University of Sheffield

Understand What Big Tech Have About You

Final Dissertation

Hayley Wing Yin Kwok

Supervisor: Dr Andrei Popescu

A report submitted in fulfilment of the requirements
for the degree of BSc in Computer Science with a Year in Industry

in the

Department of Computer Science

May 11, 2022

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Hayley Wing Yin Kwok

Signature: Hayley Wing Yin Kwok

Date: 4 Dec 2021

Abstract

Recently, there have been increasing concerns about companies' data collection, particularly for big tech companies. Users have the right to obtain a copy of their data under data privacy regulations such as GDPR (General Data Protection Regulation). These data, however, are typically returned in a variety of formats and spread across multiple folders and files. It is, hence, very difficult for users from non-technical backgrounds to understand and analyse their data.

The goal of this project is to develop an application that will help users regain control of their data by removing the technical barriers to viewing and analysing the data. The application also aims to provide its service to users in a trustworthy way, along with the freedom to customise data visualisation.

The application was built as a generic data-driven framework with .NET Blazor WebAssembly. It was deployed as a progressive web app and a static GitHub page for beta testing, with positive results.

Acknowledgements

I would like to take this opportunity to thank my supervisor, Dr Andrei Popescu, for the inspiration, guidance, and willingness to answer my never-ending list of questions throughout this project.

I would also like to thank my friends for their encouragement and belief in me to complete this project. I would like to especially thank Maria and Alex for their constant support throughout this year. Thank you for listening to every tiny update of Privasight. I would also like to thank Aiman for the inspiration for the software name Privasight.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project Background

Data is constantly collected when users are surfing the internet, especially when they are using social media. Most social media companies' business models are based on advertisements, which is why they can provide services to users for free, and in exchange they collect their data. The collected data was analysed to tailor services to customers' needs. For example, they recommend products based on what users search for on the internet and display personalised advertisements.

Individuals have the right of access, by which they are given the right to obtain a copy of their data from companies. This right is being protected by different laws across different countries. It is protected, for example, by the UK General Data Protection Regulation (GDPR) [1], GDPR in Europe [2], and the Personal Data Protection Act 2012 in Singapore [3]. A Subject access request (SAR) [4] refers to the act of making a request by or on behalf of an individual for information that they are entitled to request under the right of access. The data returned by these requests is in a variety of data formats and spread across a wide range of folders and files. For example, Google's data involves around 20 folders. The file formats include CSV, HTML, XML, TXT, MBOX, GeoJSON, KMZ, and JSON, etc. The data saved in these formats is not easy to read, making it extremely difficult for users with non-technical backgrounds to understand them. Even though some platforms, such as Facebook, allow users to choose HTML as the return data format, it is still very challenging for users to perform systematic analysis, such as calculating the total number of collected data locations, on these web pages.

1.2 Aims and Objectives

This project aims to assist users in regaining control of their data by providing users with a straightforward way of understanding their SAR data from big tech companies, as well as

the ability to perform data analysis on the data.

This involves creating an application named Privasight which stands for "Insight of Privacy". The application will display the SAR data of the user and will provide various assistance in understanding it. Privasight will, firstly, provide users with an abstract overview of their data in a dashboard with their data organised into various categories. They can, then, view the specific details by clicking on a particular category. In the detailed view, technical terms will be explained. Privasight will also include links that lead users back to their privacy settings on the company's site, allowing them to make amendments easily.

Another main objective of Privasight is to provide its services to users in a trustworthy way. Because the application will have access to personal data, users' trust in Privasight is critical. Privasight will also allow users to select how they want to view their data. These two objectives are the defining factors that set Privasight apart from other applications on the market.

1.3 Overview of the report

Chapter 2 discusses similar software and technology analysis. Chapter 3 dives deep into the project's objectives and the requirements to achieve them. Chapter 4 discusses the design and architecture of the application. Chapter 5 talks about the implementation of the design and requirements. Chapter 6 mentions how testing is done in order to validate the implementation against requirements. Chapter 7 discusses the outcome of the application and the directions for future development. Chapter 8 concludes the project.

Chapter 2

Literature and Technology Survey

2.1 Overview

This section surveys similar software and technology that can be used to develop the required application. Section 2.2 discusses similar software available on the market, and section 2.3 talks about the available platforms and development frameworks for the application.

2.2 Similar Software

This section focuses on analysing the products available on the market. They will be compared by the following criteria:

1. Data Storage

   This is about how the businesses store the user's data as well as the kind of data that is being stored.

2. The value brought by the product

   This concerns the objective of the product and the value that it provides to the user.

3. Platforms in which their services are available on

4. User reviews (if available)

Since online personal data viewing/management is a relatively new topic, there are not many products available on the market. Research has been conducted on the following systems: Jumbo [5], Udaptor [6], Datacy [7], Mine [8] and Rita Personal Data [9]. The latter three will be discussed in greater detail as they show great similarity to the proposed application.

2.2.1 Datacy

Datacy is a platform that enables users to control their data collection and sell anonymized data to specific companies. Users can also control what data is collected and when it is collected while browsing the web or using any app. Their data is tracked through the provided browser extension and web application. After collecting the data, datacy will organise and anonymize it. Users can then choose what information to sell and to whom.

In terms of data storage, according to their privacy policy [10], datacy only collects, uses, and stores users' data with their consent. The collected data will not be used for advertisement and no sensitive identifiable data will be collected. They have implemented a series of industry-standard security procedures to protect the data they process, including encryption in transit and at rest. All stored data is secured through multiple layers of asymmetric encryption. Data is ensured to be secured through the use of blockchain technology and advanced cryptography [11].

Unfortunately, since the product is currently in beta and not publicly available, there are not many user reviews. According to the comments on Product Hunt [12], it appears that many users are interested in using the product.

2.2.2 Mine

Mine is a platform that allows users to take ownership of their digital footprint. They achieved this by analysing the users' email messages' subject lines; the sender's email address; the data of the first email message received from the company; the data of the last email received from the company; the existence of attachments to the email messages; and the first few words of the email as mentioned in their privacy note [13].

Their analysis focuses on the type of data that the user shares with different companies; the frequency of the user using different companies' services; and the possibility of data breach and exposure. The analysed data is demonstrated on a dashboard while the results are grouped into categories. It also provides services that help users exercise their right to be forgotten by companies by helping them to send an email to the selected company with the use of their template.

Regarding data storage, based on their privacy note [13], it is known that a privacy-by-design principle, which has gone through Google's strict external security assessment policy [14], is being adapted. They also use physical, technical, and administrative security measures for the services that they believe best comply with applicable laws and industry standards, and only the minimum necessary data is being collected.

Based on the reviews and comments from Product Hunt [15], the product generally gets positive feedback from its users. The majority of the users are astonished by the insight given by the company, and some of them praise the user interface. There are some reports regarding the right to be forgotten email not being approved by the company holding the

user's data because of the insufficient information provided in the email or the company not trusting Mine. There are also few comments showing concern about the scope of the OAuth access that Mine requires. Currently, for Gmail access, the company requires access to the user's email messages and settings as well as their basic account info, including their primary Google Account email address and personal information. They also ask for write access to send emails to companies on behalf of the user.

2.2.3 Rita Personal Data

Rita Personal Data aims to empower users by providing a secure way of understanding and managing their data. They achieve this through their IOS/Android applications. Their applications give users three steps to get back control of their data.

The first step is "Collect" which allows the user to collect and understand their data. It currently supports two social media platforms, Google, and Twitter. They collect the data by requesting the user to log in to their Google/Twitter account inside the app. The app will then submit a request to download an archive of their data.

The second step is "View" which allows users to get an immediate overview of their digital footprint and understand what companies know about them and how they manage their data. They achieve that by calculating the Rita Score [16], a score that gives the user an understanding of their current situation and how they can improve their digital presence. The score is calculated with three parameters: "Data Ownership" which focuses on the right to access and the right to be forgotten protected by the General Data Protection Regulation (GDPR); "Data Privacy" which is about controlling what data can be accessed by who; and "Profile Safety" which is about the safety of the data storage. Other than viewing their Rita score, users can also see their data value (the amount of money Google earned from the user clicking ads), ads clicked, and companies that have their data, etc. For Twitter data, users can see their data value, a list of topics that Twitter knows they like, and a list of companies that have access to their Twitter account.

The third step is "Control" which allows users to control the access of information by companies. There is a control option that allows the user to remove the company's access to a particular aspect of the data on the view page. For instance, in the "Companies that have a copy of your data" section, the app has a "Restrict" option which helps the user to construct an email to erasure their personal data according to Art.17 GDPR.

In terms of data storage, according to their privacy policy [17], the company's app is privacy-by-design. They promise that the collected data is encrypted and stored on the user's device only. The user is the only one who has access to the data. Also, independent specialists were consulted to get feedback on the design and development of the systems. The only non-anonymised data Rita stores are the user's encrypted login credentials and the time that they created their account. All data is securely stored on an AWS server.

In terms of user reviews, the app received generally positive reviews from Product Hunt

[18], getting a 4.9/5 based on 26 reviews. Many refer this as a long-awaited data privacy management tool and appreciate the fact that their data only stays on their device. However, the app received mixed reviews on Google Play [19]. It received a 2.6/5 based on 322 reviews. There are a considerable number of concerns about logging into their account in the app and the transparency of data storage and processing. The app received a 4.1 out of 5 on the Apple App store [20] with a notable number of 1-star reviews.

2.2.4 Conclusion

Based on the research, it is known that most available applications do not achieve full transparency in data processing and storage. A large proportion of them require access to the user's account for data download. Based on the user reviews, the transparency of data storage is one of the factors users consider the most when picking a data privacy management tool. In fact, it is a factor that will prevent them from using one of these tools.

Other than that, most software focuses on giving users ownership of their data and control over who and what kind of data companies can view. A large proportion of the functionalities focus on allowing users to understand their data in conjunction with the company's defined criteria. It comes to a realisation that these tools usually show a selection of the available data instead of all data. For instance, Rita Personal Data's application only shows a subset of the collected data.

In terms of controlling data shared with companies, most tools achieve this by sending emails to the companies holding the data and changing the privacy settings of the applications from these companies. The most frequent platforms used by those tools are mobile apps and websites.

2.3 Technology Analysis

2.3.1 Introduction

This section focuses on describing the possible tech stack and the available platforms for development. It outlines which tech stack has been chosen for development and the reasons for choosing it.

2.3.2 Available Platforms

As mentioned in section 2.2, users of data privacy management systems are concerned a lot about data storage. Many Rita Personal Data users appreciate the fact that their data never leaves their device. Privasight intends to follow the same practice.

There are three platforms taken into consideration for the development: websites, mobile applications (phones and tablets), and desktop applications. The performance and capacity of local storage will be the primary consideration criterion. After evaluating the data coming

back from the subject access request to different companies like Facebook, Instagram, Twitter, WhatsApp, and LinkedIn, it is concluded that these files are generally large. Hence, relatively large local data storage will be required for the proposed application.

Mobile devices usually have less storage and smaller resolution than computers. Since Privasight aims to give a clear view of the user's data at a glance, it is preferred that the primary platform have a large resolution. For a web platform, the local storage is limited [21]. File System Access API can be the solution, but it is currently not supported by all browsers [22].

After careful consideration, Privasight aims to be a cross-platform application. It will begin as a web app, as it increases exposure and discoverability. Furthermore, being on the web makes installation much easier, increasing the incentive for users to try the application. However, due to the limited performance of browsers, Privasight will focus on delivering its services at full capacity on the desktop platform in the long run. The mobile and website versions may only support a subset of features, directing users to the desktop version for complete access.

2.3.3 Cross-Platform Framework

As stated in section 2.3.2, the application will begin as a web app and will eventually support multiple platforms. A few popular cross-platform frameworks will be discussed in the following sections.

2.3.3.1 Electron

Electron [23] is a framework that allows developers to build cross-platform native applications with web technologies like JavaScript, HTML, and CSS. It essentially enables developers to easily convert existing web apps to native applications.

The benefit is that the finished product can run on Mac, Windows, Linux, and the internet. This framework is used to build many popular applications, including Visual Studio Code and WhatsApp.

The drawback is that since the Chromium engine is used, the generated application is large in size. Electron is often criticised for its performance, RAM consumption, and security.

2.3.3.2 Progressive Web App (PWA)

Progressive Web App (PWA) [24] is a type of web app that takes advantage of both web and native app features. It allows users to install web app that can run offline on their devices.

The upside is the easy installation process. The resulting app is also more discoverable than native apps because it has a stronger presence in search engines. PWA development

requires only one codebase, which can be shared by the website, mobile app, and desktop app [25].

The downside is that it will experience the same local storage issue with the web platform as mentioned in section 2.3.2. It also fails to access all native features on the user's device.

2.3.3.3 .NET Multi-platform App UI (.NET MAUI)

.NET Multi-platform App UI (.NET MAUI)[26] is a cross-platform framework for creating native mobile and desktop apps. With .NET MAUI, developers can develop apps that run on Android, iOS, macOS and Windows from a single shared codebase. With .NET Blazor Hybrid [27], Blazor components can be shared between native and web platforms.

The pros include the ability to use a single codebase for multiple platforms. Even though it is a single codebase shared by multiple platforms, it can still have access to all platform-specific experiences and tools [28].

The cons is that Linux support is a community project. At the moment, the framework and the documentation are still in preview. The release is targeted for early Q2 of 2022.

2.3.4 Conclusion

Following the review of various platforms, it has been decided that Privasight will aim to be cross-platform. This project will concentrate on web platform development as it has the most exposure and is the easiest to install among all platforms given the current time constraints.

Microsoft intends to officially release .NET MAUI in early Q2 2022. According to the latest information, the migration of websites created in .NET Blazor to .NET MAUI applications should be straightforward, as evidenced by the official Microsoft introduction [27].

Hence, before the official release, the development will use .NET Blazor WebAssembly [29] which is a single-page app (SPA) web framework that performs most of the user interaction logic within the client's browser [30]. The final product will be available as a static website and a PWA. If time permits, the application will be ported to a desktop application using .NET Blazor Hybrid or Electron; otherwise, it will be available only as a static website and PWA.

Chapter 3

Requirements and Analysis

3.1 Introduction

Privasight's goal is to help users understand their subject access request (SAR) data, regardless of their technical knowledge. It also intends to provide its service in a way that users can rely on, as well as to give users the freedom to customise the data visualisation. This chapter focuses on the application's characteristics and requirements and how they help Privasight achieve its objectives.

The project scope is discussed in section 3.2. Section 3.3 outlines the functional requirements that enable Privasight to achieve its functional objectives. Section 3.4 discusses about the requirements needed to make Privasight trust-worthy.

3.2 Project Scope

In this project, Privasight will focus on Facebook data because there are currently no applications available for Facebook SAR data viewing. It also has the simplest data structure among the companies studied, which include Google, Facebook, Instagram, Amazon, Twitter, WhatsApp, and LinkedIn. Hence, Facebook data will be a good starting point for Privasight.

Due to the time constraints, this project will focus on developing as many features as possible. Instead of reaching their full potential, the implemented features will serve as proof of concept. For example, rather than devoting all resources to implementing the visualisation of every file in the returned zip file, Privasight will only implement a subset of it as a prototype. Three files were selected for this project. They are chosen because they are least known by users and are the most relevant to the project's objectives. Users will be told to only download these files from Facebook. Figure3.1 illustrates the expected file structure of the returned zip file. It contains two folders containing the JSON files. The JSON files will contain a single list of objects

of the same type. For instance, the ads_information/advertisers_you_'ve_interacted_with.json file contains a single list called history_v2. There are two string properties and one timestamp property for each item in the list.

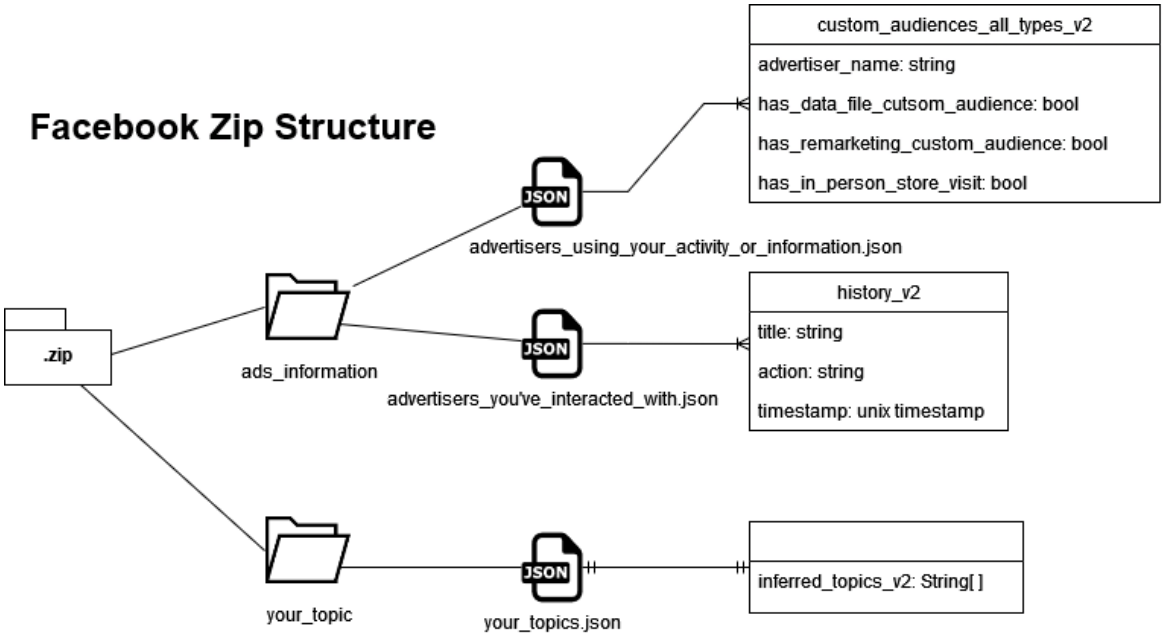


Figure 3.1: Data Structure of the expected zip file from Facebook

Privasight's long-term goal is to support multiple companies. As a result, the development focus will be on creating a generic data-driven framework. The implementation of features will strive to be as generic as possible to avoid the dependency on the data returned by Facebook.

3.3 Functional Requirements

In terms of functionality, Privasight has two goals to achieve. It aims to help users understand and analyse their data while giving them the freedom to customise how they view it. This section will discuss in detail how requirements are designed to accomplish these objectives.

3.3.1 Assist Users to View and Analyse Their Data

This objective is about how to assist users, particularly those from non-technical backgrounds, in understanding and analysing their data. The requirements for this objective are classified into three priority groups: high, medium, and low.

3.3.1.1 High Priority Requirements

This section will discuss the high priority requirements that define the success for the implementation of this objective.

Read in the return data from Facebook:

This involved taking in a zipped file from the user, unzipping the file and ingesting the JSON objects to C# objects. For this project, only a subset of the files will be processes as mentioned in section 3.2.

Create display components for the ingested data:

This function is about displaying the ingested data. The ingested data will be classified into categories based on the nature of the data. Each data category will be displayed using a card component. A card component is made up of two parts. The first part will be a summary of the data. The satisfying criteria will be a numerical summary. The second section will be a detailed view of the category. The design of the card component will be covered in greater detail in section 4.4.5.

Create description on how to download the required Facebook Data:

Because Privasight requires users to download data from Facebook and upload it to the site, it is critical that they understand how to properly download the data. This requirement is about providing detailed instructions to download the necessary data from Facebook.

Transform data to a non-technical format:

Some data is saved in a technical format. For instance, timestamps are saved in Linux timestamp format and the non-English characters are not being properly encoded. It is important to convert these back to the human-understandable format so as to assist users to understand their data.

Show all loaded data on a default dashboard:

Before users start to create their dashboards, it is vital to have a default dashboard serving as an example of what kind of data and visualisation are available. The default dashboard will show all available data. The visualisation will depend on the nature of the data.

3.3.1.2 Medium Priority Requirements

This section will discuss the medium priority requirements that will have some impact on the success of the objective's implementation.

Explain the technical terms:

Some data consists of complicated terms, such as custom audience. It is important for the application to include an explanation of these terms to assist users in obtaining a complete

understanding of their data. The HTML version of the returned SAR data includes a detailed description of each data category. The explanations will be mostly copied from there.

Compare SAR data received on different dates:

This function will allow users to see the trends and differences in various categories of the collected data. This can allow users to see the effect of a change in their privacy settings. For instance, users can see if the number of location points has changed after changing some location collection settings. This can help users to fine-tune the information fed by the platform and help them to achieve the balance between convenience and privacy. For example, by adjusting ad topics, Facebook can bring in more relevant advertisements for the user. On the technical aspect, this will involve data storage to store the ingested data so that the comparison can be done.

3.3.1.3 Low Priority Requirements

This section will discuss the low priority requirements that will have little impact on the success of the objective's implementation.

Links to change the privacy setting at each corresponding section of the dashboard:

This function is relatively less important since there are already many tools in the market helping users to change their privacy settings.

3.3.2 Ability to customise how to view the data

This objective is about giving users the ability to customise how they view their data. Since there is a lot of data available, users may not want to see it all on a single dashboard. Privasight intends to provide a variety of visualisations. A user may wish to view data in a different manner than the default display. That is why Privasight decided to allow users to customise their data visualisation. This is also the key distinction between Privasight and the other applications on the market.

Privasight aims to achieve this by allowing users to create a series of dashboards with custom settings. Users will be able to design their own dashboards in order to better analyse their data. Creating a custom dashboard entails selecting which categories of data to display on the dashboard as well as how they are displayed. Users can design a series of dashboards that are tailored to their needs.

3.4 Non-Functional Requirements

Privasight is requesting sensitive information from users. User trust in the application is critical above all else, as it ensures its usability by the users who, without trust, would

refuse to use it. Privasight intends to meet this non-functional requirement by remaining transparent, secure, and localising all processes.

3.4.1 Transparency

As concluded in section 2.2.4, transparency in data processing is the main factor that users consider when deciding whether or not to use the application. Privasight aims to be as transparent as possible in how it processes users' data. This is achieved through these features.

Privasight will explain in detail how it handles users' data on a static page, possibly on the first page they see when they visit Privasight.

The app will also strive to be open-sourced from the start. This allows users to view the source code, making it transparent.

Privasight will not request access to the user's account to obtain the SAR data. This decision is based on the suspicious reviews on services requesting access to the user's account, as discussed in sections 2.2.2 and 2.2.3. Privasight will ask the user to download their data and upload it to the site. As a result, users would have complete control over what Privasight has access to and would not have to wonder what kind of data they are actually giving Privasight by granting it access to their account.

3.4.2 Locality

The findings in section 2.2 indicate that users appreciate local data processing. This leads to the decision to keep all processing within the user's browser. With WebAssembly, it is made possible to create a static web app without the need of a central server for data processing.

To achieve full locality, data must also be stored locally. There are several options for storing user data within the browser. After investigation, IndexedDB [31] and Web Storage API[32] were selected for consideration. It will be covered in greater detail in section 5.4.

3.4.3 Security

Since the data being stored is sensitive personal data, it is critical that it is kept securely, especially since it is stored in the user's browser. Privasight aims to protect user's data by encryption. Encryption is the process of scrambling text into an unreadable format known as cipher text. Only the person who has the decryption key can restore the data to its original format [33]. Privasight will make use of symmetric encryption, which uses the same key for encryption and decryption, to protect the user's data. It will request a password from the user for encryption, and that will make the user the only person that can access their data.

3.5 Conclusion

This section has discussed different requirements and how they are designed to meet the three objectives, which are: assisting users to understand their SAR data; making users trust Privasight; and allowing users to customise data visualisation. Below is the summary of the requirements discussed. They were ranked based on their priority. Requirements with a high priority have a direct impact on the project's success. Medium and low have little to no effect.

ID	Requirement	Priority
1	Read in the return data from Facebook	High
2	Create display components for the ingested data	High
3	Create description on how to download the required Facebook Data	High
4	Transform data to a non-technical format	High
5	Show all loaded data on a default dashboard	High
6	Customise data visualisation	High
7	Explain the technical terms	Medium
8	Compare SAR data received on different dates	Medium
9	Links to change the privacy setting at each corresponding section of the dashboard	Low

Table 3.1: Summary of the Functional Requirements

ID	Requirement	Priority
1	Get user's data only by requesting them to upload the required files	High
2	Keep all processing within the user's browser	High
3	Store all data locally	High
4	Encrypt user's SAR data	High
5	Explain in detail how Privasight handles user's data on a static page	Medium
6	Be open-sourced	Medium
7	Make Privasight be a generic data-driven framework	Low

Table 3.2: Summary of the Non-Functional Requirements

Chapter 4

Design

4.1 Overview

Design is one of the most important aspects of development because it defines the structure of the codebase and provides a basic guideline for implementation. Section 4.2 discusses the data structures required for the application. Section 4.3 describes the architecture of the application and how files are organised in the codebase. Section 4.4 discusses how the various functionalities are visually organised.

4.2 Data Structures

The application will handle two types of data: the subject access request (SAR) data and the dashboard settings data. They will be saved in the browser's storage. The logic of accessing and storing will be handled by the data services, which will be described in depth in section 4.3.4.4. The following sections focus on describing their data structures.

4.2.1 Subject Access Request Data

The JSON files returned by the Facebook zipped file have varying data structures. To convert the JSON objects to C# objects, a corresponding data class for each JSON file needs to be created. Interfaces and abstract class were developed to promote abstraction and decouple the data ingestion and display from the actual data classes. They are thoroughly described in the section 4.2.1.1. Attributes are used to decouple the display logic from the data. Their data structures are described in section 4.2.1.2 and their actual effect on display is described in section 4.4.5.3. Section 4.2.1.3 discusses how the actual data classes use the attributes, interfaces, and abstract class described in the previous sections.

4.2.1.1 Abstraction of the Data

Because all data classes need to have the Title, Description, and FilePath properties for the program's functionality, the IFileWrapper interface was created to enforce this logic. When ingesting JSON files into the CompanyDataService, the static property, FilePath, will be used. Section 5.7 will go over the ingestion process in greater detail. The card component will use the Title and Description properties for display. This will be covered in greater depth in section 4.4.5.3.

This project covers three Facebook files, as mentioned in the section 3.2. As shown in figure 3.1, they are files with a single list of objects of the same type. As a result, the ISingleItemListFile<T> interface was created to represent this type of data. The display of data, which implements this interface, is described in greater detail in section 4.4.5.3.

Each item in the list will need to have a property for storing the zip file's generation date for version control. To reduce code repetition, an abstract class called DbTableObj was created. A constraint is added to the item type, T, in the ISingleItemListFile <T >interface to represent this connection.

The ISingleItemListFile interface was created to help with the type check for ISingleItemListFile<T> because, unlike Java, there are no generic wildcards for type checking in C#. It is not meant to be implemented directly.

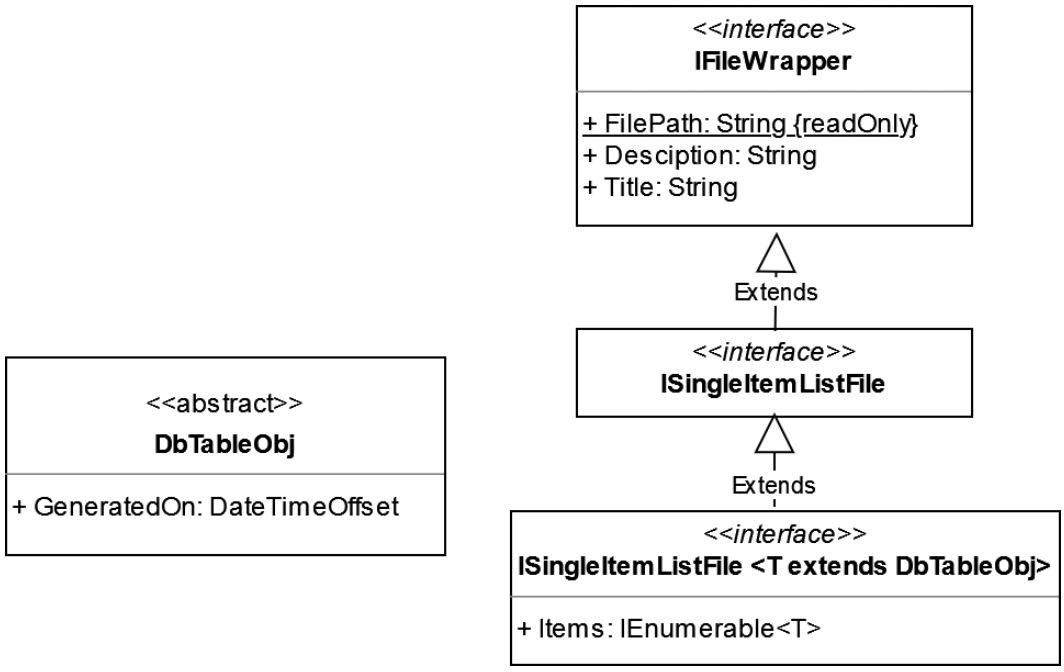


Figure 4.1: Class Diagram for the required interfaces and abstract class

4.2.1.2 Attributes for Display

Two attributes were created to decouple the detailed view of the card component from the data class. Section 4.4.5.3 will go into more detail on how the attributes affect the card's display. The emphasis in this section is on the data structure.

There are no properties in the DataListValueAttribute. That is because it is only used to indicate which property in the class the program needs to inspect when creating a DataList. This attribute will be applied to only one property in the class.

For DetailedTableDisplayDataAttribute, it has two properties. It is assumed that all properties in the item type of the list will be marked with this attribute. DisplayName is required for all properties, and Description is optional.

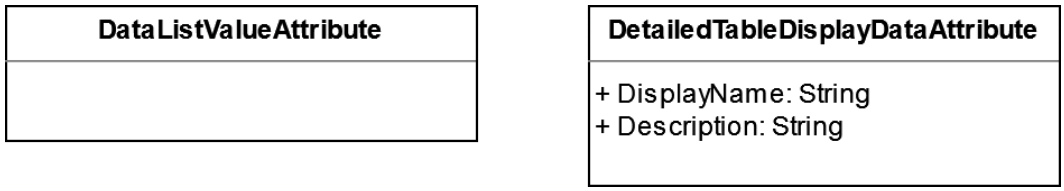


Figure 4.2: Class Diagram for the attributes that affect the display of card components

4.2.1.3 Concrete Classes for the Data

Concrete classes were created to represent the JSON files presented on figure 3.1. advertisers_you_'ve_interacted_with.json, advertisers_using_your_activity_or_information.json and your_topics.json is represented by AdvertiserYouInteractedWith, AdvertisersUsingYourActivity and InferredTopics respectively. All of these classes implement the ISingleItemListFile<T> interface and are used for the JSON deserialization and data storage.

All item classes in the list implement the DbTableObj. SharedAdvertiser and InteractedAdvertiser are created based on the data structure of the corresponding JSON object outlined in figure 3.1, with an additional field GeneratedOn. The item type of InferredTopics is a special case. Since the original data type in the JSON list is a string, which is a primitive type, the program cannot inject the GeneratedOn field into this type. The StringWrapperDbObj class and a corresponding JsonCoverter, StringWrapperDbObjConverter, were created to turn the item type from a string to an object with a string and the GeneratedOn field during the JSON deserialisation.

The properties that are marked with the colour blue in the item types are the properties that are marked with the DataListValueAttribute as mentioned in section 4.2.1.2.

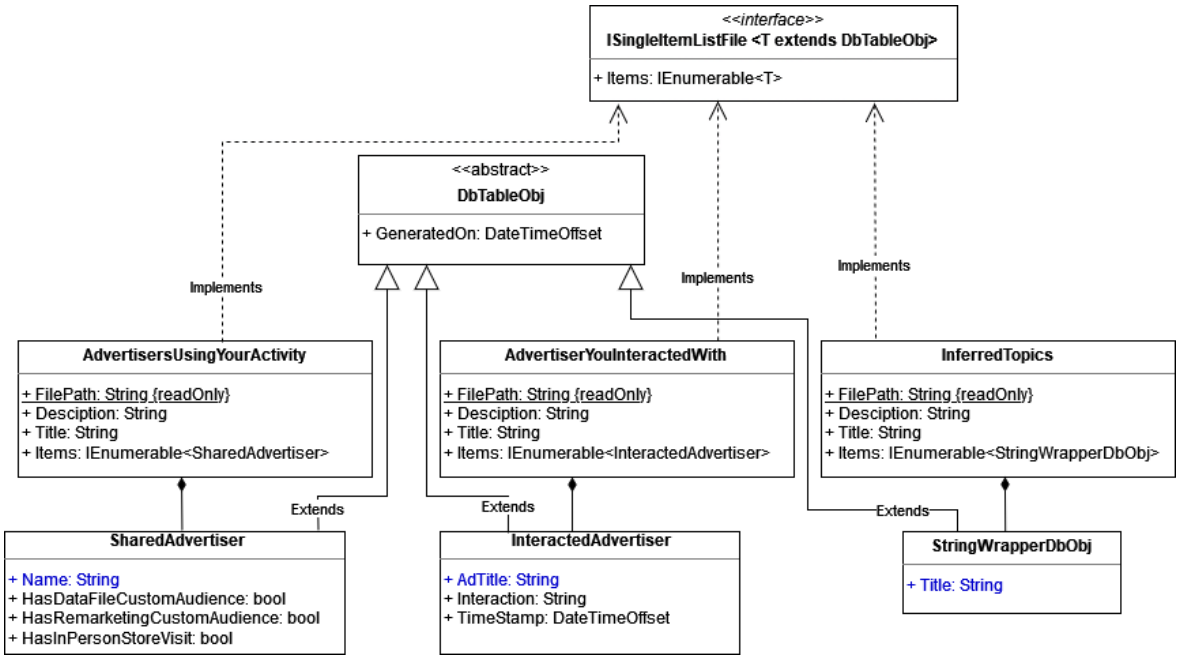


Figure 4.3: Class Diagram for the concrete data type of the SAR Data

4.2.2 Dashboard Settings

Privasight makes use of a dashboard to display the SAR data from the user. The details of the dashboard display will be covered in section 4.4.4. This section focuses on the data structures of the objects used to control the display.

On a dashboard, there are cards for the display of each JSON file in the SAR Data. In the context of this project, a JSON file in the SAR data is known as an object implementing the IFileWrapper interface, or simply a FileWrapper. The SAR data is known as a list of FileWrappers.

A DashboardSetting object is used to store the configuration of a specific dashboard. It is used to control how the SAR data is displayed on a dashboard. DashboardSetting is made up of a string property for the dashboard's title and a hash set of CardSetting objects.

A CardSetting object specifies how a card displays the corresponding FileWrapper. The data in the CardSetting object is in charge of three distinct tasks. FileWrapperTypeName determines which FileWrapper to retrieve from the CompanyDataService which is the service for accessing the SAR data (more information in section 4.3.4.4), whereas FileWrapperTitle is used for display. CardType controls the display of the abstract view of the card. Dialog is used to indicate the type of pop-up dialog to be used to display the details of the underlying list of objects stored in FileWrapper.

Both DashboardSetting and CardSetting will be implemented using record type [34] in C# since they are used solely for storing data. Record type is usually used when the fundamental

use of a class is for data storage since it follows value-based equality semantics.

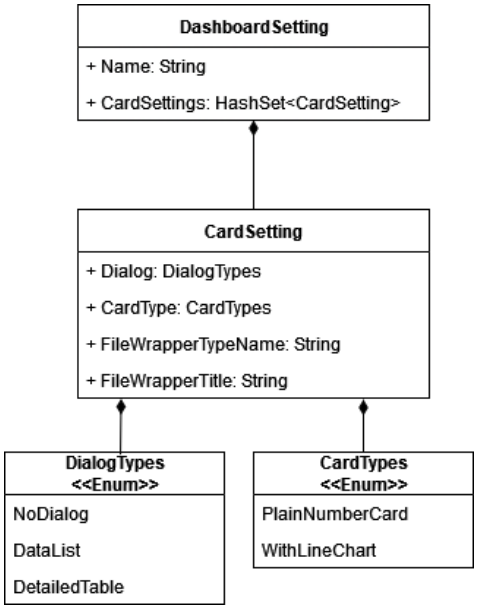


Figure 4.4: Class Diagram for Dashboard Setting

4.3 Software Architecture

This section talks about the architecture of the code base. It will start from the overall architecture of the code base (section 4.3.1) and dive deep into the architecture of each package (sections 4.3.2, 4.3.3, and 4.3.4).

4.3.1 Overall Architecture

The source code for this solution (codebase) is distributed into three packages to decouple the program's logic from the actual data. As outlined in figure 4.5, there are three packages in this solution, which are Privasight.Wasm, Privasight.Model.Facebook, and Privasight.Model.Shared. The figure also shows the dependencies of the packages.

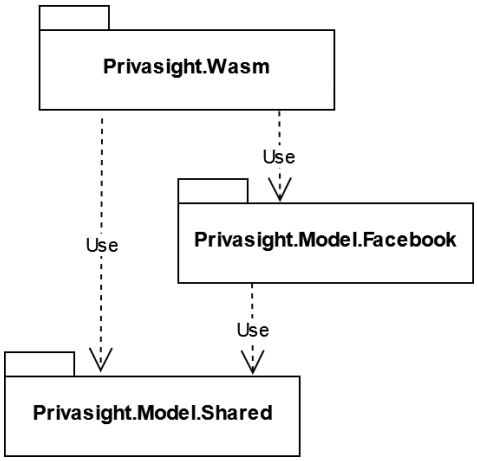


Figure 4.5: The high-level outline of the packages in the solution

4.3.2 Architecture of the Shared Model Package

Privasight.Model.Shared is a class library. It is used to store the model classes that are not specific to any company. It also contains the code that could be reused when Privasight expands to other platforms. It can avoid the dependencies between the packages of the other platforms and the WebAssembly (WASM) package.

This package has four folders, as shown in figure 4.6.

The DataStructures folder contains shared interfaces, abstract classes, and data classes that are not company specific. The specifics of these files were discussed in section 4.2.1.1.

The Converters folder contains the converters required for the custom logic for JSON de-serialization. The details of the StringWrapperDbObjConverter were described in section 4.2.1.3. Section 5.10 will go over the details of the rest.

The Display folder is where all data classes related to the display logic are kept. They were covered in greater detail in the sections 4.2.1.2 and 4.2.2.

The Helpers folder is used to store static helper classes that contain logic not specific to the WASM package.

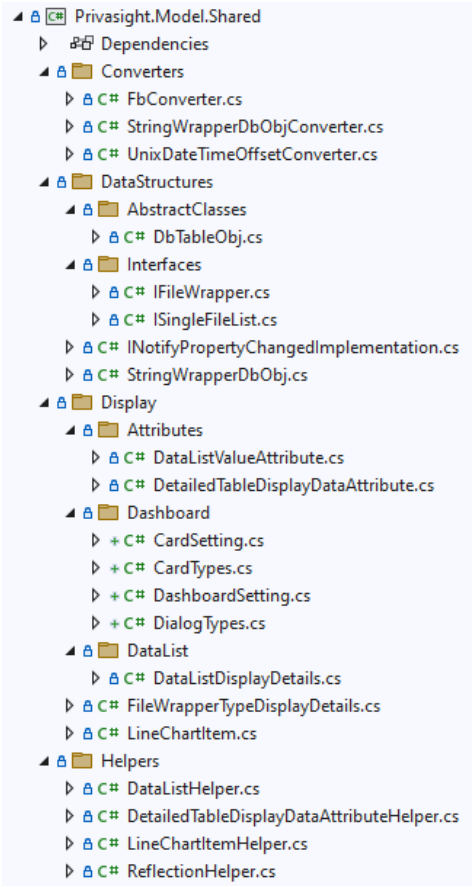


Figure 4.6: File structure of the Shared Model Package

4.3.3 Architecture of the Facebook Model Package

Privasight.Model.Facebook is a class library. It is used to store the model classes that are specific to Facebook. It has a dependency on the Privasight.Model.Shared package for the shared interfaces, classes, and converters for the creation of the concrete data classes as mentioned in section 4.2.1.3.

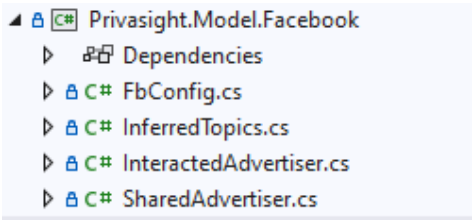


Figure 4.7: File structure of the Facebook Model package

The FBConfig class (as seen in figure 4.8) is used to store the company-specific configurations needed for data loading. It has one property, AvailableFileWrapper, to store the file path

and type of all the available concrete classes from this model package.

```
public static class FbConfig
{
    /// <summary>
    /// The supported data type (FileWrapper) for the FB zip file
    /// Key: filepath of the FileWrapper in the zip file
    /// Value: type of the file wrapper
    /// </summary>
    public static readonly Dictionary<string, Type> AvailableFileWrappers = new()
    {
        { AdvertisersUsingYourActivity.Filepath, typeof(AdvertisersUsingYourActivity) },
        { AdvertiserYouInteractedWith.Filepath, typeof(AdvertiserYouInteractedWith) },
        { InferredTopics.Filepath, typeof(InferredTopics) }
    };
}
```

Figure 4.8: FBConfig Class

4.3.4 Architecture of the WebAssembly Package

Privasight.Wasm was created with the Blazor WebAssembly App package template. All logic regarding the website is stored in this package. It has dependencies to the other two packages. Inside this package, there are six folders. Each folder has its separate responsibility which will be discussed in its own section.

4.3.4.1 wwwroot

This folder is used to store static content needed for the site. It includes the media files and the configuration files for hosting. It also includes the JavaScript file, encrypt.js, that is used for data encryption and storage. Section 5.6 will go into detail about the implementation of this file.

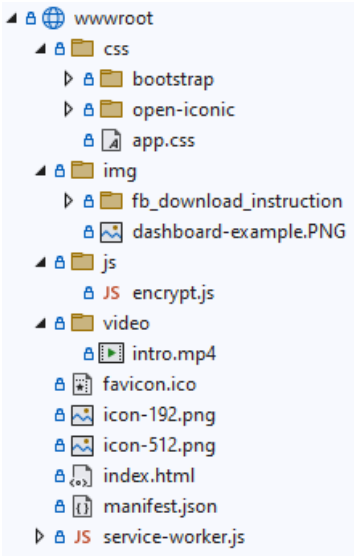


Figure 4.9: File structure of wwwroot folder in the WebAssembly package

4.3.4.2 Configs

It is critical to reduce dependency on company-specific data since this project aims to be a generic data-driven framework. This folder is the only location for storing all unavoidable company-specific dependencies. There are two files in this folder: AvailableCompany and CompanyConfigs.

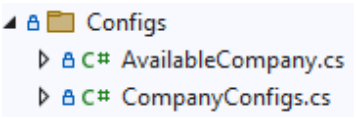


Figure 4.10: File structure of Configs folder in the WebAssembly package

AvailableCompany is an enum that specifies which companies are supported by the system. This enum currently only has one member, Facebook.

CompanyConfigs (as seen in figure 4.11) is used to store the company-specific data types. Its two properties are AvailableFileWrappers and DataDownloadInfo. AvailableFileWrappers stores the type information of the concrete data types and is the reason for the dependency on the Privasight.Model.Facebook package. The details of FBConfig.AvailableFileWrappers was described in section 4.3.3. DataDownloadInfo points the program to the correct UI component for company-specific data download instructions.

```
namespace Privasight.Wasm.Configs
{
    4 references
    public static class CompanyConfigs
    {
        3 references
        public static Dictionary<AvailableCompany, Dictionary<string, Type>> AvailableFileWrappers => new()
        {
            { AvailableCompany.Facebook, FbConfig.AvailableFileWrappers }
        };

        1 reference
        public static Dictionary<AvailableCompany, Type> DataDownloadInfo => new()
        {
            { AvailableCompany.Facebook, typeof(FBDownloadInstruction) }
        };
    }
}
```

Figure 4.11: CompanyConfigs Class

4.3.4.3 Pages

This folder is used to store the pages that the user has access to through the navigation menu. A page is typically made up of multiple UI components from the UI folder. If the required UI functionality/logic is simply placed on the page, it is usually because the page is mostly static content or the logic is tightly coupled to the page.

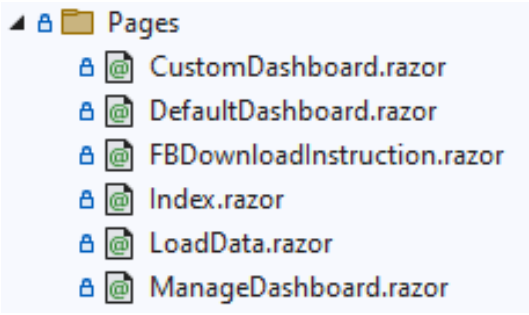


Figure 4.12: File structure of Pages folder in the WebAssembly package

4.3.4.4 Services

Dependency Injection [35] is a technique for accessing services configured in a centralised location. For this package, all services are configured in Program.cs, which is the start up program. There are two custom services with a Scoped lifetime: CompanyDataService and DashboardService. This means that all components that require the service will receive the same instance of the service. These custom services are in charge of everything related to the two types of data that the application will handle, as discussed in section 4.2.

CompanyDataService and DashboardService are responsible for the SAR data and dashboard settings data respectively. CompanyDataService has a collection of FileWrappers, whereas DashboardService has a list of DashboardSetting. These collections are grouped by the

members in the AvailableCompany enum as mentioned in section 4.3.4.2. The implementation of these services will be discussed in the section 5.5. This section focuses on a higher-level view of these services.

Whenever any component wants the SAR data or Dashboard Settings, they will go and fetch them from these data services. These services will also include methods for the data persistence logic.

ServiceUsingIndexedDb is the root class storing the shared code for the two services.

DataLoadInHelper is a static helper class that has methods to facilitate the data loading process. The details of it will be discussed in section 5.7.

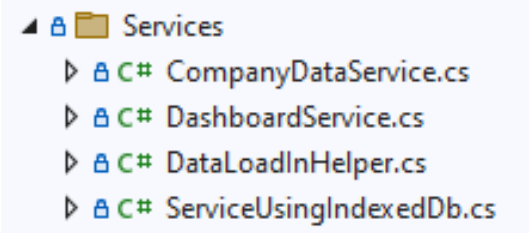


Figure 4.13: File structure of Services folder in the WebAssembly package

4.3.4.5 Shared

This folder comes with the development template. These files are used for the layout of the application. MainLayout defines the main layout, and NavMenu defines the navigation menu. NavMenu is used as the main navigation tool for users to navigate between different pages. Section 5.3 will go over its implementation.

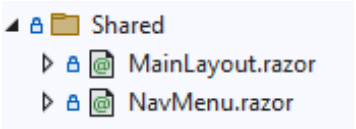


Figure 4.14: File structure of Shared folder in the WebAssembly package

4.3.4.6 UI

This folder contains the UI components. A vast majority of the files are associated with the card component. Because the card component is in charge of data display, in order to give the user the freedom to customise how they view their data, this component must be highly modular, which is why it is broken down into many pieces. Each piece has a single responsibility. For instance, LineChart is only responsible for the logic of generating a line chart.

The remaining UI components are moved from their pages to this folder, mostly because they are reused on other pages or to break down the lengthy logic within the page.

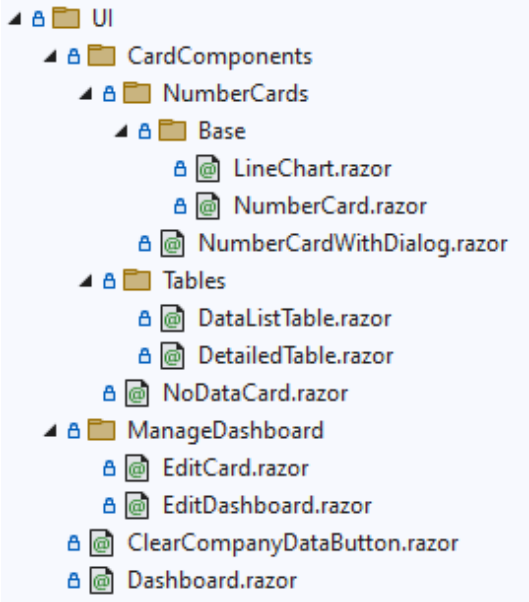


Figure 4.15: File structure of UI folder in the WebAssembly package

4.4 User Interface Design

This section talks about the user interface and front-end design of Privasight. It will go in details on how functionalities are organised into different pages and components.

4.4.1 Overall Structure

The view of the application is divided into two parts: the navigation menu and the main display area. The navigation menu, which can be seen on the left of figure 4.16, allows users to switch between different pages that correspond to different functionalities. When a page is selected, its background colour on the navigation bar will change. The main display area of the page is on the right. There will be a page title at the top and a display area at the bottom for displaying what is required for that page.

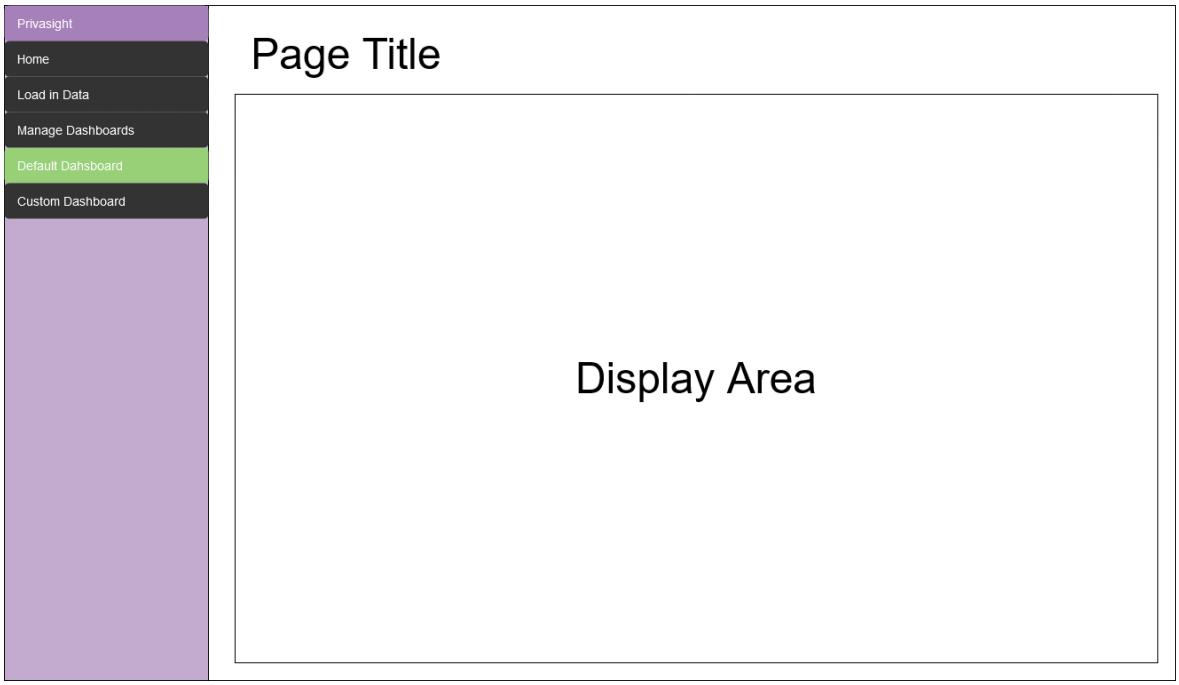


Figure 4.16: Mock-up design for the overall structure of the website, with Default Dashboard page being selected

4.4.2 Home page

This is the site's index page, which also serves as the site's landing page. This page will include information about the objectives of Privasight, a video introduction to Privasight, and a user guide on how to use Privasight. These intend to help users understand what Privasight is.

There is also a section that explains how Privasight manages the user's data. As stated in section 3.4.1, the app aims to gain user trust by detailing how the site handles their data. This section is intended to be the location for implementing this requirement.

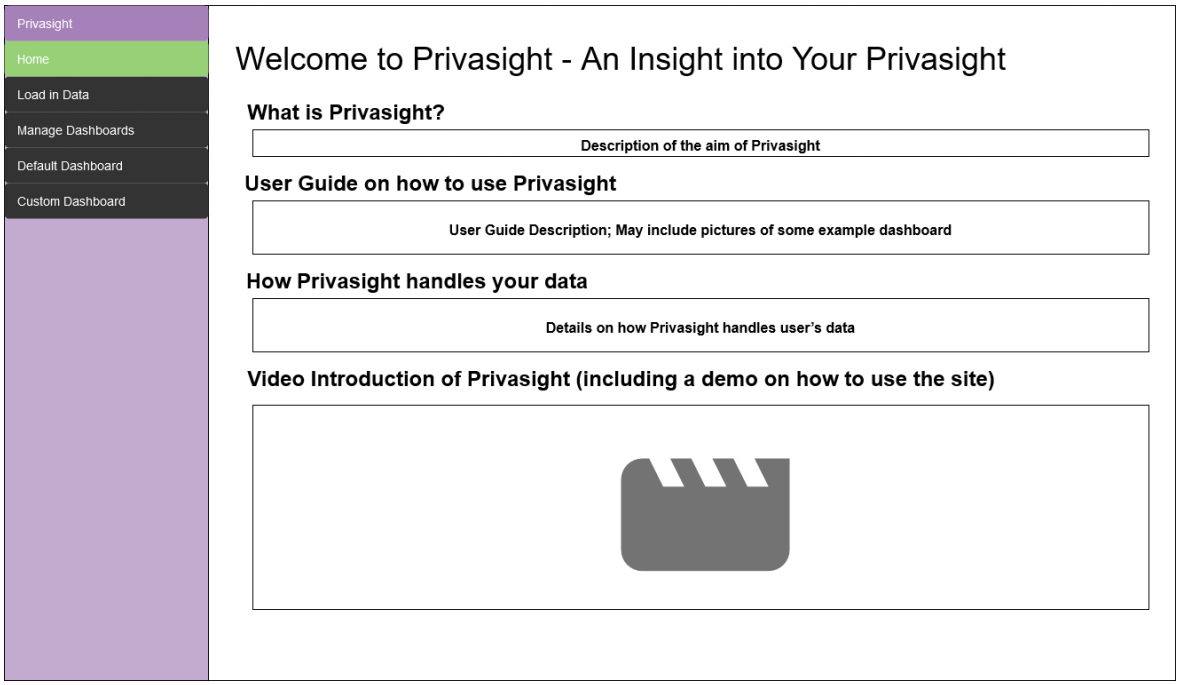


Figure 4.17: Mock-up design for the home page of the website

4.4.3 Load in Data page

This page is designed for users to upload their SAR data into the site. There will be a data loading component for implementing the functional requirement 1 "Read in the return data from Facebook". There will also be a component for implementing the functional requirement 3 "Create description on how to download the required Facebook Data". It will include the instructions for users to download the required data from Facebook.

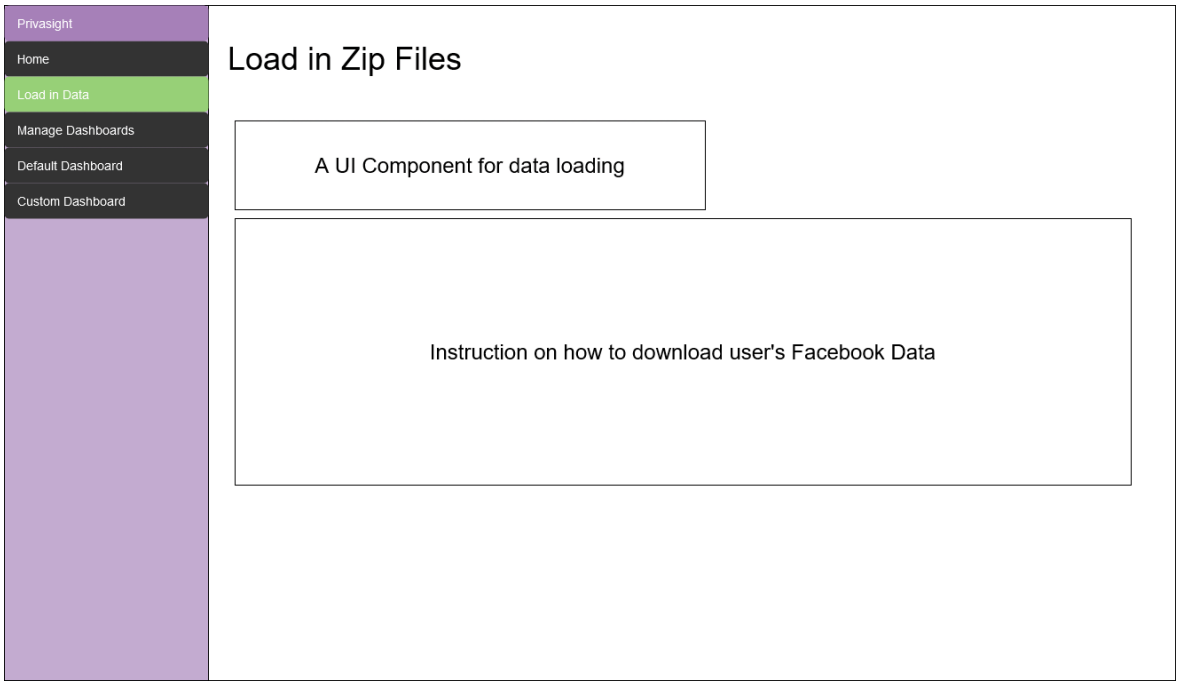


Figure 4.18: Mock-up design for the load data page

4.4.4 Dashboard Component

Privasight aims to help users understand their data by first providing an abstract overview on a single page. This abstract overview aims to allow users to have an overall summary of what is happening with their data at one glance. Privasight will achieve this through a dashboard with data organised into different categories shown as cards on the dashboard. Figure 4.19 shows the mock-up design for the dashboard.

The dashboard component requires two inputs: a DashboardSetting object, as discussed in section 4.2.2, to determine what cards to create and the details for the creation; and the corresponding Company, which is of type AvailableCompany as discussed in section 4.3.4.2. The Company property is used to determine which company's data in the CompanyDataService to retrieve in order to feed the corresponding FileWrapper object to the card component.

The dashboard component is used on the "Default Dashboard" and "Custom Dashboard" pages. Section 5.9 will discuss in details how these pages create the required input for the dashboard component within the page.

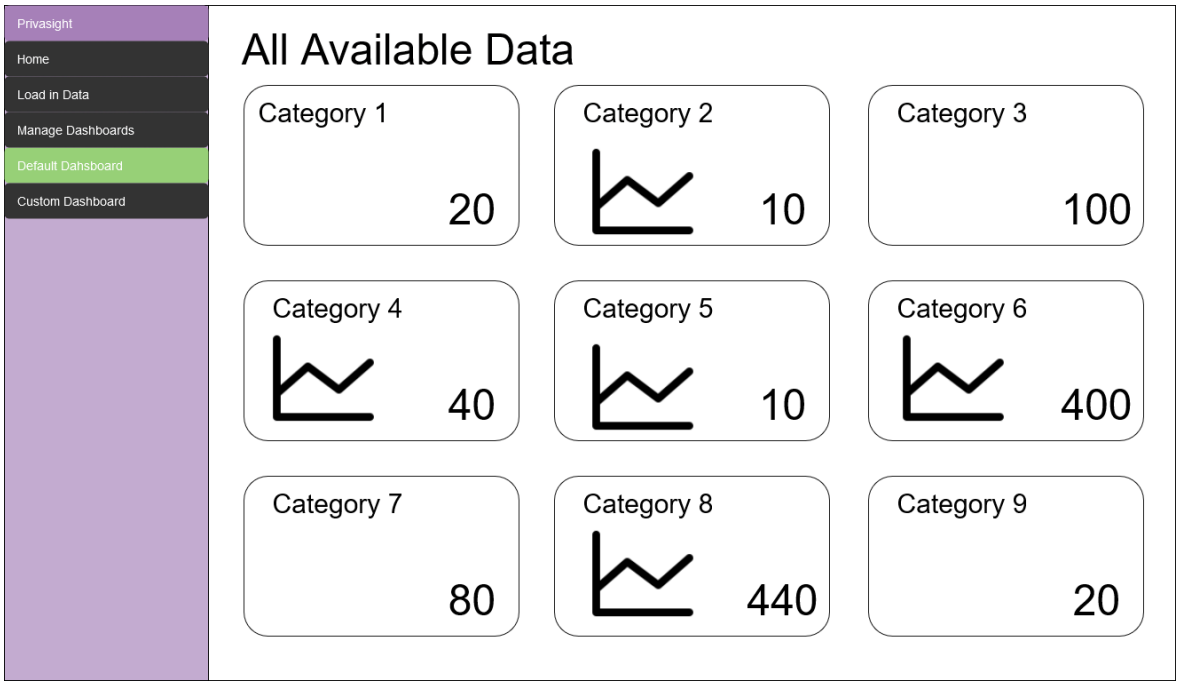


Figure 4.19: Mock-up design for Dashboard Component

4.4.5 Card Component

Inside a dashboard, there are different card components. One card will corresponds to one JSON file in the Facebook zipped file, aka a FileWrapper. A card component is responsible for two things: showing an abstract overview of the FileWrapper and showing it in details.

A card component is divided into two parts, the abstract view shown on figure 4.20 and the detailed view shown on figure 4.21a. The abstract view is the view shown on the dashboard as seen on figure 4.19. When the user click on the card, the detailed view as shown on figure 4.21a will show up as a pop-up dialog.

The abstract and detailed views are further divided into different components to allow for more flexible changes.

4.4.5.1 Inputs Parameters

As mentioned in section 4.2.1.1, all FileWrappers in this project are of type ISingleItemListFile<T extends DBTableObj>. In the current scope, the card component will only accommodate FileWrappers that implement this interface. In the future, the card component will be further generalise to accept all types of FileWrappers.

The card component takes in three inputs: a SingleItemListFile of type ISingleItemListFile<T extends DbTableObj>, a DialogTypes property called ChosenDialog, and a CardTypes property called CardType. These inputs are used to control how data is displayed on the corre-

sponding card.

4.4.5.2 Abstract View

There are three distinct components in the abstract view illustrated in figure 4.20.

The first is the category title, which has a green outline on the figure. It corresponds to the Title property of the input SingleItemListFile.

The chart component, which is highlighted in purple, is used to provide chart for the data overview. For this project, it will be a line chart with the generation date and the count of the list grouped by the date to demonstrate the overall trend of the data. The appearance of it will be determined by the CardType property.

The number component, which is highlighted in orange, is used to display a numerical summary of the list of data stored in SingleItemListFile. The number will vary depending on whether the line chart display was chosen for the card. If the line chart is selected, the most recent count on the list will be shown; otherwise, the total count of the data across all generation dates will be shown.

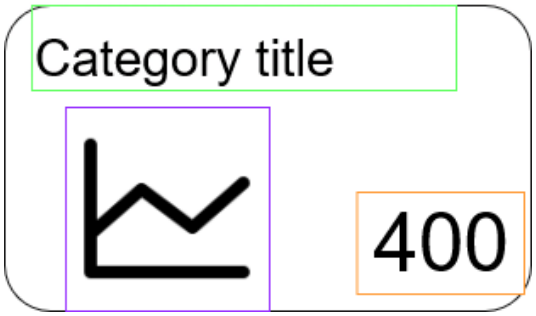


Figure 4.20: Mock-up design for Abstract view of Card Component

4.4.5.3 Detailed View

There are four parts to the detailed view, as shown in figure 4.21a.

The one at the top is the category title, which corresponds to the SingleItemListFile's Title property, as mentioned in section 4.2.1.1.

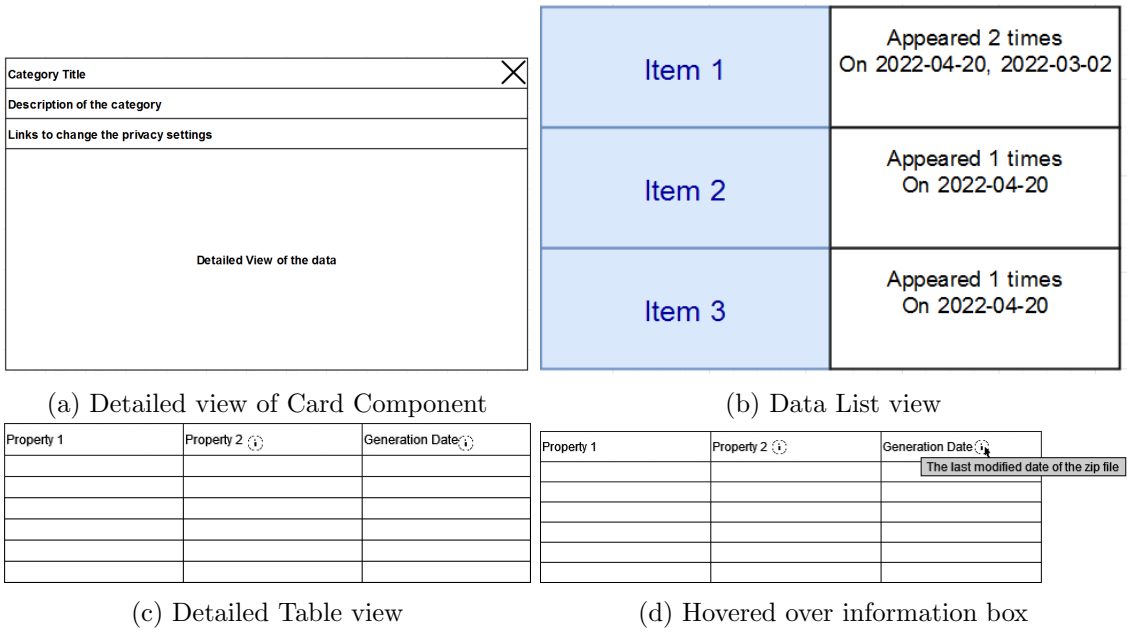
The links to change the privacy settings are grouped with the category description and saved in the SingleItemListFile's Description property.

The final component, Detailed View of the data, is used to display the details stored in the SingleItemListFile. Two views are designed for this type of data: Data List view (shown in figure 4.21b) and Detailed Table view (shown in figure 4.21c).

The Data List view focuses on showing the data with version control. The left column will be the value of the property marked with the DataListValue attribute (as mentioned in section

4.2.1.2) in the item type (T). The right column will show the number of appearances of that value in the entire list across different generation dates and the dates that it appeared on.

The Detailed Table view focuses on showing the user everything. It will display all the properties of the underlying object. The data saved in the DetailedTableDisplayDataAttribute determines the header of each column in the table. This attribute stores two properties, DisplayName and Description, as mentioned in section 4.2.1.2. As an example, the "Generation Date" shown in the last column of the table in figure 4.21c would be the DisplayName stored in the attribute. When the user hovers over the "i" symbol, the Description from the attribute appears for more information about the property, as shown in figure 4.21d. The "i" symbol will only appear if the property has a Description.



(a) Detailed view of Card Component

(b) Data List view

(c) Detailed Table view

(d) Hovered over information box

Figure 4.21: Mock-up design for Detailed view of Card Component

4.4.6 Manage Dashboard Setting Page

This page allows users to create and manage the settings for their customised dashboard. It is designed to implement the functional requirement 6 "Customise data visualisation".

As seen at the top of the figure 4.22, there is an area for showing the list of existing Dashboards (outlined in blue). This area will have a data binding to the existing list of DashboardSettings from the DashboardService. When the user clicks on a dashboard on the list, the program will pass the selected DashboardSetting to the EditDashboard Component (the box outlined in orange). This component is responsible for displaying the details of the dashboard. If the user clicks the "Create New Dashboard" button, null will be passed into EditDashboard

to let the component know the user chose to create a new dashboard. If the "Delete Dashboard" button is clicked, the selected Dashboard will be instantly deleted from the existing list of DashboardSettings in the DashboardService. For all the changes made within the EditDashboard Component, user has to click the "Create/Edit Dashboard" button to persist the changes into the DashboardService.

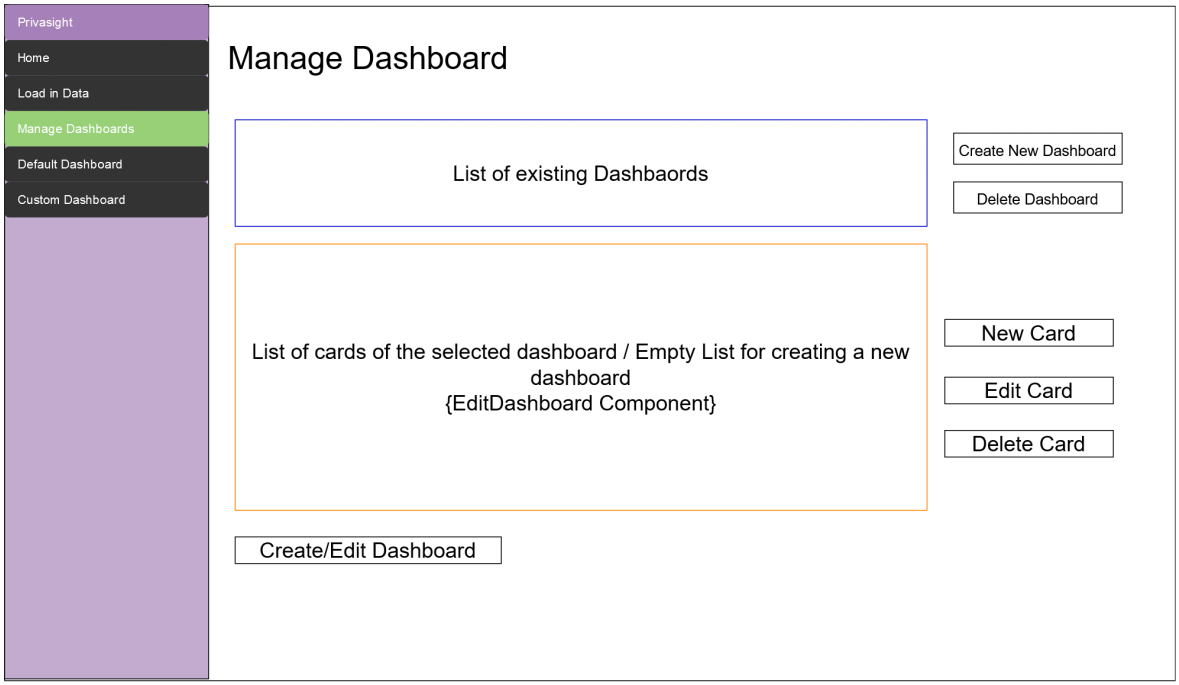


Figure 4.22: Mock-up design for the Manage Dashboard page

The EditDashboard component will show the list of existing cards of the input DashboardSetting or an empty list in the case of creating a new dashboard. User can press the buttons on the right to either create a new card, edit the selected card, or delete the selected card. When "New Card" or "Edit Card" button is clicked, a dialog with the EditCard component will show up. The difference between "New Card" and "Edit Card" is the CardSetting object being fed into the EditCard component. For New Card, it will be null to instruct the EditCard component to initialise a new CardSetting component. For Edit card, it will be the CardSetting of the selected card.

The mock up of the EditCard component can be seen on figure 4.23. It is a step by step design. User will have to choose the data type to display from the collection of AvailableFileWrappers saved in CompanyConfigs as mentioned in 4.3.4.2. Then, user will need to choose from a card type from all members of the CardTypes enum (detailed in section 4.2.2). The last step is to choose a dialog type from all members of the DialogTypes enum (detailed in section 4.2.2). For the last two steps, a demo display of those component will be generated and shown in the display area for user to pick from.

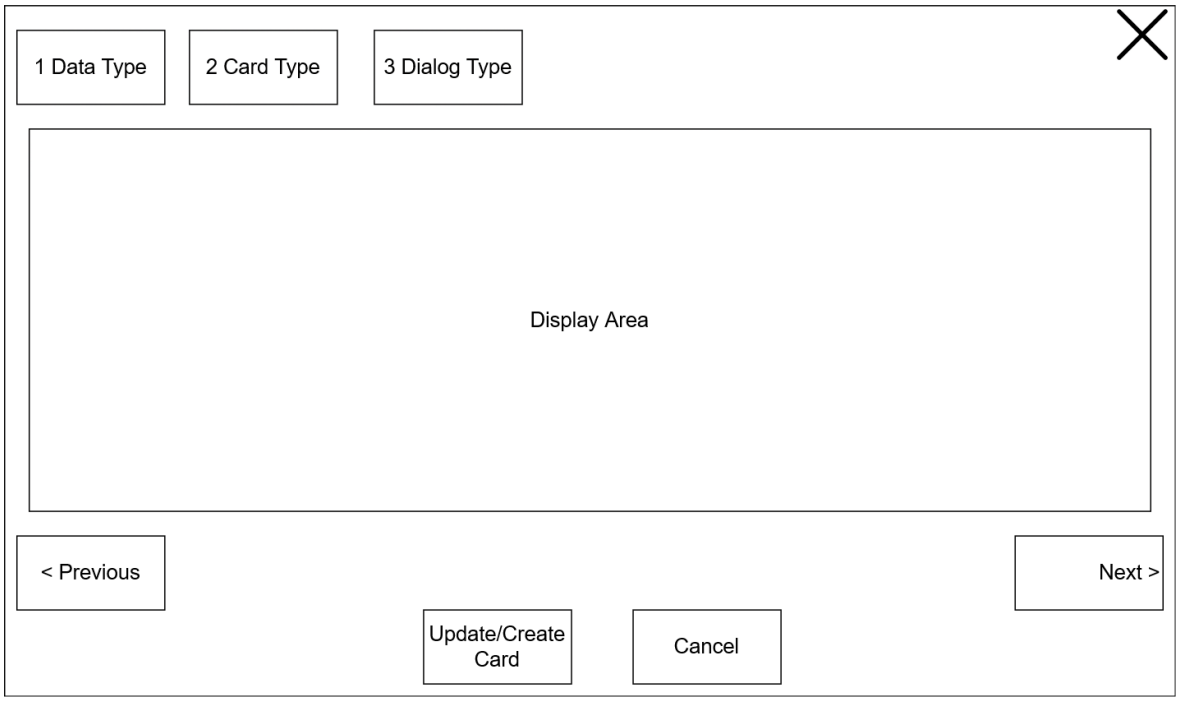


Figure 4.23: Mock-up design for the EditCard Component

4.4.7 UI Library Choice

Different UI libraries were compared. It includes MatBlazor [36], DevExpress Blazor UI Components [37], Syncfusion Blazor Components [38] and Radzen Blazor Components [39]. Radzen Blazor Components was chosen because it covers almost all required UI components. It is a library with a set of 60+ free and open source native Blazor UI controls. It is free for commercial use. It has 1.3 million downloads on Nuget [40], the package manager for .NET.

Chapter 5

Implementation

5.1 Overview

This chapter focuses on the analysis of the implementation. This chapter will start by discussing the implementations of components that are not tied to a particular requirement. After that, the implementation of the functional requirements will be discussed.

Some functionalities have complicated logic between the user and different components and services. Sequence diagrams will be used to help the discussion. The syntax in this tutorial of sequence diagrams [41] has been followed.

The site was deployed as a static GitHub page [42] to make sure that it would work across multiple machines. A CI/CD pipeline was built using GitHub Actions to ensure that the live website is always in sync with the codebase. CI/CD stands for Continuous Integration, which automatically builds and integrates code changes within the repository, and Continuous Deployment, which automatically deploys code changes to customers [43].

All screenshots used in this section will be taken from the live website hosted on GitHub.

5.2 Data synchronisation on UI

It is important to keep data in sync across the site. The INotifyPropertyChanged interface has been used to notify components that a property value has changed. Since the implementation of this interface is the same despite the class, a default implementation, INotifyPropertyChangedImplementation, has been created. The details can be seen in figure 5.1.

```
public abstract class INotifyPropertyChangedImplementation : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler? PropertyChanged;

    4 references
    protected void OnPropertyChanged([CallerMemberName] string? propertyName = null)
    {
        PropertyChanged?.Invoke( sender: this,  e: new PropertyChangedEventArgs(propertyName));
    }
}
```

Figure 5.1: INotifyPropertyChangedImplementation Class

Since the state of the application stays only in the data services, only these two objects are required to inherent INotifyPropertyChangedImplementation. When the state of the properties in these services changes, they are required to call the OnPropertyChanged method to notify the changes to the components using these properties.

The components using these properties are required to have an event listener on the ProretyChanged event of the services. An example of the event listener implementation can be found in NavMenu, as shown in figure 5.2. The event listener is added upon component initialization and removed during the component disposal process. The event listener simply calls the StateHasChanged method when required to notify the UI of the changes.

```
public void Dispose() => _dashboardService.PropertyChanged -= PropertyHasChanged;

protected override async Task OnInitializedAsync()
{
    _dashboardService.PropertyChanged += PropertyHasChanged;
    if (_dashboardService.DashboardSettings == null)
    {
        await _dashboardService.SetDashboardSettingsFromStorage();
    }
}

private void PropertyHasChanged(object? sender, PropertyChangedEventArgs args)
{
    if (args.PropertyName == nameof(_dashboardService.DashboardSettings))
    {
        StateHasChanged();
    }
}
```

Figure 5.2: Event Listener implementation in NavMenu

5.3 Navigation System

The user will navigate to different pages of the site through the navigation menu. It is designed to always stay on the left of the site. Figure 5.3 shows the screenshot of the navigation menu

with two custom dashboards from the live website [42].

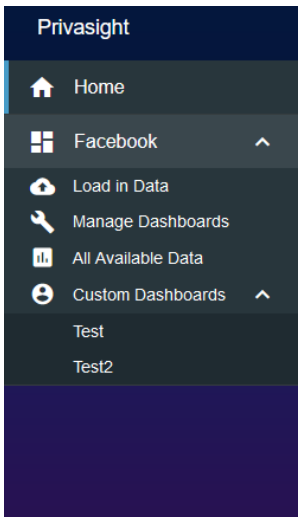


Figure 5.3: Screenshot of the Navigation Menu on the live website, taken on 09-05-2022

The navigation menu has been generalised to make sure that the system is not tied to the Facebook data. As seen in figure 5.4, it generates the link to the pages based on the members in the AvailableCompany enum. It feeds the company value to different pages through the URL. That is how the components know what company they are handling. This value will be used when the component needs to fetch data from the CompanyDataService and DashboardService. The navigation area for the custom dashboards will only be generated when there is dashboard setting in the DashboardService. The company value and the index number of the dashboard setting in DashboardService will be fed into the custom dashboard page.

```
<div class="@NavMenuCssClass" @onclick="ToggleNavMenu">
    <nav class="flex-column">
        <RadzenPanelMenu>
            <RadzenPanelMenuItem Text="Home" Icon="home" Path=" " />
            @foreach (AvailableCompany company in Enum.GetValues(typeof(AvailableCompany)))
            {
                <RadzenPanelMenuItem Text=@company.ToString() Icon="dashboard" Expanded="true">
                    <RadzenPanelMenuItem Text="Load in Data" Path="@($"{company}/loadData")" Icon="backup"/>
                    <RadzenPanelMenuItem Text="Manage Dashboards" Path="@($"{company}/manageDashboards")" Icon="build"/>
                    <RadzenPanelMenuItem Text="All Available Data" Path="@($"{company}/dashboards/default")" Icon="assessment"/>
                    @if (_dashboardService.DashboardSettings != null &&
                         _dashboardService.DashboardSettings.ContainsKey(company) &&
                         _dashboardService.DashboardSettings[company].Count != 0)
                    {
                        <RadzenPanelMenuItem Text="Custom Dashboards" Icon="account_circle">
                            @for (var i = 0; i < _dashboardService.DashboardSettings[company].Count; i++)
                            {
                                <RadzenPanelMenuItem Text="@_dashboardService.DashboardSettings[company][i].Name" Path="@($"{company}/dashboards/{i}")"/>
                            }
                        </RadzenPanelMenuItem>
                    }
                </RadzenPanelMenuItem>
            }
        </RadzenPanelMenu>
    </nav>
</div>
```

Figure 5.4: Code for generating the navigation menu in NavMenu

5.4 Data Storage

Data storage is an essential feature for this application. It is about storing the dashboard settings and ingested SAR data in the user's browser. Throughout the development process, there were three approaches used. The final approach used is "IndexedDb with IDB-Keyval".

5.4.1 First Approach: SQLite Database in IndexedDB

At the start of the development, the plan was to have a SQLite database stored in the IndexedDB [31] within the user's browser. That was shown possible in this Microsoft introduction [44].

The main advantage of this approach is that the data storage architecture and code can be reused when the application expands to other platforms. Besides, with the use of a SQL database, the code for this function will be a lot more organised than the other approaches.

However, several drawbacks have been identified after the implementation of this approach.

To begin with, the compiled code had grown exponentially with the use of SQLite and Entity Framework, which is an object-relational mapping framework that enables data access without the need of handling the specific logic of the underlying database tables and columns [45]. The website's initial loading time (the time it takes to download the web assembly to the user's machine) had significantly increased.

The cached data on the user's machine, which includes the loaded SAR data and the cached assembly for the site, has grown from 13.8 MB to 55.8 MB. It is important to note that the site currently only stores three data categories. This would not be ideal when the app expands its support to include all data categories from Facebook.

The time it takes the app to load data from the browser's storage has also increased noticeably. The site is currently working with a small subset of all possible data. This will become a serious drawback when the app expands.

With all these identified downsides, this approach was abandoned.

5.4.2 Second Approach: localStorage with Blazored LocalStorage

Before the implementation of data encryption, the application stores all data in localStorage to solve the issues with the first approach.

localStorage is part of the Web Storage API [32]. The Web Storage API makes it possible to save key/value pairs in the browser. Within the API, there are two mechanisms: sessionStorage, which keeps a separate storage area for the duration of the page session; and localStorage,

which keeps the data even when the browser is closed and reopened. localStorage was chosen as it is a better fit for the use case of this application.

Blazored LocalStorage [46] is a library that gives Blazor applications access to the browser's localStorage without the need to write JavaScript code. It also handles serialisation and deserialization of values. It is open-sourced under the MIT licence and has 1.6 million downloads on Nuget [47], the package manager for .NET.

The implementation strategy for this approach is to use the methods in Blazored LocalStorage in the data services to handle the data storage and retrieval. For data storage, the services will provide the data and the key to the library. The library will then serialise the C# object to JSON and store the serialised string to localStorage. For data retrieval, the services will provide the library with the key and type of the expected C# objects. The library will get the string back from localStorage based on the key and deserialise the string into C# objects based on the given type parameter.

During the implementation, there was one issue encountered. Due to the limited support for polymorphic deserialization of the default JSON library (System.Text.Json), the serialisation of loaded SAR data was deemed impossible without a custom converter. Due to the time constraint, another JSON library, Newtonsoft.Json (Json.Net), has been added to the project as a solution.

Newtonsoft.Json [48] had its first release in June 2006. Since then, it has been the go-to library for JSON serialisation and deserialisation before the introduction of the official Microsoft support, "System.Text.Json". It is open-sourced under the MIT licence and is free for commercial use. It also has 1.9 billion downloads on Nuget [49].

5.4.3 The final Approach: IndexedDB with IDB-Keyval

After the implementation of data encryption, the data type in which the data needed to be stored has been converted from a string to a JavaScript ArrayBuffer object. This will be covered in greater detail in Section 5.6. Since localStorage only allows string storage, the storage has to be switched to IndexedDB.

IndexedDB [31] is a low-level API for client-side storage of large amounts of structured data, such as files/blobs. When compared to the Web Storage API, it can store complex structured objects other than just string.

The JavaScript library, IDB-Keyval [50], is used to facilitate the usage of IndexedDB. It is a simple promise-based key-value store implemented with IndexedDB. Essentially, it makes IndexedDB similar to localStorage. It is the simplified version of idb [51], which is the most popular package for IndexedDB on npm [52], the package manager for JavaScript.

The dependency on the Blazored LocalStorage library has been removed. The code that is responsible for data storage has been moved from the data services to encrypt.js, which will be discussed in detail in sections 5.5 and 5.6.

From the data storage and retrieval standpoint, the data services are now only responsible for data serialisation and deserialisation. The encryption and storage are handled by encrypt.js. For data storage, the services only need to serialise the data into a string and send it to encrypt.js. For data retrieval, the services will ask encrypt.js for the serialised string and deserialise it.

5.5 Data Services

As mentioned in section 4.3.4.4, this application contains two data services, CompanyDataService and DashboardService. This section will cover the implementation of these services.

The aim of these services is to handle the logic surrounding the access and storage of the SAR data and the dashboard settings respectively. Both services share a similar structure, and the shared code is stored in the ServiceUsingIndexedDb class, which mainly consists of the inheritance of the INotifyPropertyChangedImplementation and the initialisation of the JsRunTime for the communication with encrypt.js. The encrypt.js file consists of the logic to store and retrieve data in IndexedDB.

5.5.1 CompanyDataService

The outline of the methods and properties in the CompanyDataService can be seen in figure 5.5.

The property, DataLoadingStatus, is used to show the data loading status to the user, while LoadingData is used to make sure that the user cannot load two files into the system at the same time. These two properties are saved in this service to make sure that the loading status persists even when the user switches between pages during the data loading process. These two properties are only saved in the memory since they do not need to persist between sessions.

AvailableData is used to stored the ingested FileWrappers. The key of this dictionary is used to denote the company of the data stored in the value. This is designed to facilitate the expansion of the supported companies and decouple the system from Facebook data. The value of the AvailableData dictionary is another dictionary for storing the ingested FileWrappers. The key is the type name of the FileWrapper to make the retrieval of the FileWrapper easier.

AvailableData is the only data that needs to be persisted into browser storage. Methods have been developed to support this requirement. The details of encrypt.js will be covered in section 5.6. In this section, only the name of the methods of this file will be mentioned.

SetAvailableDataFromStorage is used to get the data from the browser storage by invoking the getCompanyData method in encrypt.js and set the returned value to AvailableData within

the service. After the initial call of this method, AvailableData will only be updated when the UpdateAvailableData or RemoveAvailableData method is called.

The private method UpdateAvailableDataInStorage is used by UpdateAvailableData and RemoveAvailableData to call the setCompanyData method in encrypt.js.

UpdateAvailableData is used to update the AvailableData of a particular company. It is only used on the "Load in Data" page as that is the only place for users to input their SAR data (FileWrappers) into the system. If the service has not retrieved the data stored in the browser's storage before this process, the method will invoke the SetAvailableDataFromStorage method. After that, the existing data will be grouped with the newly loaded data. The grouped data will then be sent to encrypt.js using UpdateAvailableDataInStorage for data persistence.

RemoveAvailableData is used to remove all ingested FileWrappers of the given company. It is used by the ClearCompanyDataButton that is shown on the top of each dashboard page and the "Load in Data" page. It will call the UpdateAvailableDataInStorage method to set the data of the given company in the storage to empty.

```
public class CompanyDataService : ServiceUsingIndexedDb
{
    5 references
    public string DataLoadingStatus { get; set; } = "";
    3 references
    public bool LoadingData { get; set; }

    private Dictionary<AvailableCompany, Dictionary<string, IFileWrapper>>? _availableData;

    /// <summary> Key: the company Value: Store the transformed FileWrappers (Json o ...
    17 references
    public Dictionary<AvailableCompany, Dictionary<string, IFileWrapper>>? AvailableData
    {
        get => _availableData;
        private set
        {
            _availableData = value;
            OnPropertyChanged();
        }
    }

    0 references
    public CompanyDataService(IJSRuntime jsRuntime) ...

    1 reference
    public async Task UpdateAvailableData(AvailableCompany company, Dictionary<string, IFileWrapper> newData)...

    2 references
    public async Task SetAvailableDataFromStorage()...

    1 reference
    public async Task RemoveAvailableData(AvailableCompany company)...

    2 references
    private async Task UpdateAvailableDataInStorage()...
}
```

Figure 5.5: Abstract overview of CompanyDataService

5.5.2 DashboardService

Figure 5.6 outlines the properties and methods in the DashboardService class.

DashboardSettings is used to store the list of DashboardSetting grouped by the AvailableCompany enum. This is also the only property in this class that needs to be saved into the browser's storage.

ExistingDashboardNames is a property to facilitate the retrieval of the list of existing dashboard names. It is used by the "Manage Dashboard" page to ensure the dashboard name that the user enters when they are updating or creating a dashboard is unique.

SetDashboardSettingsFromStorage is used to get the DashboardSettings from the browser's storage. It will call the getDashboardSettings method in encrypt.js.

UpdateDashboardSettings is used to either add, update, or delete a DashboardSetting from DashboardSettings depending on the action parameter. This method will call the setDashboardSettings method in encrypt.js to update the DashboardSettings in the memory.

Despite being stored in the encrypt.js file, the setDashboardSettings and getDashboardSettings methods have nothing to do with encryption. Since DashboardSetting is not sensitive data, it can be stored in plain text. The setDashboardSettings method simply stores its parameter (the serialised string) into indexedDB using the IDB-Keyval library. getDashboardSettings retrieves the value stored in the indexedDB using the library and returns it to the caller.

```
public class DashboardService : ServiceUsingIndexedDb
{
    7 references
    public enum UpdateAction
    {
        Add,
        Update,
        Delete
    }

    private Dictionary<AvailableCompany, IList<DashboardSetting>>? _dashboardSettings;

    27 references
    public Dictionary<AvailableCompany,IList<DashboardSetting>>? DashboardSettings
    {
        get => _dashboardSettings;
        private set
        {
            _dashboardSettings = value;
            OnPropertyChanged();
        }
    }

    private Dictionary<AvailableCompany, IEnumerable<string>>? _existingDashboardNames;
    3 references
    public Dictionary<AvailableCompany, IEnumerable<string>>? ExistingDashboardNames...

    0 references
    public DashboardService(IJSRuntime jsRuntime) ...

    3 references
    public async Task UpdateDashboardSettings(AvailableCompany company, DashboardSetting newSetting, UpdateAction action,
        DashboardSetting? oldSetting = null)...

    4 references
    public async Task SetDashboardSettingsFromStorage()...
}
```

Figure 5.6: Abstract overview of DashboardService

5.6 Data Encryption

The focus of this section will be on data encryption. All the code for encryption is in encrypt.js. This file is sometimes referred to as the Encryption Service. This section will also cover some data storage logic since it is being done within the encryption logic.

As mentioned in section 4.2, there are two types of data that this application will handle. They are DashboardSettings and the SAR data. Since SAR data is the personal data of the user, it is required to be encrypted to protect the user's privacy. DashboardSettings is just the configuration of the application and hence does not require encryption.

The Web Cryptography API [53] was used for the encryption. It is a JavaScript API for performing basic cryptographic operations in web applications. This API is used because the official C# encryption library is not supported in Blazor WebAssembly. The JavaScript file, encrypt.js, was developed to perform the encryption.

The current state-of-the-art encryption method, AES-128-CBC, is used. AES stands for the Advanced Encryption Standard, which is the name of the encryption algorithm. 128 stands for the key size in bits, and CBC stands for the Cipher Block Chaining, which is the mode of encryption. PBKDF2 (Password-Based Key Derivation Function 2) is used to derive the encryption key from the password provided by the user. This function requires a salt for the

key generation. During the password creation process, a random salt will be generated and saved in the browser's storage for later use. For the encryption to work, the user is required to create a password of at least 16 characters.

The encryption function in the Web Cryptography API used will represent the encrypted message as a JavaScript object of the ArrayBuffer type. This ArrayBuffer object will be saved in the data storage, and no one without the password will be able to decrypt it.

5.6.1 Storage of AvailableData

Figure 5.7 denotes a typical flow of the process for setting the AvailableData through the Encryption Service when no encryption key is present in the memory.

The flow starts with some processes related to setting the AvailableData of the CompanyDataService. The CompanyDataService will then send a request to update the AvailableData in the browser's storage alongside the updated data to the Encryption Service by calling the setCompanyData method.

The setCompanyData method will start the encryption key generation process. The process begins with the Encryption Service requesting the salt from the data storage. If it is undefined, it means that the password is never being generated. The service will ask the user to create a password and generate a salt at random. The service will save the salt in the data storage for future use. If the salt is defined, the Encryption Service will ask the user for the existing password. With the received password, the service will start the generation of the encryption key based on the password and salt. The key will then be saved in memory for later use during the session.

After that, the Encryption Service will encrypt the input data and send the encrypted ArrayBuffer to the data storage for persistence.

Upon the completion of the process, an OnPropertyChanged notification of the AvailableData will be sent to the UI. If any error happens during the process, an error notification will be sent to the user. Or else, the UI will do whatever it has to do with this data update.

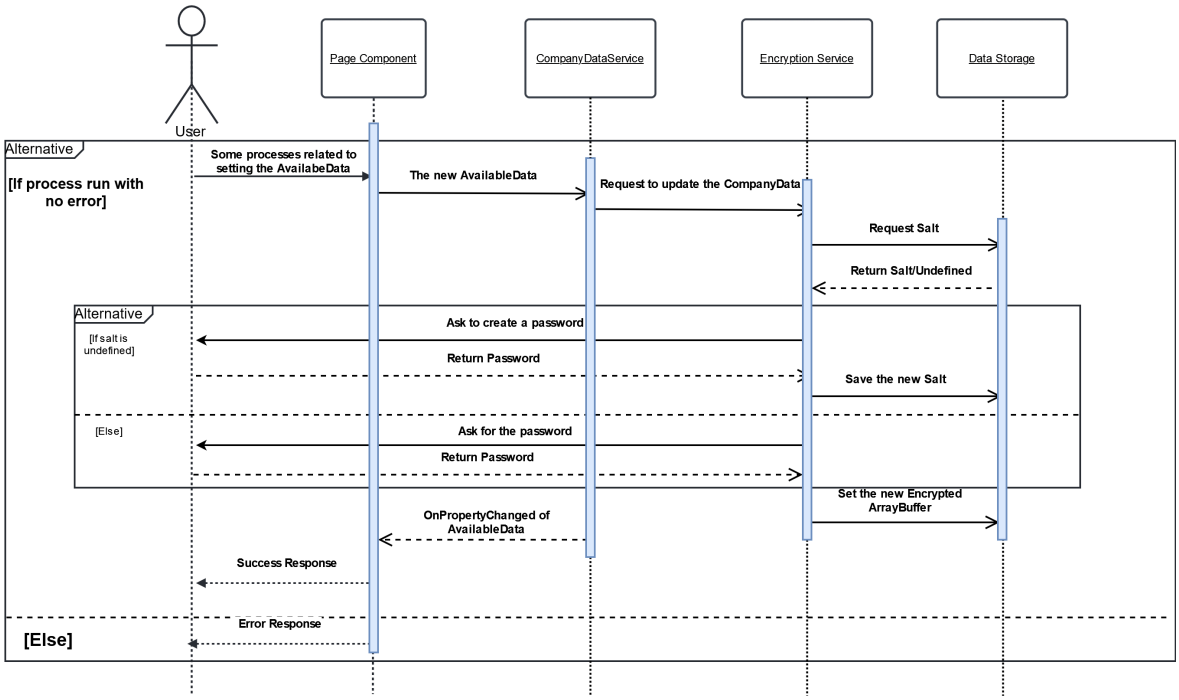


Figure 5.7: Sequence Diagram of a flow that set the AvailableData when the encryption key is not present in the memory

Figure 5.8 denotes a typical flow of the process for setting the AvailableData through the Encryption Service that occurs when the encryption key is present in the memory.

It starts with a similar process like the previous flow (figure 5.7). Some processes related to setting the AvailableData of the CompanyDataService happened. The CompanyDataService sent a request to update the AvailableData in the memory alongside the updated data to the Encryption Service by calling the setCompanyData method.

Given that the encryption key already exists, the Encryption Service will skip the key generation process and encrypt the data with the existing key. The encrypted ArrayBuffer will then be sent to the data storage. Upon the completion of the process, an OnPropertyChanged notification on the AvailableData will be sent to the UI as a notification. The user will either see the expected update on the UI or receive an error notification if any error happens during the process.

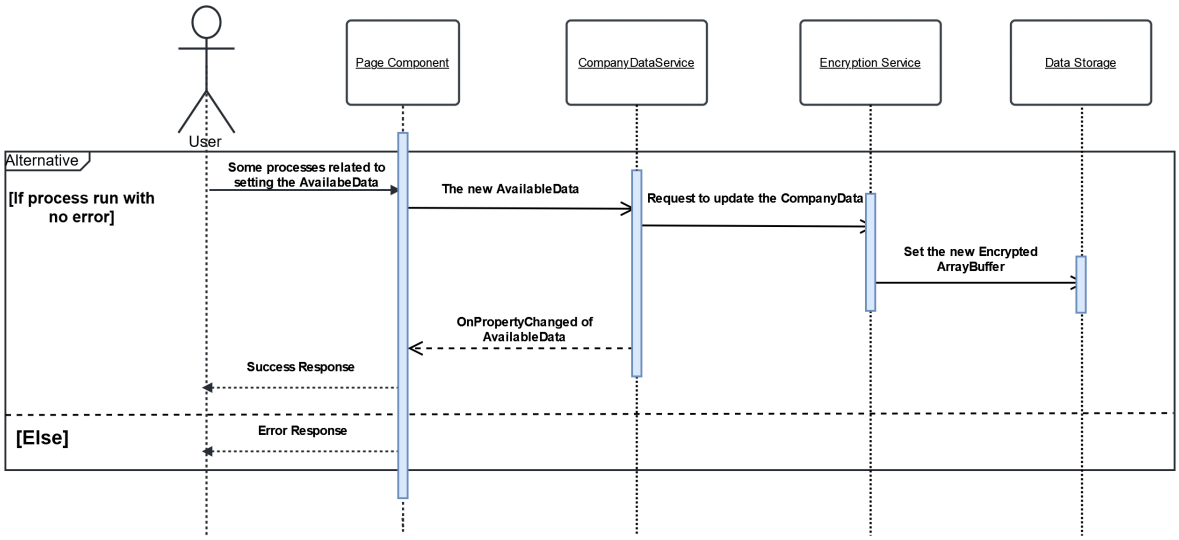


Figure 5.8: Sequence Diagram of a flow that set the AvailableData when the encryption key is present in the memory

5.6.2 Retrieval of AvailableData

Figure 5.9 denotes a typical flow of the process for retrieving the AvailableData from the Encryption Service when no encryption key is present in the memory.

It starts with the user requesting something related to the AvailableData through some page components, which is expected to be either "Load Data in" page, or "Default Dashboard" page, or "Custom Dashboard" page. The page component will then request the AvailableData from the CompanyDataService. The CompanyDataService will send a request to the Encryption Service to retrieve the existing data.

The retrieval process, which happens within the getCompanyData method, will start with the generation of the encryption key, which is the same process mentioned before.

After that, the Encryption Service will request the encrypted ArrayBuffer from the data storage. If undefined is returned, the service will simply return an empty string to the CompanyDataService. If an ArrayBuffer is returned, the Encryption Service will decrypt the ArrayBuffer back to the JSON string using the encryption key and return it to the CompanyDataService.

The CompanyDataService will then return the deserialised AvailableData to the page component. The OnPropertyChanged method will be called within the AvailableData setter to notify all UI components about the changes. The component will display the updated dashboard or show a "data loaded in successfully" message to the user depending on what page it is. If any error happens during this entire process, an error response will be shown to the user.

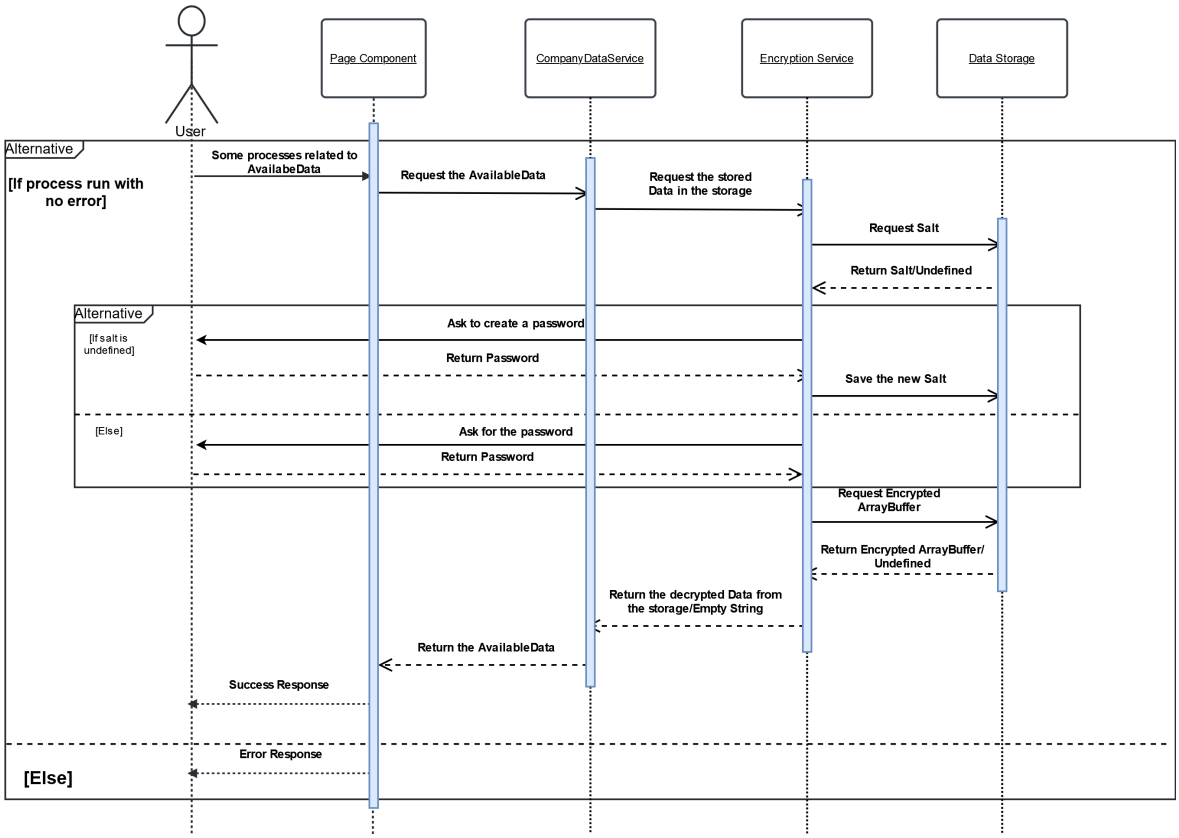


Figure 5.9: Sequence Diagram of a flow that retrieve the AvailableData when the encryption key is not present in the memory

Figure 5.10 denotes a typical flow of the process for retrieving AvailableData from the Encryption Service that occurs when the encryption key is present in the memory.

It starts with a similar process like the previous flow (figure 5.9). The page component requests the AvailableData and then the CompanyDataService requests it from the Encryption Service by calling the getCompanyData method.

Given that the encryption key is already in the memory, the Encryption Service will skip the key generation process and ask for the encrypted ArrayBuffer from the data storage. The service will then return either the decrypted message or an empty string to the CompanyDataService. The page component will then give the corresponding response to the user.

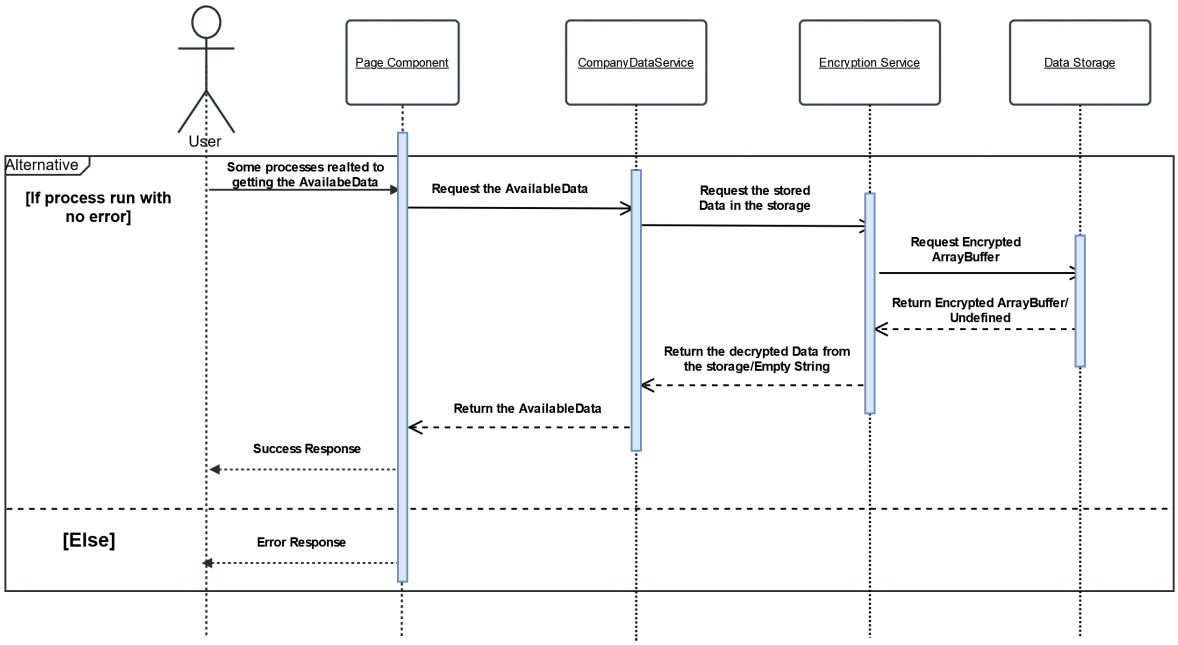


Figure 5.10: Sequence Diagram of a flow that retrieve the AvailableData when the encryption key is present in the memory

5.7 Data Ingestion of the return data from Facebook

SAR data is being loaded in by the user on the "Load in Data" page. The user can upload a zip file of the SAR data from Facebook to the application through this page. The expected interaction between the user, the load data page (the application), and services is documented in figures 5.11 and 5.12. The difference in the figures is on whether the CompanyDataService has retrieved the AvailableData from the data storage before the data loading process.

The flow of both figures begins with the user loading the zip file in via the "upload file" feature as described in the Microsoft Doc [54]. This feature will take in the zip file from the user and return an InputFileChangeEventArgs object which contains the file input by the user.

The eventArgs object and the AvailableFileWrappers property from CompanyConfigs, as mentioned in section 4.3.4.2, will be passed to the TransformJsonToObj function in the static helper class, DataLoadInHelper. The function will open the stream of the zipped file from the eventArgs and unzip it using ZipArchive, which contains a collection of file streams. It will then loop through the collection of files and check if the file matches any file paths (the keys) in the AvailableFileWrappers. When a match is discovered, the helper will deserialise the file stream using the type (the value) stored in the AvailableFileWrappers. The details of this function can be seen in figure 5.13.

After receiving the C# object version of the SAR data, the LoadData page will call another

function in DataLoadInHelper to inject the last modified date into the GeneratedOn field in the FileWrappers.

Following that, the page will send this data to CompanyDataService to persist it in the data storage using the UpdateAvailableData method. This is where the flow of the two figures differs.

The flow in figure 5.11 is relatively simpler. In this version of the flow, the CompanyDataService has already requested the AvailableData in the browse's storage before this process, meaning that the encryption key has already been generated. The CompanyDataService will group the newly loaded data and the existing AvailableData stored in it as the new AvailableData. It will then serialise the grouped data and send the serialised JSON string to the Encryption Service. The service will then encrypt this string and update the data in the browser's storage.

The flow in figure 5.12 is more complicated. In this version, the CompanyDataService has not requested the data stored in the storage before this process. It implies that the encryption key has not been generated. The CompanyDataService will ask the Encryption Service for the data stored in the data storage. The Encryption Service will start by generating the encryption key, which is the same process as described in section 5.6.1. After the key generation process and the decryption of data, the CompanyDataService will receive the JSON string of the existing data from storage. The service will deserialise it back to a C# object and group it with the new data. It will then serialise it to a JSON string and send it back to the Encryption Service. The Encryption Service will encrypt the data and update the encrypted ArrayBuffer in the data storage.

For both flows, if no errors occur during this entire process, a "data loaded in successfully" message will be displayed to the user at the end; otherwise, a failure message will be displayed.

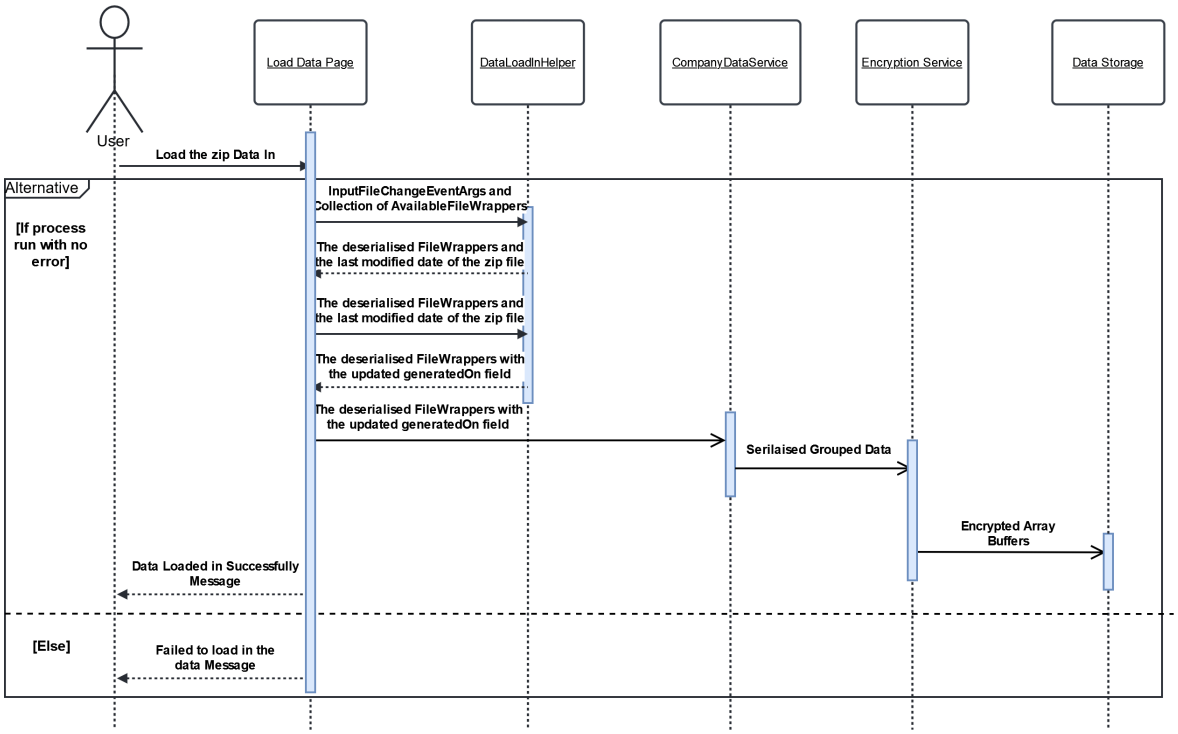


Figure 5.11: Sequence Diagram for Load Data Page with data from storage loaded into the CompanyDataService before the data loading process

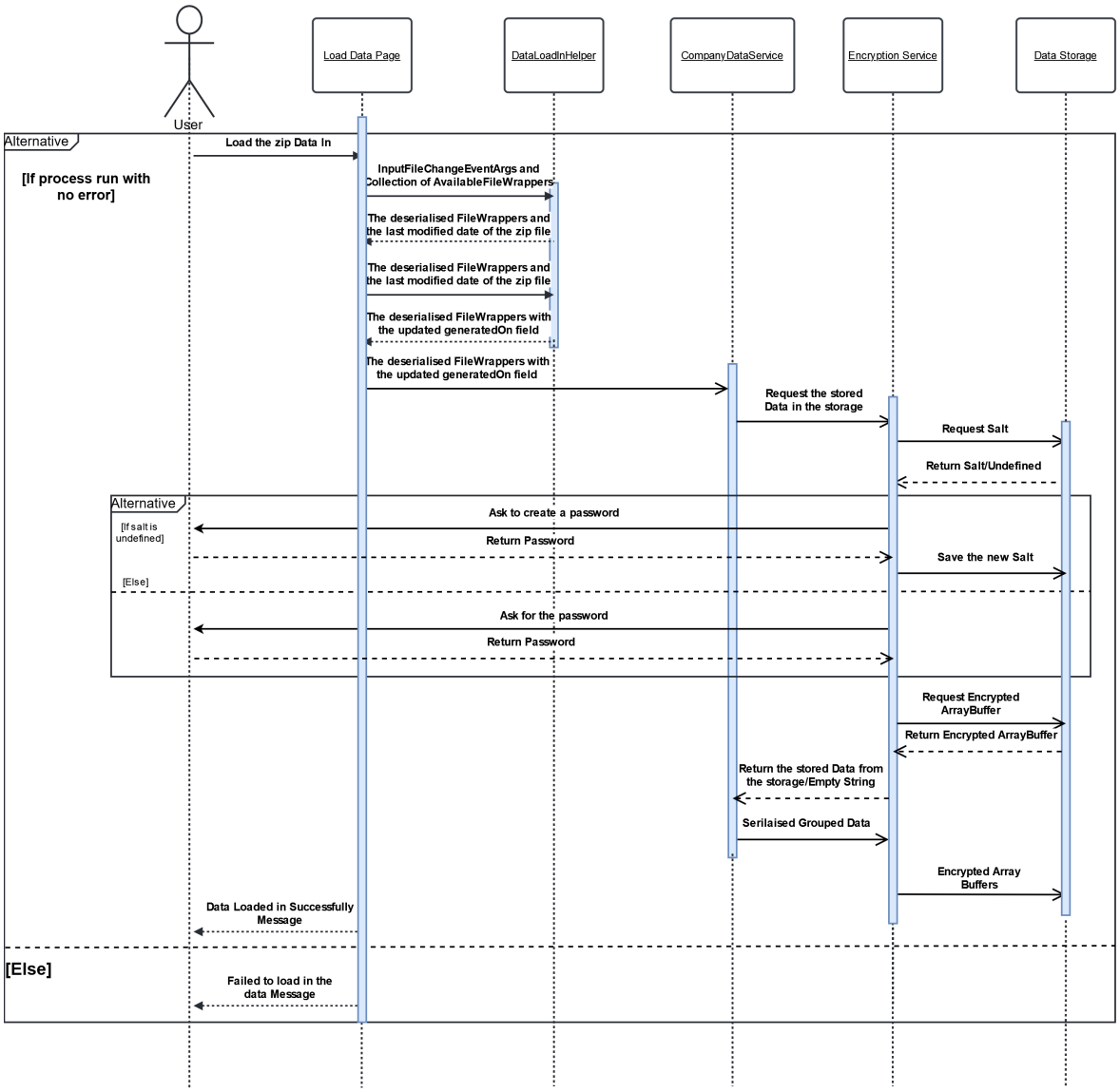


Figure 5.12: Sequence Diagram for Load Data Page with data from storage not loaded into the CompanyDataService before the data loading process

```
1 reference
public static async Task<(Dictionary<string, IFileWrapper> newData, DateTimeOffset generationDate)> TransformJsonToObj
  (InputFileChangeEventArgs e, Dictionary<string, Type> availableFileWrappers)
{
    var newData = new Dictionary<string, IFileWrapper>();
    var options = new JsonSerializerOptions();
    options.Converters.Add(item: new FbConverter());
    options.Converters.Add(item: new StringWrapperDbObjConverter());

    await using var stream = new MemoryStream();
    await e.File.OpenReadStream(MaxFileSize).CopyToAsync(stream);

    var generationDate = e.File.LastModified;

    using var archive = new ZipArchive(stream);
    foreach (var entry in archive.Entries)
    {
        if (!availableFileWrappers.TryGetValue(entry.FullName, out var wrapperType)) continue;
        var unzippedEntryStream = entry.Open();

        if (await JsonSerializer.DeserializeAsync(unzippedEntryStream, wrapperType, options) is IFileWrapper wrapperObj)
        {
            newData.TryAdd(wrapperType.Name, wrapperObj);
        }
    }

    return (newData, generationDate);
}
```

Figure 5.13: TransformJsonToObj function in DataLoadInHelper class

5.8 Display Components for the Ingested Data

Dashboard Component and Card Component are implemented similar to the design in sections 4.4.4 and 4.4.5. Figures 5.14 and 5.15 show screenshots of the dashboard components from the live website [42]. Figures 5.16 and 5.17 show screenshots of the card component's detailed view from the live website [42].

The implementation of the card component make use of several UI components from the chosen UI library. In order to decouple the display from the actual data classes, Reflection was used in the generation of the data list view and detailed table view components. Reflection allows the retrieval of type information, and hence the list of properties, of a class at run time. It also allows the retrieval of the value stored in attributes. The generation of DataList component is similar to DetailedTable component. Figure 5.18 shows the code of DetailedTable component as an example of the use of Reflection. The GetProperties method in ReflectionHelper returns a list of PropertyInfo of the item type of the list. The PropertyInfo is used to retrieve the value stored in the DetailedTableDisplayDataAttribute to generate the header information for the table. It is also used to supply the required information to the UI library.

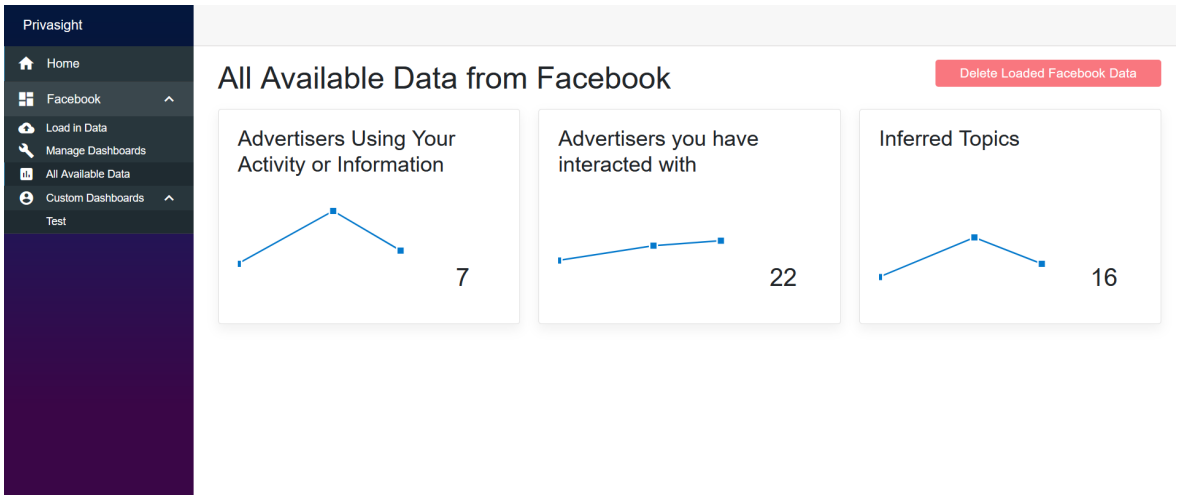


Figure 5.14: Screenshot of the Default Dashboard on the live website, taken on 08-05-2022

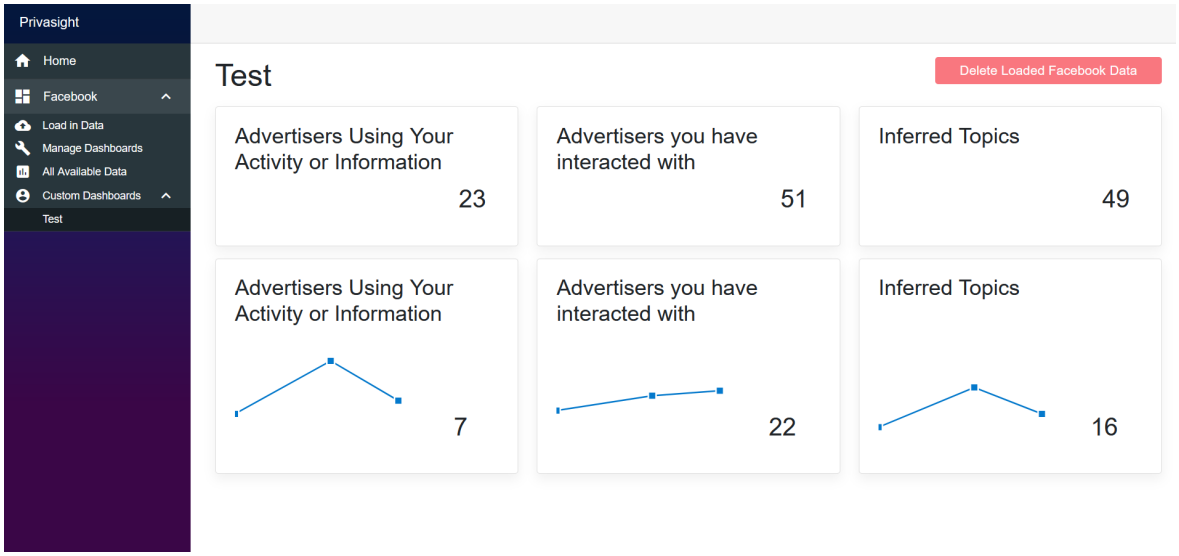


Figure 5.15: Screenshot of a Custom Dashboard on the live website, taken on 08-05-2022

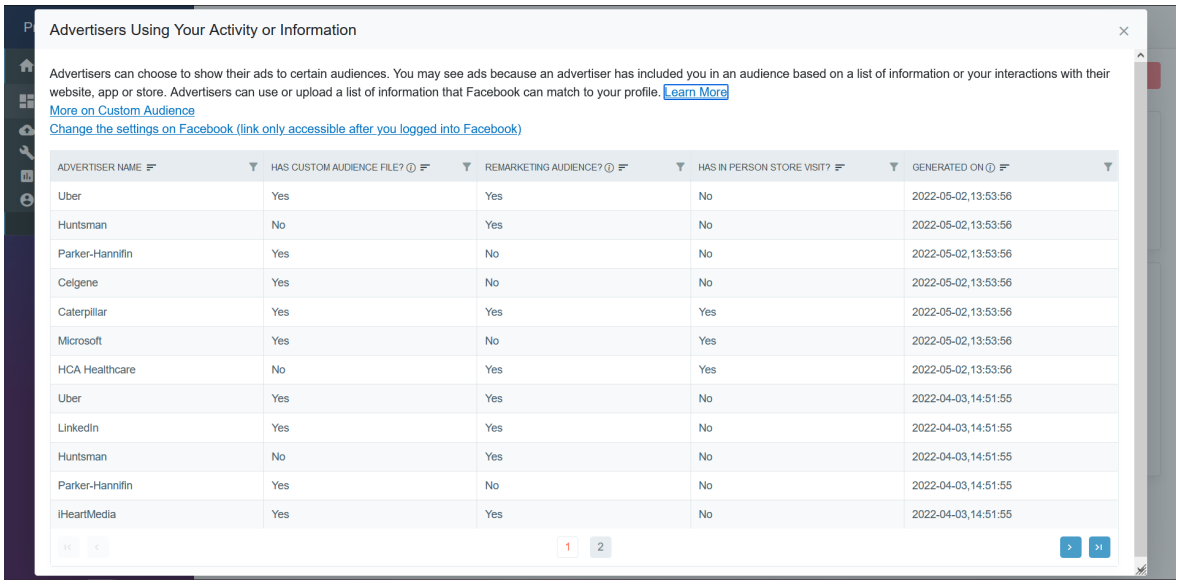


Figure 5.16: Screenshot of the Detailed Table view of the card component on the live website

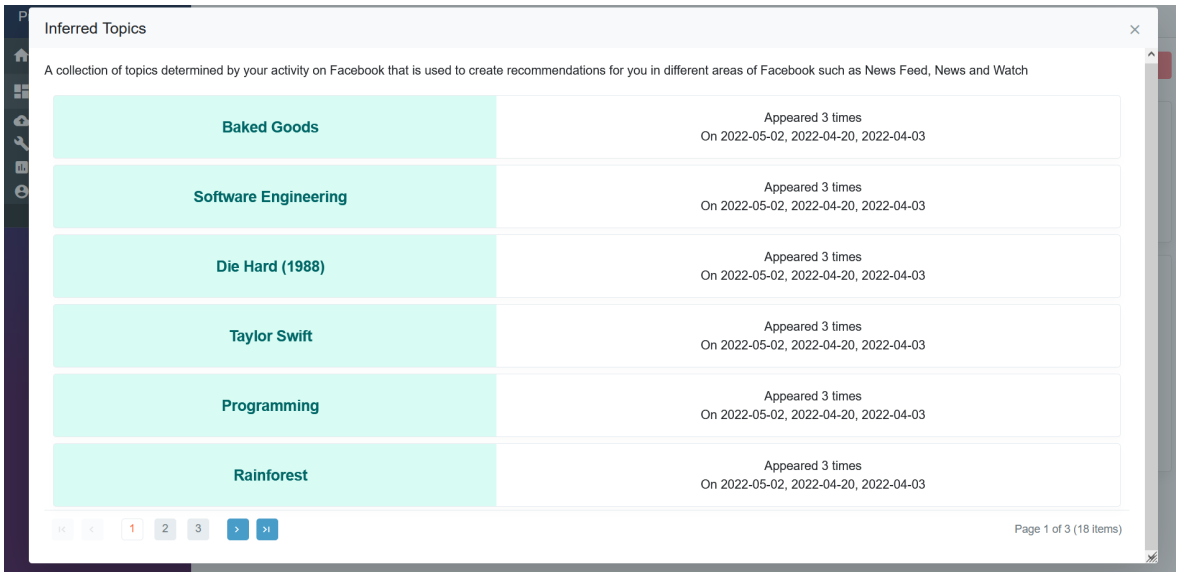


Figure 5.17: Screenshot of the Data List view of the card component on the live website

```
@using System.Reflection
@using Privasight.Model.Shared.Helpers

@typeparam TItem

@if (!string.IsNullOrWhiteSpace(Description))
{
    <p>@((MarkupString)Description)</p>
}

<RadzenDataGrid AllowFiltering="true" AllowColumnResize="true" FilterMode="FilterMode.Advanced" AllowPaging="true" AllowSorting="true"
                Data="@Items" TItem="TItem" LogicalFilterOperator="LogicalFilterOperator.Or" PageSize="12" >
    <Columns>
        @foreach (var propInfo in PropertyInfos)
        {
            var dto = DetailedTableDisplayDataAttributeHelper.GetDTO(propInfo);

            <RadzenDataGridColumn TItem="TItem" Property="@propInfo.Name">
                <HeaderTemplate>
                    @if (!string.IsNullOrWhiteSpace(dto.Description))
                    {
                        <span data-toggle="tooltip" title="@dto.Description">@dto.DisplayName <i class="oi bi-info-circle"/></span>
                    }
                    else
                    {
                        @dto.DisplayName
                    }
                </HeaderTemplate>
                <Template>
                    @DetailedTableDisplayDataAttributeHelper.FormatPropertyString(propInfo, context)
                </Template>
            </RadzenDataGridColumn>
        }
    </Columns>
</RadzenDataGrid>

@code {
    private TItem FirstItem => Items.First();
    private IEnumerable<PropertyInfo> PropertyInfos => ReflectionHelper.GetProperties(FirstItem!);

    [Parameter]
    public string Description { get; set; } = "";
    [Parameter]
    public IEnumerable<TItem> Items { get; set; } = null!;
}
```

Figure 5.18: DetailedTable.radzor

5.9 Dashboard Pages

There are two page components that will use the Dashboard Component to create a dashboard to show the corresponding dashboard data to the user. They are the DefaultDashboard page component and the CustomDashboard page component.

For both pages, the Company property is set on their URL and passed into the Dashboard component as mentioned in section 5.3.

On the "Default Dashboard" page, the program will look through all available FileWrappers of the corresponding company in the CompanyDataService and generate the corresponding CardSetting object based on the data type of the FileWrapper. After generating all of the necessary CardSetting objects, the page will generate a DashboardSetting object and feed it into the page's Dashboard Component. Figure 5.19 shows the details of the implementation of the DefaultDashboard page component and how it creates the corresponding DashboardSetting.

```
<PageTitle>
    @_title| Privasight
</PageTitle>

<Dashboard Setting="GetDefaultDashboardSetting()" Company="_company"/>

@code {
    private string _title = "All Available Data";
    private AvailableCompany _company;

    [Parameter]
    public string UrlCompany { get; set; } = "";

    protected override void OnInitialized()
    {
        _company = Enum.Parse<AvailableCompany>(UrlCompany);
        _title = $"All Available Data from {_company}";
    }

    private DashboardSetting GetDefaultDashboardSetting()
    {
        var cardSettings = new HashSet<CardSetting>();

        foreach (var fileWrapperType in CompanyConfigs.AvailableFileWrappers[_company].Values)
        {
            CardSetting? card = null;

            if (fileWrapperType.GetInterfaces().Contains(typeof(ISingleItemListFile<StringWrapperDbObj>)))
            {
                var title :string = ReflectionHelper.GetTitleFromFileWrapper(fileWrapperType);
                card = new CardSetting(title, fileWrapperType.Name, DialogTypes.DataList, CardTypes.WithLineChart);
            }
            else if (fileWrapperType.GetInterfaces().Contains(typeof(ISingleItemListFile)))
            {
                var title :string = ReflectionHelper.GetTitleFromFileWrapper(fileWrapperType);
                card = new CardSetting(title, fileWrapperType.Name, DialogTypes.DetailedTable, CardTypes.WithLineChart);
            }

            if (card != null)
            {
                cardSettings.Add(card);
            }
        }

        return new DashboardSetting(cardSettings, _title);
    }
}
```

Figure 5.19: DefaultDashboard.radzor

The "Custom Dashboard" page will take an integer as an input in its URL as mentioned in section 5.3. This number corresponds to the index number of the required DashboardSetting stored in the DashboardService. The page will fetch that DashboardSetting and feed it into the page's Dashboard Component. Figure 5.20 shows the details of the implementation of the CustomDashboard page component and how it fetches the corresponding DashboardSetting object.

```razor
@page "/{UrlCompany}/dashboards/{Id:int}"
@using Privasight.Model.Shared.Display.Dashboard
@using Privasight.Wasm.Configs
@using Privasight.Wasm.Services
@using Privasight.Wasm.UI

@inject DashboardService _dashboardService

<PageTitle>@_dashboardSetting?.Name | Privasight</PageTitle>

@if (_dashboardSetting != null)
{
    <Dashboard Setting="_dashboardSetting" Company="_company"/>
}

@code {
    private DashboardSetting? _dashboardSetting;
    private AvailableCompany _company;

    [Parameter]
    public string UrlCompany { get; set; } = "";

    [Parameter]
    public int Id { get; set; }

    protected override void OnParametersSet()
    {
        _dashboardSetting = _dashboardService.DashboardSettings?[_company][Id];
    }

    protected override async Task OnInitializedAsync()
    {
        _company = Enum.Parse<AvailableCompany>(UrlCompany);
        if (_dashboardService.DashboardSettings == null)
        {
            await _dashboardService.SetDashboardSettingsFromStorage();
        }
    }
}
```

Figure 5.20: CustomDashboard.radzor

5.10 Transformation of Data in Technical Format

As mentioned in the section 3.3, some data from the zip file is stored in a technical format. Two JsonConverters, FbConverter and UnixDateTimeOffsetConverter, were created to deal with this issue.

FBConverter is used to handle the incorrectly encoded characters. The issue with the data is

that when Facebook outputs the data, it has decoded the UTF-8 encoded data with Latin-1 instead. So in order to revert this mistake, all string data has to be re-encoded to Latin-1 and then decoded as UTF-8. This converter was added to the custom converters list before the deserialisation of the JSON files, as shown in figure 5.13. All strings from the JSON files will be fed into this converter to correct the encoding.

UnixDateTimeOffsetConverter is used to convert the Unix timestamp to the DateTimeOffset object in C#. Properties that use Unix timestamp will be marked with this converter in their model class. An example is shown in figure 5.21. The Timestamp property is marked with the converter using the JsonConverter attribute.

```
2 references
public class InteractedAdvertiser : DbTableObj
{
    [DataListValue]
    [JsonPropertyName("title")]
    [DetailedTableDisplayData("Title", description: "The title of the Ad")]
    0 references
    public string? AdTitle { get; set; }

    [JsonPropertyName("action")]
    [DetailedTableDisplayData(nameof(Interaction), description: "The interaction with the Ad")]
    1 reference
    public string? Interaction { get; set; }

    [JsonConverter(typeof(UnixDateTimeOffsetConverter))]
    [JsonPropertyName("timestamp")]
    [DetailedTableDisplayData("Interacted on")]
    0 references
    public DateTimeOffset TimeStamp { get; set; }
}
```

Figure 5.21: InteractedAdvertiser Class

5.11 Customisation of Data Visualisation

The "Manage Dashboards" page contains all the logic for users to create their custom dashboards. This page is built in the same way as the design described in section 4.4.6. The data on this page is linked to the DashboardService. This page is also the only place where the user can enter DashboardSettings into the application.

Figures 5.22 and 5.23 show the screenshots of the page from the live website [42].

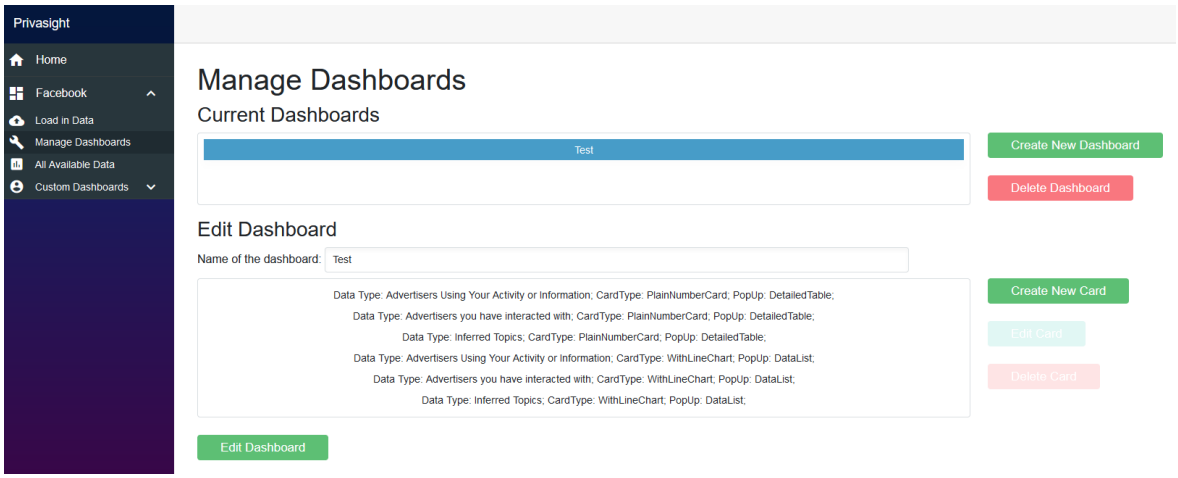
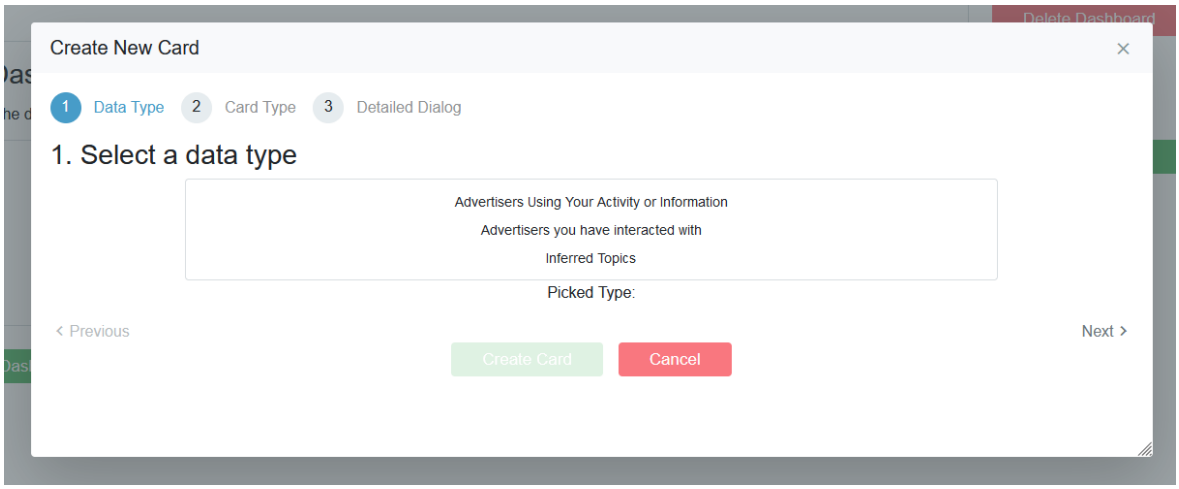
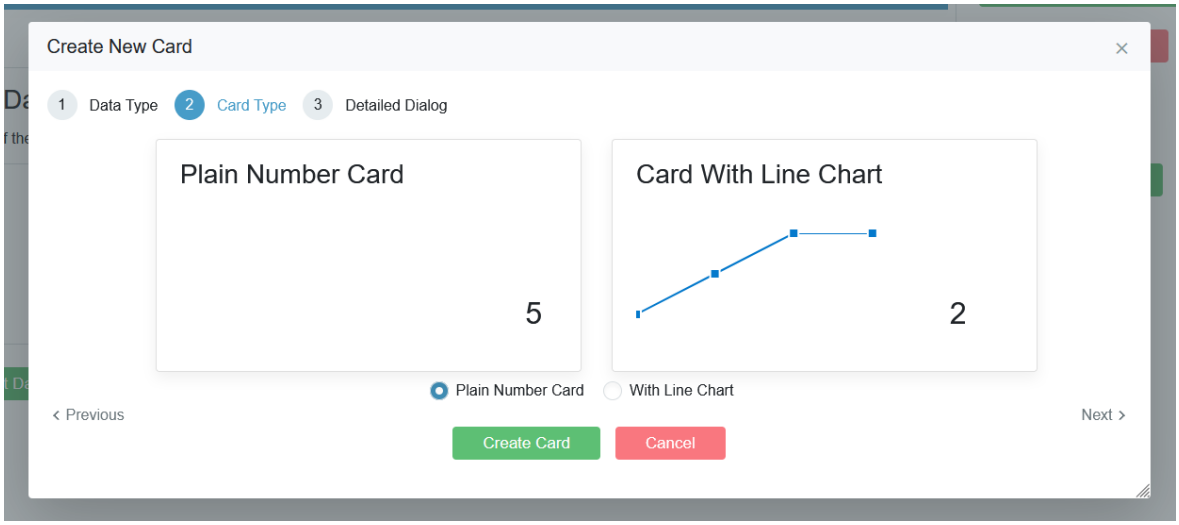


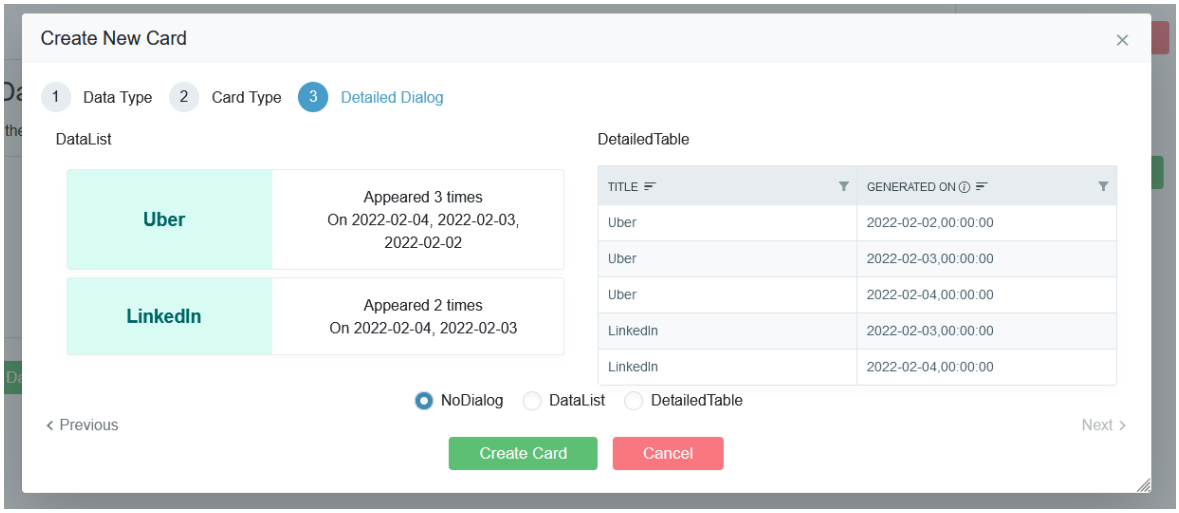
Figure 5.22: Screenshot of the Manage Dashboards page on the live website, taken on 09-05-2022



(a) Step 1 of EditCard



(b) Step 2 of EditCard



(c) Step 3 of EditCard

Figure 5.23: Screenshot of the EditCard component on the live website, taken on 09-05-2022

Chapter 6

Testing

6.1 Overview

Testing is a way to validate the implementation of the application. Various testing methodologies were investigated. Based on the project's nature and time constraints, beta testing was determined to be the most valuable one among all since it can be used to collect the feedback from end-users and validate the requirements from the end-users perspective. Users' feedback can be used to evaluate the customer satisfaction with the product and determine the future development focus.

Before the start of the beta test, manual test was done based on the requirement list to make sure that the site will work as expected during the beta test.

6.2 Beta Testing

The beta test started after the completion of the implementation of most requirements. It was done anonymously using Google Forms. There are no assumptions about the test participants. The questionnaire can be found in the appendix A. Users were given the link to the site [42] hosted on GitHub for testing.

The questionnaire contains 14 questions: 11 multiple-choice questions and 3 optional open-ended questions. They are grouped into two parts.

The questions in the first part mostly concern a specific function.

Question 1 is about whether the user understands the aim of Privasight based on the information on the home screen. The responses indicate that the current description is mostly adequate.

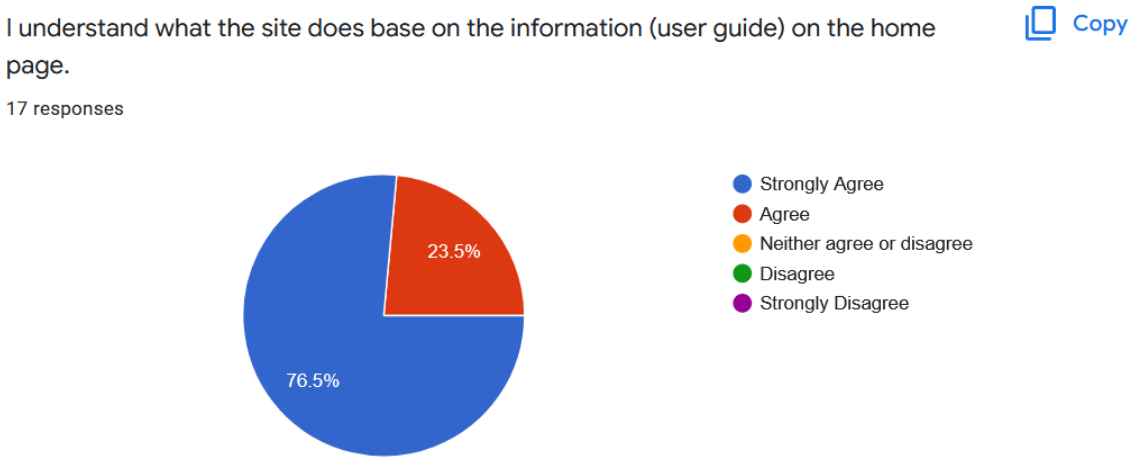


Figure 6.1: Responses to question 1 of the beta testing questionnaire

Question 2 is about whether the implementation of non-functional requirement 5, "Explain in detail how Privasight handles user's data on a static page", helps to achieve the objective regarding user trust on Privasight. Based on responses to the open-ended question, some technical terms used could be confusing. That could be the reason for the decrease in the percentage of participants who strongly agree with this statement.

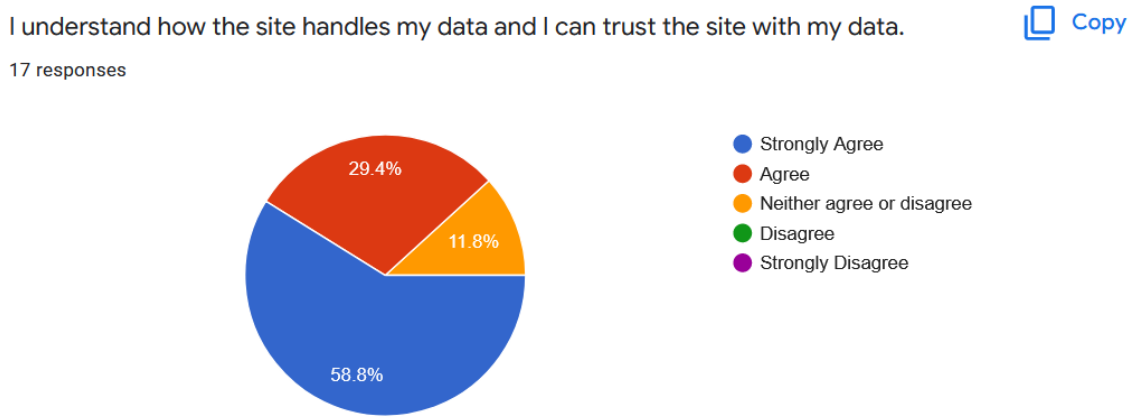


Figure 6.2: Responses to question 2 of the beta testing questionnaire

Question 3 asks if the user understands the instructions for downloading data from Facebook (functional requirement 3), and question 4 asks if the application can read in the downloaded data (functional requirement 1). Both statements have a large majority of strong agreement, implying that both requirements were properly implemented.

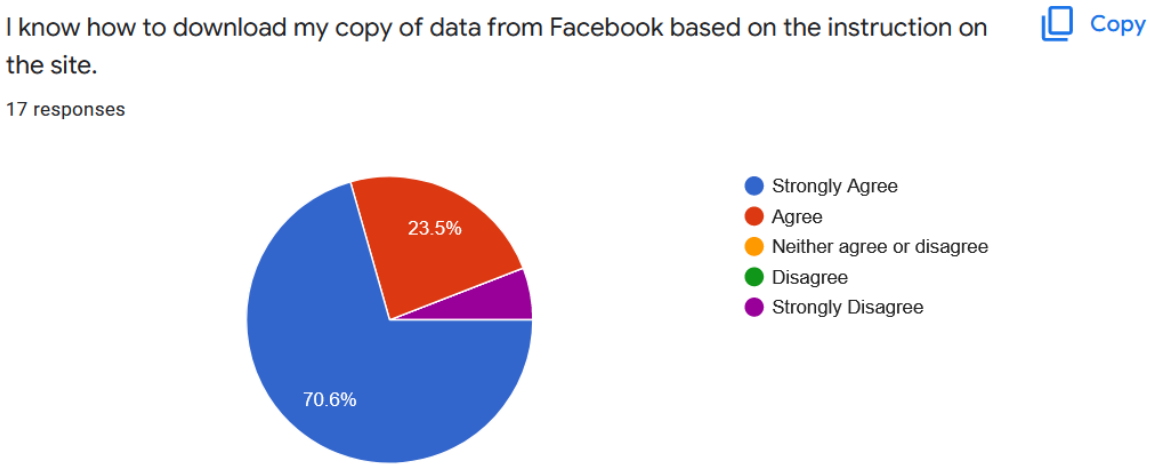


Figure 6.3: Responses to question 3 of the beta testing questionnaire

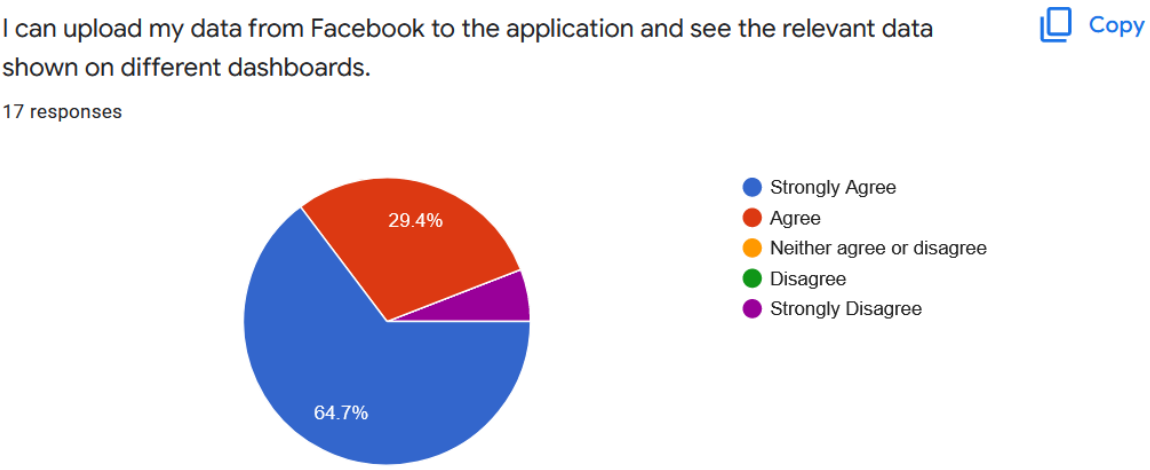


Figure 6.4: Responses to question 4 of the beta testing questionnaire

Question 5 asks whether the display components (functional requirement 2) contribute to the objective of assisting users in understanding their data. The majority of responses indicate that it does.

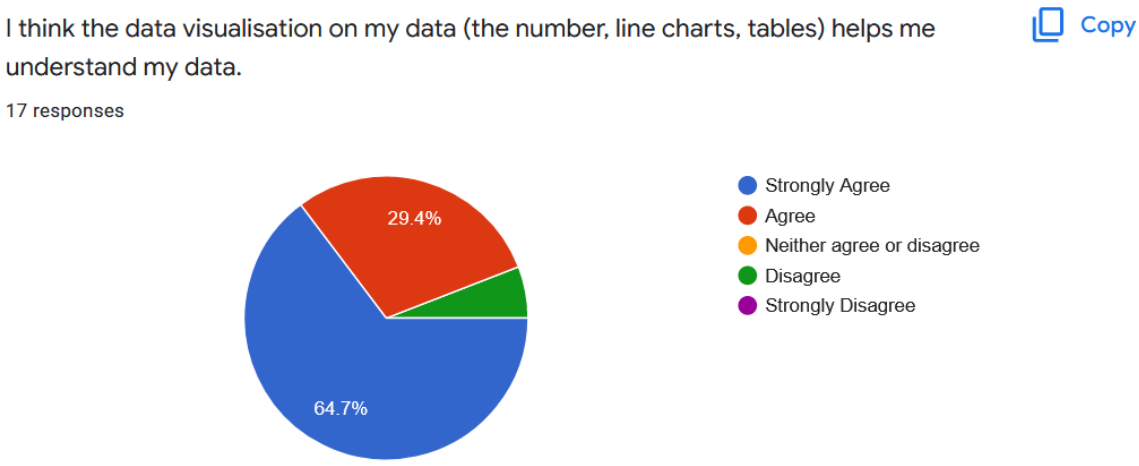


Figure 6.5: Responses to question 5 of the beta testing questionnaire

Question 6 is about whether functional requirement 6, "customise data visualisation", is properly implemented, and question 7 is concerning whether the user thinks that this is a nice feature. Based on the responses from the open-ended questions, some changes are needed for the UI on the "Manage Dashboard" page, but in general, it is a nice feature.

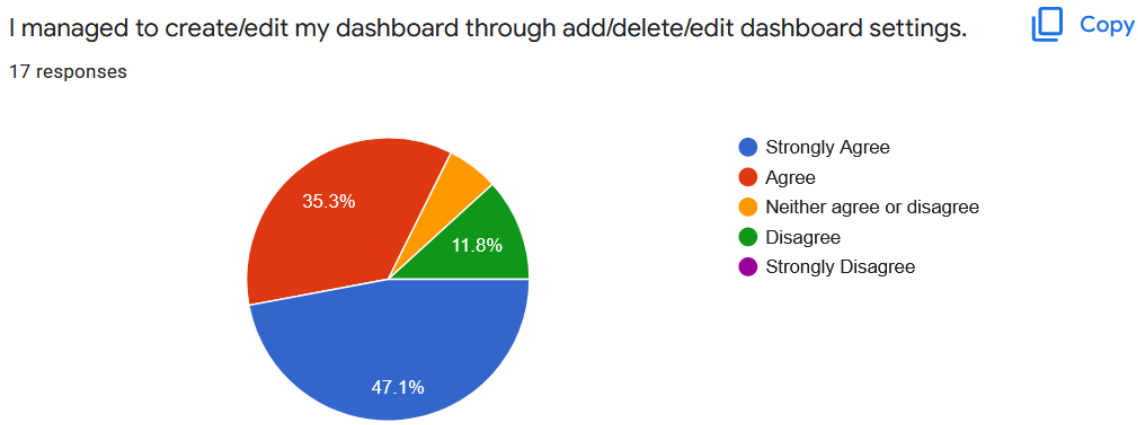


Figure 6.6: Responses to question 6 of the beta testing questionnaire

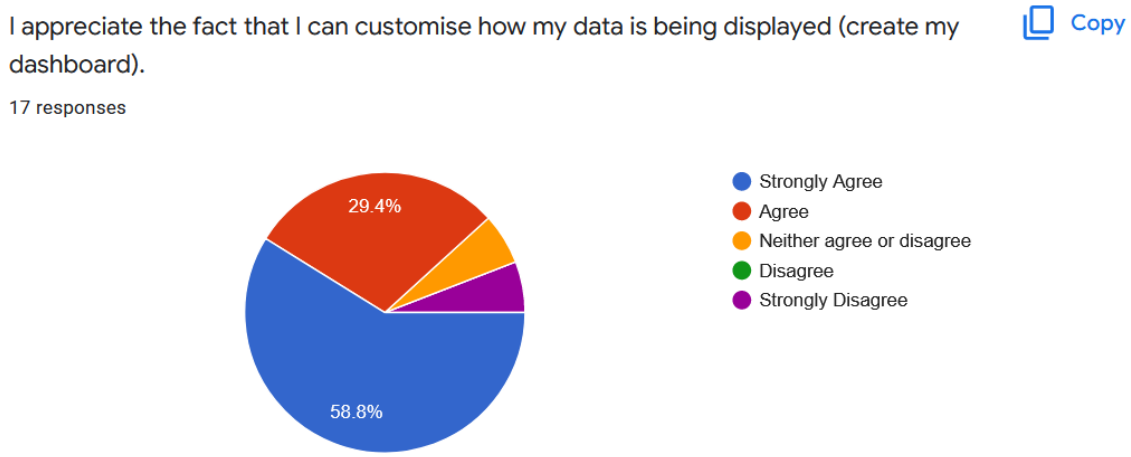


Figure 6.7: Responses to question 7 of the beta testing questionnaire

Question 8 is about whether users appreciate the data encryption feature and the responses show that most users do.

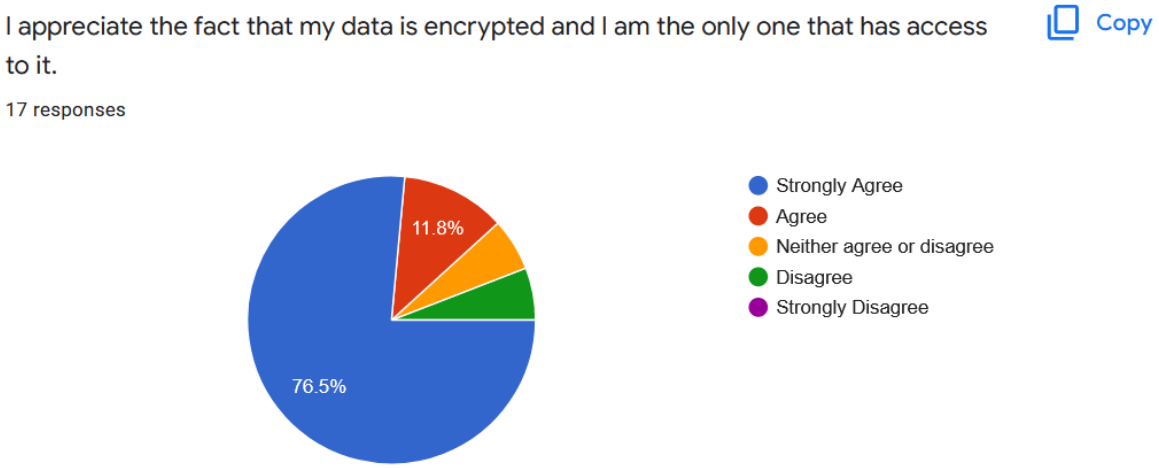


Figure 6.8: Responses to question 8 of the beta testing questionnaire

The second part of the questionnaire focuses more on the overall evaluation of Privasight.

Based on the responses to questions 9 and 10, it is reasonable to assume that the user experience is pleasant. Some UI flaws, according to the open-ended responses, may have contributed to the drop in strong agreement.

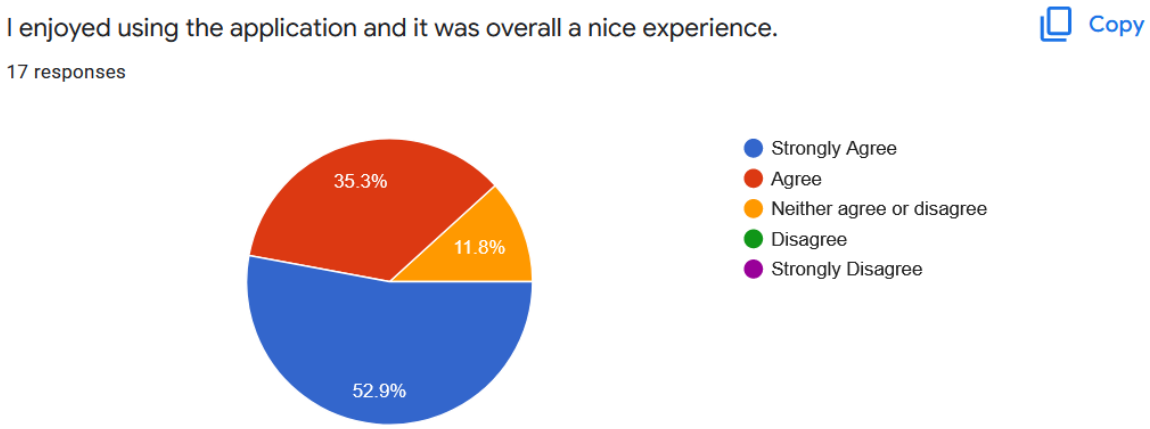


Figure 6.9: Responses to question 9 of the beta testing questionnaire

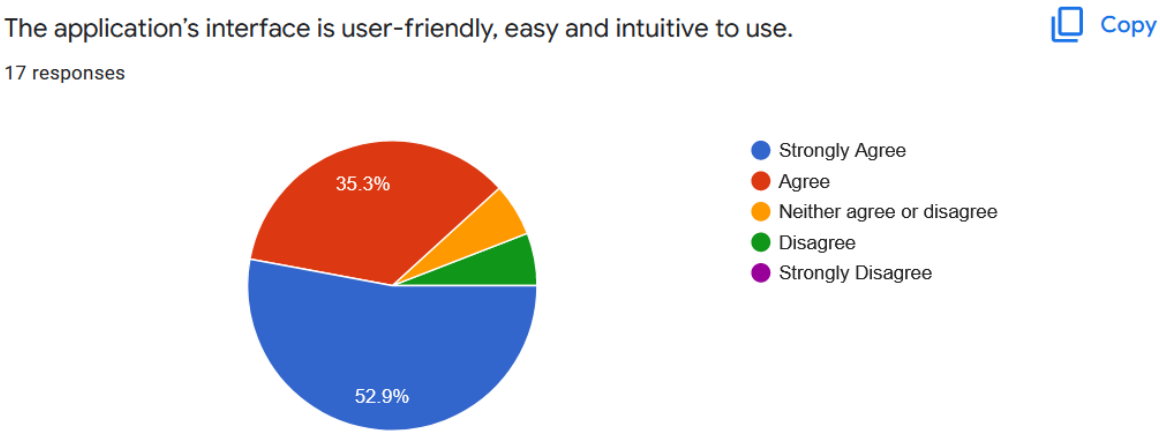


Figure 6.10: Responses to question 10 of the beta testing questionnaire

Question 11 inquires about the tool's usefulness, and the majority responded that it is useful, and they will use it again.

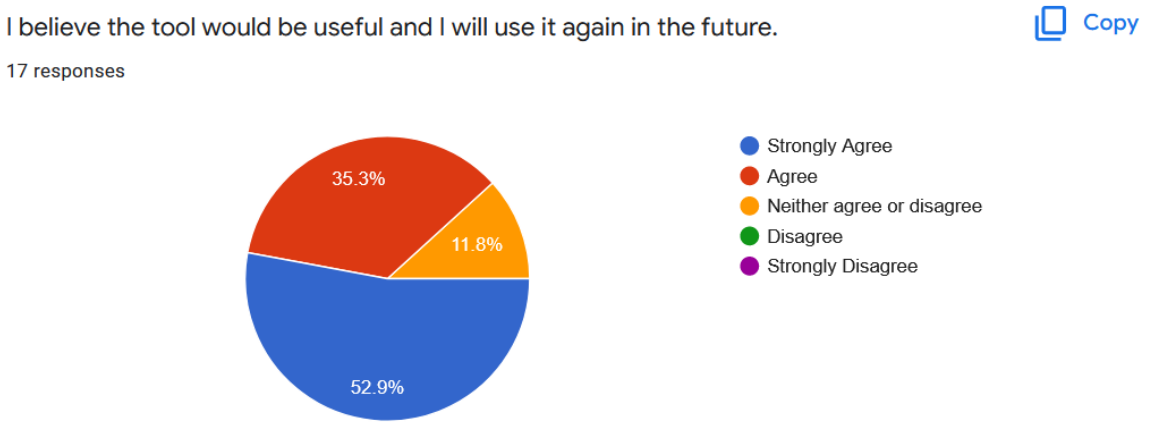


Figure 6.11: Responses to question 11 of the beta testing questionnaire

The rest of the questions are optional open-ended questions. The full responses can be found in the appendix A. Below is a summary of the responses.

The first question is about what the user likes the most about the application. Most talked about the display of their data in a straightforward manner and the data visualisation. Some also appreciate the locality of the site, the UI/UX, and the user guide.

The second question is about what part of the site should be changed. There are some opinions about the display, such as the display of links and text alignment. There are also some complaints about the password and errors regarding the encryption system. There are also some comments about the lack of guidelines on how to navigate through the website.

The third question is for general comment. There are many comments on how they think that this is a great tool. There are also error reports on the password/encryption system. There is also one comment saying that Facebook did not output the data into a zip file. This is an issue that requires further investigation for future development.

6.3 Conclusion

Based on the beta test results, it is fair to conclude that, despite some flaws in the UI and functionalities, the application is a useful tool that customers value.

Chapter 7

Results and Discussion

7.1 Overview

This chapter aims to evaluate the final outcome of this project. Section 7.2 evaluates the completion and test status of the requirement list outlined in chapter 3. Section 7.3 talks about how generic the application is. Section 7.4 discusses the application's technical debts. Section 7.5 suggests the directions of future development.

7.2 Completion and Test Status of Requirements

Table 7.1 and 7.2 outlined the completion status of the requirements. Most are completed as planned. All requirements were manually tested and some were even tested by beta users mentioned in section 6.2.

Requirements marked as "Manual Tested" were completed and present on the site during beta testing. Users were not explicitly asked to test them, so they are not marked as "Beta Tested".

ID	Requirement	Priority	Completion	Test
1	Read in the return data from Facebook	High	Completed	Beta Tested
2	Create display components for the ingested data	High	Completed	Beta Tested
3	Create description on how to download the required Facebook Data	High	Completed	Beta Tested
4	Transform data to a non-technical format	High	Completed	Manual Tested
5	Show all loaded data on a default dashboard	High	Completed	Beta Tested
6	Customise data visualisation	High	Completed	Beta Tested
7	Explain the technical terms	Medium	Completed	Beta Tested
8	Compare SAR data received on different dates	Medium	Completed	Manual Tested
9	Links to change the privacy setting at each corresponding section of the dashboard	Low	Completed	Manual Tested

Table 7.1: Completion status of the Functional Requirements

ID	Requirement	Priority	Completion	Test
1	Get user's data only by requesting them to upload the required files	High	Completed	NA
2	Keep all processing within the user's browser	High	Completed	Beta Tested
3	Store all data locally	High	Completed	Manual Tested
4	Encrypt user's SAR data	High	Partially Completed	Beta Tested
5	Explain in detail how Privasight handles user's data on a static page	Medium	Completed	Beta Tested
6	Be open-sourced	Medium	Not Completed	NA
7	Make Privasight be a generic data-driven framework	Low	Completed	NA

Table 7.2: Completion status of the Non-Functional Requirements

The source code is currently not open-sourced but it is ready for release.

Data encryption is the last implemented feature. Due to the time constraint, it is not fully completed. The implementation of error handling is limited. It is the reason for most errors reported in the beta test.

7.3 Genericness of the application

The non-functional development goal is to create a generic data-driven framework. After development, it can be concluded that this goal has been achieved. The front-end of the application is highly modular so as to accommodate new visualisations. The dependencies of the actual data are limited to the two files within the Configs folder as mentioned in section 4.3.4.2.

Only two things need to be added for the expansion of data categories from Facebook: the actual model class extending the FileWrapper interface and the type reference to the class in FBConfigs (as mentioned in section 4.3.3).

For the expansion of supported companies, a similar package like the Facebook Model package needs to be created. It will include the actual data classes and a configuration class akin to FBConfigs. The name of the company need to be added to the AvailableCompany enum and the CompanyConfigs needs to be updated accordingly.

7.4 Technical Debts

Technical debt is the extra development work caused when development emphasis is placed on time over quality. There are two technical debts identified for this project.

The first one is the dependencies on two JSON libraries. As mentioned in section 5.4, Newtwonsoft.Json was introduced to the project to solve the issues related to polymorphic deserialization. Before that, the default library (System.Text.Json) was used and is still in use for data ingestion. The use of two libraries will easily cause confusion during future devel-

opment and is generally not a good practice. It is suggested to remove the dependency on Netwonsoft.Json in future development by writing the required custom converter.

encrypt.js is the source of the second technical debt. This file contains the logic for data storage and encryption, which violates the single responsibility principle. It will easily cause confusion, especially as the filename suggested, this file should only be responsible for encryption, but functions that only concern storage, setDashboardSettings and getDashboardSettings, are also in this file. In future development, this file should be broken down into at least two files to separate the responsibilities.

7.5 Future Directions

This section highlights the directions for future development.

7.5.1 Follow Up on Beta Test's Result

There are several flaws highlighted by the beta testers. These are the things that should be handled at the start of future development.

The most important one is on the guidance of the expected flow of behaviour. Few participants responded that they did not know where to navigate after uploading their data. A skippable pop-up introduction should be added to instruct users through the process.

There were also comments on the "Manage Dashboard" page regarding the update of the edit of dashboards. Currently, updates are only loaded into the system when the update button is clicked. This should be changed to an instant update after the user makes the changes.

There were also many opinions and error reports regarding the Encryption Service. As mentioned before, this feature is not fully implemented. Future development should start with fixing the error handling of the functions in this service. Also, users are currently required to create an at least 16-character password, which is complained by some beta users. Research and updates should be conducted in the future to strike a balance between security and user experience.

7.5.2 Testing

Due to the time constraints, not many resources were advocated for testing. The site is currently only being manually tested in a black box manner. Unit testing and integration testing should be added to the project to make sure that each component works as expected and that they work together properly as well.

The run of the tests should be added to the CI/CD pipeline after the completion of these tests to make sure that no breaking changes will be pushed to the client.

7.5.3 Expansion on Data Sources and Visualisation

Currently, the application only handles three data categories from Facebook. As mentioned in section 7.3, the app is capable of doing a lot more. Future development should start by expanding on the supported data categories from Facebook. In terms of the expansion of supported companies, Instagram would be a good start since it shares a lot of data structures with Facebook.

For now, the application only supports a few types of data visualisation. Expansion of the visualisation types would not be difficult since the front-end is highly modular. Figure 7.1 illustrates the future vision for the dashboard.

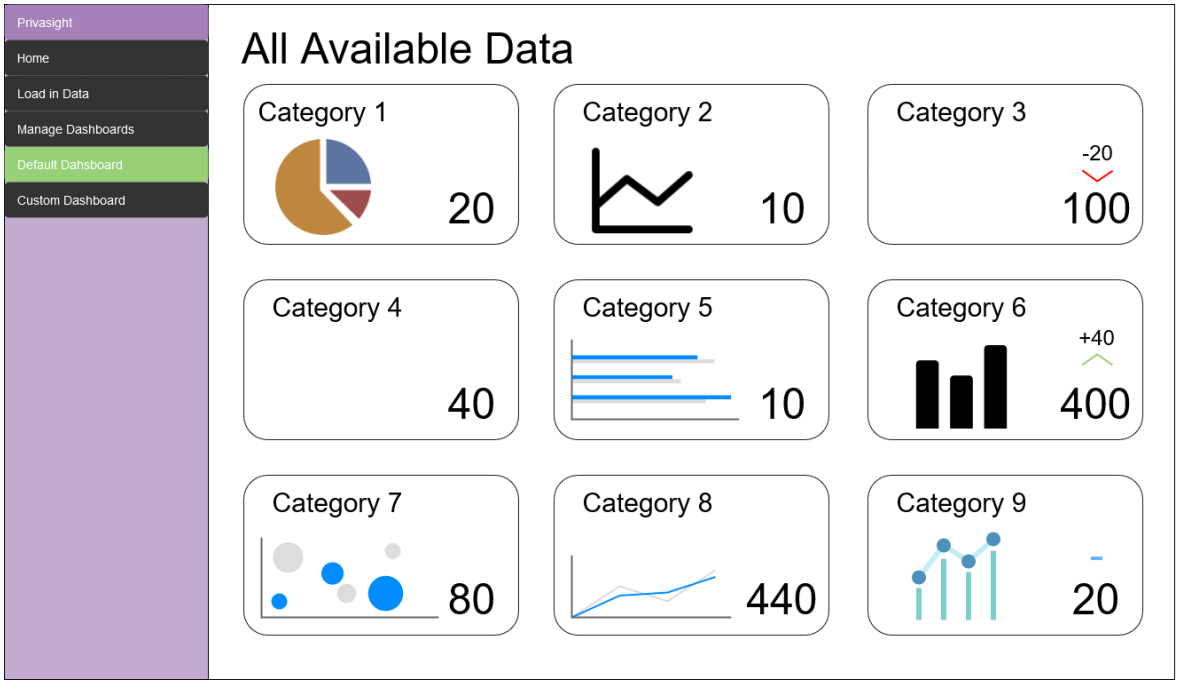


Figure 7.1: Mock-up design for the future dashboard

7.5.4 Expansion on Platforms

Since the application runs within a browser, its capabilities are limited. With .NET MAUI, expansion to the desktop environment should be simple, allowing Privasight to accomplish more.

Chapter 8

Conclusions

The project started with the aim of helping users understand their subject access request (SAR) data. Based on the findings of the survey in chapter 2, two objectives were derived: to provide the services to users in a trustworthy way and to give users the freedom to customise their data visualisation. Requirements were defined in chapter 3. Design (chapter 4) and implementation (chapter 5) were done with the requirements in mind.

The final product is a static web app hosted on GitHub [42]. Users can also download the application as a progressive web app for offline use. As the results in chapter 6 suggested, this application was well implemented and valued by the end-users. It is worthwhile to mention that most test participants appreciate the three objectives and the measures that were done to achieve them.

Based on the current progress, the application, Privasight, can be seen as a minimum viable product. After the improvement is made based on the beta test findings, Privasight is ready to be released to the public to bring some actual value to the real world.

References

[1] Information Commissioner's Office. *Guide to the General Data Protection Regulation.* URL: https://www.gov.uk/government/publications/guide-to-the-general-data-protection-regulation (visited on December 8, 2021).

[2] European Union. *Art. 15 GDPR - Right of access by the data subject.* URL: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:02016R0679-20160504&from=EN#tocId22 (visited on December 8, 2021).

[3] Republic of Singapore. *Personal data protection act 2012 - Singapore Statutes Online.* URL: https://sso.agc.gov.sg/Act/PDPA2012 (visited on December 8, 2021).

[4] Information Commissioner's Office. *How do we recognise a subject access request (SAR)?* URL: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/right-of-access/how-do-we-recognise-a-subject-access-request-sar/ (visited on December 8, 2021).

[5] Jumbo. *Jumbo: Take back control of your data and privacy.* URL: https://www.withjumbo.com/ (visited on May 3, 2022).

[6] Udaptor. *UDAPTOR - Main page.* URL: https://udaptor.io/index.html (visited on May 3, 2022).

[7] datacy. *datacy - The future of data.* URL: https://datacy.com/ (visited on December 8, 2021).

[8] Mine. *Mine - the future of data ownership.* URL: https://saymine.com (visited on December 8, 2021).

[9] Rita Team. *Rita - It's your data, own it.* URL: https://ritapersonaldata.com/ (visited on December 8, 2021).

[10] datacy. *datacy - Privacy policy.* URL: https://datacy.com/privacy-policy (visited on December 8, 2021).

[11] datacy. *datacy - FAQ.* URL: https://datacy.com/faq (visited on December 8, 2021).

[12] Product Hunt. *Datacy - Make your data earn for you. Control what to sell & to whom — Product Hunt.* URL: https://www.producthunt.com/posts/datacy (visited on December 8, 2021).

[13] Mine. *Privacy Policy - Mine.* URL: https://saymine.com/privacy-policy (visited on December 8, 2021).

[14] Andy Wen. *Investing in the security of our API ecosystem: updates on the security assessment — Google Cloud Blog*. URL: https://cloud.google.com/blog/products/identity-security/investing-security-our-api-ecosystem-updates-security-assessment (visited on December 8, 2021).

[15] Product Hunt. *Mine - Discover and control what the internet knows about you — Product Hunt*. URL: https://www.producthunt.com/posts/mine-2 (visited on December 8, 2021).

[16] Rita Personal Data. *Introducing: The Rita Score*. URL: https://ritapersonaldata.medium.com/introducing-the-rita-score-8a1e72d1d8ec (visited on December 8, 2021).

[17] Rita Team. *Rita - Privacy Policy*. URL: https://ritapersonaldata.com/privacypolicy.html (visited on December 8, 2021).

[18] Product Hunt. *Rita Personal Data - Collect, view and control your data — Product Hunt*. URL: https://www.producthunt.com/posts/rita-personal-data (visited on December 8, 2021).

[19] Google Play. *Rita Personal Data - Apps on Google Play*. URL: https://play.google.com/store/apps/details?id=com.ritapersonaldata.rita (visited on December 8, 2021).

[20] Apple App Store. *Rita Personal Data on the App Store*. URL: https://apps.apple.com/us/app/rita-personal-data/id1538361872 (visited on December 8, 2021).

[21] Pete LePage. *Storage for the web*. URL: https://web.dev/storage-for-the-web/ (visited on December 8, 2021).

[22] Pete LePage and Thomas Steiner. *The File System Access API: simplifying access to local files*. URL: https://web.dev/file-system-access/ (visited on December 8, 2021).

[23] OpenJS Foundation and Electron contributors. *Electron — Build cross-platform desktop apps with JavaScript,HTML, and CSS*. URL: https://www.electronjs.org/ (visited on December 8, 2021).

[24] Dennis Sheppard. "Introduction to Progressive Web Apps". In: *Beginning Progressive Web App Development*. Apress, 2017, pp. 3–10. DOI: 10.1007/978-1-4842-3090-9_1. URL: https://doi.org/10.1007/978-1-4842-3090-9_1.

[25] MSEdgeTeam et al. *Overview of progressive Web Apps (PWAs) - Microsft Edge Development—Microsoft Docs*. URL: https://docs.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium (visited on December 8, 2021).

[26] David Britch and J. Conrey. *What is .NET MAUI? - .NET MAUI—Microsoft Docs*. URL: https://docs.microsoft.com/en-us/dotnet/maui/what-is-maui (visited on December 8, 2021).

[27] James Montemagno and Eilon Lipton. *Introduction to .NET MAUI Blazor*. URL: https://docs.microsoft.com/en-us/shows/xamarinshow/introduction-to-net-maui-blazor--the-xamarin-show (visited on December 8, 2021).

[28] David Britch. *Single project - .NET MAUI—Microsoft Docs*. URL: https://docs.microsoft.com/en-us/dotnet/maui/fundamentals/single-project (visited on December 8, 2021).

[29] Luke Latham et al. *ASP.NET Core Blazor—Microsoft Docs*. URL: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0 (visited on May 5, 2022).

[30] Steve Smith et al. *Choose Between Traditional Web Apps and Single Page Apps —Microsoft Docs*. URL: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps (visited on December 8, 2021).

[31] Mozilla.org. *IndexedDB API*. URL: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API (visited on May 5, 2022).

[32] Mozilla.org. *Web Storage API*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API (visited on May 5, 2022).

[33] Norton. *What is encryption and how does it protect your data?* URL: https://us.norton.com/internetsecurity-privacy-what-is-encryption.html (visited on May 4, 2022).

[34] Bill Wagner et al. *Use record types - C# tutorial—Microsoft Docs*. URL: https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/tutorials/records (visited on May 7, 2022).

[35] Luke Latham et al. *ASP.NET Core Blazor dependency injection—Microsoft Docs*. URL: https://docs.microsoft.com/en-us/aspnet/core/blazor/fundamentals/dependency-injection?view=aspnetcore-6.0 (visited on May 7, 2022).

[36] MatBlazor. *MatBlazor*. URL: https://www.matblazor.com/ (visited on May 8, 2022).

[37] Developer Express Inc. *Blazor UI components*. URL: https://www.devexpress.com/blazor/ (visited on May 8, 2022).

[38] Syncfusion. *Blazor components examples & demos*. URL: https://blazor.syncfusion.com/demos/ (visited on May 8, 2022).

[39] Radzen. *Free Blazor Components*. URL: https://blazor.radzen.com/ (visited on May 8, 2022).

[40] NuGet. *NuGet Gallery — Radzen.Blazor 3.18.7*. URL: https://www.nuget.org/packages/Radzen.Blazor/ (visited on May 8, 2022).

[41] Lucidchart. *UML sequence diagram tutorial*. URL: https://www.lucidchart.com/pages/uml-sequence-diagram (visited on May 8, 2022).

[42] Hayley Kwok. *Privasight - An insight into Your Privacy*. URL: https://hayley-kwok.github.io/Privasight (visited on May 9, 2022).

[43] GitHub. *Continuous Integration and Continuous Delivery (CI/CD) Fundamentals — GitHub Resources*. URL: https://resources.github.com/ci-cd/ (visited on May 9, 2022).

[44] Steve Sanderson. *New Blazor WebAssembly capabilities in .NET 6*. URL: https://www.youtube.com/watch?v=kesUNeBZ1Os (visited on May 9, 2022).

[45] Microsoft. *Entity Framework documentation*. URL: https://docs.microsoft.com/en-us/ef/ (visited on May 9, 2022).

[46] Blazored. *Blazored LocalStorage: A library to provide access to local storage in Blazor applications*. URL: https://github.com/Blazored/LocalStorage (visited on May 9, 2022).

[47] NuGet. *NuGet Gallery — Blazored.LocalStorage 4.2.0*. URL: https://www.nuget.org/packages/Blazored.LocalStorage/ (visited on May 9, 2022).

[48] Newtonsoft. *Introduction — Json.NET*. URL: https://www.newtonsoft.com/json/help/html/introduction.html (visited on May 9, 2022).

[49] NuGet. *NuGet Gallery — Newtonsoft.Json 13.0.1*. URL: https://www.nuget.org/packages/Newtonsoft.Json/ (visited on May 9, 2022).

[50] Jake Archibald et al. *idb-keyval: A super-simple-small promise-based keyval store implemented with IndexedDB*. URL: https://github.com/jakearchibald/idb (visited on May 9, 2022).

[51] Jake Archibald et al. *idb: IndexedDB, but with promises*. URL: https://github.com/jakearchibald/idb (visited on May 9, 2022).

[52] Npmcharts.com. *Compare npm downloads for idb, localForage, localForage, localforage, dexie, zangoDB, zangoDB, zangodb, pouchdb, jsstore, lovefield and idb-keyval - npmcharts*. URL: https://npmcharts.com/compare/idb,localForage,localForage,localforage,dexie,zangoDB,zangoDB,zangodb,pouchdb,jsstore,lovefield,idb-keyval?ref=hackernoon.com&interval=30 (visited on May 9, 2022).

[53] World Wide Web Consortium (W3C). *Web Cryptography API*. URL: https://www.w3.org/TR/WebCryptoAPI/ (visited on May 10, 2022).

[54] Luke Latham et al. *ASP.NET Core Blazor file uploads—Microsoft Docs*. URL: https://docs.microsoft.com/en-us/aspnet/core/blazor/file-uploads?view=aspnetcore-6.0&pivots=webassembly (visited on May 5, 2022).

Appendices

Appendix A

Beta Testing Related Documents

This appendix includes the documents related to the beta test. They are presented in the order listed: the questionnaire used and the responses to the open-ended questions in the questionnaire.

"Understand What Big Tech Have About You" Dissertation

You are being invited to take part in an undergraduate dissertation project as an evaluator. Before you decide whether or not to participate, it is important for you to understand why the research is being done and what it will involve. Please take time to read the following information carefully and discuss it with others if you wish.

*Required

1. What is the project's purpose?

This project aims to provide users with a user-friendly way to understand the data they received from submitting the Subject Access Request (SAR) entitled by the GDPR to big tech companies and allow them to perform analysis on the data.

These will involve a tool that shows users and assists them in understanding the SAR data. The tool will have an explanation of technical terms and links directing users back to their privacy settings so that they can change these settings easier.

2. What will I have to do if I take part?

You will be provided with a link to the prototype website. The website should be compatible with any modern browsers.

The research aims to evaluate the experience of using the tool and its effectiveness. There are descriptions and clear instructions on how to use the site on the home page of the site. Once you finished experimenting with the site, you will be asked to participate in a survey by answering and providing feedback regarding your thoughts on the site. The whole process should take approximately 20 minutes.

3. What will happen to the data collected, and the results of the research project?

All responses are anonymous, no identifiable personal data will be collected, and email addresses are not collected for this survey. All data will be handled in compliance with UK GDPR, stored on the University of Sheffield Google Drive, and will be destroyed upon completion of the dissertation project. By completing and submitting this form, you are consenting to your data being processed as described above.

The result of this survey will be analysed and discussed in my undergraduate dissertation project at the University of Sheffield. Summarized results may be used in subsequent research publications, research talks and for teaching.

If you are happy with the above, please continue. If you have any further questions, please contact me at wykwok1@sheffield.ac.uk. Thank you for your time.

Here is the link to the prototype website https://hayley-kwok.github.io/Privasight/
If you have visited this site before, please press Ctrl + F5 to make sure that you are on the latest version before the start of the test. It is recommended to visit the site

Questions regarding a specific function

on a computer.

It is being tested on modern browsers like Mircosoft Edge, Google Chrome and Mozilla Firefox. (There might be some issues with private window on Firefox. If anything happens, please try it out on Edge.)

If there are any issues accessing the site, please contact me at wykwok1@sheffield.ac.uk.

It is recommended to finish these questions in sequential order.

1. I understand what the site does base on the information (user guide) on the home page. *

Mark only one oval.

- ◯ Strongly Agree
- ◯ Agree
- ◯ Neither agree or disagree
- ◯ Disagree
- ◯ Strongly Disagree

2. I understand how the site handles my data and I can trust the site with my data. *

Mark only one oval.

- ◯ Strongly Agree
- ◯ Agree
- ◯ Neither agree or disagree
- ◯ Disagree
- ◯ Strongly Disagree

3. I know how to download my copy of data from Facebook based on the instruction on the site. *

Mark only one oval.

- Strongly Agree
- Agree
- Neither agree or disagree
- Disagree
- Strongly Disagree

4. I can upload my data from Facebook to the application and see the relevant data shown on different dashboards. *

Mark only one oval.

- Strongly Agree
- Agree
- Neither agree or disagree
- Disagree
- Strongly Disagree

5. I think the data visualisation on my data (the number, line charts, tables) helps me understand my data. *

Mark only one oval.

- Strongly Agree
- Agree
- Neither agree or disagree
- Disagree
- Strongly Disagree

6. I managed to create/edit my dashboard through add/delete/edit dashboard settings. *

Mark only one oval.

- Strongly Agree
- Agree
- Neither agree or disagree
- Disagree
- Strongly Disagree

7. I appreciate the fact that I can customise how my data is being displayed (create my dashboard). *

Mark only one oval.

- Strongly Agree
- Agree
- Neither agree or disagree
- Disagree
- Strongly Disagree

8. I appreciate the fact that my data is encrypted and I am the only one that has access to it. *

Mark only one oval.

- Strongly Agree
- Agree
- Neither agree or disagree
- Disagree
- Strongly Disagree

Overall Evaluation

9. I enjoyed using the application and it was overall a nice experience. *

Mark only one oval.

- Strongly Agree
- Agree
- Neither agree or disagree
- Disagree
- Strongly Disagree

10. The application's interface is user-friendly, easy and intuitive to use. *

Mark only one oval.

- Strongly Agree
- Agree
- Neither agree or disagree
- Disagree
- Strongly Disagree

11. I believe the tool would be useful and I will use it again in the future. *

Mark only one oval.

- Strongly Agree
- Agree
- Neither agree or disagree
- Disagree
- Strongly Disagree

12. What did you like the most about the application?

13. What did you like the least about the application or is there anything you will change about the application?

14. Any other comment?

Question 1: What did you like the most about the application?

Answer 1:	Being able to know which advertisers I have interacted with
Answer 2:	I like how I can see all of my data, the interface is super friendly, I love all the options available to me. I also love that I am given the option to immediately change my settings in Facebook, that is extremely useful.
Answer 3:	the table
Answer 4:	Relatively straight forward, clean front-end and it's quick. Tutorial on how to download the data is useful. Also like the assurance given on the homepage on a fairly technical level how the security of this works. Love the Privasight name some genius must've thought of that.
Answer 5:	How approachable and user-friendly it is. I think every area was covered thoroughly, and there was not a moment when I was confused.
Answer 6:	The application is very intuitive and the user guide is very useful and clear
Answer 7:	I like how easy it is to understand and use .
Answer 8:	Graphical display of input data
Answer 9:	It's so simple to use, and prioritises privacy by not sending the data over a network
Answer 10:	It has a good that there was a video that showed the "How to" it made it easier to use the site and visually showed how to use it
Answer 11:	Knowing what data facebook has about myself.
Answer 12:	I can see the data used by FB
Answer 13:	HELPFUL
Answer 14:	The interface that show when did the data appeared on my feed (the icons and structure looks nice)
Answer 15:	the friendly interface and the intuitive controls
Answer 16:	I really liked that I could view my data in an easy to read format. It was eye-opening to see how many companies use my data. The colours and overall look of the website is also very appealing. Definitely will be using it again in the future.
Answer 17:	Ease of use - very well explained usage instructions, very clear description of how data is handled by the application, and overall a nice and smooth UX.

Question 2: What did you like the least about the application or is there anything you will change about the application?

Answer 1:	The load data button is not very appealing but that is so minor, it doesn't even bother me
Answer 2:	Maybe a note on the front-end, try to align and keep the text centered as modern web designs usually are, makes the users eyes move left to right across the screen less haha.
Answer 3:	The 16 character password . Not sure if its neccessary for the encryption method to work. But I managed to bypass it just by cancelling the popup many time and still managed to upload my data :P
Answer 4:	Yeah, two things (don't need to add them but general design ideas) 1. In the tutorial to download the fb file, the first point says "Go to this link", and basically every UI guideline in the world will tell you to name the <a> tags more verbosely and clearer, e.g. "Go to <Facebook Personal Data Dashboard>", where everything in the <> is the clickable link. Second thing, after uploading the zip it's good that the Data loading Status tells you the progress, but after it switched to loading finished I didn't know where to click. I think it'd be a good idea to change that text to "Loading finished. Please go to All Available Data

	tab on the left", or even better have a button that would redirect there.
Answer 5:	nil
Answer 6:	All good, potentially adding a comparision of average data to the user's specific data
Answer 7:	N/A
Answer 8:	I think there are a lot of words and people can start get confused with the technical terms used on the application, using less technical language and breaking it up with visual aids would make it easier to read through.
Answer 9:	Add the ability to visualize more data insights. For example, data about news feed, messages, and facebook marketplace.
Answer 10:	It's not direct that I have to click all available data to view the data I uploaded. I had to watch the video again to find where the generated dashboard is. There is dashboard I can see but no options to edit the one I currently have. So I could not test the editing function. Fonts are too small to read with ease.
Answer 11:	NTH
Answer 12:	I am not sure how relevant is to know what other companies use/ know about me. It is right that I can visualize what type of data they have used but this will not change my the way I interact with other platforms (such as Facebook(Meta)).
Answer 13:	the need to remember the password
Answer 14:	There were some parts where I didn't know how to navigate the website, for example, when creating the dashboard, I have to press the create dashboard button first, then create a new card and then press another create dashboard button again. I didn't realise that I would have to press that final button as I thought that that would just make another dashboard.
Answer 15:	There seemed to be a slight problem, when editing a card from a dashboard, the card doesn't actually update unless you then click the 'edit dashboard' button after clicking the 'update card' button. Also, after entering the password after pressing the 'delete loaded facebook data' button, the dialog says the password was incorrect (it wasn't), but the data is still deleted. The same happens when uploading data to the application, where the dialog says the password is incorrect but the data is successfully uploaded.

Question 3: Any other comment?

Answer 1:	I strongly believe that this is and incredible and useful tool that I would like to use in my daily life.
Answer 2:	1. I think the website is a bit difficult to comprehend for those who possesses low literacy level. It would be great if you can add a bit infographics / flow chart to tell people how to use your site. 2. After I uploaded the data on one page, I had to guess what is the next step and tried to press on "Manage Dashboards". Though I can "guess", but for user experience perspective, it would be better if you can set up something to guide the user to go to another page.
Answer 3:	Good job Hayley! Maybe nice to expand this website in the future and include more than just Facebook as your title Big Tech suggests. I'd be interested to see my Google data (like Google has their own solution called Google Dashboard)
Answer 4:	I am terrified about what Facebook knows about me, but I guess that's a good thing. I will actually tell my mom to use this website to check her FB account, and let her more aware of how her careless decisions on social media benefit the companies. Overall, I think this project is a real eye-opener.
Answer 5:	Nil
Answer 6:	NAH
Answer 7:	Good work on the website looks very professional
Answer 8:	NO
Answer 9:	There are a few improvements that can be made: multiple users to be able to

	use the same system, a few more explicit error messages (alerts) when something is not inputted right [such as: when creating a password, if it has less than 16 chars, I will just be asked again to input a password and I will not be told why did the previous password not worked to be created]; improved security such as: a user should not be able to just refresh the website and try as many times as he wants; furthermore, the web-page should be hosted instead of an addition to the GitHub web-page.
Answer 10:	On the other hand, I think that overall the project was able to inform the user (in this case me) of what types of data a company used about me.
Answer 11:	None
Answer 12:	I struggled to upload my file to the website, as it wasn't downloaded in the format that it was meant to be done. For example, mine was never zipped, hence I had to zip it myself. But since the downloaded content was an encompassing file with the two separate files (ads_info and your_topics) within it, I was zipping the encompassing file that showed me no data. I later realised that I need to just zip the two separate files together and it worked!
Answer 13:	Great idea and great application!