samples be obtained from participants?: No
Is pain or more than mild discomfort likely to result from the study?: No
Could the study induce psychological stress or anxiety in participants or third parties?: No
Will the study involve prolonged or repetitive testing?: No
Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No
Are there risks of significant damage to physical and/or ecological environmental features?: No
Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: No
Does the project involve animals in any way?: No
Could the research outputs potentially be harmful to third parties?: No
Could your research/artefact be adapted and be misused?: No

Will your project or project deliverables be relevant to defence, the military, police or other security organisations and/or in addition, could it be used by others to threaten UK security?: Yes
Please briefly explain any security implications that may arise from the project/deliverables.: This project does not introduce any security threats to any security organisation and cannot be used by third parties to introduce security threats however as a cyber security project cyber security organisations will be interested in the outcomes. The final outcome of my project has the potential to cause organisations to change their security policy although this will not happen at the point of project closure.
Please read and confirm that you agree with the following statements: I confirm that I have considered the implications for data collection and use, taking into consideration legal requirements (UK GDPR, Data Protection Act 2018 etc.), I confirm that I have considered the impact of this work and and taken any reasonable action to mitigate potential misuse of the project outputs, I confirm that I will act ethically and honestly throughout this project

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor comments: **The involvement of human participants is simply to elicit requirements and potentially evaluate the final artefact. I am satisfied that the documentation evidences appropriate ethical consideration and I trust you to address points of feedback prior to distribution regarding Uni contact details, ethics review reference number and details of the disadvantages and advantages of of participating.**

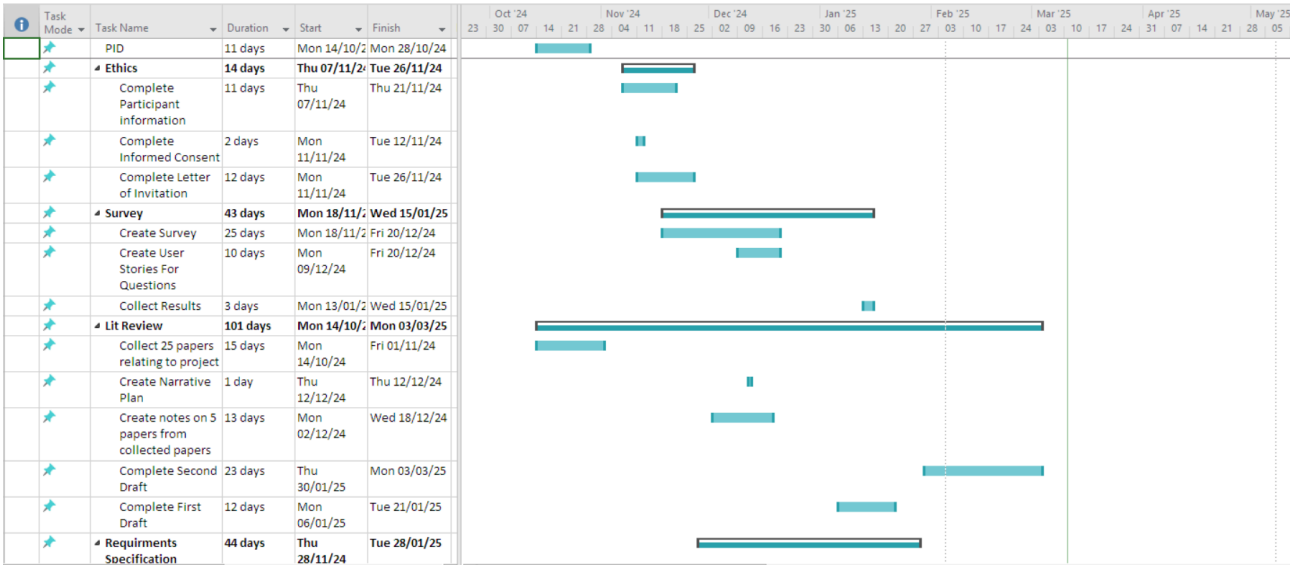
Supervisor's Digital Signature: **david.williams2@port.ac.uk** Date: **21/11/2024**

Faculty Ethics Committee Review

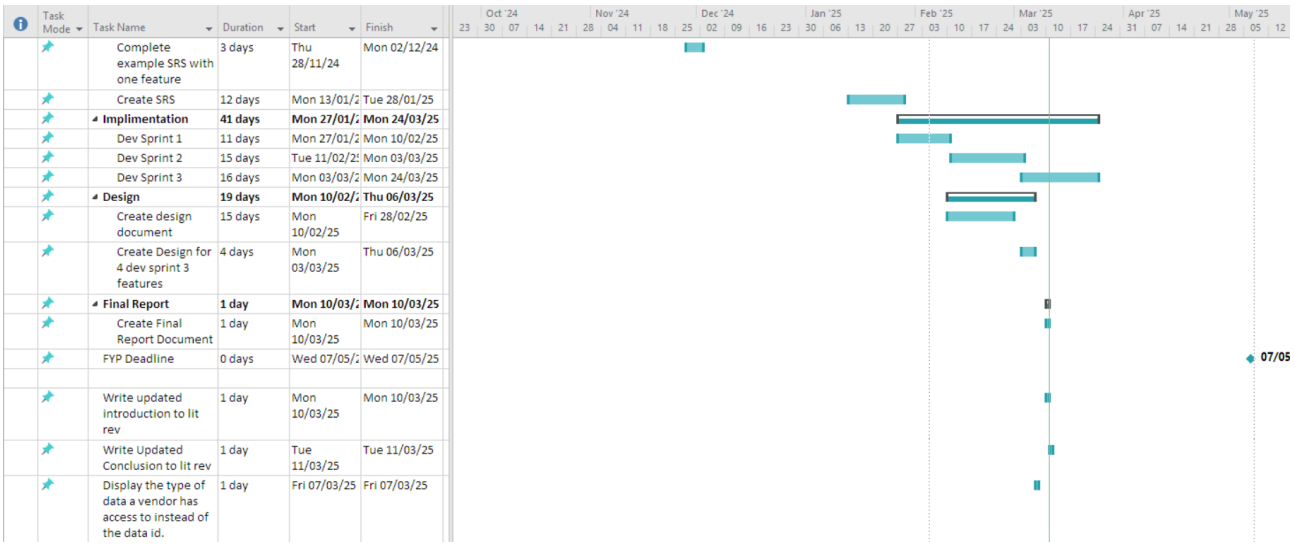
Ethics Rep comments:

Faculty Ethics Committee Member's Digital Signature(s): **kirsten.smith@port.ac.uk** Date: **21/11/2024**

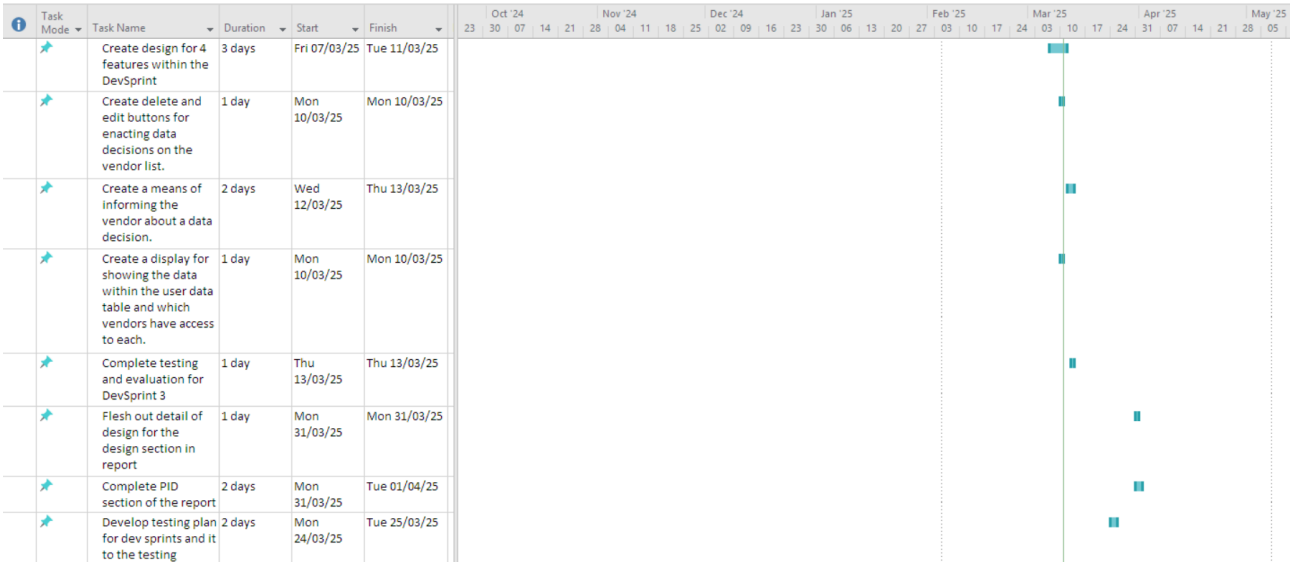
Appendix D: Mid Project Gantt Chart



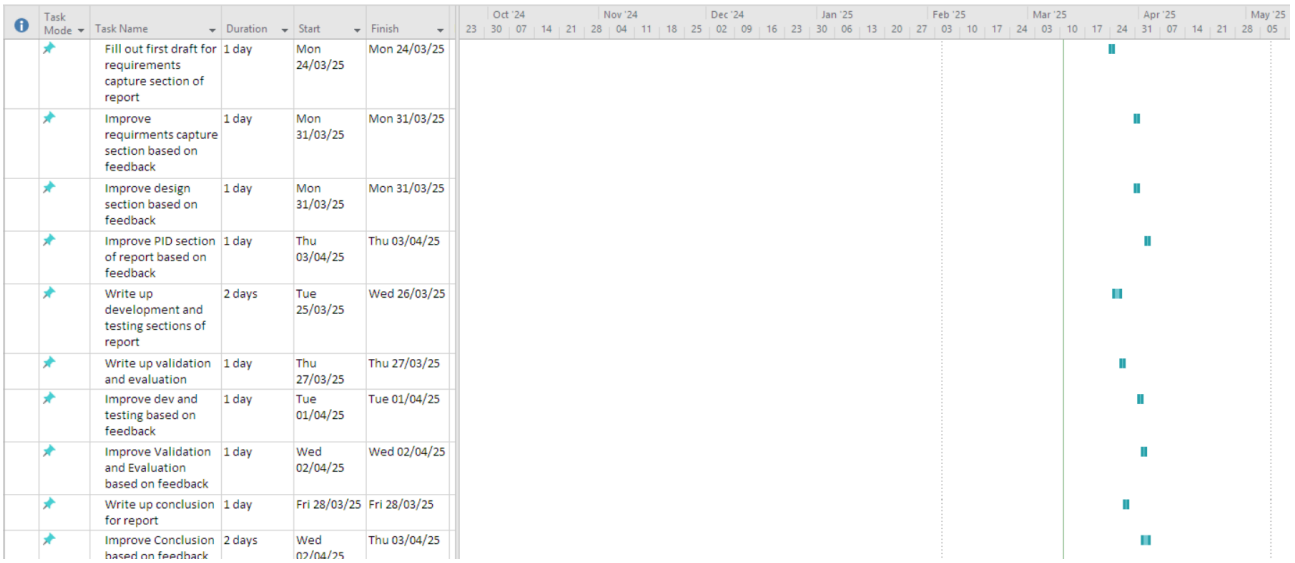
Task Mode	Task Name	Duration	Start	Finish
	PID	11 days	Mon 14/10/2	Mon 28/10/24
	Ethics	**14 days**	**Thu 07/11/2**	**Tue 26/11/24**
	Complete Participant information	11 days	Thu 07/11/24	Thu 21/11/24
	Complete Informed Consent	2 days	Mon 11/11/24	Tue 12/11/24
	Complete Letter of Invitation	12 days	Mon 11/11/24	Tue 26/11/24
	Survey	**43 days**	**Mon 18/11/2**	**Wed 15/01/25**
	Create Survey	25 days	Mon 18/11/2	Fri 20/12/24
	Create User Stories For Questions	10 days	Mon 09/12/24	Fri 20/12/24
	Collect Results	3 days	Mon 13/01/2	Wed 15/01/25
	Lit Review	**101 days**	**Mon 14/10/2**	**Mon 03/03/25**
	Collect 25 papers relating to project	15 days	Mon 14/10/24	Fri 01/11/24
	Create Narrative Plan	1 day	Thu 12/12/24	Thu 12/12/24
	Create notes on 5 papers from collected papers	13 days	Mon 02/12/24	Wed 18/12/24
	Complete Second Draft	23 days	Thu 30/01/25	Mon 03/03/25
	Complete First Draft	12 days	Mon 06/01/25	Tue 21/01/25
	Requirments Specification	**44 days**	**Thu 28/11/24**	**Tue 28/01/25**



Task Mode	Task Name	Duration	Start	Finish
	Complete example SRS with one feature	3 days	Thu 28/11/24	Mon 02/12/24
	Create SRS	12 days	Mon 13/01/2	Tue 28/01/25
	Implimentation	**41 days**	**Mon 27/01/2**	**Mon 24/03/25**
	Dev Sprint 1	11 days	Mon 27/01/2	Mon 10/02/25
	Dev Sprint 2	15 days	Tue 11/02/25	Mon 03/03/25
	Dev Sprint 3	16 days	Mon 03/03/2	Mon 24/03/25
	Design	**19 days**	**Mon 10/02/2**	**Thu 06/03/25**
	Create design document	15 days	Mon 10/02/25	Fri 28/02/25
	Create Design for 4 dev sprint 3 features	4 days	Mon 03/03/25	Thu 06/03/25
	Final Report	**1 day**	**Mon 10/03/2**	**Mon 10/03/25**
	Create Final Report Document	1 day	Mon 10/03/25	Mon 10/03/25
	FYP Deadline	0 days	Wed 07/05/2	Wed 07/05/25
	Write updated introduction to lit rev	1 day	Mon 10/03/25	Mon 10/03/25
	Write Updated Conclusion to lit rev	1 day	Tue 11/03/25	Tue 11/03/25
	Display the type of data a vendor has access to instead of the data id.	1 day	Fri 07/03/25	Fri 07/03/25



Task Mode	Task Name	Duration	Start	Finish
	Create design for 4 features within the DevSprint	3 days	Fri 07/03/25	Tue 11/03/25
	Create delete and edit buttons for enacting data decisions on the vendor list.	1 day	Mon 10/03/25	Mon 10/03/25
	Create a means of informing the vendor about a data decision.	2 days	Wed 12/03/25	Thu 13/03/25
	Create a display for showing the data within the user data table and which vendors have access to each.	1 day	Mon 10/03/25	Mon 10/03/25
	Complete testing and evaluation for DevSprint 3	1 day	Thu 13/03/25	Thu 13/03/25
	Flesh out detail of design for the design section in report	1 day	Mon 31/03/25	Mon 31/03/25
	Complete PID section of the report	2 days	Mon 31/03/25	Tue 01/04/25
	Develop testing plan for dev sprints and it to the testing	2 days	Mon 24/03/25	Tue 25/03/25



Task Mode	Task Name	Duration	Start	Finish
	Fill out first draft for requirements capture section of report	1 day	Mon 24/03/25	Mon 24/03/25
	Improve requirments capture section based on feedback	1 day	Mon 31/03/25	Mon 31/03/25
	Improve design section based on feedback	1 day	Mon 31/03/25	Mon 31/03/25
	Improve PID section of report based on feedback	1 day	Thu 03/04/25	Thu 03/04/25
	Write up development and testing sections of report	2 days	Tue 25/03/25	Wed 26/03/25
	Write up validation and evaluation	1 day	Thu 27/03/25	Thu 27/03/25
	Improve dev and testing based on feedback	1 day	Tue 01/04/25	Tue 01/04/25
	Improve Validation and Evaluation based on feedback	1 day	Wed 02/04/25	Wed 02/04/25
	Write up conclusion for report	1 day	Fri 28/03/25	Fri 28/03/25
	Improve Conclusion based on feedback	2 days	Wed 02/04/25	Thu 03/04/25

Appendix E: Survey

Possible Features For PDV

This form is for us to understand what features our target audience feel are most important for a personal data storage application.

up2097531@myport.ac.uk Switch accounts

Not shared

* Indicates required question

What We Would Like To Achieve

We are trying to create a personal data storage app called a "Personal Data Vault" (PDV) so users can own and control their personal data instead of relying on third parties to do this for them. The idea is that users can store their data like a person would store their valuable items in a safe in their home giving them full control on what happens to their data and who has access to it. This has the added bonus of creating a decentralised method of storing our personal data, so that when a data breach happens it will affect a smaller set of people, as opposed to hundreds or thousands.

Informed Consent *

Before we collect any data we must acquire your consent.
Title of Project: How users can take control of their personal data using PDV.

Name and Contact Details of Researcher(s): Harrison Morley - UP2097531@myport.ac.uk

Name and Contact Details of Supervisor: David Williams - david.williams2@port.ac.uk

University Data Protection Officer: Samantha Hill, 023 9284 3642 or information-matters@port.ac.uk

Ethics Committee Reference Number: TETHIC-2024-109660

Participant Information: Click Here

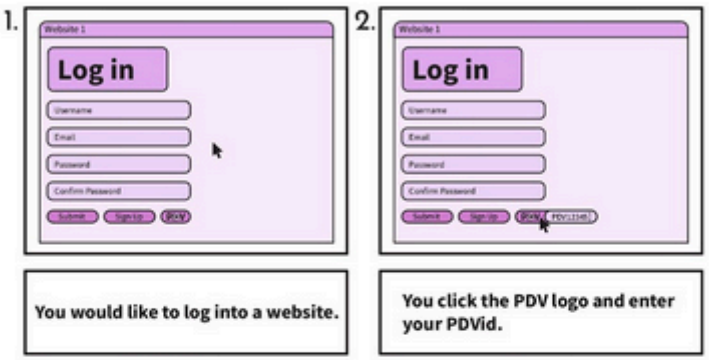
Letter Of Invitation: Click Here

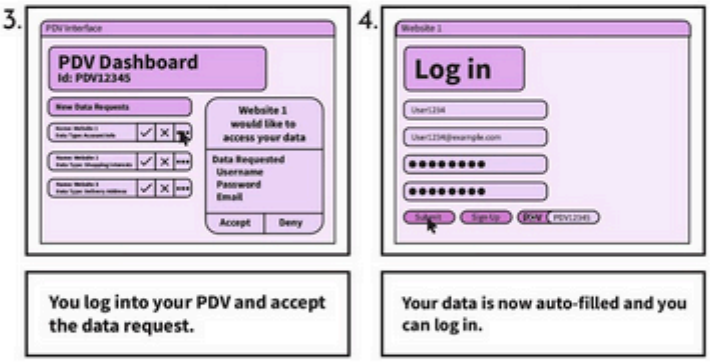
	Agreement
I confirm that I have read and understood the information sheet (version 2.9 dated November 2024) for the above study. I have had the opportunity to consider the information, ask questions and have had these answered satisfactorily.	☐
I understand that my participation is voluntary and that I am free to withdraw at any time without giving any reason until my answers are submitted.	☐
I understand that data collected during this study will be processed in accordance with data protection law as explained in the Participant Information Sheet.	☐
I am age 18 or above.	☐
I agree to take part in the above study.	☐

User Stories

Here are a few features of our app, and how they could be used. Let us know what you think about each of them.

User Story One





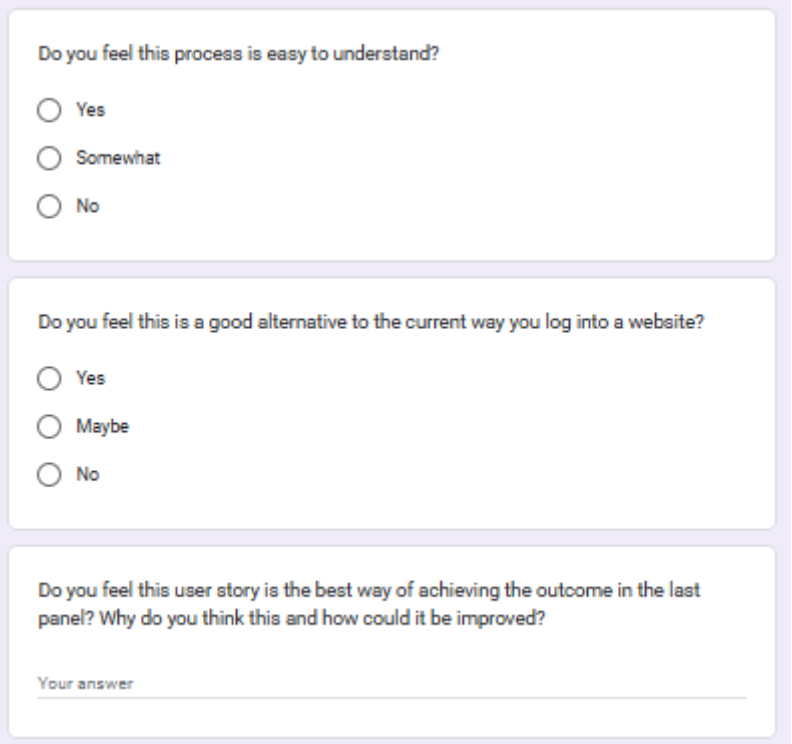
How important do you feel this feature is for our app?

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

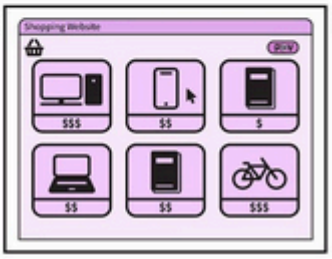
How often do you feel you would use your PDV in this way?

- ◯ Never
- ◯ Occasionally
- ◯ Often
- ◯ Always


Do you feel this process is easy to understand?
Yes
Somewhat
No
Do you feel this is a good alternative to the current way you log into a website?
Yes
Maybe
No
Do you feel this user story is the best way of achieving the outcome in the last panel? Why do you think this and how could it be improved?
Your answer

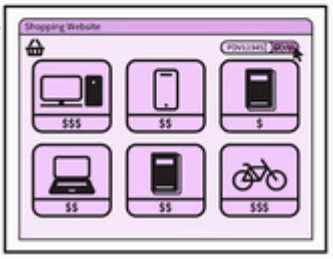
User Story Two

1.



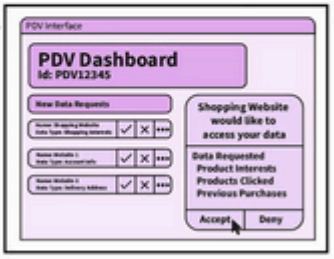
You are scrolling though an online shopping site.

2.



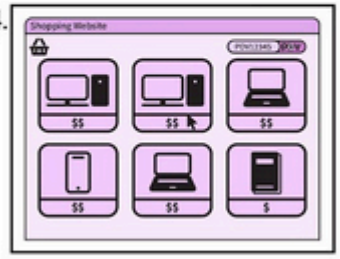
You click the PDV logo and enter you PDVid for reccomendations tailored to you.

3.



You log into your PDV and accept the data request.

4.



You now get better product recommnedations while scrolling.

How important do you feel this feature is for our app?

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

How often do you feel you would you use your PDV in this way?

- ◯ Never
- ◯ Occasionally
- ◯ Often
- ◯ Always

Do you feel this process is easy to understand?

- ◯ Yes
- ◯ Somewhat
- ◯ No

Do you feel this gives you more control over your targeted advertising and recommendations?

- ◯ Yes
- ◯ Maybe
- ◯ No

Do you feel this user story is the best way of achieving the outcome in the last panel? Why do you think this and how could it be improved?

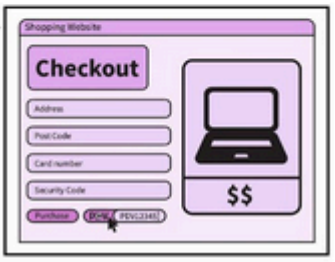
Your answer

User Story Three

1.

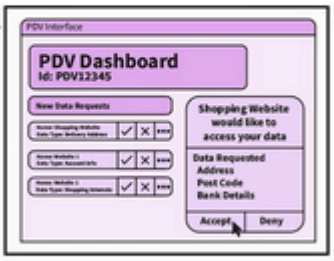
You are buying a product online.

2.



You click the PDV logo and enter your PDVid.

3.



You log into your PDV and accept the data request.

4.

Your data is now auto-filled and you can continue your purchase.

How important do you feel this feature is for our app?

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

How often do you feel you would you use your PDV in this way?

- ◯ Never
- ◯ Occasionally
- ◯ Often
- ◯ Always

Do you feel this process is easy to understand?

- ◯ Yes
- ◯ Somewhat
- ◯ No

Do you feel this process is quicker and easier than how you currently fill out delivery details?

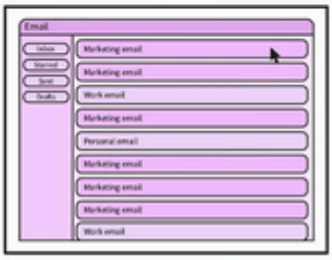
- ◯ Yes
- ◯ Maybe
- ◯ No

Do you feel this user story is the best way of achieving the outcome in the last panel? Why do you think this and how could it be improved?

Your answer

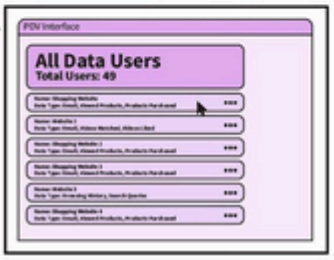
User Story Four

1.



You would like to reduce the amount of third parties who have access to your infomration.

2.

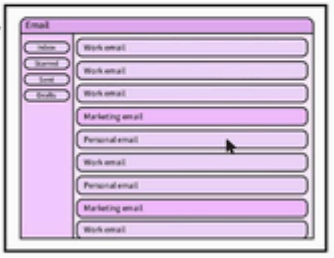


You log into your PDV and see your current data users.

3.

You stop access to some of the third parties with your infomation.

4.



Third parties will now have to re-request your data next time you use their services.

How important do you feel this feature is for our app?

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

How often do you feel you would use your PDV in this way?

- ◯ Never
- ◯ Occasionally
- ◯ Often
- ◯ Always

Do you feel this process is easy to understand?

- ◯ Yes
- ◯ Somewhat
- ◯ No

Do you feel this gives you more control over data that you have already submitted to third parties?

- ◯ Yes
- ◯ Maybe
- ◯ No

Do you feel this user story is the best way of achieving the outcome in the last panel? Why do you think this and how could it be improved?

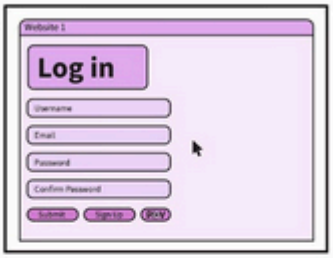
Your answer

Rank These Requirement

Here are a few requirements we've created for each of our user stories. Please rank how important you feel each one is.

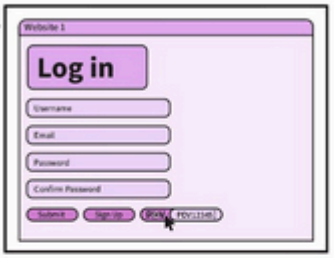
User Story One

1.



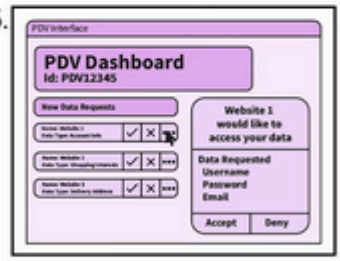
You would like to log into a website.

2.



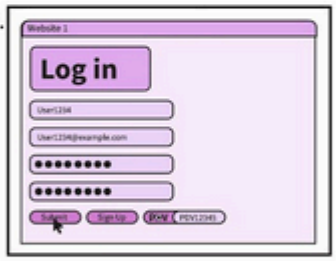
You click the PDV logo and enter your PDVid.

3.



You log into your PDV and accept the data request.

4.



Your data is now auto-filled and you can log in.

Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).

Your information is auto filled when you accept a data request.

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).

You can quickly accept or deny recent data requests on your PDV homepage.

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).

You can input an ID rather than an email and password to a third party for them to request your data.

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).

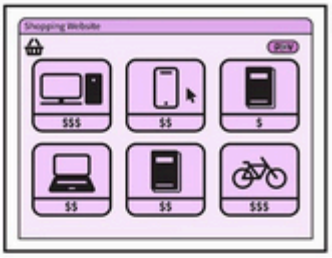
You can create custom groups of data types that you can choose from, such as: address information, username and password groups and whole accounts.

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

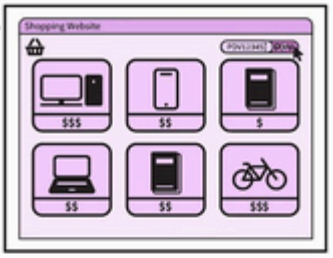
User Story Two

1.



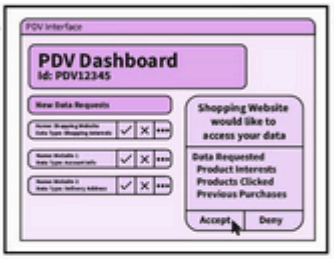
You are scrolling though an online shopping site.

2.



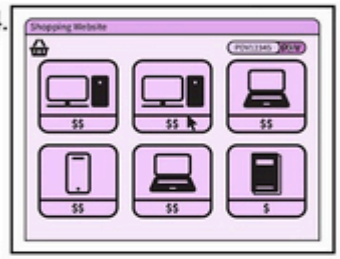
You click the PDV logo and enter you PDVid for reccomendations tailored to you.

3.



You log into your PDV and accept the data request.

4.



You now get better product recommnedations while scrolling.

Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).

You can choose to accept or deny specific types of requested data instead of having to approve or reject all of it at once.

1 2 3 4 5

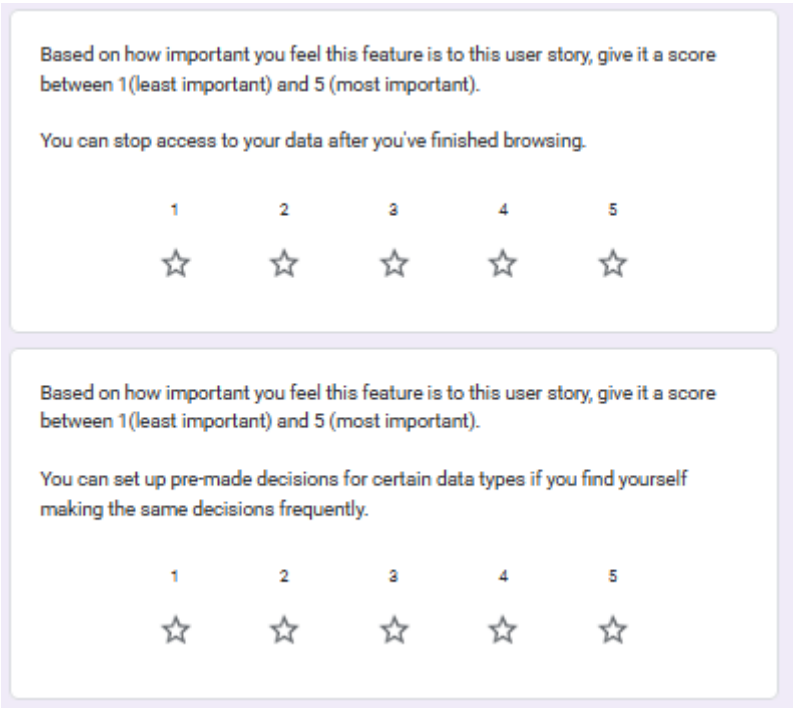
☆ ☆ ☆ ☆ ☆

Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).

You can see the data that has been collected before you make a decision on a request.

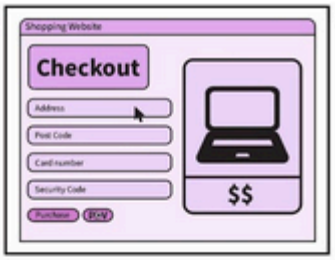
1 2 3 4 5

☆ ☆ ☆ ☆ ☆


Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).
You can stop access to your data after you've finished browsing.
1 2 3 4 5
Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).
You can set up pre-made decisions for certain data types if you find yourself making the same decisions frequently.
1 2 3 4 5

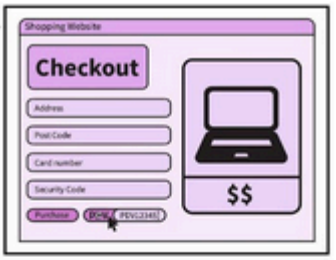
User Story Three

1.



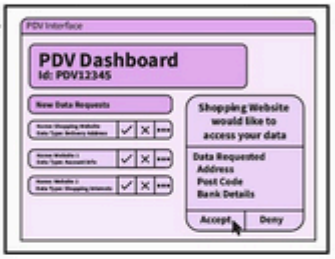
You are buying a product online.

2.



You click the PDV logo and enter your PDVid.

3.



You log into your PDV and accept the data request.

4.

Your data is now auto-filled and you can continue your purchase.

Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).

You can quickly accept or deny recent data requests on the PDV homepage.

1 2 3 4 5

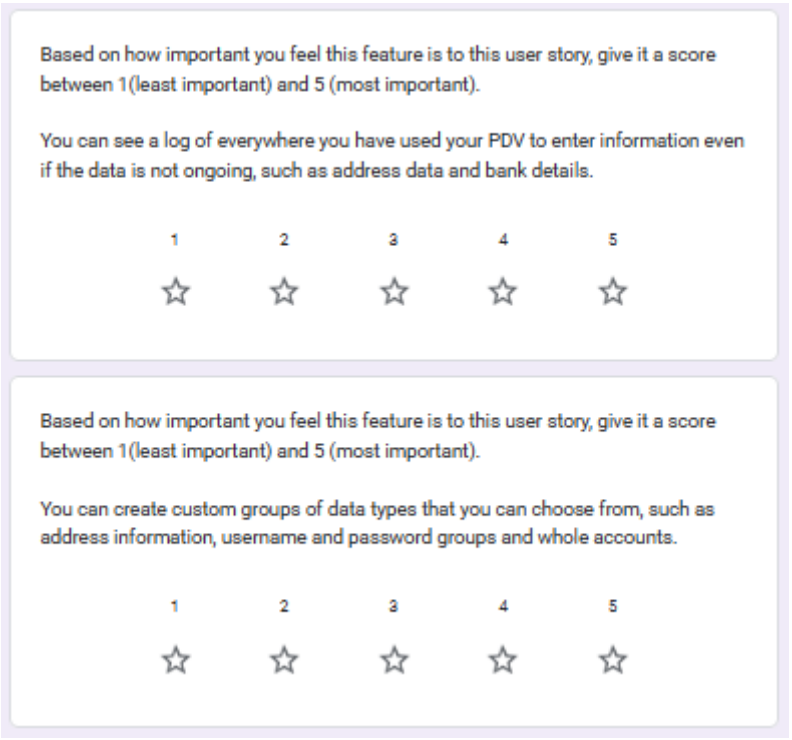
☆ ☆ ☆ ☆ ☆

Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).

You can have information partially filled out if you do not want certain information stored on your PDV such as card details.

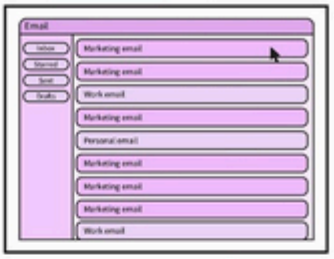
1 2 3 4 5

☆ ☆ ☆ ☆ ☆


Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).
You can see a log of everywhere you have used your PDV to enter information even if the data is not ongoing, such as address data and bank details.
1 2 3 4 5
Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).
You can create custom groups of data types that you can choose from, such as address information, username and password groups and whole accounts.
1 2 3 4 5

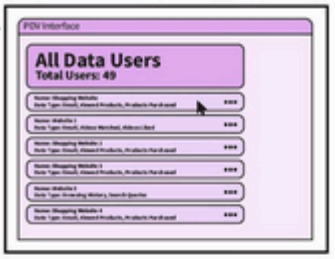
User Story Four

1.



You would like to reduce the amount of third parties who have access to your infomration.

2.

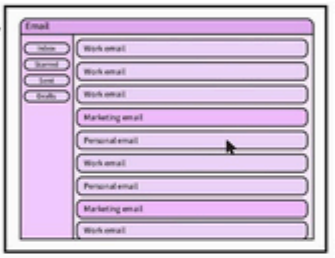


You log into your PDV and see your current data users.

3.

You stop access to some of the third parties with your infomation.

4.



Third parties will now have to re-request your data next time you use their services.

Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).

You can quickly see how many third parties have ongoing access to your data.

1 2 3 4 5

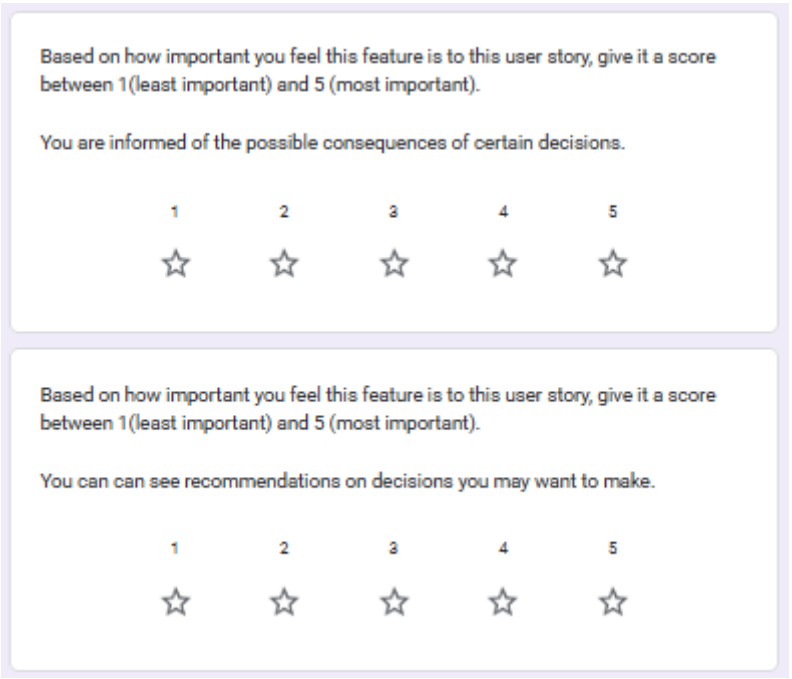
☆ ☆ ☆ ☆ ☆

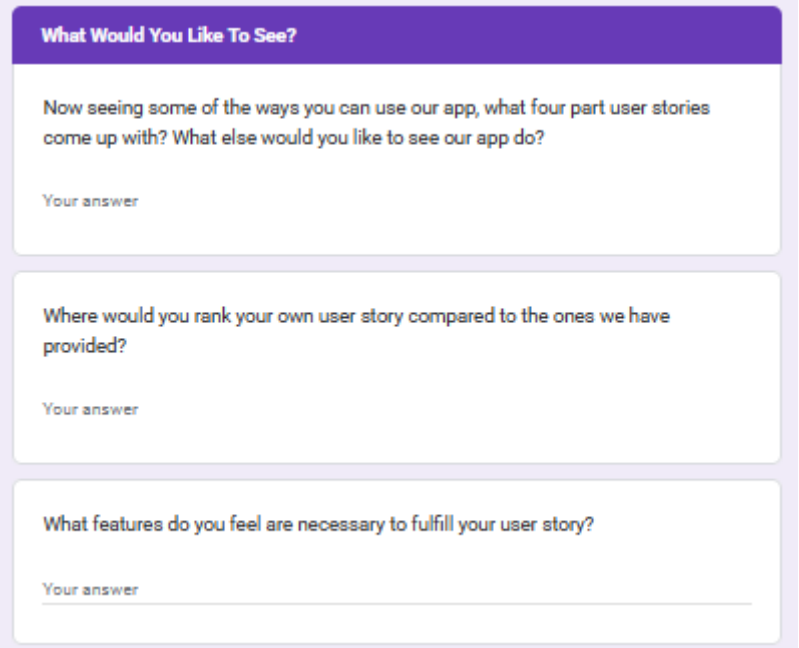
Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).

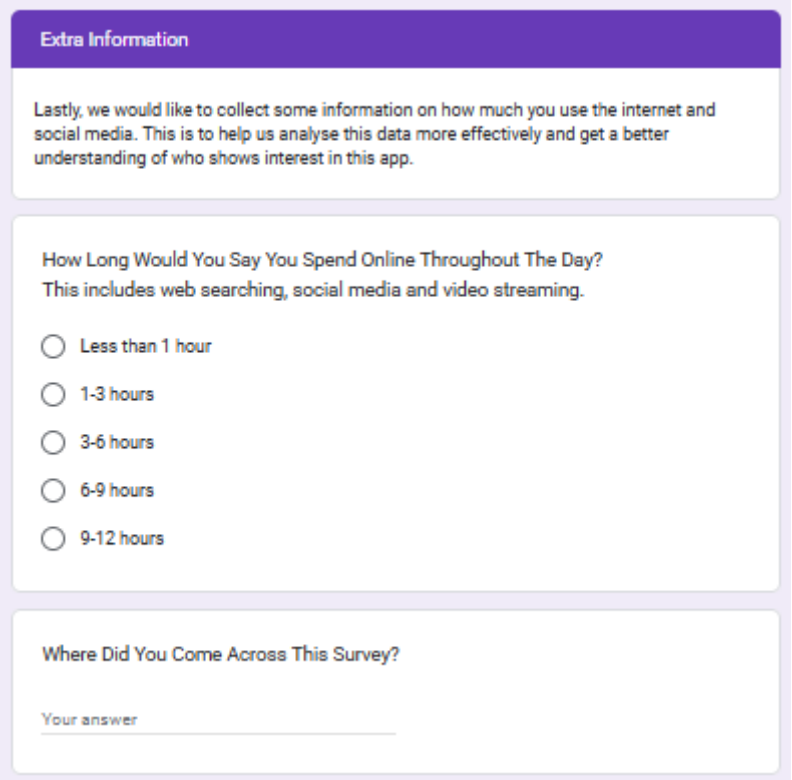
You can filter third parties based on how long they've had access to your data, what data they are using and when they last used your data.

1 2 3 4 5

☆ ☆ ☆ ☆ ☆


Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).
You are informed of the possible consequences of certain decisions.
1 2 3 4 5
Based on how important you feel this feature is to this user story, give it a score between 1(least important) and 5 (most important).
You can can see recommendations on decisions you may want to make.
1 2 3 4 5


What Would You Like To See?
Now seeing some of the ways you can use our app, what four part user stories come up with? What else would you like to see our app do?
Your answer
Where would you rank your own user story compared to the ones we have provided?
Your answer
What features do you feel are necessary to fulfill your user story?
Your answer



Appendix F: Software Requirements Specification

Software Requirements Specification

Project: Making Data Subjects better Data Controllers: Engineering a data manager for Data Subjects to track who holds what data.

Introduction

The outcome of this project is to create an application for users to store their personal data in a storage medium they own instead of relying on third parties to do this for them. The aims of this are

to reduce the amount of data leakage in data breaches and to give users more control over their data. This SRS describes the requirements for the application itself and not the project as a whole.

We have split the overall application into four “user stories” which describe a collection of features that will be included. We then asked participants in a survey to rank each user story based on what they feel is most important therefore this document follows that priority order.

User Story 4 - The Data Manager

Overall Description

This set of features describes a data manager that allows a user to log into the application and see what third parties are using what data. The data manager will show in total how many vendors have access to the users data and will allow the user to stop access.

Functional Requirements

The user must be able to:

- Submit an ID for their Data Manager to a vendor for a data request.
- Be able to see recent data requests and accept or deny them.
- See a collection of all vendors and their ongoing data access.
- See a collection of data that is held within the manager.
- Filter third parties based on how long they've had access to their data, what data they are using and when they last used it.
- See information about the consequences of denying a vendor access to their data.
- See recommendation on what decisions they could make.

Non-Functional Requirements

- All recommendations and information must be easy to understand.
- Decision making must be simple and streamlined and cannot require too many inputs.
- Key information such as the amount of vendors using data should be clearly displayed.

System Features

Must Haves

Feature 1 - Data Manager Id

Description

A system for identifying each unique data manager and a way to send data requests to that ID. This will be visible at all times in the data manager.

Stimulus and Response

There is no stimulus and response for this feature.

Feature 2 - Data Request System

Description
A system for sending and receiving data requests to the Data Manager from a vendor using the Data Manager Id system. This will use the HTTPS protocol for communication and JSON for the data itself.

Stimulus and Response
User submits Data Manager Id to a vendor → The vendor sends a request with the type of data they want to access, how often they want access, what the data will be used for and an authentication hash using the TLS protocol in the JSON format → The user sees the request on their Data Manager and accepts → The Data Manager sends the data using the JSON format using the TLS protocol to the vendor.

User submits Data Manager Id to a vendor → The vendor sends a request with the type of data they want to access, how often they want access, what the data will be used for and an authentication hash using the TLS protocol in the JSON format → The user sees the request on their Data Manager and denys → The Data Manager sends a denial message to the vendor using the TLS protocol.

Feature 3 - Vendor List
Description
Allows the user to see a list of all current data requests and third parties with ongoing data access. They should also be able to turn off ongoing data access for any vendor in the list as well as be able to accept or deny data requests.

Stimulus and Response
User logs into Data Manager account → A list of all data requests are displayed with a summary of what is being requested for how long with quick buttons to accept or deny the request → user clicks on the data request → A more detailed view of the data request is showing all information about the request with buttons allowing them to accept or deny the request.

User logs into Data Manager account → A list of all data requests are displayed with a summary of what is being requested for how long with quick buttons to accept or deny the request → The user clicks the quick accept button → the request is accepted and the request moves into the ongoing list.

User logs into Data Manager account → A list of all data requests are displayed with a summary of what is being requested for how long with quick buttons to accept or deny the request → The user clicks the quick deny button → the request is denied and the list item disappears.

User logs into Data Manager account → A list of all data requests are displayed with a summary of what is being requested for how long with quick buttons to accept or deny the request → user clicks on the ongoing usage tab → A list of all ongoing data usage is displayed with a summary of what data is being used, by who and for how long with quick buttons to stop access → The user clicks on a list item → A more detailed view of all of the information about the data usage is displayed with a button to stop access.

User logs into Data Manager account → A list of all data requests are displayed with a summary of what is being requested for how long with quick buttons to accept or deny the request → user clicks on the ongoing usage tab → A list of all ongoing data usage is displayed with a summary of what data is being used, by who and for how long with quick buttons to stop access → The user clicks on a quick deny button on a list item → Access to the data is revoked from the vendor and the list item disappears.

Feature 4 - User Data List

Description

Allows the user to see all of their data stored on the data manager and to make decisions on that data. The information displayed should be at least the data itself and the vendors that have access to it.

Stimulus and Response

User clicks user data tab → A list of all data stored within the data manager is displayed with the data and the vendors that have access to that data → User clicks the delete button for a piece of data → Every vendor linked to that data receives a request for that data to be deleted.

User clicks user data tab → A list of all data stored within the data manager is displayed with the data and the vendors that have access to that data → User clicks the edit button for a piece of data → The user enters the data they want to change it to and presses enter → Every vendor linked to that data receives a rectification request for that data.

User clicks user data tab → A list of all data stored within the data manager is displayed with the data and the vendors that have access to that data → User clicks on the filter button → A filter tab appears and the user selects which filters they would like to apply → The list updates to show only instances of data that fit with the filter.

Should Haves

Feature 1 - Vendor Counter

Description and Priority

Priority 1

This allows the user to see at a glance how many third parties are using their data as well as a list showing which vendors are using specific data.

Stimulus and Response

The user logs into their Data Manager account → A counter is displayed at the top of the page showing how many third parties are using their data and a list of which data each vendor is using.

Feature 2 - Informed Consequences

Description and Priority

Priority 2

When a user wants to stop a vendor's access they must be able to clearly see how this decision may affect their experience with that vendor.

Stimulus and Response

The user clicks on the stop access button on one of the third parties in their vendor list → A pop up appears informing the user how it may change their experience with the vendor → The user clicks okay and the vendor's access is revoked.

The user clicks on the stop access button on one of the third parties in their vendor list → A pop up appears informing the user how it may change their experience with the vendor → The user clicks cancel and the pop up closes with no change.

Feature 3 - Vendor Filter

Description and Priority

Priority 3

The user can use a filter to customise which third parties they can see in the vendor list. Filters would include time since access was allowed, vendor name, data used and when the vendor last used their data.

Stimulus and Response

The user clicks on the filter button → A side tab appears allowing them to select which filters they would like to apply

The users selects one or more filters and clicks apply → The vendor list updates to only show third parties that fit with the parameters

Feature 4 - Decision Recommendations

Description and Priority

Priority 4

The user can see a tab of actions that are recommended for them to take such as disallowing vendors that haven't used their data for a while or disallowing access of high value data. They should be able to either individually select which vendors they would like to deny access to or be able to deny all with one button. They will also be able to ignore individual recommendations or ignore all with a single button like denials.

Stimulus and Response

The user clicks on the recommendation button → A side tab appears showing vendors that haven't used the users data or are using high value data → The user denies access to the vendors they would like to deny access to → The vendors no longer have access to the specified data.

The user clicks on the recommendation button → A side tab appears showing vendors that haven't used the users data or are using high value data → The user clicks deny all→ All vendors on the list no longer have access to the specified data.

Appendix G: Dev-Sprint 1 Doc

DevSprint Plan

DevSprint Number: 1

Sprint Duration: 27/01/2025 - 10/02/2025 (21 days)

Description

This document will contain the development goals of the sprint, documentation of the development, testing reports and evaluation of the features developed in the sprint.

This is the first sprint in the development of the PDV. It will focus on getting the fundamental features working for the app before we can develop anything around our requirements. Achieving all of the goals in this sprint is vital to continuing the project.

Sprint Goals

1. Get a VM instance running
2. Create a way to send data remotely to VM
3. Have VM store received data
4. Set up a server hosting a webpage on the VM

Initial Ideas

- A very lightweight version of linux might be best for this project, we need an OS with a low space requirement initially so we can maximise the space available for storage however this also needs to be able to perform the functions of the application quickly to not affect the users experience.
- When creating a way to send data to the VM this means sensor data **NOT** a way to request this data.
- The webpage at this time will only be a place holder however this will become the UI for the PDV. If we can have a webpage that can be accessed from other devices this means we can continue development of the UI using a web server.
- The development will take place on a personal device however testing should be done around portability as we want to test the PDV on different host devices in the future.

Development Log

25/01/25 - Started Development

- I have set up an Ubuntu live server running on a virtual box vm so far I have managed to set this up with an apache2 web server that can be accessed from anywhere with a placeholder page. This is however a set up that would need to be repeated if a new vm were created and there was a different network as I have used port forwarding.
- There were initially some issues with finding a suitable OS for this project as I was having trouble getting more lightweight linux distros working therefore I have settled with Ubuntu Live Server as using it is intuitive and allows me to continue development. Despite this its size is still relatively small at around 2.4GB however I am aware that smaller operating systems are available.

26/01/25 - Testing Development Environment

- I started looking for ways to send data from one device to another and it seems like HTTPS will be the best protocol for communicating data between the data collector and the PDV.

This is useful as I already have experience developing HTTPS servers and we will already have one running anyway. Because of this I will be switching from using apache2 to creating my own server. This will allow me to customise the server functions a lot more and will hopefully mean there are less redundant operations.
- I will be using node js and npm to run the server and host the web pages.
- All development will be done off VM as it is easier for me to control, I will then upload all files to github and pull them on the VM for testing.
- Node and NPM are now set up on the VM so I can run the server and javascript straight away when testing is needed.
- Once development of the server is completed I will make it a service process so it automatically starts when the VM is booted

27/01/25 - Node development
- Started on creating the node server which will be a continuous development. I have also created a mysql database to store vendor information.
- From the responses of the survey and reading papers for my literature review I have found that there is more of a need for a system that manages where your data is being held and being able to enact decisions based on that rather than having a place to store your data. Focusing development on this aspect would make it a lot easier and will still hit most of the targets for the project. If time is left over near the end of developing that deliverable then the storage aspect could be added with relative ease as the main groundwork would have been built. I am going to discuss this with my supervisor and see how to document this more formally.
- Currently the server can add, remove and count the third parties within the database. The functions within the server work as expected however the add and remove functions need to be tested from a remote client before they are complete.

29/01/25 - Testing remote requests
- I have now moved the current web server and related files to the VM and have set up an sql server on there. All functions of the server work and I have set up a test client to make remote http requests however I have come up against a cors error to do with content access control.

03/02/25 - Remote Data
- The server running on the VM can now receive data from a remote client and store it in the database. The client must have the correct url which is in place of the PDVid at the moment. At the moment I feel the PDVid will be a unique 8 digit code in the url of the PDV. I need to find a way to generate this randomly for each instance as well as apply to the url. In theory this now achieves all the goals within this sprint however to achieve this I allowed all origins to request data from the server to get around the CORS errors I was coming across which is very insecure. The next step is to find a way to get around the CORS errors in a more secure way.

Testing

WP-01: Placeholder webpage must show when the VM url is entered into a browser.
Gherkin:
Given: The user has the correct url for the page.
And: The VM is running.
When: The user enters the url in any web browser on any network.
Then: The webpage will be displayed in the user's browser.
Test Passed?: Yes

RD-01: When a button is clicked on the client side a data entry must be added into the VM database.
Gherkin:
Given: The client has the VM url.
And: The VM is running.
When: The add button is clicked on the client side.
Then: The VM should receive the name and ID of the client and store it in its database.
Test Passed?: Yes

VC-01: When the page is refreshed an updated count of all of the instances within the VMs database should be displayed on the screen.
Gherkin:
Given: The VM is running.
And: The SQL database is running.
When: The user refreshes the VM webpage.
Then: The server should query the database and display the updated count on the webpage.
Test Passed?: Yes

Evaluation

Verification

Goal 1 - Get a VM instance running.
The initial idea for this was to get a very lightweight version of linux running so the application would be as small as possible to alleviate size requirements and also free up space so the storage we do use can be focused on the data. We ended up going for the Ubuntu Server 24.04.2 LTS which is about 3GB in size which is not the smallest possible OS. We tried to make some other lightweight operating systems work however we ran into problems with the VirtualBox VM software.

Goal 2 - Create a way to send data remotely to the VM

This has been achieved however not in the way we first intended. Our survey and research concluded that the management of data is of higher priority than the storage of it therefore we have a way to receive information from the vendor such as identification, the data that has been submitted to that vendor and the category of data they hold. This would have been developed further down the line in the old design.

Goal 3 - Have the VM store received data

This has been achieved using a mySQL database on the VM. Currently there is only one table which stores all vendor information. This can be split into different tables in the future when the need for them is there and more data attributes are stored.

Goal 4 - Set up a server hosting a web page on the VM

This has been achieved; currently a placeholder page is shown when the VM ip is entered into the browser. The placeholder page when loaded displays a count of all the instances inside the sql database.

Validation

Goal 1 - Get a VM instance running.

A VM may not be needed now for the new design as it is taking the form of an application more than a server however for testing purposes we will still use the vm to simulate two distinct systems, the data manager and the vendor.

Goal 2 - Create a way to send data remotely to the VM.

This partially meets with the requirement for being able to track where a user's data is; however the format for third parties to send this data needs to be formalised and a full working example of this needs to be developed. In general this is a step toward fulfilling most of the requirements that involve communication with the server such as acquiring data from the vendor and sending data from the server.

Goal 3 - Have the VM store received data.
This again does not target a specific requirement however is a large step in targeting and requirement that involves enacting data decisions and informing the user.

Goal 4 - Set up a server hosting a webpage on the VM.
Again this is only a stepping stone to fulfilling any inform and enact requirements as this is the main UI the user will use to view and make decisions on their data.

Appendix H: Dev-Sprint 2 Doc

DevSprint Plan

DevSprint Number: 2

Sprint Duration: 11/02/25 - 2/03/25 (21 days)

Description

This document will contain the development goals of the sprint, documentation of the development, testing reports and evaluation of the features developed in the sprint.

Dev sprint 2 will focus on expanding the work from sprint 1. This will include adding some features from our user stories.

Sprint Goals

1. Create the ID system so that individual data managers can be identified.
2. Create a display for all vendors within the database
3. Make a way to capture user-submitted data and send it to the data manager.
4. Create a way for the user to verify new data that has been sent to the data manager.

Initial Ideas

- My idea for creating the ID system is to embed the id within the PDV url. Currently to send data the user has to submit something along the lines of http://xxx.xxx.xxx.xxx:xxx however this is insecure. The format I would like to achieve would be https://pdv/12345678, the last number being the ID however currently I am unsure how to achieve this.
- Creating the display should be pretty simple; it will most likely be a query to the database from the server which gets sent to the page and formatted there; however this may slow down when the amount of data within the database increases.
- For the third goal I think this will have to be done on the vendor side as it will be sent via the HTTP post request that's currently in place however I can see this needing to be updated in the future due to the size of data and possible security concerns.

Development Log

16/02/25 - Vendor list and query speed testing.

- I wanted to test the speed of the select all query when the amount of data increased within the database. Looking at how many websites a user visits per week 200 seemed a good place to start (https://www.demandsage.com/website-statistics/). Each record only holds a name and an ID so more testing will need to be done once more data is stored. I also think that more instances need to be tested in the 10s of thousands range.
- The vendor list was as simple as I thought it'd be and with the way I have set it up filtering should be relatively easy to implement as well.
- Within this sprint I think the hardest task is going to be setting up the ID system. The only way I think I could achieve this is through DNS within the VM however this seems overly complicated for the current application design. The main benefit to doing it with DNS is that it will allow for expansion into the storage aspect of the PDV in the future.

17/02/25 - Database updates and data transmission.

- I've created another table within the to store the users submitted data and metadata. I wanted to link a single vendor in the vendor info table with multiple data ids through a relational database however it would have been impossible doing this dynamically as there are too many data types to consider therefore I think i'm going to try and store a list of IDs as a string which can then be parsed and interpreted as a list within the server.
- Each instance in the user data table will be a separate piece of data with an ID, the type of data it is e.g address, card details and the data itself. The IDs allow me to link each specific data to a vendor and having each piece of data stored as separate instances allows for verification in the future.
- I still need to implement the linking of the tables through the server however this will not take long.
- The capture of user data is done on the client/vendor side and is pushed to the data manager rather than the data manager requesting it.

23/02/25 - Linked tables.

- The server can now add a list of data IDs to a specific vendor linking the two tables.
- Next I will try to add a method of verifying new data.

24/02/2025 - Data Verification

- Data that is sent to the manager now is initially stored inside a buffer table within the database. The server then searches if the newdata appears in the user data table, if so it is deleted from the buffer then the already existing data id is linked to the vendor. If it is a new piece of data not stored in the user data table the user must verify it on the webpage before

it is linked with a vendor and sent to the user data table. If the user rejects the pending data it is deleted and not linked.
- There may be an easier way to compare the user data table and the buffer table directly using one sql query so I will work on that If I have time.
- Individual vendors can now be deleted from the vendor list and database using the delete buttons on their respective list entries.
- I need to work on a way to delete vendors that have no linked data after their pending data has been rejected however this should be relatively simple.
- The next major feature to work on is the ID system which I think is going to be very difficult and might run over into the next sprint but we will see.

Testing

DMID-01: A vendor should be able to send unique data to two separate data managers using different IDs
Gherkin:
Given: The vendor has a valid PDVid.
When: The vendor sends the data to the url.
Then: The data manager with the corresponding ID should receive the JSON payload.
Test Passed?: Yes

DV-01: If the data manager receives a new set of user data for the first time it should ask the user to confirm it is theirs.
Gherkin:
Given: The vendor sends a payload to a valid url.
When: The data manager does not recognise an item of data.
Then: The data manager should ask the user to verify the data before it is stored in the database.
Test Passed?: Yes

VL-01: A user should be able to see a list of all the vendors and their information on the data managers webpage.
Gherkin:
Given: The database has at least one data entry.
When: The user views the data manager webpage.
Then: The page should show a formatted list of the data within the database.
Test Passed?: Yes

RD-02: When the user submits data to the vendor that data should be sent to the data manager.
Gherkin:
Given: The user has entered a valid url.
And: Submitted data to the vendor.
When: The vendor sends the information payload.
Then: The payload should contain the data the user has submitted.
Test Passed?: Yes

VL-02: Testing the speed of the vendor list function when all vendors are queried.
This was tested using the following code:

```
start = performance.now();
    vendorList();
```

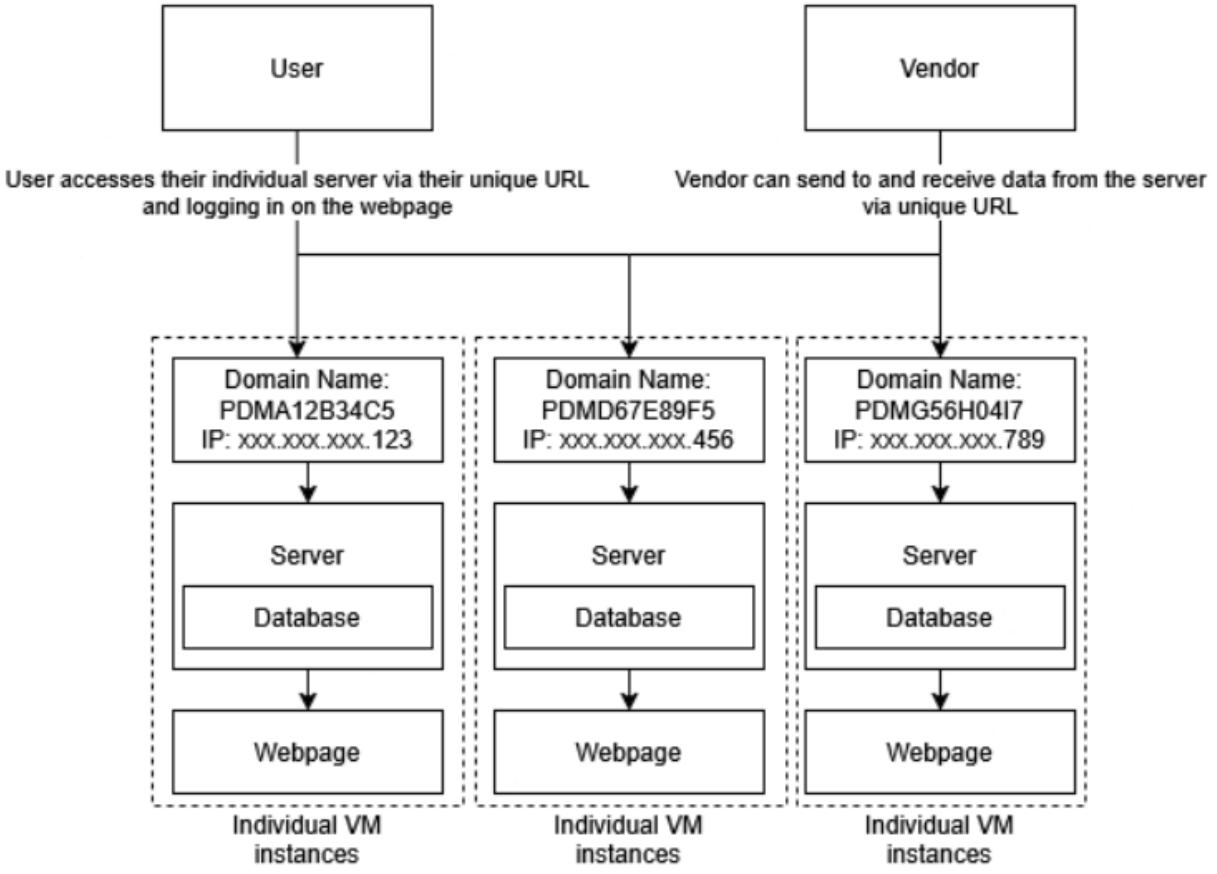
```
end = performance.now();
timeTaken = end - start;
console.log("Function took " + timeTaken + " milliseconds");
```

Number of Instances in database	Time to query and display (millisecond average after 10 queries)
200	1.4 ms
500	1 ms
1000	1.5 ms
2000	1.6 ms

Evaluation

Verification

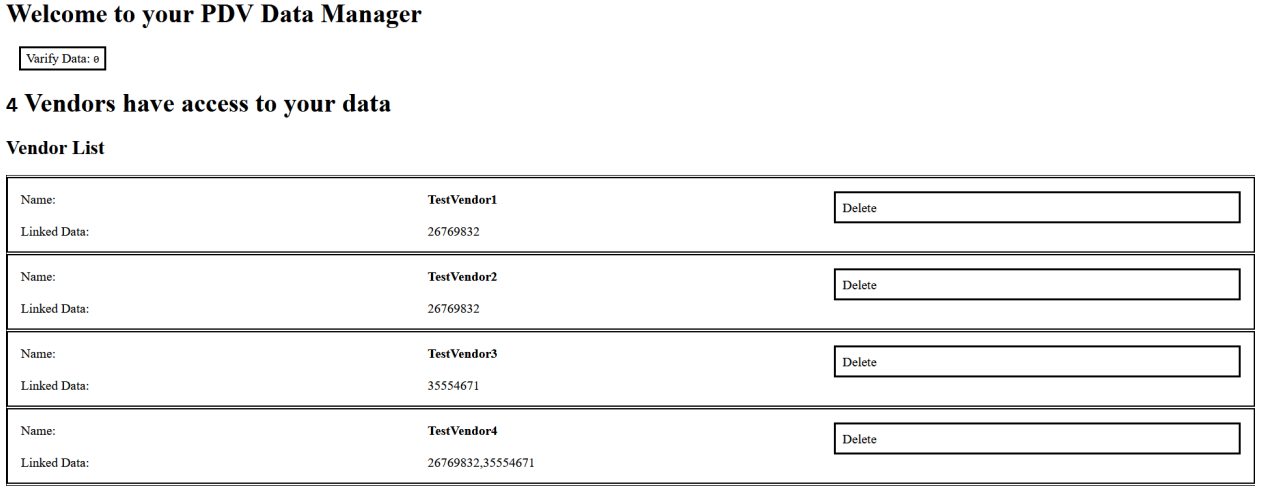
Goal 1 - Create the ID system so that individual data managers can be identified.
This does not work in the way I originally intended and more research is needed to find a way of achieving this or a new design will have to be created. Here is the intended design diagram for this feature.



The vendor can send data to a specific data manager as long as it resides on a unique port or ip and the user can access a specific data manager through this ip however providing a raw ip to a vendor and using this as access to a webpage on an open network is insecure.

The reason this was unsuccessful was that I did not have the resources to achieve the design. I intended to have a domain name server hosted on the VM that assigns a unique domain name to each instance. Through my research I found I would not be able to do this publicly as I would need to purchase the rights to each domain. The second line of thought would be to have a central domain then have extensions to that domain using a unique id such as www.pdm.com/1234ABCD however this would require a central host that then hosts the manager instances which would not satisfy the requirements for hosting on a user owned device and would be closer to the solid pods idea of having a pod provider. I am not completely closed off to this idea however this would be a last resort option.

Goal 2 - Create a display for all vendors within the database.
This has been achieved and will be fleshed out in the future. The vendor list displays all vendors that are stored within the vendor_info database as below.



The list is automatically updated when the page is refreshed. The vendor information includes the data the vendor has access to in its linked data field however the id is displayed currently where it should show the type of data e.g. address. This satisfies step 4 in the Acquire and Inform section of the design diagram to some extent however as mentioned before the linked data need to show the type of data instead of the data id.

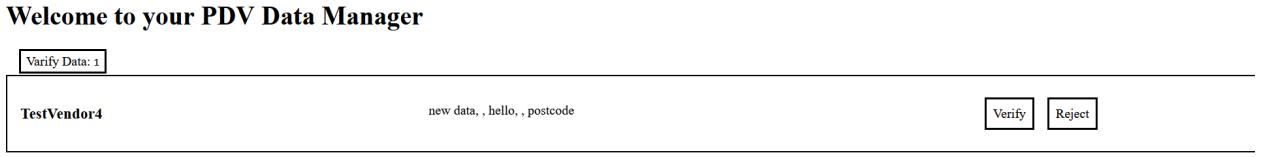
Goal 3 - Make a way to capture user-submitted data and send it to the data manager.

This has been achieved however this is done on the vendor side and pushed to the data manager instead of requested by the manger itself. This works and is a good, easy solution but it could be updated to have the data manager request instead of receive so the vendor doesn't have to implement as much.

Goal 4 - Create a way for the user to verify new data that has been sent to the data manager.
Although not directly in the design I felt this feature was needed if the manager were to store the data the user was submitting. If someone were to enter an incorrect url and send their data to the wrong manager the vendor and data would automatically be stored within that manager. I felt a

good solution to this would be to have a way for the user to have to verify any data that isn't recognised by the manager. If the user verifies that it is theirs then they can accept it and it will be linked to the vendor and be sent to the user_data table. If they reject the data then it will be deleted from the database and will not be added to the vendor instance.

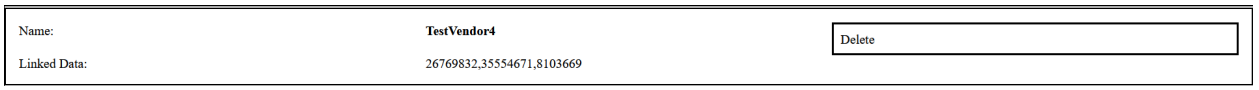
When a new set of data is sent to the manager the user must verify it in the pending data list which is accessed using the verify data button.



The user can then click verify and the data id will be added to the vendor_info linked_data field for the respective vendor.

Vendor instance before data is verified.

Vendor instance after data is verified.



Validation

Goal 1 - Create the ID system so that individual data managers can be identified.
This achieves functional requirements 1 and 2 "Submit an ID for their Data Manager to a third party for a data request." and "Be able to see recent data requests and accept or deny them."

As mentioned in verification this has not been achieved to design as the ID is not layman friendly. Using the ip and port of the machine is an easy way to implement this feature however for the average user it isn't very friendly and may be quite daunting.

Goal 2 - Create a display for all vendors within the database.

This almost completely satisfies functional requirement 3 "See a collection of all vendors and their ongoing data access" and non-functional requirement 3 "Key information such as the amount of vendors using data should be clearly displayed". We have a list of all vendors that have access to any data from the user. It is implied that the data is ongoing as they are still holding it and in the future vendors will be removed if a deletion request is made. More info for each vendor could be added in the future to strengthen the fulfilment of this requirement however this is not a high priority.
Goal 3 - Make a way to capture user-submitted data and send it to the data manager.

This is a stepping stone to fulfilling functional requirement 4 "See a collection of data that is held within the manager". There is currently no way to view this data as this will be developed in the next sprint however the data is there to be displayed. The metadata collected should also be sufficient for clear displaying which will also satisfy non-functional requirement 3.

Goal 4 - Create a way for the user to verify new data that has been sent to the data manager.

As mentioned in verification this does not target a specific requirement however we felt it was needed as it is possible for data that isn’t the users to be accidentally sent to their data manager. Within this feature there are decision buttons which do follow the guide of non-functional requirement 2 “Decision making must be simple and streamlined and cannot require too many inputs”. Each operation only takes one click from the tab and it is clear what the functions do.

Appendix I: Dev-Sprint 3 Doc

DevSprint Plan

DevSprint Number: 3

Sprint Duration: 03/03/25 - 23/03/25 (21 days)

Description

This document will contain the development goals of the sprint, documentation of the development, testing reports and evaluation of the features developed in the sprint.

This sprint is largely about fulfilling the enact design and polishing the inform aspects of the data manager.

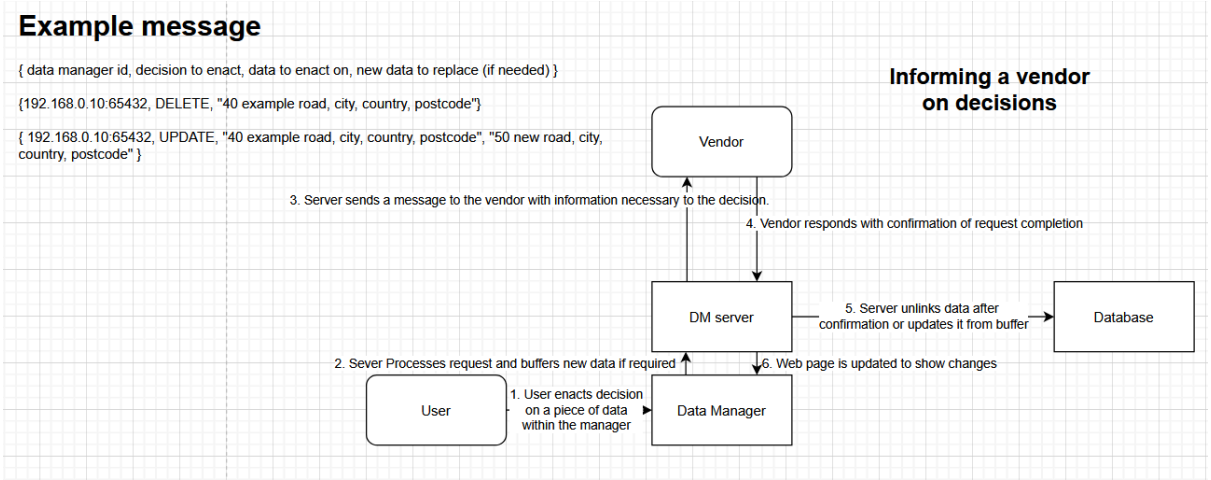
Sprint Goals

1. Display the type of data a vendor has access to instead of the data id.
2. Create delete and edit buttons for enacting data decisions on the vendor list.
3. Create a display for showing the data within the user data table and which vendors have access to each.
4. Create a means of informing the vendor about a data decision.

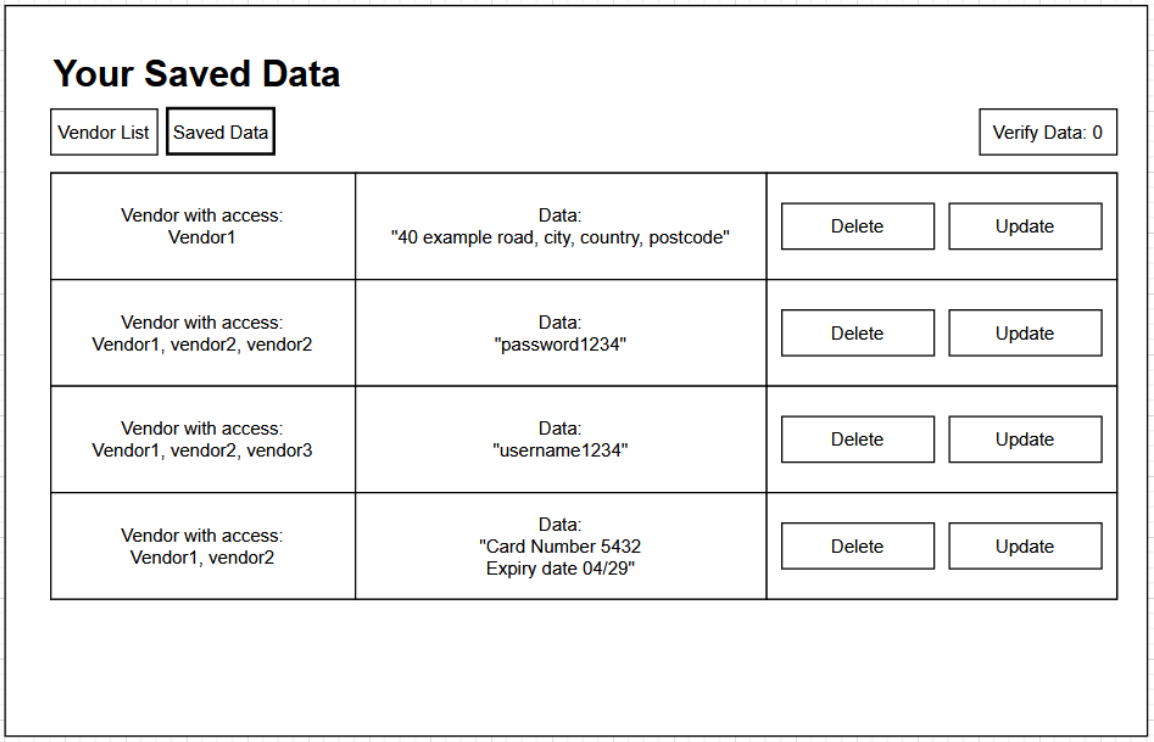
Initial Ideas

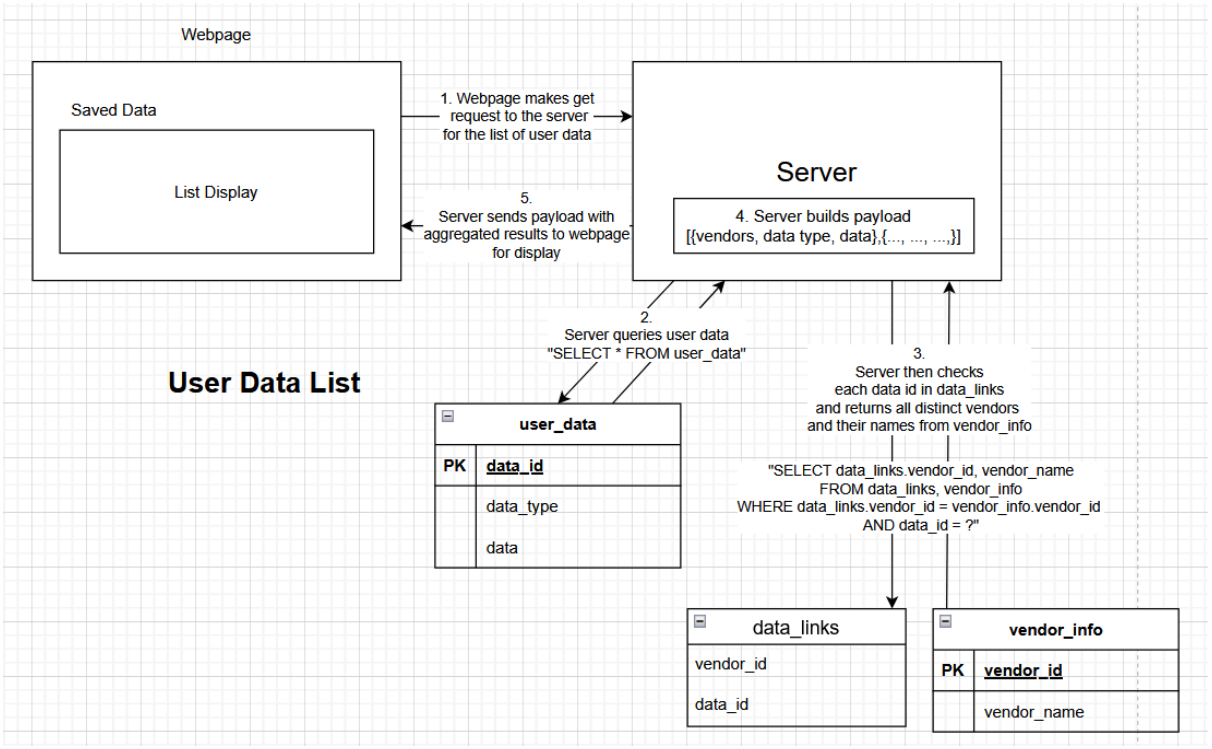
- As this sprint is mostly polishing what exists so far I think it will be easy to get through most of the goals. The one that will take the most time will probably be goal 4 as I am going to make the burden on the vendor as light as possible.
- I will most start styling using css more heavily as well to try and get the app to look more finished however this isn't a priority and will only be done if there is any spare time.

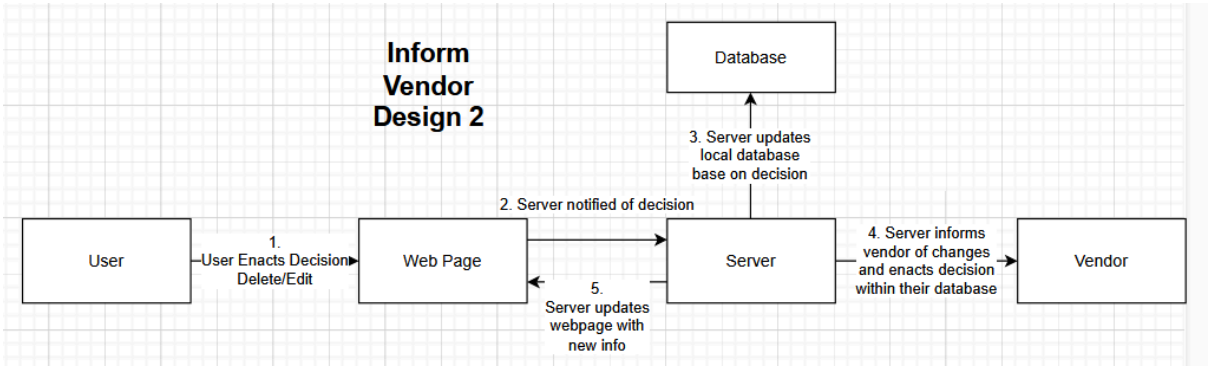
Feature designs



User Data List Design On Page







Development Log

07/03/25 - Showing linked data types

- The linked data on the vendor list now shows as the data type that is linked rather than the data id itself which is a lot more digestible for a user. This may however be incredibly slow especially as the vendor list and linked data for each vendor grows. This is because I am running a select query for each data id on every vendor so there is a very heavy burden on the server. I will test the scalability near the end of the sprint as part of the evaluation.
- The next step is to create a display for the user data.

09/03/25 - Database rework and user data display

- I decided that the method I was using to link the data was way too complicated and would have slowed down the app significantly as more data was added. To fix this I decided to make another table containing every data link. This way I can use more advanced queries to access more relevant data rather than having to combine many simple ones then aggregating this using the server. This should greatly decrease the load on the server and database. Unfortunately this meant rewriting almost every function within the server to use more complex queries and updated functions however it should be worth it as now linking between tables should be a lot easier.

- I've started working on the data display. At the moment I have all data from the user data table being displayed and I need to find a way of showing all of the vendors linked to each individual piece of data. I can think of a solution to this however it is incredibly slow and inefficient so I will try and come up with a better one before I start Implementing it.

13/03/25 - Editing and Deleting data

- The user can now edit and delete specific data from the database however the vendor has no knowledge of this so far.
- When a user deletes a piece of data all the associated vendor links are deleted. If a vendor I left with no data links it is removed from the vendor list but not the database.
- The user can edit an individual piece of data however there are no checks on if the new data already exists or not so there can be duplicates which I plan to change.

17/03/25 - Informing the vendor

- I realised that the current system I have for enacting user decisions isn't the best. It fits more into the second design than the first which means a user can enact a decision locally however there is no confirmation the vendor has actually enacted it on their side therefore this needs to be changed. Luckily this shouldn't be too difficult as it just requires reordering the processes however this may require another table in the database. Before this however I think I will develop the functionality for informing the vendor so then all of the pieces are there for me to order at the end.
- When a data decision is made the vendor is now informed. I intend to simulate a delayed response to a decision being enacted on the vendor side as this would be more accurate to an actual use case.
- I now need to implement a way to delay updating the database locally until the vendor confirms it has been done on their side. To do this I will most likely use the buffer table and add a tag that shows why the data is in there. E.g pending verification, pending edit, pending delete. This will make the design more like the first diagram.

22/03/25 - Informing the vendor 2

- The enact system now waits for the vendor to confirm that they have changed the data on their end before any local changes are done making the feature more like the first design.
- There was a possibility that if data is edited to be the same as another then they would automatically merge because of my check data function however this wasnt the case so I will have to create a way to merge data if two instances of it exist.
-

Testing

Scenario: Updated database layout and edited functions accordingly

ID	Test	Input	Expected Output	Actual Output	Notes
RD-03	Vendor sends data to pdm.	Payload {name: "ThisIsATest, datatype: "card", data: "12345,123,1234"}	Vendor name and id is inserted into vendor_info table and the data, datatype and generated data id is inserted into	Server crashed trying to split linked data. Correct data had been inserted into the database however.	This was due to the getVendorList function not being edited yet. Looking at the database however the data has been entered.

			the data_buffer		
DV-02	Vendor sends already existing data to pdm.	Payload {name: “ThisIsATest, datatype: “card”, data: “12345,123,1234”}	If a link already exists nothing should change inside the pdm	Server tries to create a new data link with NULL id.	To fix this I need to try and create a function that checks if a link already exists. Updated insert link query to check if exact link already exists. When trying to link data that already exists but not linked to any vendor the linkData function will not be able to find an ID to link to.
VL-03	Vendor List should show data types each vendor has access to.	Payload { name: "ThisIsATest" , datatype: "address", data: "Test, Address, , , " }	Vendor list should display the data types the vendor has linked to them. In the case of vendor “ThisIsATest” the linked data should show “address, card”	As expected	
DV-03	When a new vendor is added if the data submitted already exists inside the user_data table it should link the existing data.	Payload {name: “ThisIsATest2 , datatype: “card”, data: “12345,123,1234”}	New vendor list instance appear in the vendor list with a data link to already existing data	New vendor is created but the data insert function can't find the id from the name.	For some reason the query cant access new data despite the fact I can see the vendor instance inside the database. Updated the linkData function to keep trying to find the id until it finds one.

DV-04	When a new piece of data has been verified it should appear in the saved data list	Verifying data { name: "Vendor2", datatype: "card", data: "newCardData, 12, 12345" }	New verified data should appear in the saved data list with all attributes	As expected	
DV-05	When a new piece of data is verified in the data manager it should appear in the user data list with the vendor it is linked to.	Verifying data { name: "Vendor1", datatype: "card", data: "NewCardDetails, 10/25, 092" }	When the verify button is clicked the data should appear within the user data list with one linked vendor named “Vendor1”	As expected	
UDL-01	When a new vendor is added with the same data submitted the linked vendors inside the user data tab for the respective data should update	Verifying data { name: "Vendor2", datatype: "card", data: "NewCardDetails, 10/25, 092" }	When the data is sent to the data manager the linked vendors for the data should update showing “Vendor1, Vendor2”	As expected	
UDL-02	When a vendor is deleted all links to that data should be removed and the name should disappear from any data it was	Deleting Vendor1	When the vendor is deleted “Vendor1” should disappear from the data within the user data tab	As expected	

	once linked to.				
UDL-03	When a piece of data is deleted from the user data list it should disappear from any vendor that it is linked to in the vendor list	Deleting { datatype: "card", data: "NewCardDetails, 10/25, 092" }	When the data is deleted “address” should be removed from “Vendor1”	As expected	When only one piece of data is linked to the vendor and is deleted the vendor still appears in the list.
VL-04	When a vendor has no linked data it should not appear in the vendor list	Adding vendor 4 with new data	When the payload is sent to the server the new data should appear in the verify data list however the vendor count and vendor list should not show the new vendor until the data is verified	As expected	
ED-01	When enter is pressed after a piece of data has been edited the data id and new data should be sent to the server	Payload {“id: 70918528, newData: editedCardDetails, 11/25, 092”}	Server should receive the correct id and newData	As expected	
IV-01	When inform vendor function is called the	Sending payload {source:”192.168.0.90:8080”, data	The test vendor server should receive the info, log it in	Invalid url error from the fetch function within the	This may be because the url is incomplete, I will add “http://” to the start of the URL.

	vendor at the given url should receive the data then respond with a confirmation message	“1234, 123, 12”, decision:“DELETE”} to url “192.168.0.90:65431”	its own console then send “Data Received” back to the data manager server.	inform vendor function	
IV-01	^	Sending payload {source:”192.168.0.90:8080”, data “1234, 123, 12”, decision:“DELETE”} to url “http://192.168.0.90:65431”	^	Vendor server is receiving undefined data	The vendorURL is created and stored within the server database so it can be automated to always include the “http://”. The undefined data issue was due to express.json function not being included in vendor recv function.
IV-02	When a vendor is informed of a data edit the new data should be moved to the data buffer table	Updating data {1234,123,12} to {12345,1234,123}	When the data is submitted the data should appear in the buffer table with the purpose tag “pending vendor confirmation”	Data was entered into the table twice as there were two vendors linked to it although the vendor is specified so this is acceptable.	
IV-03	When a decision is sent to a vendor the server should get a response after 3000 ms	Sending update request	When the request is sent the vendor should respond with the decision that was requested and the vendor it is for after 3000 ms. The server will	Fetch failed error	This was due to the fetch on the vendor end trying to use HTTPS instead of HTTP

			then output “updating data for vendor”		
IV-03	^	^	^	As expected	Server received two confirmations as there were two vendors linked to data
IV-04	Testing vendor confirmation for delete decision	Sending delete request	Vendor should respond after 3000 ms with the data that should be deleted.	As expected	I would like to change this to send and receive the data id as this will decrease the amount of data being sent
IV-05	Vendor should send data id instead of vendor name when confirming request	Sending edit request	Vendor should respond after 3000 ms with the data id to be edited	As expected	I am going to change the addToBuffer function to include an optional parameter for data id so it can be searched.
D2B-01	When addDataToBuffer is called without data id as parameter it should generate a random one.	`addDataToBuffer("type", "data", "vendor", "test");`	All data should appear in data_buffer table with random id	Data does not appear in buffer	This worked after server restart
D2B-02	When addDataToBuffer is called with a data id it should enter data with that id	`addDataToBuffer("type", "data", "vendor", "test", 12345678);`	All data should appear in data_buffer with id “12345678”	As expected	

VL-05: A user must be able to see the type of data a vendor has access to on the vendor list, if they have access to multiple of the same type of data the data type should not appear twice.
Gherkin:

Given: The vendor has at least one linked data id in their database entry
When: The user views their list entry on the data manager
Then: The data type of the linked data should be displayed on the list.
But: If the vendor has multiple of the same type of data.
Then: Only one tag for that data type should appear on the vendor list entry.
Test Passed?: Yes

ED-02: A user should be able to prompt update their data within the data manager and have it changed inside the user data table.
Gherkin:
Given: The user has at least one item of data in the user data table.
When: The user clicks the edit button on a data entry in the user data display and fills in the data fields.
Then: That piece of data should be updated within the user data table.
Test Passed?: Yes

IV-06: A user should be able to prompt a rectification request to a vendor for a single piece of data.
Gherkin:
Given: The user edited a piece of data within the user data table using the edit button.
When: The user clicks submit.
Then: A data rectification request should be sent to the vendor with the updated data.
Test Passed?: Yes

DD-01: The user should be able to delete data from the data manager using the delete button.
Gherkin:
Given: The user has at least one item of data within the user data table.
When: The user clicks the delete button on the data manager.
Then: The data should be deleted from the data manager and user data table.
Test Passed?: Yes

IV-07: The user should be able to prompt a data erasure request for a single piece of data using the delete button.
Given: The user has at least one item of data within the user data table and it is linked to at least one vendor.
When: The user clicks the delete button on the piece of data.
Then: A data erasure request should be sent to the vendor.
Test Passed?: Yes

UDL-04: The user should be able to view a full list of all the data within the user data table and which vendors are linked to each.
Given: The user has at least one item of data within their user data table and one vendor linked to that data.
When: The user clicks on the stored data tab.
Then: A full formatted list of user data should be displayed on the page with the data that is stored, what vendors have access to it and delete and edit buttons.
Test Passed?: Yes

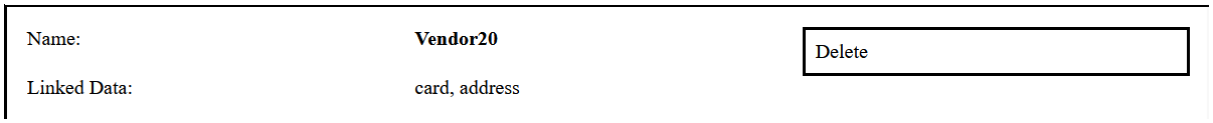
Evaluation

Verification

Goal 1 - Display the type of data a vendor has access to instead of the data id.

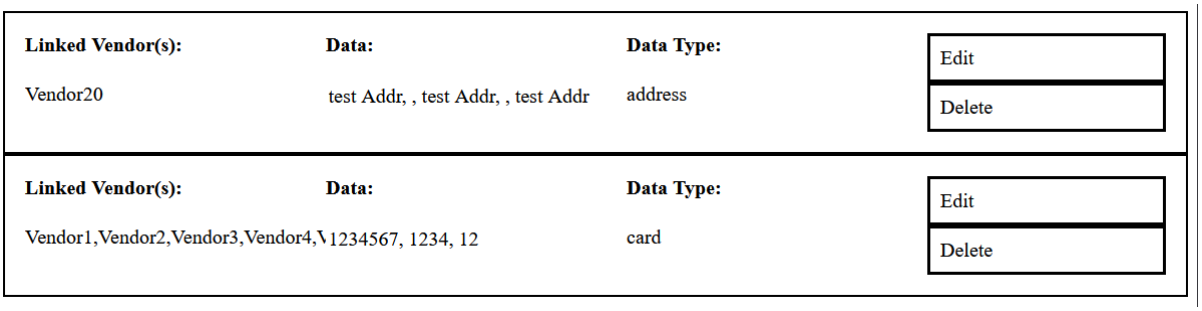
This works exactly as explained however some reworking to the vendor list function was needed. The vendor list function on the webpage client first fetches a list of every vendor within the vendor info database from the server and then iterates through each instance. In each iteration the webpage client then fetches all data types associated with the vendor it is building the item for. This is inefficient as it is doing n+1 fetch requests and queries every time the vendor list is called for (n being the amount of vendors inside the database). Timing this function takes 1ms to complete with 20 vendors however I anticipate this to increase as the amount of vendors gets into the triple digits.

Despite this the feature still works as stated in the goal and only shows one instance of each data type even if the vendor has access to multiple instances of that data type.



Goal 2 - Create delete and edit buttons for enacting data decisions on the vendor list.

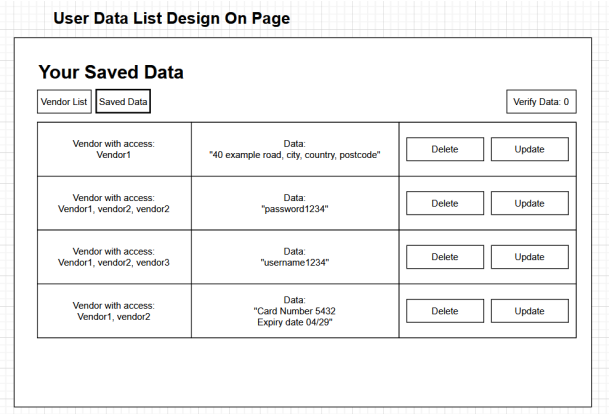
If considering this goal exactly as worded it has not been achieved as there is only one button on each instance of the vendor list which deletes the vendor from the vendor_info database and its associated data links however there are data decision buttons on the user data list.

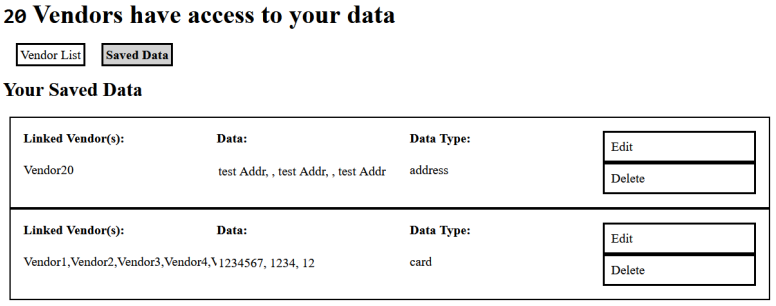


Each instance of data has an edit and delete button attached to it which allows you to make the respective decisions on them. The limitation to this at the moment is that when a decision is made it is enacted across every vendor that is linked to that data. For example with the second data instance in the screenshot above if a user were to edit that piece of data a request would be sent out to vendor1, vendor2, vendor3 and so on.

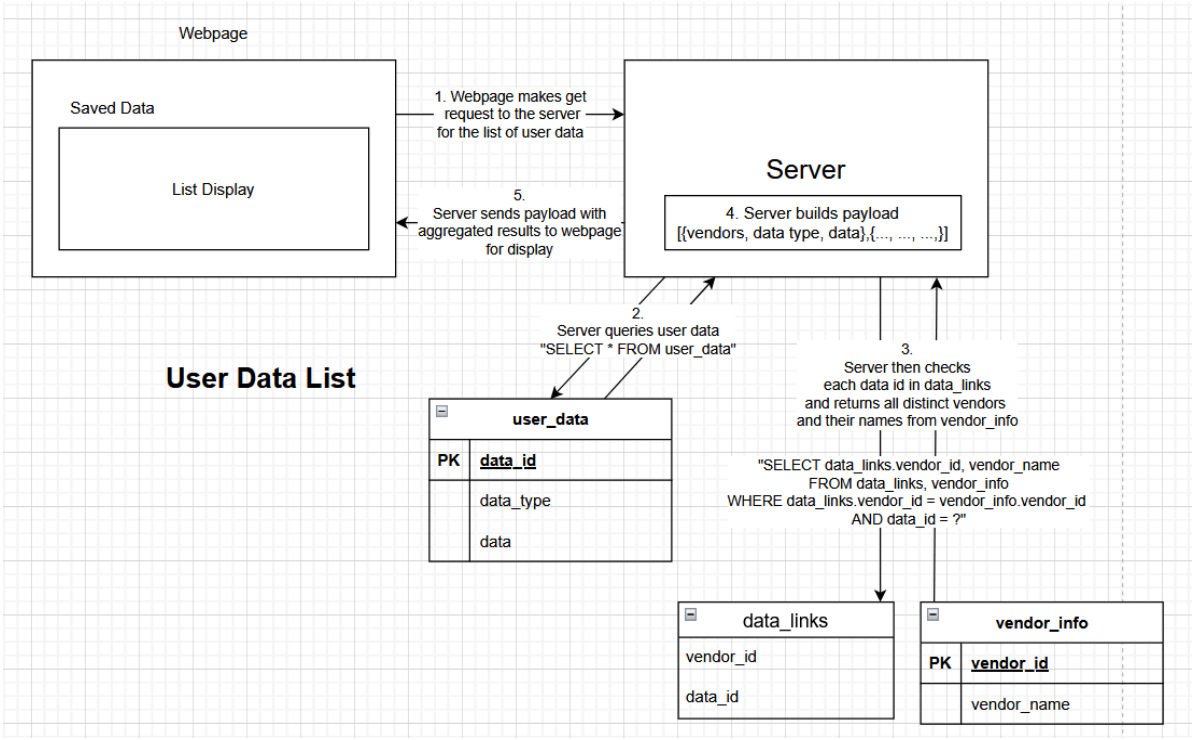
Goal 3 - Create a display for showing the data within the user data table and which vendors have access to each.

This has been achieved, on the left is the wireframe for this feature and on the right is how this has been implemented in the application.
There are some minor differences such as the addition of the data type field however this seemed necessary to add for additional context and the possibility of filters in the future.





The function itself also works as designed in the data flow diagram as well.

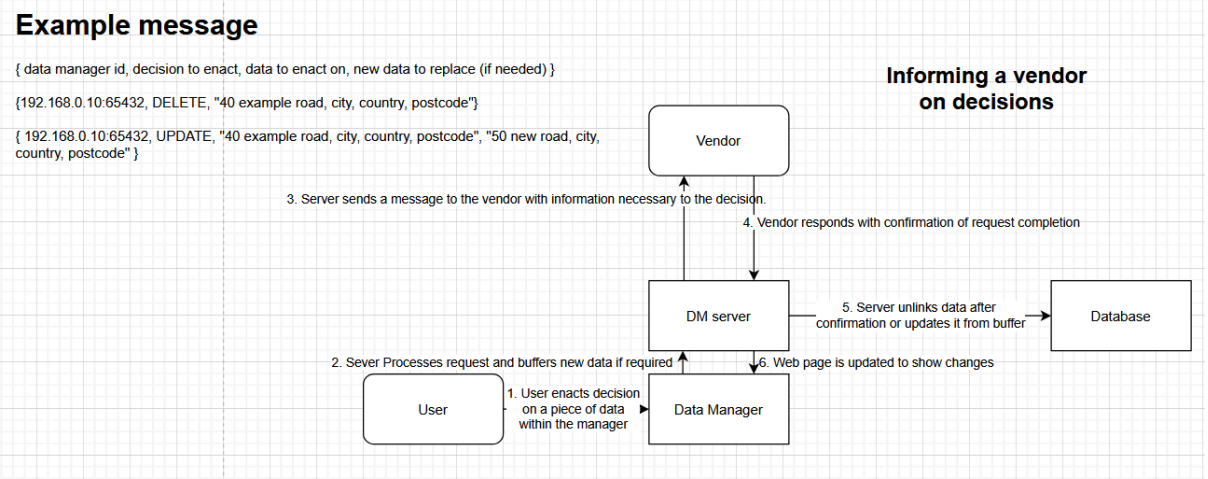


On the webpage client the list builder is almost exactly the same as the vendor list with the exception of the second fetch function for each instance as all data aggregation is done on the server side as outlined in the design. This is good as it lightens the load of the web browser and leaves the hard work to the server which will most likely be running on a machine that can handle it. Some work needs to be done on visualising the vendors as the text clips if the amount of vendors linked is too large however this is a small concern and should be fixed relatively easily.

Goal 4 - Create a means of informing the vendor about a data decision.

There were two possible methods that could have been used to achieve this however it seemed like the first design albeit more complicated was better. This is because if a user were to make a decision and have it enacted locally before the request is sent to the vendor the information could

be wrong for up to a month due to the fact that a data controller has up to a month to fulfil GDPR requests. The first design involves buffering decisions locally then sending the request to the vendor before they are enacted. Once the vendor sends a confirmation to the data manager that the request has been fulfilled then the decision is unbuffered and enacted locally. This means that the data manager always tracks data that the vendors actually have rather than data they should have.



Known Bugs

- When multiple instances of the same new data are verified, they are registered in the user_data table as separate IDs, even though they should be grouped under a single ID. A possible fix for this could be to check if the data exists in the buffer before being entered like the verification process.
- When a piece of data is edited the user may end up editing the data to one that already exists. This creates two instances of that data inside the user_data table when there should only be one. A merge function should be implemented to merge the two instances under one id.

Validation

Goal 1 - Display the type of data a vendor has access to instead of the data id.

This applies to non-functional requirement 3 for user story 4: ”Key information such as the amount of vendors using data should be clearly displayed.”
Allowing the user to see text relating to the type of data that a vendor has rather than an ambiguous id that relates to a specific type of data is a lot more digestible for an individual. To take this a step further the user could name data instead of having just the data type so they can see exactly what data the vendor has instead of a single type of data.

Goal 2 - Create delete and edit buttons for enacting data decisions on the vendor list.

This applies to non-functional requirement 2 for user story 4: “Decision making must be simple and streamlined and cannot require too many inputs.”
As decision buttons are directly attached to each instance in the user data list it requires a maximum of three inputs to make a decision in the case of editing. In both cases delete and edit

only require one click to initiate the decision. When editing the text input it auto focused meaning the user just has to type the new data then press enter to submit.

The decisions also do not require pop ups or extra tabs so it is easy to see where the changes take place which also loosely applies non-functional requirement 2 of user story 4 as seeing where data is changing is a form of clearly displaying information.

Goal 3 - Create a display for showing the data within the user data table and which vendors have access to each.

This applies to functional requirement 4 and feature 4 of user story 4 “See a collection of data that is held within the manager.”
When clicking on the user data list tab a list of all the data within the data manager is displayed as outlined before. Information such as the data, datatype and vendors who have access to the data are displayed on the list.

Looking at the stimulus and response scenarios All of them are achieved with the exception of the last one which outlines a filter so the user can refine which data they would like to see when they have hundreds of possible instances.

Goal 4 - Create a means of informing the vendor about a data decision.

This applies to non-functional requirement 2 for user story 4: “Decision making must be simple and streamlined and cannot require too many inputs.” as this is directly linked to goal 2

This feature forms the functionality behind goal 2 therefore inherits the requirements it satisfies. All of the functionality for this feature is done via the server therefore no user input is needed.

This also applies to non-functional requirement 3 for user story 4: ”Key information such as the amount of vendors using data should be clearly displayed.” however does not satisfy it completely.

When a user initiates a request the decision is buffered so it can be enacted locally once confirmation of completion has been received however there is no way of knowing if the process is going correctly on the user side and there is no indication that the data is awaiting confirmation of a decision therefore this needs to be developed.

Appendix J: Dev-Sprint 4 Doc

DevSprint Plan

DevSprint Number: 4

Sprint Duration: 26/03/25 - 16/04/25 (21 days)

Description

This will be the final official development sprint. Some low technical risk features may be added after this to improve the presentation and demonstration of this application however they will not be substantial additions.

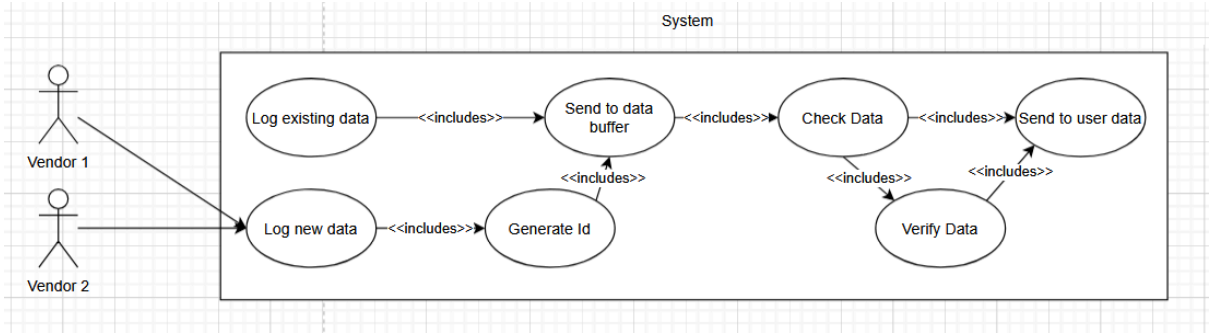
Within this sprint I will be focusing on polishing features like the awaiting decision tab, filters and updated visualisation of data.

Sprint Goals

1. Fix known bugs identified in sprint 3
2. Complete the awaiting decision visualisation feature and create a tab to show awaiting data.
3. Add filters to the vendor list so the user can display certain vendors based on the data they hold.
4. Add filters to user data list based on the vendor list filters
5. Create 3 new type of visualisation of data

Initial Ideas

This is a use case diagram showing how the system for inserting data into the data manager currently works.



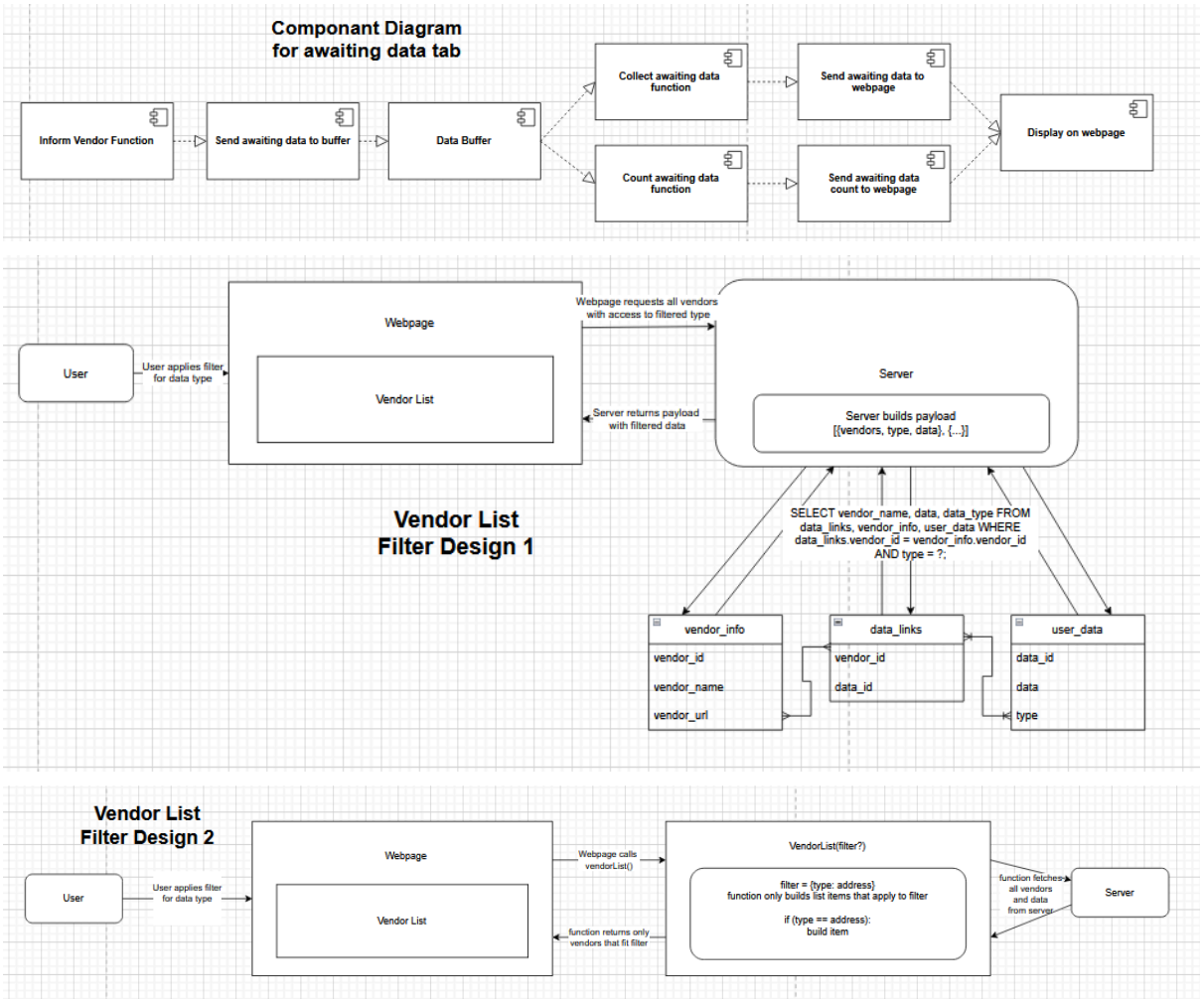
When a piece of data the manager hasn't seen before is sent to the manager an id for it is created then it is sent to the buffer table. Once in the buffer table the data is checked to see if it exists in the user data table. If not then the data waits in the buffer until it is verified. If an existing piece of data is entered without an id then it follows the same path as a new piece of data but when the check data function finds it within the user data table it is removed from the buffer.
The problem we have with this system is that when a new piece of data is waiting in the buffer it never exists in the user data table therefore if the same piece of new data is entered before it has been verified there will be two instances of the same data waiting to be verified under separate Ids.

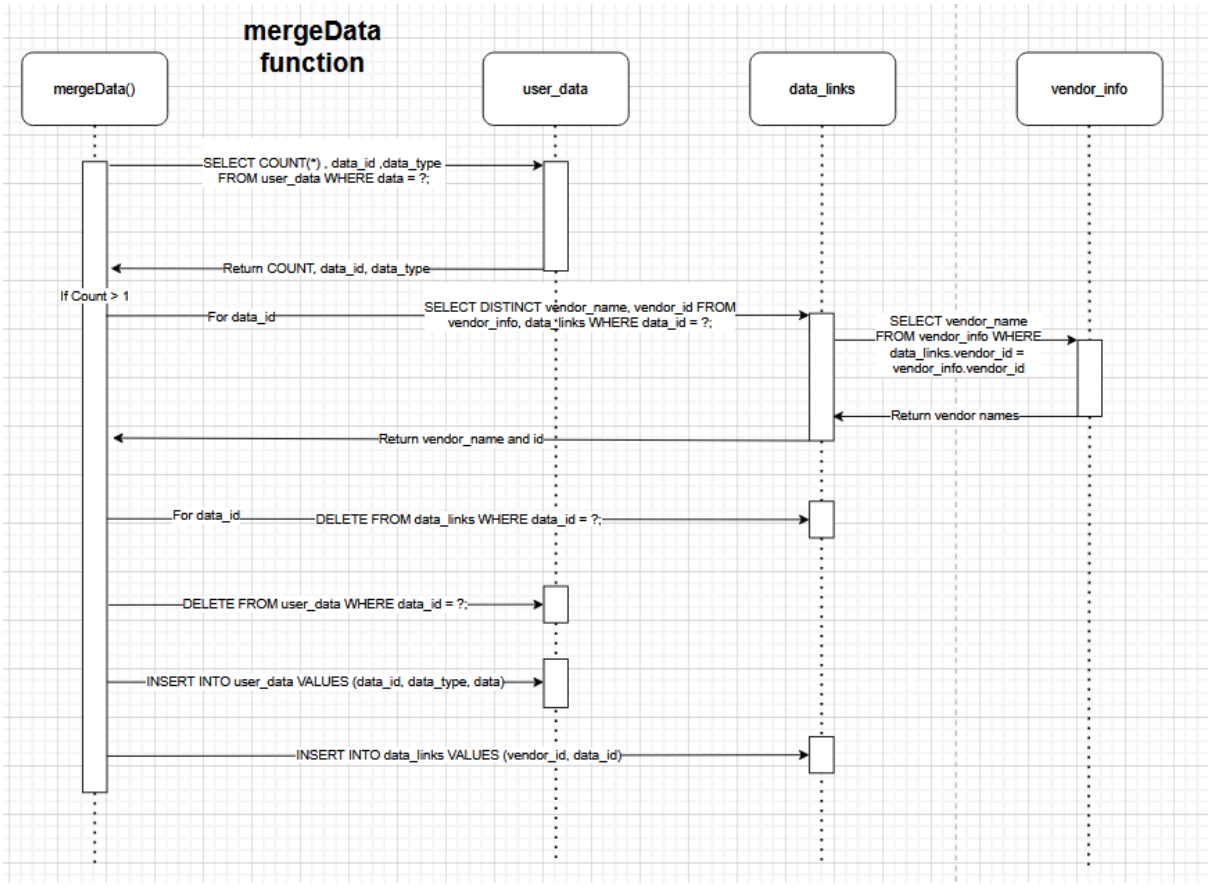
The data is not checked again after the check data function so when they are both eventually verified they will both exist in the user data table under different Ids.

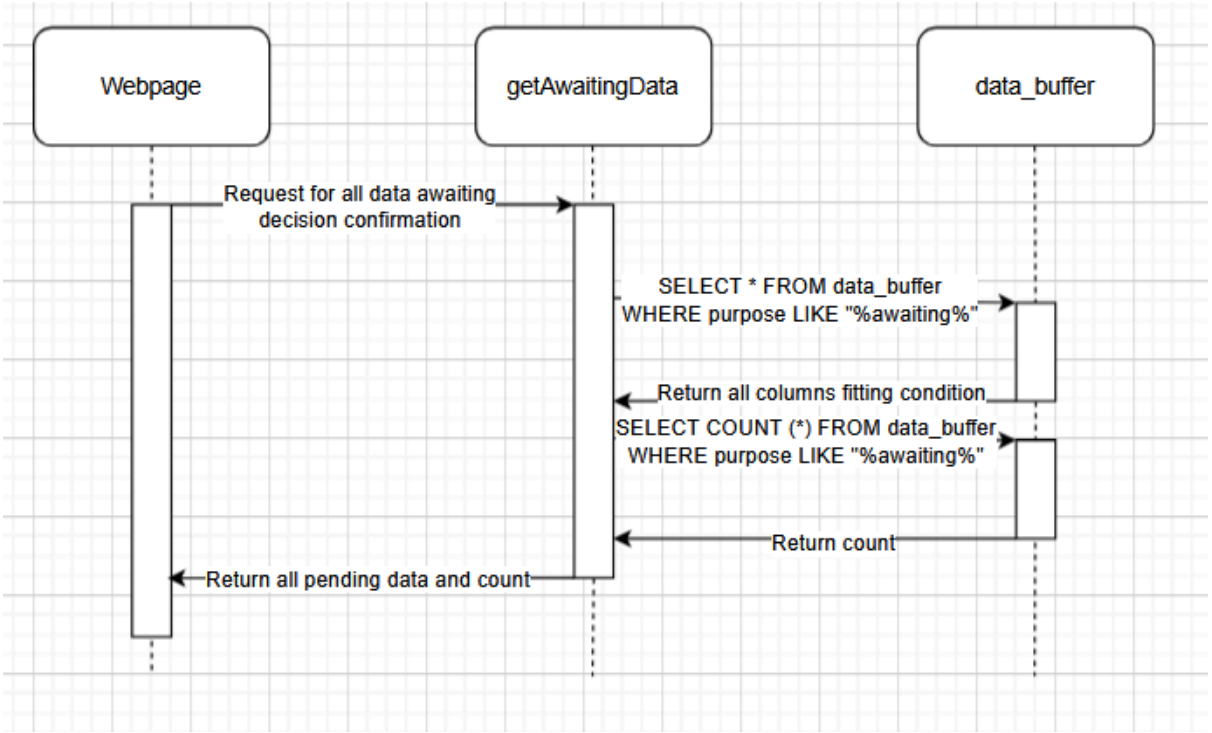
There are two solutions for this problem. We could check the data as it enters the buffer to see if it already exists in there before it is entered or we could implement a merge function that checks for multiple instances of data within user data and merges all vendor links to one instance and deletes the others.

The second option may be the better one as there is another way for two instances of data to exist in the user data table caused by the edit data button. This problem would solve both issues.

Feature designs




mergeData function
mergeData()
user_data
data_links
vendor_info
SELECT COUNT(*) , data_id ,data_type FROM user_data WHERE data = ?;
Return COUNT, data_id, data_type
If Count > 1
For data_id
SELECT DISTINCT vendor_name, vendor_id FROM vendor_info, data_links WHERE data_id = ?;
SELECT vendor_name FROM vendor_info WHERE data_links.vendor_id = vendor_info.vendor_id
Return vendor names
Return vendor_name and id
For data_id
DELETE FROM data_links WHERE data_id = ?;
DELETE FROM user_data WHERE data_id = ?;
INSERT INTO user_data VALUES (data_id, data_type, data)
INSERT INTO data_links VALUES (vendor_id, data_id)


Webpage
getAwaitingData
data_buffer
Request for all data awaiting decision confirmation
SELECT * FROM data_buffer WHERE purpose LIKE "%awaiting%"
Return all columns fitting condition
SELECT COUNT (*) FROM data_buffer WHERE purpose LIKE "%awaiting%"
Return count
Return all pending data and count

Development Log

27/03/25 - Merge Data

- Started working on the merge data function which in theory is complete however I have come across an error that states there are too many connections to the sql.

01/04/25 - Complete merge data

- The merge data function is now complete and has been applied to the necessary areas which now fixed both known issues from sprint 3. There was an issue while implementing this where there were too many connections to the sql server due to the fact I was creating a new connection every time I started a function. This meant the amount of connections was reaching near 100. To fix this I've created one global connection that all functions use which is a drastically better solution.

02/04/25 - Awaiting data tab

- The awaiting data tab now works as it should however only auto updates when an item enters the buffer not when it leaves. An Item will only disappear from the count and tab when the page is refreshed which is inconsistent and I would like to change however this is low priority.
- The button to open and close the tab also hasn't been fully implemented so the list is shown all the time when items are in it however this is a quick fix as I just need to add the event logic.
- The implementation works a little bit differently to the design as I have counted the number of instances using the .length method on the returned array instead of doing a whole query to the database which should be a little bit faster.

03/04/25 - Filters

- Filters have now been implemented. FIlters include vendor, data type and data. Filtering by vendor will show all vendors that match including partial matches. This is the same with data type and data.
- This was implemented client side as it was easy to access all of the list items quickly and just apply a hidden class to any instances that don't fit the filter.

Testing

Merge Data function

ID	Test	Input	Expected Output	Actual Output	Notes
D2UD-01	(Unit) addDataToUserData() should create an instance of the given data directly to the user_data table	addDataToUserData(1234, test, testData)	An instance of (1234, test, testData) should appear inside the user_data table	As expected	
MD-01	(Unit) mergeData() should log a count of	mergeData("testData")	The console should return two raw data packets with	SQL error: "In aggregated query without	I believe this is due to the fact im trying to return aggregated and

	duplicate instances, the data ids and data_type of the given data only if there are multiple instances		the respective information	GROUP BY, expression #2 of SELECT list contains nonaggregated column 'pdm.user_data.data_id'; this is incompatible with sql_mode=only_full_group_by"	non-aggregated values. To fix this I will use two separate queries.
MD-01	^	^	^	Count is returned but second query is not running	The if statement wasn't passing because I didn't specify the packet to check.
MD-02	(Unit) mergeData() should now log a list of all the vendors linked to each instance of the data	mergeData(testingData, ,)	The console should contain two lists containing one vendor each.	Console logged two lists with promise data inside	To fix I will need to add an await for the vendors list so the promise can fulfil before it is printed
MD-03	(Integration) mergeData() should delete all instances of data the data and its links using deleteData()	mergeData(testingData, ,)	Both instances of data should be deleted from user_data	As expected	
MD-04	(Integration) After deleting the data instances mergeData() should create a new instance of data using the first id and type using	mergeData(testingData, ,)	One instance of the data should be left in the user_data table with no links	As expected	

	addDataToUserData()				
MD-05	(Integration) mergeData() should create a data link for every vendor using linkData()	mergeData(testingData, ,)	Two data links should be created for the two vendors to the same id.	SQL error too many connections	Adding connection end calls to functions that don't have them.
SQL-01	(Unit) Updated the connectSQL function to create a pool of 10 instead of a single connection.	Calling connectSQL( )	The too many connection error should no longer appear	As expected	The error appeared again at a different point in the code The connection error was fixed by creating one global connection. The issue was that a new connection was being created every time a query was called.
MD-06	(Integration) All vendors connected to the multiple instances of data should appear next to the left over instance	Calling mergeData(testingData, ,)	All vendors should appear next to one instance of the data	No data links created	This was due to the fact I was trying to access vendor.vendor_name which didn't exist. I instead needed to just access vendor within the link data for loop.
DV-06	(Integration) When a piece of data is verified from the data buffer that already exists in user data the two pieces should be merged	Verifying (testingData, ,)	Only one instance of the data should appear with a new link	MergeData() is called before data is put into user_data so data is not merged	Placed mergeData() lower down in the function which worked.
ED-02	(Integration) When a piece of data is edited to equal another they	Editing (dataToEdit, ,) to (editedData, ,)	Vendor1 who is linked to dataToEdit should appear next to editedData	As expected after refresh	

	should be merged		after it has been edited. The dataToEdit instance should disappear from the list.		

Get Awaiting Data Function

ID	Test	Input	Expected Output	Actual Output	Notes
AD-01	(Unit) getAwaitingData() should log all data in the data buffer that is awaiting confirmation.	Calling getAwaitingData()	The console should return a list of two items the first is a count of 0 and the second is a list of all data awaiting confirmation	Function did collect correct information however did not put it into the list outside of the query function.	Made the query a promisified function with callback so I could pass variables outside of it
AD-02	(Unit) getAwaitingData() should log a count off all of the data awaiting confirmation	Calling getAwaitingData()	The console should return a list of two items the first is a count of 1 and the second is the data item awaiting edit	As expected	
IV-08	(Integration) When a decision is requested by the user the getAwaitingData function should be called after the data has been put into the buffer	Requesting edit on data (testData, ,)	The awaiting confirmation count should show 1 on the webpage	As expected	This doesnt update automatically when the decision has been confirmed. This only happens after refresh.
IV-09	(Integration) When a decision is made a list	Requesting edit on data (testDataEdited, ,)	A list item should appear under the awaiting	No list item appears	The template hadn’t been defined

	item of the awaiting data should be shown on the webpage		data button showing the information of the data awaiting confirmation including what decision it's waiting for.		

User Data and Vendor Filter

ID	Test	Input	Expected Output	Actual Output	Notes
FIL-01	(Unit) When building the list instances on both lists each instance should contain a dataset tag with the vendor name(s), data type and data associated with it.	Calling vendorList() and userDataList( )	All instances should show a data tags with vendors, data type and data within the element viewer on browser	As expected	
FIL-02	(Unit) When clicking on the filter button the correct filter tab should show.	Clicking on the filter button	When the vendor list is currently showing the vendor set of filters should be shown.	Both sets were shown at the same time	The hidden class needed to be added by default to the element
FIL-03	(Unit) When filter by vendor is selected and apply is clicked the console should log the value that is in the matching text	Applying vendor1 when vendor is selected	The console should log “vendor1”	As Expected	

	box				
FIL-04	(Unit) When filter by vendor is selected and applied all instances that don't contain the name inputted on the text field should be hidden	Appling vendor 1 when vendor is selected	All vendor instances except for vendor 1 should be hidden	Uncaught TypeError: instance.dataset.vendor.contains is not a function	Changed contains to includes. Hidden class has been added to the correct instances however there has been no effect. Hidden class is overwritten by listItem class. Added !important to hidden css to fix
FIL-05	(Unit) When a filter value is entered that doesn't exist the user should be alerted	Applying vendor4 when vendor is selected	A pop up saying there are no matches to the filter should appear	As expected	
FIL-06	(Integration) Run tests 3 - 5 on data type and data tags	Running respective inputs on the different dataset tags	^	As expected	
FIL-07	(Integration) Run tests 3 - 5 on all data tags on the user data list	Running respective inputs of the different dataset tags within the user data list	^	As expected	

Appendix K: Requirements Traceability Matrix

Requirement ID	Description	Priority	Implementation	Tests	Status
A-01	The user must be able to submit an ID for their manager to a third party for a data request.	HIGH	DMID	DMID-01, RD-01, RD-02, RD-03	Achieved
I-01	The user must be able to see recent data requests and accept or deny them.	HIGH	None	None	Not Achieved
I-02	The user must be able to see a collection of data that is held within the manager.	HIGH	UDL, DV	(UDL-01 to UDL-04), RD-03, ED-01, DD-01, D2UD-01, (MD-01 to MD-05), (FIL-01 to FIL-07), (DV-01 to DV-06), D2B-01, D2B-02	Achieved
I-03	The user must be able to see a collection of all vendors and their ongoing data access.	HIGH	VL	(VL-01 to VL-05), RD-02, RD-03, DV-03, UDL-03, (FIL-01 to FIL-06)	Achieved

I-04	The user must be able to filter third parties based on how long they've had access to their data, what data they are using and when they last used it.	MEDIUM	VFIL	FIL-01 to FIL-06	Achieved
I-05	The user must be able to see information about the consequences of denying a third party access to their data.	MEDIUM	IC	None	Not Achieved
I-06	The user must be able to see recommendations on what decisions they could make.	LOW	DR	None	Not Achieved
I-07	All recommendations and information must be easy to understand.	HIGH	DR	None	Failed

I-09	Key information such as the amount of vendors using data should be clearly displayed.	HIGH	VL, UDL, AD, VC, VFIL, UDFIL	(VL-01 to VL-05), RD-02, RD-03, DV-03, (UDL-01 to UDL-04), DV-04 DV-05, ED-01, DD-01, D2UD-01, (MD-01 to MD-05), (FIL-01 to FIL-07), AD-01, AD-02, IV-08, IV-09, VC-01	Achieved
E-01	Decision making must be simple and streamlined and cannot require too many inputs.	MEDIUM	DD, ED, DELV	DD-01, ED-01, ED-02, (IV-01 to IV-09), UDL-02	Achieved
I-10	Assist the user in comprehen ding the extent of their data.	HIGH	VL, VC, VFIL, UDL, UDFIL	(VL-01 to VL-05), RD-02, RD-03, DV-03, (UDL-01 to UDL-04), DV-04 DV-05, ED-01, DD-01, D2UD-01, (MD-01 to MD-05), (FIL-01 to FIL-07), VC-01	Achieved
I-11	Allow the user to	HIGH	VL, UDL, AD	(VL-01 to VL-05),	Achieved

	monitor the location of their data and how it is being processed.			RD-02, RD-03, DV-03, (UDL-01 to UDL-04), DV-04 DV-05, ED-01, DD-01, D2UD-01, (MD-01 to MD-05), (FIL-01 to FIL-07), AD-01, AD-02, IV-08, IV-09	
E-02	Allow a user to control their data though GDPR rights in a central place.	HIGH	DD, ED, DELV, DR	DD-01, ED-01, ED-02, (IV-01 to IV-09), UDL-02	Achieved
I-12	Inform the users on the impact of their data decisions.	HIGH	IC	None	Failed

Appendix L: Initial Log Of Risks

The risks were minimal for the project as the team consisted of two people and the application was developed on one device.

Table 1
Log Of Risks

Risk	Likelihood	Impact	Mitigation
Device Failure	Low	Delay in development and missed deadlines.	Backup all code and documents. When writing reports or supporting documents work on

			the cloud where possible.
Unable to contact supervisor.	Medium	Work moving out of scope or created at a low quality. Setbacks in progress if large sections have to be edited.	Plan regular meetings with the supervisor ahead of time and development in smaller sections so work does not progress too far during periods of no feedback.
Ill Health.	Medium	Delay in development and missed deadlines.	Allow extra time during sprints so that if a few days need to be taken off they can be.
Software at the end of the development stage is untestable or doesn’t meet the minimum viable product.	Low	No software deliverable and no evaluation of software and requirements.	Plan for each development sprint to create at least one testable feature relating to the requirements.