DIGITALWAVES – THE DIGITAL FOOTPRINT ASSISTANT

COMP3000 Computing Project

(s) Harry Ormandy
harry.ormandy@students.plymouth.ac.uk

Contents

Abstract

The modern technology-enabled world is awash with data. With massive volumes of data being generated by internet users, personal data being available online has become a real issue. Searching for this data online can be complicated and overwhelming for the average user due to the variety of sources it comes from. If left available, cyber criminals can use this personal information to commit crimes such as identity theft and fraud.

DigitalWaves is a deployed server service that aims to solve this problem. It provides a user-friendly interface for users to enter their personal information, which is then used to query various popular APIs for matches. Once the user has found matches for their data in the results, they can review their score and how to improve their digital footprint for each entry.

The application was developed using the agile development model – this was effective as it allowed a dynamic and iterative development style which could pivot quickly based on client feedback and interaction. Initial background research and user story creation meant that requirements could be abstracted, from which a sprint plan was created to structure the further design, development and testing phases. The Django Python framework was used for development due to its built in scalability and security (Django, 2024). SQLite was used for databases, and JavaScript was used for communication between back-end and the user interface. Sprints were devised so an MVP was delivered early in the development process which then evolved based on feedback and further abstraction of requirements.

Unit testing was used to confirm a robust application, allowing the removal of several bugs. User Acceptance Testing confirmed a user-friendly experience, with 100% of users being able to complete the tasks and 50% stating they would use the application in the future.

The project concluded to be a success, producing an application with an intuitive User Interface that simplified the process of searching for personal information online. User Acceptance Testing demonstrated the goals were met, with 100% of users feeling confident using the application. Further improvements and expansions will be made on this application in the future, with the expansion of data sources and APIs, as well as commercialisation and AI incorporation for enhanced support in data removal.

Project repository:

The application code can be accessed through the repository link:

https://github.com/harry976/Comp3000Project

Chapter 1: The Problem

With the internet increasing in size by a factor of 300 from 2005 to 2020, totalling to 40,000 exabytes globally in 2020 (Gantz, J. F. et al, 2011), the quantity of personal information available online is staggering. Most of this will be unmonitored and unaddressed by users due to the complexity and size of the problem. The cost of cybercrime is increasing annually, costing an annual estimate of £27 billion (Cabinet Office, 2009) and predicted to rise, highlighting the ever-growing threat of lacklustre cyber security. With identity theft reports reaching over 1 million in 2023 (Akin, 2019), cyber-enabled crime is becoming more dangerous, with these crimes being driven by data breaches, exposing personal information.

Each existing tool uses Open Source Intelligence (OSINT) techniques to gather information from publicly available sources about an individual. When used by an attacker, these techniques can enable common cyber crimes such as phishing and spear phishing, which, when tailored to the victim with their data, seem even more realistic. Personal data can also be used in social engineering attacks, using data such as interests or personal connections to build trust and pretending to be someone the victim knows. These techniques create a large attack surface, which for most individuals, is never mitigated. This highlights the need for a tool that helps ordinary users to use OSINT to defend themselves and reduce their digital footprint, reducing the likelihood of cyber attacks having the detail to convince a user and be successful.

For the general non-technical user, current state of the art methods to addressing this concern are confusing. Powerful industry-standard tools such as Maltego and Shodan have intimidating User Interfaces (UIs) that allow a technical user freedom in investigations, but provide little support for a general user. Other tools such as HaveIBeenPwned are useful to a general user and provide a simple UI, however are limited in the data they present. This gap in the existing tool market poses a significant risk to an everyday user, as they are not aware of the personal data about them online. Furthermore, if they were to try and address their digital footprint, no tool would give them the depth of information or simple User Experience (UX) that they need to understand the process of identifying personal information, and how to respond to it.

Despite the availability of tools like Maltego and HaveIBeenPwned, a significant gap remains in providing user-friendly solutions to non-technical users to manage their digital footprint.

Following the initial topic research and identification of the gap in the current state of the art, the high-level project objectives and scope were defined:

1. Research and identify key APIs or data sources for information gathering
2. Create a simple, intuitive UI for a non-technical user
3. Implement security parameters for user account data

4. Provide different suggestions for users on how to improve their digital footprint
5. Track and display the user's progress through multiple scans
6. Enable users to view and delete their data

By making the process of finding personal information less technical through an easy-to-use UI, more everyday users will be able to manage their digital footprint effectively. The improvement in digital footprint hygiene from general internet users will greatly help to reduce the quantity of personal data sold by data brokers and used in cyber-enabled crimes such as identity theft and fraud, reducing the cost on the individual user and the contribution these crimes have to the global cybercrime bill.

When creating an Open Source Intelligence (OSINT) tool, ethical concerns must be addressed. OSINT tools can be used to scrape information without consent, even if the data is publicly available. To ensure the application is used ethically, the Menlo report (Kenneally, E. et.al, 2012) was used to guide strong user protection principles. Data privacy laws must also be considered to ensure the application complies to national and international laws. GDPR mandates user consent and minimal data storing. Processing of personal data will be justified and local. Failure to comply with these laws could result in fines of up to 20 million euros or 4% of the companies' annual turnover, whichever is higher (European Union, 2016).

For this project, the dissertation supervisor, Dr Nathan Clarke, acted as the client.

Chapter 2: Current state of the art

2.1: Existing product research

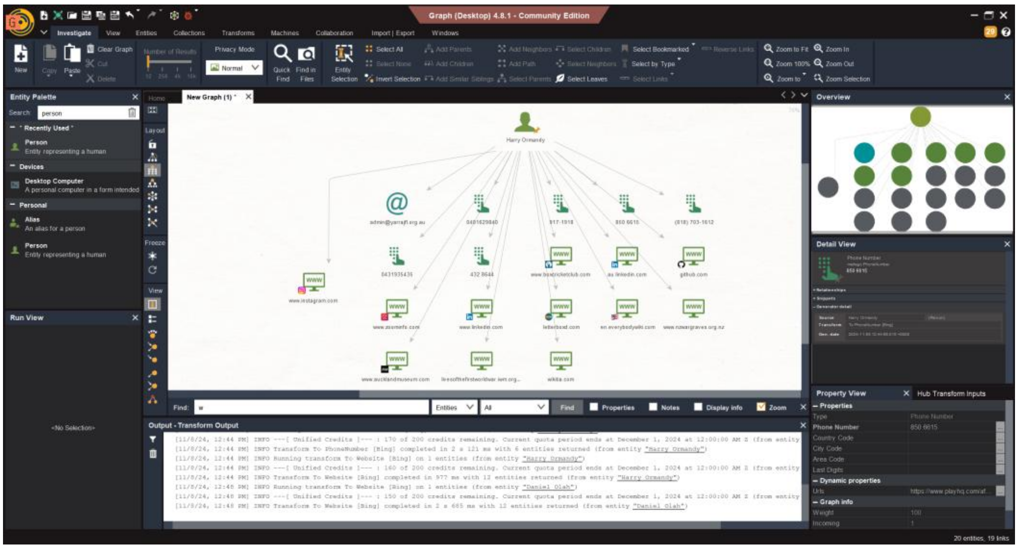
This section aims to study the existing products, to further identify gaps in the current state of the art.

Maltego

Maltego is an Open Source Intelligence tool that allows for mapping and linking of information to aid in investigations.

Pros	Cons
Graph-based GUI for information linking	Technically focussed
Easy to link APIs to	Difficult to decide what APIs are needed
Automatic searching for data when a transformer is selected	Can be a paid service – APIs may need paid keys
Technical information is easily found	Limited functionality on community edition
	Inefficient for searching for personal data

Whilst Maltego is an efficient tool for technical OSINT and mapping out data, it is not designed for the specific purpose of tracking digital footprints through social media, websites and companies. This is technically feasible, but inefficient in practice. Through a test search using the name “Harry Ormandy” with the base features included in the community edition, all that could be found was a GitHub account. There were also many false entries. It is not user friendly for a non-technically minded person, as transformers must be used to search for each type of information.



```
[11/8/24, 12:44 PM] INFO ---[ Unified Credits ]--- : 170 of 200 credits remaining. Current quota period ends at December 1, 2024 at 12:00:00 AM Z (from entity
[11/8/24, 12:44 PM] INFO Transform To PhoneNumber [Bing] completed in 2 s 121 ms with 6 entities returned (from entity "Harry Ormandy")
[11/8/24, 12:44 PM] INFO Running transform To Website [Bing] on 1 entities (from entity "Harry Ormandy")
[11/8/24, 12:44 PM] INFO ---[ Unified Credits ]--- : 160 of 200 credits remaining. Current quota period ends at December 1, 2024 at 12:00:00 AM Z (from entity
[11/8/24, 12:44 PM] INFO Transform To Website [Bing] completed in 977 ms with 12 entities returned (from entity "Harry Ormandy")
[11/8/24, 12:48 PM] INFO Running transform To Website [Bing] on 1 entities (from entity "Daniel Olah")
[11/8/24, 12:48 PM] INFO ---[ Unified Credits ]--- : 150 of 200 credits remaining. Current quota period ends at December 1, 2024 at 12:00:00 AM Z (from entity
[11/8/24, 12:48 PM] INFO Transform To Website [Bing] completed in 2 s 685 ms with 12 entities returned (from entity "Daniel Olah")
```

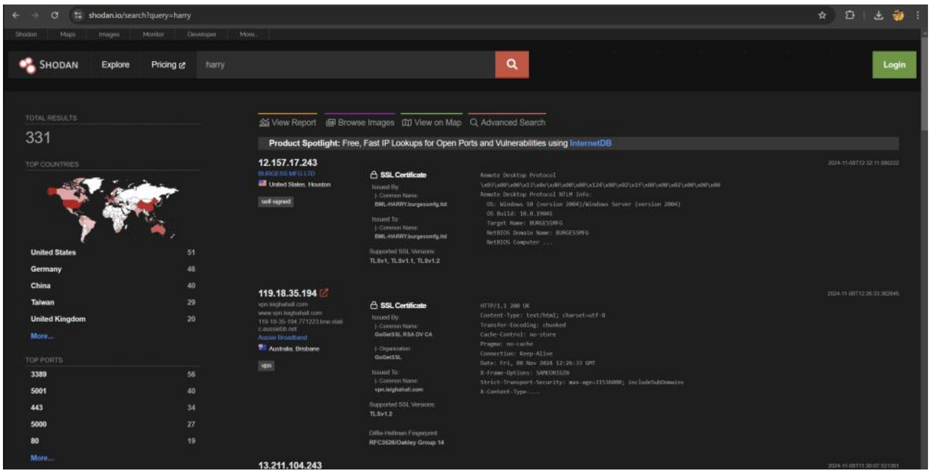
Figure 1: Maltego's dashboard in use

Shodan

Shodan is an online OSINT tool that focuses on IoT and any internet connected devices.

Pros	Cons
IoT searches for devices	Limited social media and internet carving features
Provides ports, IP addresses and more for devices	Can be a paid service
Geolocation data	Unintuitive UI design
Integration using API	Possible inconsistent data

Whilst Shodan is a powerful tool, it is not appealing to an inexperienced user. Searching for possible personal information revealed internet connected devices with a similar name. This is not what an application focusing on a digital footprint should focus on, however it could have useful features such as discovering devices owned by a user.



TOTAL RESULTS
331

TOP COUNTRIES

United States	51
Germany	46
China	40
Taiwan	29
United Kingdom	20

More...

TOP PORTS

3389	56
5001	40
443	34
5000	27
80	19

More...

Product Spotlight: Free, Fast IP Lookups for Open Ports and Vulnerabilities using InternetDB

12.157.17.243
BURGESS MFG LTD
United States, Houston
self-signed

2024-11-08T12:32:11.880222

SSL Certificate
Issued By:
|- Common Name:
BML-HARRY.burgessmfg.ltd
Issued To:
|- Common Name:
BML-HARRY.burgessmfg.ltd
Supported SSL Versions:
TLSv1, TLSv1.1, TLSv1.2

```
Remote Desktop Protocol
\x03\x00\x00\x13\x0e\xd0\x00\x00\x124\x00\x02\x1f\x08\x00\x02\x00\x00\x00
Remote Desktop Protocol NTLM Info:
  OS: Windows 10 (version 2004)/Windows Server (version 2004)
  OS Build: 10.0.19041
  Target Name: BURGESSMFG
  NetBIOS Domain Name: BURGESSMFG
  NetBIOS Computer ...
```

119.18.35.194
vpn.leighahall.com
www.vpn.leighahall.com
119-18-35-194.771223.bne.static.aussiebb.net
Aussie Broadband
Australia, Brisbane
vpn

2024-11-08T12:26:33.382945

SSL Certificate
Issued By:
|- Common Name:
GoGetSSL RSA DV CA
|- Organization:
GoGetSSL
Issued To:
|- Common Name:
vpn.leighahall.com
Supported SSL Versions:
TLSv1.2
Diffie-Hellman Fingerprint:
RFC3526/Oakley Group 14

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Cache-Control: no-store
Pragma: no-cache
Connection: Keep-Alive
Date: Fri, 08 Nov 2024 12:26:33 GMT
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-Content-Type-...
```

13.211.104.243

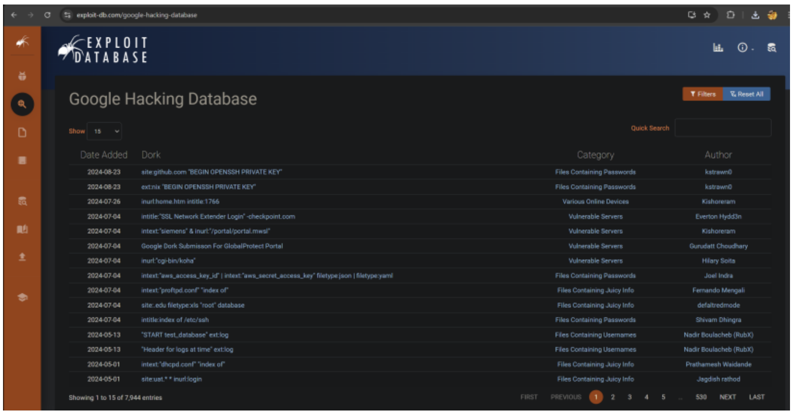
2024-11-08T11:30:07.521361

Figure 2: Shodan search dashboard

Google dorks

ExploitDB is an online tool that allows users to search through premade google advanced search filters to find leaked information like usernames and passwords.

Pros	Cons
Free	Ethical issues due to sensitivity
Useful web data discovery	Old data can be misleading
Versatile	Complex for new users
Can be integrated into other tools	More focused on exploits than personal data security



Google Hacking Database

Show 15

Date Added	Dork	Category	Author
2024-08-23	site:github.com "BEGIN OPENSSH PRIVATE KEY"	Files Containing Passwords	kstrawn0
2024-08-23	ext:nix "BEGIN OPENSSH PRIVATE KEY"	Files Containing Passwords	kstrawn0
2024-07-26	inurl:home.htm intitle:1766	Various Online Devices	Kishoreram
2024-07-04	intitle:"SSL Network Extender Login" -checkpoint.com	Vulnerable Servers	Everton Hydd3n
2024-07-04	intext:"siemens" & inurl:"/portal/portal.mwsl"	Vulnerable Servers	Kishoreram
2024-07-04	Google Dork Submisson For GlobalProtect Portal	Vulnerable Servers	Gurudatt Choudhary
2024-07-04	inurl:"cgi-bin/koha"	Vulnerable Servers	Hilary Soita
2024-07-04	intext:"aws_access_key_id" | intext:"aws_secret_access_key" filetype:json | filetype:yaml	Files Containing Passwords	Joel Indra
2024-07-04	intext:"proftpd.conf" "index of"	Files Containing Juicy Info	Fernando Mengali
2024-07-04	site:.edu filetype:xls "root" database	Files Containing Juicy Info	defaltredmode
2024-07-04	intitle:index of /etc/ssh	Files Containing Passwords	Shivam Dhingra
2024-05-13	"START test_database" ext:log	Files Containing Usernames	Nadir Boulacheb (RubX)
2024-05-13	"Header for logs at time" ext:log	Files Containing Usernames	Nadir Boulacheb (RubX)
2024-05-01	intext:"dhcpd.conf" "index of"	Files Containing Juicy Info	Prathamesh Waidande
2024-05-01	site:uat.* * inurl:login	Files Containing Juicy Info	Jagdish rathod

Showing 1 to 15 of 7,944 entries

Figure 3: ExploitDB dashboard

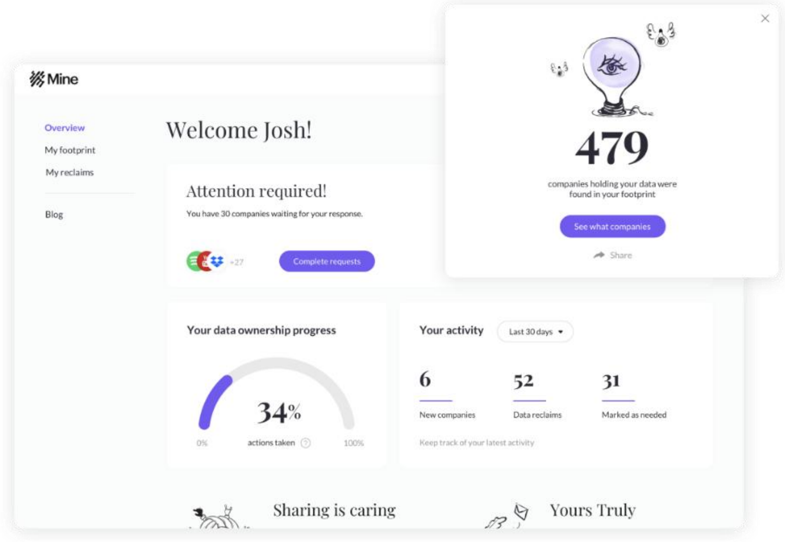
Whilst using the Google hacking database could uncover leaked usernames and passwords, it could be complicated to implement, as specific queries would have to be selected depending on the information the user provides. This application is difficult for an inexperienced user, as it focuses on the technical side of OSINT, with emphasis on finding vulnerable servers, leaked passwords and sensitive information. These features could be beneficial to integrate into other tools, however it does not have enough features to be an effective standalone OSINT tool for addressing a digital footprint.

MIME

Mime is an online application that specifically focuses on the automated removal of user data from companies.

Pros	Cons
Automated search and removal of data	Limited control over 3rd party websites
Legal compliance	Inconsistent data removal
Provides alerts to a user	Public records not taken into account
User friendly	Requires access to the users email account

Whilst MIME can be an effective and established digital footprint removal tool, several issues arise with its use. Due to the automation of the service, it has inconsistent success rates and limited control over third parties that may not comply with automated requests. The main issue with this level of automation in a service is that when creating the user account, it requires access to 'read the email' of a user. This is a breach of the users privacy, and would make a large number of users uncomfortable if they read the conditions of sharing their information with the service. It has many good features, however its downsides do not make it a viable choice.



Welcome Josh!

Attention required!

You have 30 companies waiting for your response.

479

companies holding your data were found in your footprint

Keep track of your latest activity

Figure 4: MIME dashboard

Summary of existing products

Whilst each pre-existing tool offers extensive helpful features, the majority focus on the technical side of OSINT, offering complex UIs and information on internet-enabled technology. These features do not align with the product vision, highlighting the current market's need for a non-technical digital footprint hygiene tool. The application that offers the most similarity is 'MIME', however its automation of processes requires for access to information that users may not be comfortable providing, such as email inbox access. Its automation also does not give users insight into how this information can affect them online; hence, it would need to be used constantly, as it does not encourage the user to improve.

2.2: Existing Literature review

Digital footprint tools use Open Source Intelligence (OSINT) techniques to uncover information online about a target from various sources. These can include social media, public records or news websites. The rapid increase in the use of online platforms has led to internet users having increasingly large digital footprints, and little awareness of the extent or implications this can have. Whilst OSINT techniques can be challenging for a non-technical user, they enable skilled individuals to exploit personal data for malicious purposes.

Human factors play a large part in users rejecting advances in security recommendations. Modern security tools are often aimed at technical and/or corporate users, prioritising flexibility over ease of use. (Herley. K, 2009) argues that user will ignore advances in security advice due to the costs in time and effort, and that, for the average user, these will outweigh the benefits. This effect is clear on cyber security breaches overall, seen by the continued prevalence of breaches caused by phishing

(GOV.UK, 2024) - for average users, the time cost is too high to study each email in depth, so users accept the fact that some emails may not be legitimate.

Similarly, in a corporate environment, (Beautement A. et al, 2008) argues that a trend exists in employees not following security policy and possibly causing breaches due to the anticipated cost to them and the benefits to the organisation. Furthermore, the study suggests that individuals have a “compliance budget”, and that employees have a limited capacity to comply with security policies, and overloading this can lead to non-compliance. Understanding these trends is vital when studying human factors within security tooling and why individuals may not comply to security advice. Most average users will take the chance in that they do not perceive an active threat in the consequences of if a breach was to occur with their data – and hence do not expend the additional effort in following security policy. Given these human tendencies, digital footprint tools that are aimed at an average user need to take into account that many users will not engage with a complicated system unless it is easy to use, and provides clear benefits to them.

Usability is critical in a non-technical user benefitting from a security tool. Difficulties arise due to most tools being designed for security professionals, producing a negative user experience for an average user. A compromise that is often made is between usability and security. (Sasse. M. A, et al., 2001) argues that security features often conflict with Human Computer Interaction (HCI) principles due to human factors such as ambushing for unaided recall of information. This reflects how poor usability often leads to neglect of the tool or incorrect use, rendering its security features ineffective. (Nielsen. J, 1994) provides insight on how tooling can avoid these pitfalls, by implementing system status visibility, error prevention and recovery and consistency. A vital part of usability principles also expanded upon is learnability – a user needs to be able use the tool intuitively first time, and not be bothered by ‘first time use’ hints and tips when they are familiar with how the application functions. Ensuring this means that non-technical users will feel far more comfortable in using the application, and less likely to deliberately circumvent its security features out of frustration and not benefit from the tool.

When considering making OSINT techniques more readily usable by non-technical users, the ethical use of these techniques must be considered. Whilst all data that is collected is publicly available, this does equal ethical neutrality. (Keneally. E. et al, 2012) describes, in the Menlo report by US government, ethical guidelines for research using network and information technologies. The technology must not harm, whilst maximising benefits for the user and minimising risks. Informed consent and privacy must always be provided, to ensure the user understands the privacy they are entitled to. Finally, respect for public interest must be achieved by maintaining accountability and transparency of the processes that the application uses. Third party application

Terms of Service violations must also be considered, as these TOS agreements often include data scraping as a breach. (Zimmer. M, 2010) explores these ethical concerns further, suggesting that publicly available data can be scraped in unethical ways, such as facial recognition datasets. Aggregation of data on specific person or persons can also equivalate to profiling, constituting a possible security risk.

In conclusion, existing research shows that modern security tooling is being developed with the functionality to protect users. However, they are mostly focused on a technical and corporate target demographic, meaning the non-technical user is uncatered for. A lack of consideration of human factors and a poor implementation of HCI principles has led to users becoming frustrated with security-focused tooling and deliberately circumventing security features designed to help them. Whilst making tools such as OSINT more available raises important ethical concerns, guidance can be followed to help avoid misuse. This research demonstrates a clear gap for a tool that balances usability, ethical data scraping and effective personal data recovery for the average non-technical user. Whilst the current research backs the existence of such a tool, one is yet to be developed.

2.3: LSEP considerations

When considering legalities with an OSINT-based application, the primary concern should be adherence to Data protection laws, such as GDPR (European Union, 2016) and the Data Protection Act (DPA) in the UK (GOV.UK, 2018). Users must maintain the right to access or delete their data, which should be stored locally wherever possible. There must also be consent and transparency on the processing of the data, to ensure users understand how their data is being used. They must maintain their right to withdraw consent at all times, should they wish to. As the application interacts with third-party services, Terms of Service (TOS) agreements also need to be considered. Many social media sites prohibit data scraping or automation, meaning there may be a risk of legal liability due to a breach of these TOS agreements. Due to this, users of the application could unknowingly be breaching TOS agreements, reflecting poorly on the application. To tackle this issue, official APIs are used where provided to avoid data scraping, keeping the application TOS compliant.

How the application affects society and users in general must also be considered. By creating an application that makes OSINT techniques simpler, the digital divide between technical and non-technical users is significantly narrowed. This was achieved through a simple and intuitive UI and UX, meaning that users with limited technical literacy can understand the information shown to them, and why. This does come at a cost of flexibility of the application, as users are only provided with information of sources that have been included before download, however more flexibility would lead to scope creep. Users also need to feel confident in the tool and how their information is being collected. Whilst all the data is public, aggregation and presentation of it in one place to a user could feel invasive. This was tackled by establishing trust through transparency – all data is stored locally, no confirmed entries are permanently stored, and data is used following third party TOS agreements.

Whilst OSINT techniques only scrape for publicly available information, this does not mean that it is ethical to collect, aggregate or display it. As discussed in the Menlo report (Kenneally, E. et. Al, 2012), the users must understand the data collected and why. Furthermore, the tool must maintain a respect for law and public interest, whilst maximising the benefit it provides. This is achieved through the User's ability to control what information is stored and used in searches. Users can delete or edit their data, aligning with GDPR's 'right to be forgotten' (European Union, 2016). Prevention of Misuse is also an important ethical consideration. The tool must avoid having a dual-use potential, not enabling crimes such as stalking or blackmail. This will be addressed by ensuring the user agrees to not search for information on anyone but themselves. Users must also create an account to ensure accountability.

The tool was developed to align with industry best practices. Honesty in capability was implemented, to ensure the user does not obtain false hope. The tool achieves this by providing advice to the user, and requiring them to rescan for their information to check for themselves if the information has been removed, rather than telling them so. The tool can also be used in an educational role, promoting digital security awareness and empowering non-technical users. This will be achieved through embedding tips on good privacy hygiene and how users can improve, encouraging long-term use of the tool, allowing non-technical users to become more informed on digital security over time.

Chapter 3: Requirements Analysis and Design

3.1: User Stories

Based on the gaps identified in existing product research and literature review, the following user stories represent the key needs a user would have of the application.

“As a user…”:

- I want to view what personal data about me is publicly accessible online, so that I can understand what people can see about me
- I want the application to explain the risks of having a large digital footprint, so I can understand why it matters
- I want the application to suggest specific steps I can take to reduce my digital footprint and improve my privacy
- I want the application to have a simple and intuitive interface, so I can use it without technical skills
- I want to see a score or indicator of my digital footprint progress, so I can track my improvements
- I want to know how my data is stored and processed, so I feel safe using the application

3.2: Requirements abstraction

The initial project aims were further developed to include additional detail and to align with the SMART goals framework – ensuring each aim was Specific, Measurable, Achievable, Realistic, and Time-bound:

- Research and identify at least 4 key APIs or data sources for gathering user data by the end of the planning sprint
- Create a simple, intuitive UI for a non-technical user by the end of January 2025
- Implement security parameters for user account data by the end of January 2025
- Provide at least 4 different suggestions for users on how to improve their digital footprint by March 2025
- Track and display the user’s progress through multiple scans by April 2025

The following requirements were abstracted following the user stories identified in 3.1, the gaps identified in product research and the high level SMART objectives. The requirements were split into functional and non-functional requirements to aid in clarity.

The functional requirements defined the core features and actions that the application needs to perform, focusing heavily on developing a non-technical UX:

1. The application must help the user understand why a digital footprint matters

2. The application must provide advice to the user on how to reduce their digital footprint, tailored to the type of information found
3. The application must calculate, track and display a progression score to show the user improving over time
4. The user must create an account to use the application to ensure accountability
5. User searches should be locally stored for progression score calculation and trend tracking
6. The application must only allow searches of the user
7. The application must perform API GET calls to gather public information
8. The information entered by the user should be used as parameters in API searches
9. The application should include a help section to help the user understand the application and digital footprint risks

The non-functional requirements define how the application will perform and behave to a user. Following the user stories, these will focus on the usability heuristics of the application:

1. The application should follow HCI principles wherever possible, ensuring a user-friendly design
2. The application must be easy to use, understand and navigate
3. The UI must be intuitive to a use and consistent
4. The system should respond quickly to avoid user frustration, and provide user feedback for background processes
5. The application should encourage long-term use through an education-focused UX and feedback loops

Using the background literature review and LSEP consideration, legal and ethical constraints were also abstracted. These clearly outline the laws and ethical standard the application needs to comply with:

1. The application must comply with GDPR and the UK DPA
2. A clear terms of service policy must be provided
3. Users must give consent before data is collected, stored or processed
4. Users must be able to withdraw consent at any time, observing their right to be forgotten
5. The application must not process data for purposes beyond the users own searches
6. The application must use official APIs wherever possible to avoid violation of third-party TOS agreements
7. All collected data should be stored locally where possible
8. Users must agree to not use the application to search for anyone but themselves

3.3: UML diagrams

Unified Modelling Language (UML) was used to help abstract the outlined requirements into defining the system architecture. This allowed for the core user interactions and logic to be visualised, and supported communication to the client, providing clear visualisation of the design approach. Although a variety of UML diagrams were considered, two were fully developed and referred to heavily throughout the product backlog creation and development of the prototype.

The system architecture diagram provides a high-level overview of the structure of the application, highlighting the key components that interact with one another. This helped in visualising how the different layers of the application communicated. A non-monolithic structure to an application is essential for industry-standard systems, as they allow for modularity, security and scalability. Separating layers also ensures future feature expansion is also possible, as each layer can be modified individually. Key layers were identified; the front-end, back-end processing, and the data layer.

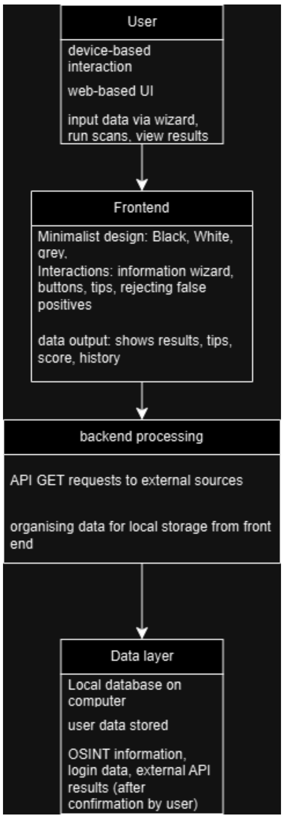

User
device-based interaction
web-based UI
input data via wizard, run scans, view results
Frontend
Minimalist design: Black, White, grey,
Interactions: information wizard, buttons, tips, rejecting false positives
data output: shows results, tips, score, history
backend processing
API GET requests to external sources
organising data for local storage from front end
Data layer
Local database on computer
user data stored
OSINT information, login data, external API results (after confirmation by user)

Figure 5: Architecture diagram

The user interaction flowchart heavily influenced the low-fidelity prototype creation, as it provides the main user journey from login and account creation to searching for their digital footprint. The flowchart was particularly helpful in identifying the logical branches and user decisions that needed to be implemented in prototype designs and the final product early on in the development process. The diagram was also used to great effect in abstracting the requirements further into smaller, more manageable tasks that could be completed in the implementation stage. A comprehensive flowchart also helped to manage scope creep, as it ensured that all user actions were accounted for during the implementation, prioritising key interactions over ad-hoc, unstructured feature additions made without a plan.

Whilst the main user journey of the final product differs slightly from the initial design presented in the flowchart, the diagram serves as a valuable reference point during development, identifying the key user actions and decision points. The flexibility of agile software development allowed for the project to evolve slightly and be adjusted during development to cater for blockers and client suggestions, however the overall structure presented by the flowchart remained relevant.

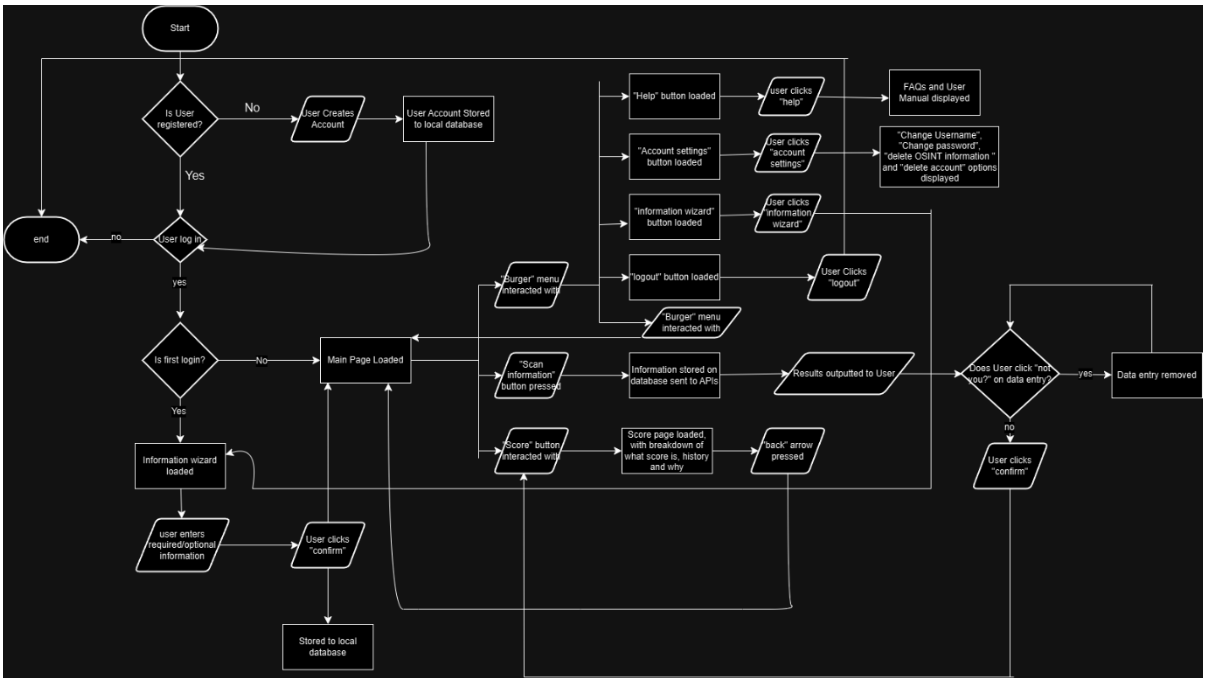

Start
Is User registered?
No
User Creates Account
User Account Stored to local database
Yes
end
no
User log in
yes
Is first login?
No
Main Page Loaded
Yes
Information wizard loaded
user enters required/optional information
User clicks "confirm"
Stored to local database
"Burger" menu interacted with
"Scan information" button pressed
"Score" button interacted with
"Help" button loaded
user clicks "help"
FAQs and User Manual displayed
"Account settings" button loaded
User clicks "account settings"
"Change Username", "Change password", "delete OSINT information " and "delete account" options displayed
"information wizard" button loaded
User clicks "information wizard"
"logout" button loaded
User Clicks "logout"
"Burger" menu interacted with
Information stored on database sent to APIs
Results outputted to User
Does User click "not you?" on data entry?
yes
Data entry removed
no
User clicks "confirm"
Score page loaded, with breakdown of what score is, history and why
"back" arrow pressed

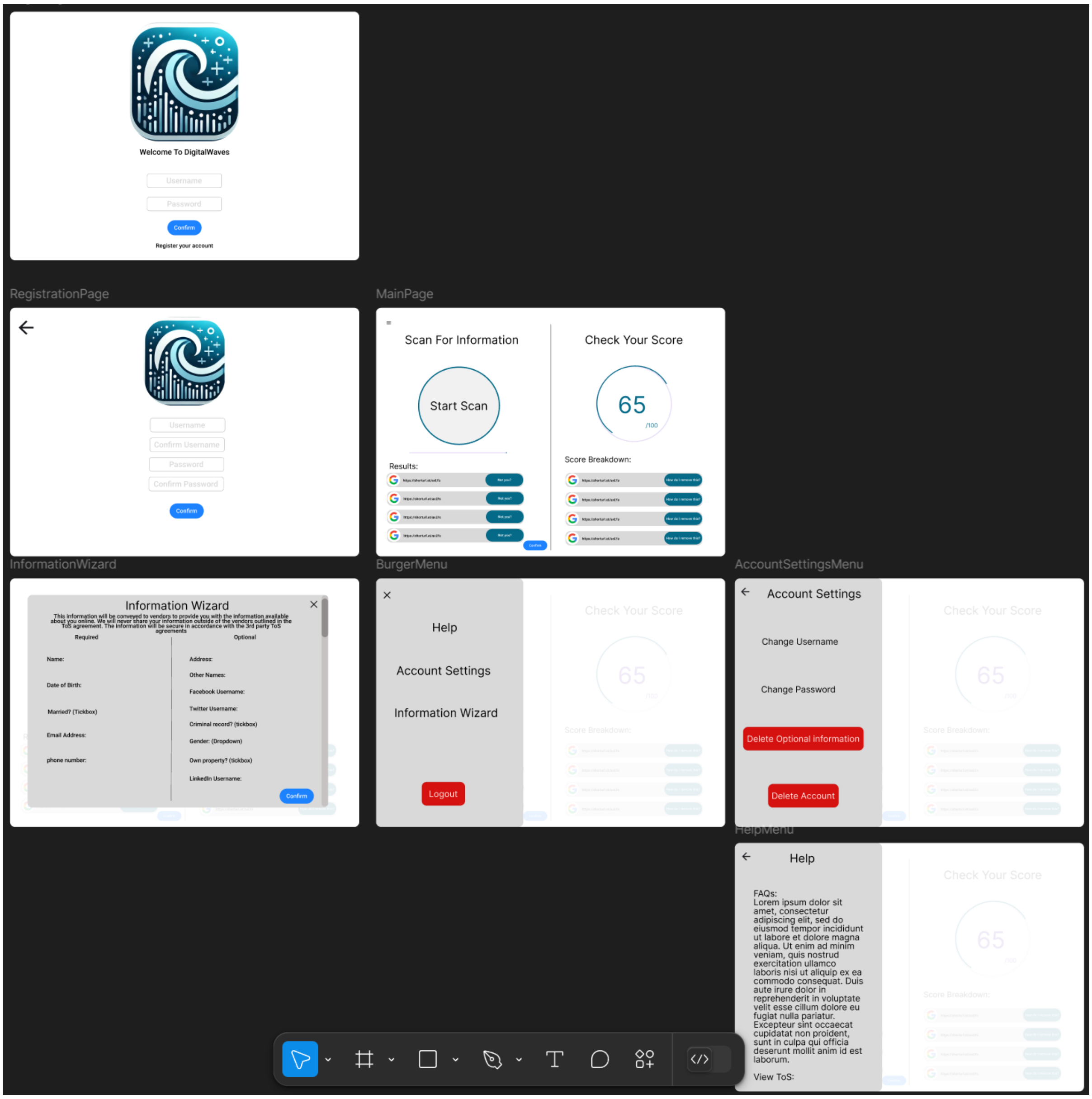
Figure 6: User flow diagram

3.4: Low and High Fidelity Designs

The UML diagrams and product backlog were used heavily in the design prototyping of the application. Creating designs before the implementation phase was critical in ensuring an informed structure and layout, particularly concerning the UI. With the application targeting non-technical users, the designs needed to align with HCI principles and user stories wherever possible. This ensured that the design provided an intuitive UX, minimising points of frustration wherever possible. This was an iterative process, with the client being consulted on designs to refine ideas and areas of the UI wherever needed.

Fundamental low fidelity designs were created initially, focusing on the basic layout of the UI. Figma was used to create digital wireframes of the essential features of the UI, such as the login and registration pages, as well as the main home page where searches and results can be interacted with, as well as the users score. This enabled the design to develop quickly, visualising the flow of the user, aligning closely with the logic provided by the flowchart UML diagram. These designs, whilst containing no navigation or functionality, were also able to identify early usability issues or layout challenges. This was helpful in the following implementation stage, where changes were made to the results section, as the designs only catered for a small scroll view of results. However, in the implementation of the MVP, due to the amount of information aggregated, it was deemed that this was inefficient and a tabs system with multiple sections was implemented.

The foundational wireframes created in Figma were developed further into higher-fidelity prototypes. These included consideration to colour theory and fonts, as well as logo and branding designs. The designs also included interactive elements, providing a simulation of the user journey. This simulation was paramount in ensuring a positive UX, as consistency across different components of the application was achievable, further improving the UI by ensuring no areas of the application could be different and confuse a user. Whilst the prototypes had no data-driven mechanics to them, they were critical in gathering feedback on the look and feel of the application before implementation began. A near-final representation of the UI was critical in aiding implementation, as clear specifications were then available to quickly develop an MVP based on the prototypes.



Information Wizard

This information will be conveyed to vendors to provide you with the information available about you online. We will never share your information outside of the vendors outlined in the ToS agreement. The information will be secure in accordance with the 3rd party ToS agreements

Required	Optional
Name:	Address:
	Other Names:
Date of Birth:	Facebook Username:
Married? (Tickbox)	Twitter Username:
	Criminal record? (tickbox)
Email Address:	Gender: (Dropdown)
phone number:	Own property? (tickbox)
	LinkedIn Username:

FAQs:
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

View ToS:

Figure 7: High fidelity prototypes developed in Figma

The design phase was critical in combining the user requirements with the technical implementation. The structured design approach ensured that the user experience was closely aligned with the project goals and user stories initially created. The use of Figma enabled quick prototyping and efficient iteration over the designs, which was complementary to the iterative feedback received in regular sprint meetings with the client, ensuring that the design was within scope before committing to development.

Chapter 5: Implementation

5.1: Methodology discussion

Agile is a popular method of project methodology to implement, being used in over 71% of software development teams worldwide (https://digital.ai/, 2024). Its use of iteration gives it the necessary flexibility to effectively pivot the project based on user feedback and client communication, which is particularly effective in user-driven application development such as DigitalWaves. The development was split into 2 week sprints, with tasks being assigned from the product backlog. Following each sprint, a meeting was had with the client to ensure progress was being made and the original requirements were being met, and the project was not moving out of scope. This was particularly effective following the early delivery of the MVP, meaning subsequently the product could be added to, with additional features being discussed with the client to improve the product further.

The Software Development Life Cycle is a framework that guided the development of DigitalWaves, ensuring a comprehensive approach to each stage. Using agile within this cycle allows for each stage to benefit from iteration and feedback, helping the project to stay responsive to user needs and client suggestions.

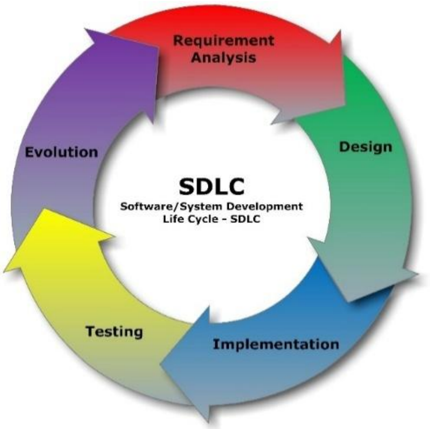


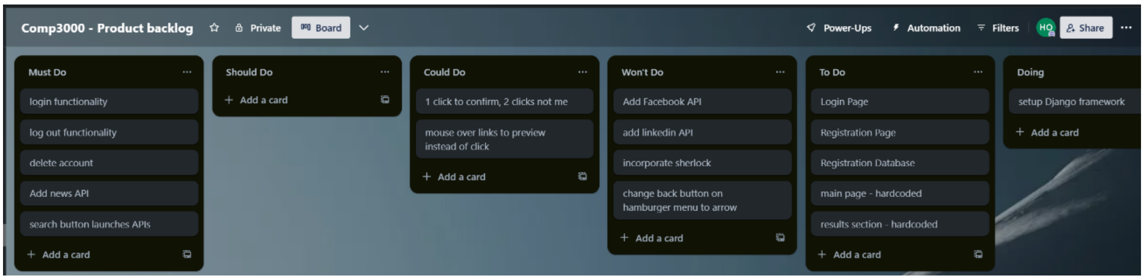
Figure 8: the Software Development Life Cycle

Due to the uncertainty on exact user needs early in the development process, agile was particularly useful in developing a user-focused project such as DigitalWaves. The early development of working prototypes of the application and communication with the client every sprint ensured that the required value was delivered. This constant communication was also effective at refining and reprioritising requirements where needed to allow for user's needs to be catered for, particularly for non-technical users. The sprint reviews with the client also informed the planning of the next sprint, allowing for constant progress to be made and ensuring the project did not suffer from scope creep.

Other methodologies such as waterfall could have been used, however it would not have mapped as well to the project as agile did. Whilst waterfall is effective, it relies on clear end-goals being defined at the start for a linear development cycle, and hence is a much more rigid structure and not suitable to evolving requirements. The use of regular sprint reviews in agile also allows for a lower risk approach as blockers can be fixed quickly or the project can be pivoted.

5.2: Project management

Project management was achieved through the use of a product backlog created using Trello, which was referred to heavily throughout development. The functional and non-functional requirements were abstracted into manageable, achievable tasks. These were then prioritised based on a MoSCoW structure – must, should, could, won't. This helped greatly in prioritising tasks that were essential for the Minimum Viable Product (MVP) delivery, as well as organising tasks that were critical for product delivery, but not essential for proof of concept.



Comp3000 - Product backlog

Must Do
- login functionality
- log out functionality
- delete account
- Add news API
- search button launches APIs

Should Do

Could Do
- 1 click to confirm, 2 clicks not me
- mouse over links to preview instead of click

Won't Do
- Add Facebook API
- add linkedin API
- incorporate sherlock
- change back button on hamburger menu to arrow

To Do
- Login Page
- Registration Page
- Registration Database
- main page - hardcoded
- results section - hardcoded

Doing
- setup Django framework

Figure 9: initial product backlog screenshot

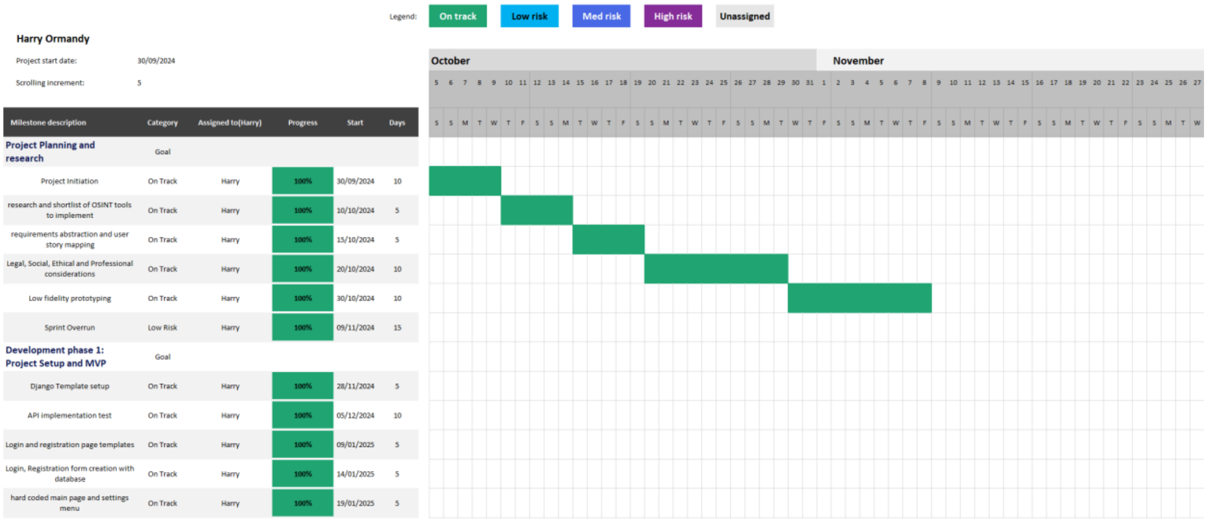
The completed Trello product backlog can be accessed via: https://trello.com/b/S5k7FgSM/comp3000-product-backlog

Each abstracted task was small enough to be assigned to a sprint, meaning constant progress could be made and value could be delivered to the client. The use of the MoSCoW structure was beneficial in tracking the priorities of each task, which allowed for the effective delivery of an MVP early into development. Furthermore, the priorities allowed for management of scope creep by tracking future ideas, but at a low priority. A less organic, structured approach to the project management ensured that low priority features were not focused on mid-sprint, further allowing for constant value to be delivered.

5.3: Time management

General time management for the project used the agile methodology with scrum. The project was organised into sprints lasting 2 weeks, which would conclude with a sprint review with the client. The regular client reviews enabled reflection on sprint and project progress, allowing for pivoting of requirements or reprioritisation of tasks if necessary. This process ensured that the following sprint was informed by the last, and tasks could be quickly adjusted if blockers were met. A Gantt chart was used to organise sprints, split into a planning phase and multiple development phases, prioritising the MVP's early delivery. The planning phase was used to research prior tasks, consider feasibility of the product and inform product requirements. Prototyping was also completed in this phase, which allowed for adjustment of individual tasks from the product backlog, before they were assigned to sprints.

To further ensure effective progress, tasks were aligned to the SMART goals framework – Specific, Measurable, Achievable, Relevant and Time bound. This ensured that each task had a realistic and clear deliverable within each sprint.



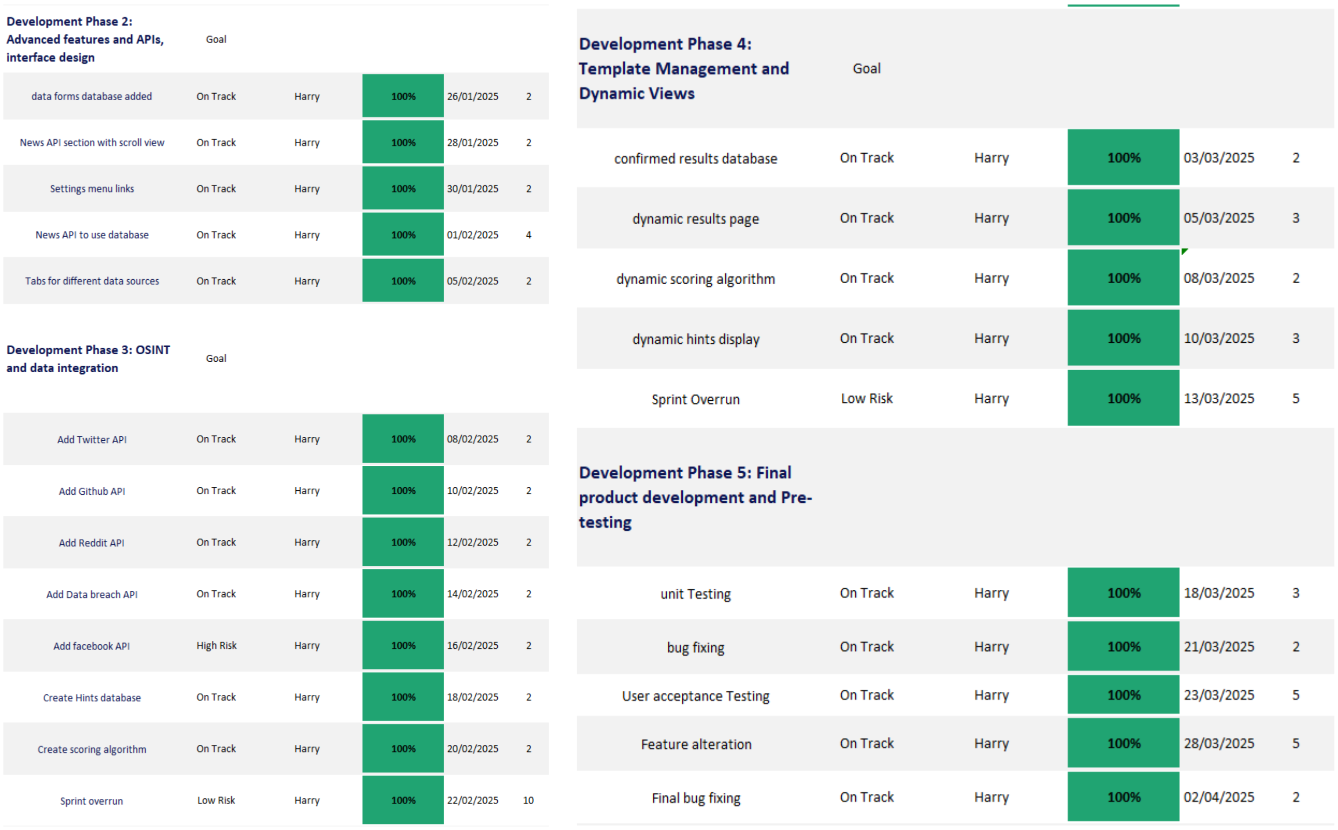
Harry Ormandy

Project start date: 30/09/2024

Scrolling increment: 5

Milestone description	Category	Assigned to(Harry)	Progress	Start	Days
Project Planning and research	Goal				
Project Initiation	On Track	Harry	100%	30/09/2024	10
research and shortlist of OSINT tools to implement	On Track	Harry	100%	10/10/2024	5
requirements abstraction and user story mapping	On Track	Harry	100%	15/10/2024	5
Legal, Social, Ethical and Professional considerations	On Track	Harry	100%	20/10/2024	10
Low fidelity prototyping	On Track	Harry	100%	30/10/2024	10
Sprint Overrun	Low Risk	Harry	100%	09/11/2024	15
Development phase 1: Project Setup and MVP	Goal				
Django Template setup	On Track	Harry	100%	28/11/2024	5
API implementation test	On Track	Harry	100%	05/12/2024	10
Login and registration page templates	On Track	Harry	100%	09/01/2025	5
Login, Registration form creation with database	On Track	Harry	100%	14/01/2025	5
hard coded main page and settings menu	On Track	Harry	100%	19/01/2025	5

Figure 10: Planning and MVP Gantt chart



Development Phase 2: Advanced features and APIs, interface design	Goal				
data forms database added	On Track	Harry	100%	26/01/2025	2
News API section with scroll view	On Track	Harry	100%	28/01/2025	2
Settings menu links	On Track	Harry	100%	30/01/2025	2
News API to use database	On Track	Harry	100%	01/02/2025	4
Tabs for different data sources	On Track	Harry	100%	05/02/2025	2

Development Phase 3: OSINT and data integration	Goal				
Add Twitter API	On Track	Harry	100%	08/02/2025	2
Add Github API	On Track	Harry	100%	10/02/2025	2
Add Reddit API	On Track	Harry	100%	12/02/2025	2
Add Data breach API	On Track	Harry	100%	14/02/2025	2
Add facebook API	High Risk	Harry	100%	16/02/2025	2
Create Hints database	On Track	Harry	100%	18/02/2025	2
Create scoring algorithm	On Track	Harry	100%	20/02/2025	2
Sprint overrun	Low Risk	Harry	100%	22/02/2025	10

Development Phase 4: Template Management and Dynamic Views	Goal				
confirmed results database	On Track	Harry	100%	03/03/2025	2
dynamic results page	On Track	Harry	100%	05/03/2025	3
dynamic scoring algorithm	On Track	Harry	100%	08/03/2025	2
dynamic hints display	On Track	Harry	100%	10/03/2025	3
Sprint Overrun	Low Risk	Harry	100%	13/03/2025	5

Development Phase 5: Final product development and Pre-testing					
unit Testing	On Track	Harry	100%	18/03/2025	3
bug fixing	On Track	Harry	100%	21/03/2025	2
User acceptance Testing	On Track	Harry	100%	23/03/2025	5
Feature alteration	On Track	Harry	100%	28/03/2025	5
Final bug fixing	On Track	Harry	100%	02/04/2025	2

Figure 11: Development phase Gantt charts

The flexible approach to time management was particularly beneficial when blockers were encountered and adaptation was needed. The regular sprint meetings ensured that expectations were managed effectively, and allowed time to address challenges. Some features that were originally planned had to be reconsidered after further research revealed they weren't feasible. The flexibility of the sprints allowed for the tasks to be moved to later sprints, following a discussion within the sprint meetings, and an alternative and successful approach found. This adaptability of time management was critical in the success of the project.

5.4: Cost considerations

Student Licenses and free tiers allowed for no cost to be incurred for the development of the application. The development tools used were as follows:

- Code Development – Visual Studio 2022
- Version Control – GitHub
- Prototyping and HCI principle experimentation – Figma
- Task management – Trello
- Time management – excel

During development, costs were avoided where possible due to the aggregation of expenses, specifically when dealing with third party data sources. This imposed some barriers, such as rate limitations on multiple APIs that were communicated with. These could have been avoided by paying for premium access to the APIs, but this would have been a considerable expense and out of scope. Costs were however, incurred when interacting with some data sources. The 'Have I Been Pwned' API, which returns information relating to any data breaches an email has been detected in, cost a non-commercial fee of $3.99 per calendar month.

Hosting the service was also free, as the application is run locally. If the application were to scale commercially, server hosting, commercial licensing and data/legal compliance would have to be considered. Each API and data source used in production could also be purchased at premium rates to scale the rate limits to commercial standards.

5.5: Implementation Overview

The requirements analysis and design phase provided the clear project goal. This was used to focus early development on MVP features, and iterate through it to add further project aims identified in the design and user stories.

The core functionality identified to be implemented was:

- User account creation and login system
- Personal data input and storage
- Automated third-party API queries
- User-driven confirmation of relevant entries
- Presentation of results with a digital footprint score calculated
- Additional custom hints based on entry data type for how to remove the entry or further secure the entry

Agile methodology was heavily implemented during this phase of development. A sprint-based structure was used, with 2 weeks being allocated to each sprint. Initial development sprints focused heavily on the early delivery of the MVP, which included the accounts system, basic back-end databases and hardcoded results section and a news-based API query. Subsequent sprints iterated over the product, focusing on additional data sources, polishing the UI and the addition of dynamically presenting results to the user.

The tools and technology used for the product were:

- Backend development – Django (Python)
- Frontend development – HTML/CSS/JavaScript
- APIs: third party data sources – X (formerly twitter), HaveIBeenPwned, GitHub, Reddit, Google CSS
- Data storage – SQLite databased using Django ORM with hashed passwords and custom tables added

Python’s Django was used to develop the project due to the frameworks flexibility and scalability it provides. The framework focuses on providing the basis for a non-monolithic structure, separating frontend, backend and databases from each other to ensure modularity. Whilst this was unfamiliar technology and hence required further time and research at the beginning of development, the framework provided features that eased development overall and aided in delivery of a strong final product that demonstrates meaningful user interaction and educational value.

5.6: Frontend implementation

For initial MVP creation, the main page structure and navigation was essential. Base templates were created using HTML and CSS to recreate the prototype designed in Figma for the web application.

The frontend was heavily influenced by the high-fidelity prototypes, as well as Jakob Nielsen’s usability heuristics, specifically the ‘Consistency and Standards’ concept, ensuring a uniform design with predictable UI behaviour (Nielsen. J, 1994).

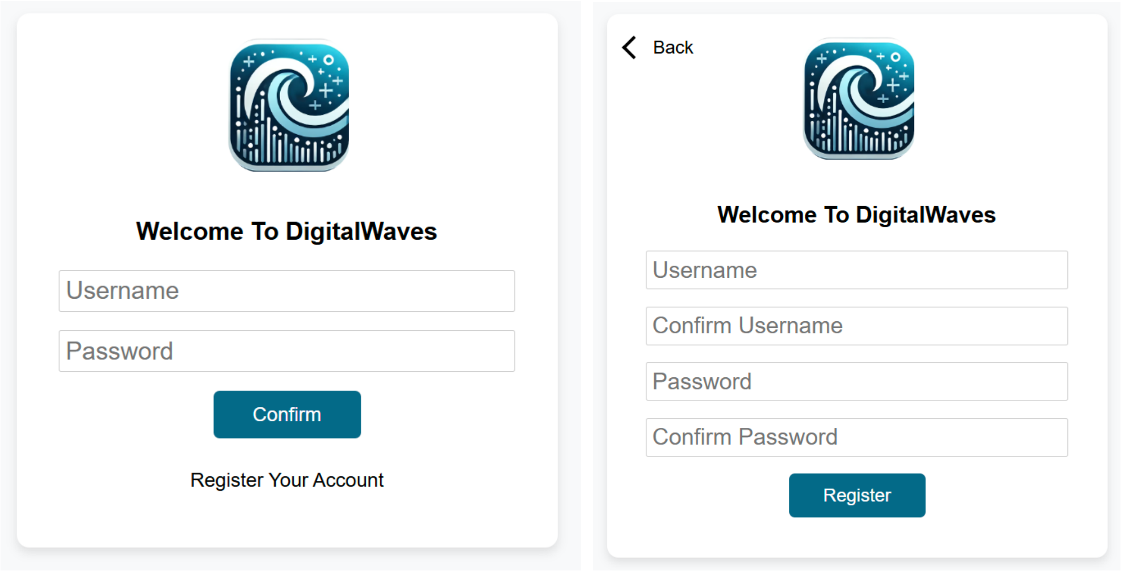


Figure 12: Login and registration templates

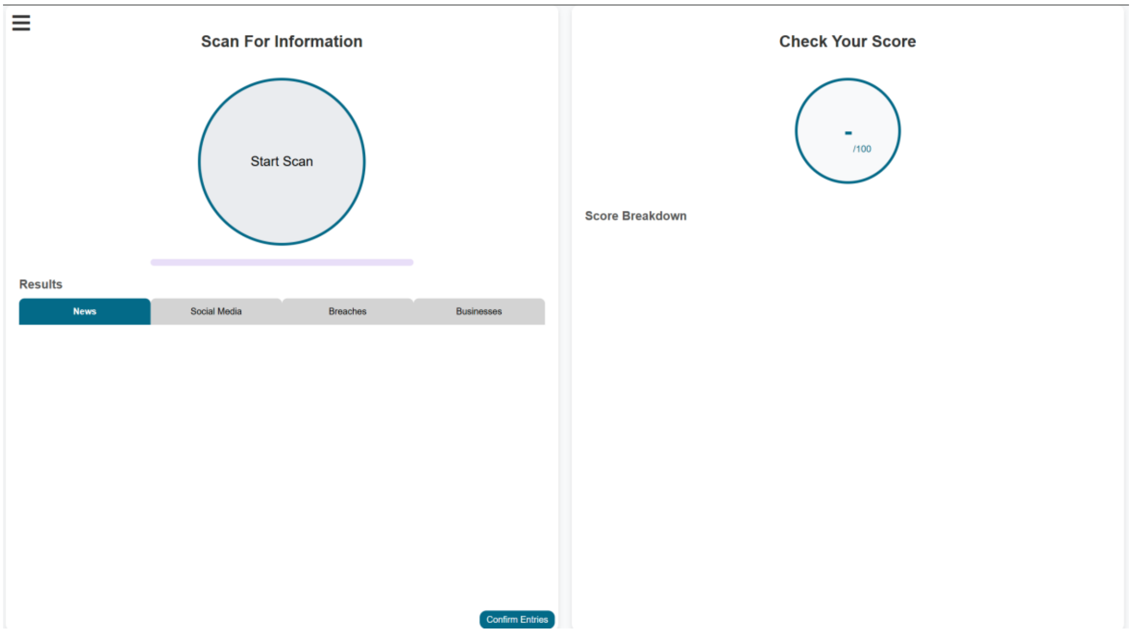
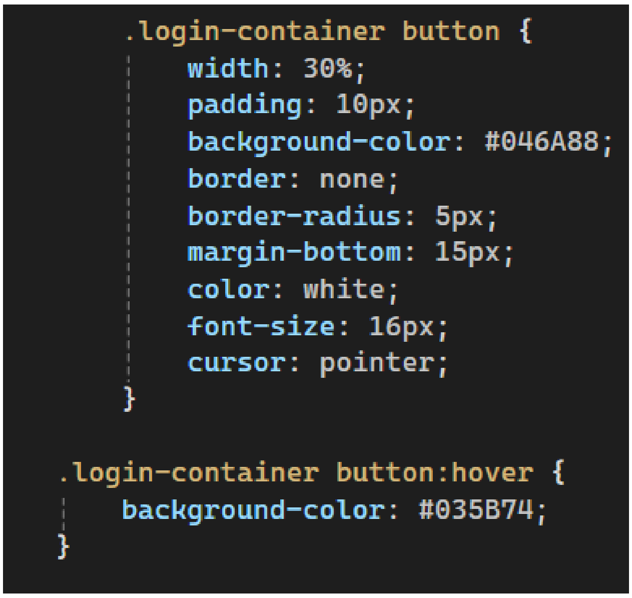


Figure 13: home scanning page template

These templates served as a basis for displaying information to the user throughout the project. As most of the data display was dynamically created, the templates were mostly empty. To closely resemble the design and align with HCI principles, pseudo-classes were used in the HTML design to enable reactive elements that provide the user with visual feedback, including buttons that dynamically darken when hovered over. This added a more professional feel to the application, improving the user experience.

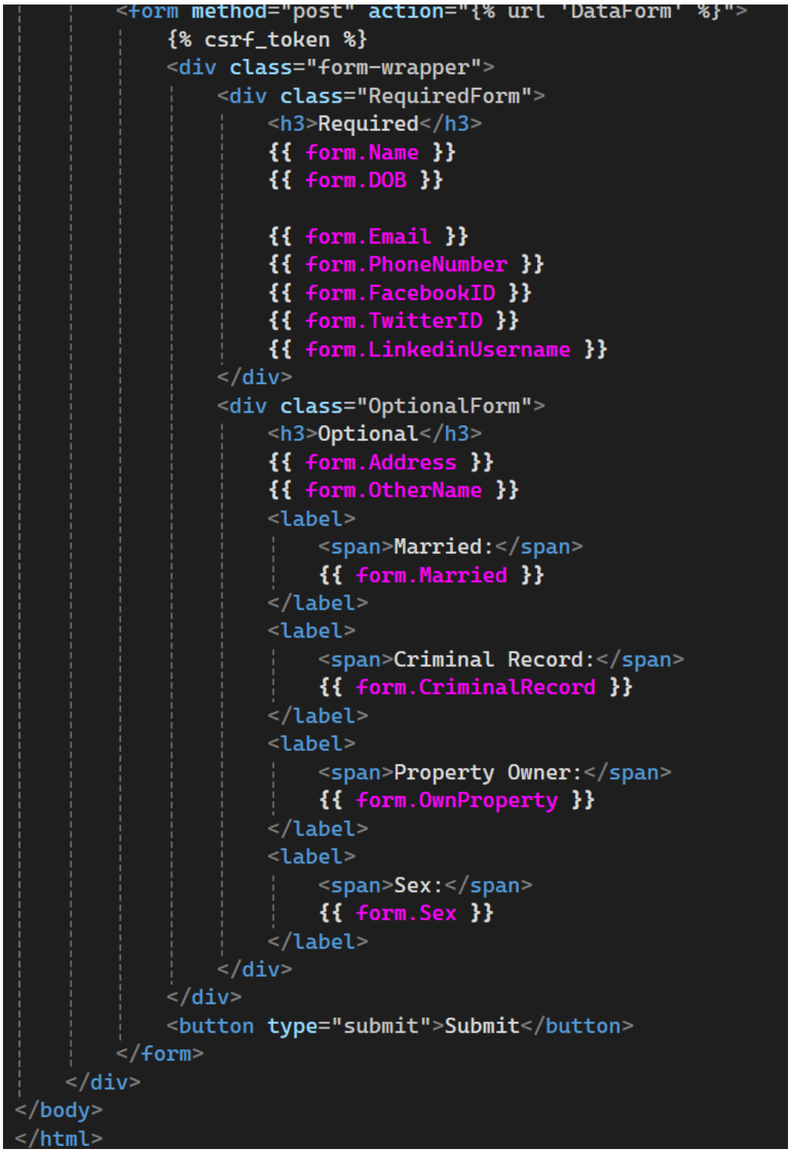


```
    .login-container button {
        width: 30%;
        padding: 10px;
        background-color: #046A88;
        border: none;
        border-radius: 5px;
        margin-bottom: 15px;
        color: white;
        font-size: 16px;
        cursor: pointer;
    }

  .login-container button:hover {
      background-color: #035B74;
  }
```

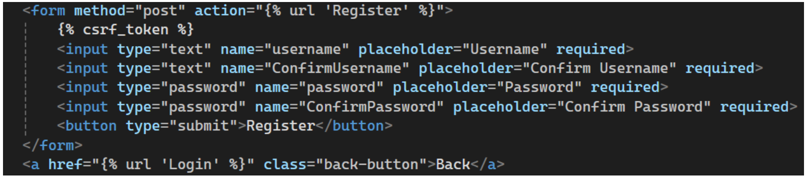
Figure 14: using pseudo-classes for user feedback

CSS was used within the templates to recreate the design conceived in the prototypes. To ensure a positive UX, consistent colour theory and logo placement was used throughout the main pages, as well as clear placement of buttons and other interactive elements.

Forms and data input was critical implementation to retrieve data from the user. HTML forms were used to collect data from the user for registration for an account, and the user information form.



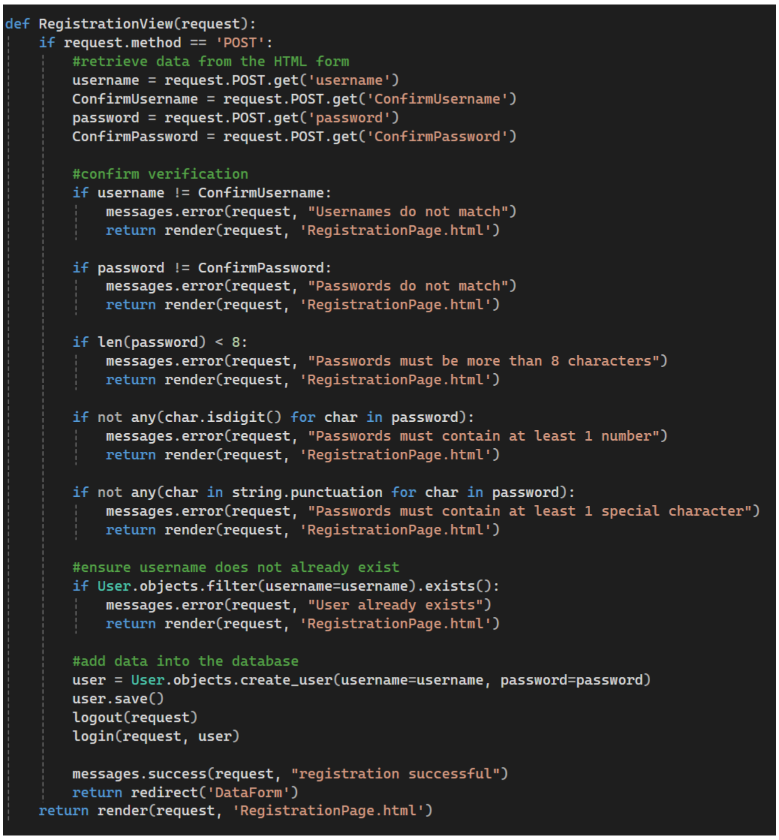
```
        <form method="post" action="{% url 'DataForm' %}">
            {% csrf_token %}
            <div class="form-wrapper">
                <div class="RequiredForm">
                    <h3>Required</h3>
                    {{ form.Name }}
                    {{ form.DOB }}

                    {{ form.Email }}
                    {{ form.PhoneNumber }}
                    {{ form.FacebookID }}
                    {{ form.TwitterID }}
                    {{ form.LinkedinUsername }}
                </div>
                <div class="OptionalForm">
                    <h3>Optional</h3>
                    {{ form.Address }}
                    {{ form.OtherName }}
                    <label>
                        <span>Married:</span>
                        {{ form.Married }}
                    </label>
                    <label>
                        <span>Criminal Record:</span>
                        {{ form.CriminalRecord }}
                    </label>
                    <label>
                        <span>Property Owner:</span>
                        {{ form.OwnProperty }}
                    </label>
                    <label>
                        <span>Sex:</span>
                        {{ form.Sex }}
                    </label>
                </div>
            </div>
            <button type="submit">Submit</button>
        </form>
    </div>
</body>
</html>
```



```
<form method="post" action="{% url 'Register' %}">
    {% csrf_token %}
    <input type="text" name="username" placeholder="Username" required>
    <input type="text" name="ConfirmUsername" placeholder="Confirm Username" required>
    <input type="password" name="password" placeholder="Password" required>
    <input type="password" name="ConfirmPassword" placeholder="Confirm Password" required>
    <button type="submit">Register</button>
</form>
<a href="{% url 'Login' %}" class="back-button">Back</a>
```

Figure 15: HTML user information and account creation forms for user inputs

During the creation of the registration form, additional validation and verification was put in place to ensure the user created a secure account. The user is required to input their password twice, and ensure their password met minimum standards. Should they fail to do this, they were presented with an error message describing what they have done incorrectly.



```python
def RegistrationView(request):
    if request.method == 'POST':
        #retrieve data from the HTML form
        username = request.POST.get('username')
        ConfirmUsername = request.POST.get('ConfirmUsername')
        password = request.POST.get('password')
        ConfirmPassword = request.POST.get('ConfirmPassword')

        #confirm verification
        if username != ConfirmUsername:
            messages.error(request, "Usernames do not match")
            return render(request, 'RegistrationPage.html')

        if password != ConfirmPassword:
            messages.error(request, "Passwords do not match")
            return render(request, 'RegistrationPage.html')

        if len(password) < 8:
            messages.error(request, "Passwords must be more than 8 characters")
            return render(request, 'RegistrationPage.html')

        if not any(char.isdigit() for char in password):
            messages.error(request, "Passwords must contain at least 1 number")
            return render(request, 'RegistrationPage.html')

        if not any(char in string.punctuation for char in password):
            messages.error(request, "Passwords must contain at least 1 special character")
            return render(request, 'RegistrationPage.html')

        #ensure username does not already exist
        if User.objects.filter(username=username).exists():
            messages.error(request, "User already exists")
            return render(request, 'RegistrationPage.html')

        #add data into the database
        user = User.objects.create_user(username=username, password=password)
        user.save()
        logout(request)
        login(request, user)

        messages.success(request, "registration successful")
        return redirect('DataForm')
    return render(request, 'RegistrationPage.html')
```

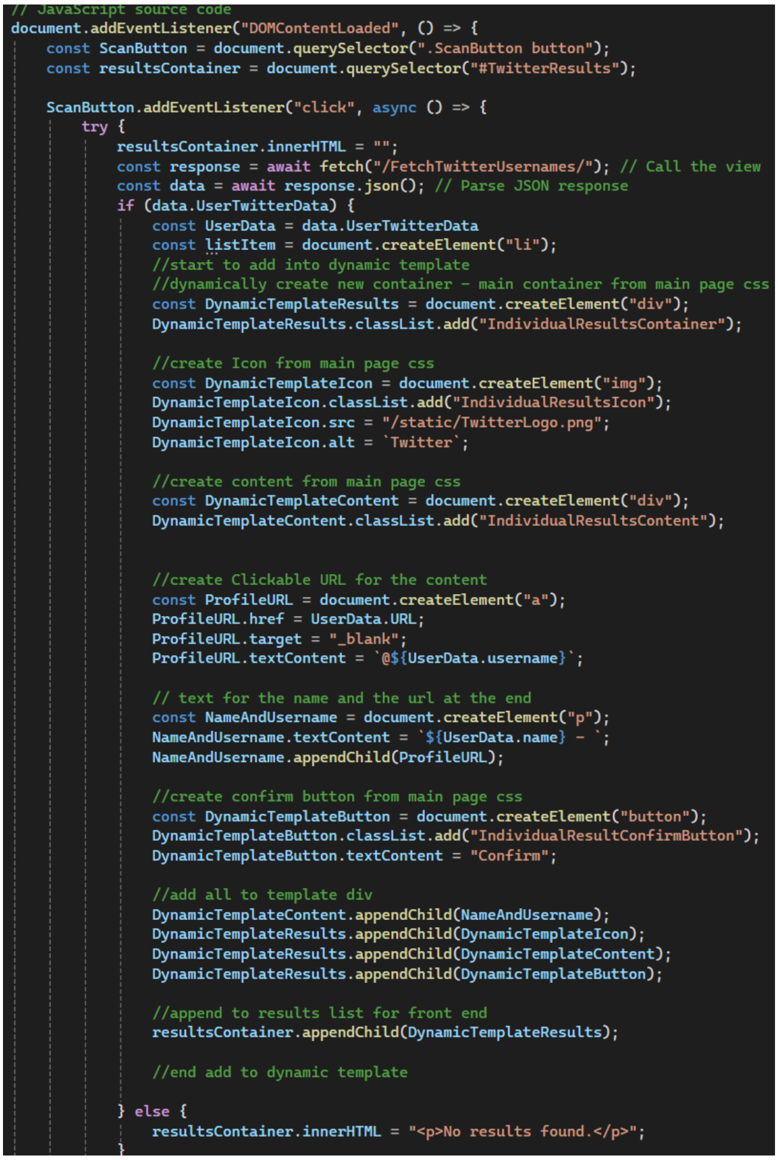
Figure 16: registration account validation and verification

The Django framework also included Cross-Site Request Forgery protection tokens that could be implemented when retrieving information from a user via any input boxes. This adds additional security and prevents malicious users from authenticating themselves on the platform and gaining unauthorised access to other elements of the application. These were disabled during development, however this would be enabled during deployment to reinforce security.

Once the user has registered an account and filled out the information form, they are redirected to the home page. From here, they can navigate to the settings page, or scan and explore scans results and their score. The user initiates a scan by clicking the scan button, which triggers back-end processes to retrieve information based on their information provided, and then display this to the user in a user friendly fashion.

JavaScript was used heavily in development to load dynamic content, add event listeners to buttons and ensure the front end Dynamic Object Model (DOM) was updated. Results were retrieved from the back-end in JSON form, which was then broken down into the information that needed to be presented to the user. This information then populated a template that was created in CSS to present the information to the user in a HCI-friendly fashion.

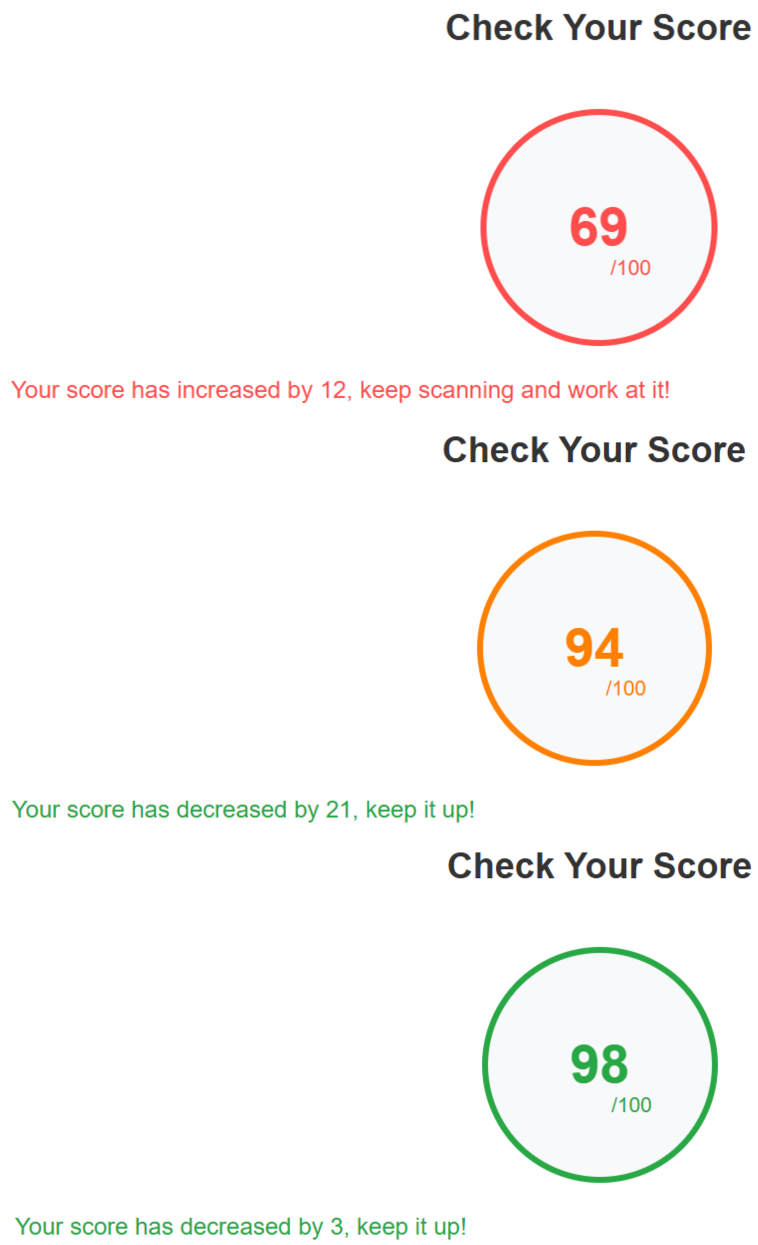
Within each template, there was also an interactive button that allowed the user to confirm an entry, or find hints on how to remove the entry, dependant on the section of the application the result was related to. Each entry was then dynamically created in HTML, loaded into the DOM and displayed in a scroll view within a tab, dependant on the type of information loaded in. This dynamic population of results was used in both the results and the score section of the application, encompassing a large amount of the front-end development.



```
// JavaScript source code
document.addEventListener("DOMContentLoaded", () => {
    const ScanButton = document.querySelector(".ScanButton button");
    const resultsContainer = document.querySelector("#TwitterResults");

    ScanButton.addEventListener("click", async () => {
        try {
            resultsContainer.innerHTML = "";
            const response = await fetch("/FetchTwitterUsernames/"); // Call the view
            const data = await response.json(); // Parse JSON response
            if (data.UserTwitterData) {
                const UserData = data.UserTwitterData
                const listItem = document.createElement("li");
                //start to add into dynamic template
                //dynamically create new container - main container from main page css
                const DynamicTemplateResults = document.createElement("div");
                DynamicTemplateResults.classList.add("IndividualResultsContainer");

                //create Icon from main page css
                const DynamicTemplateIcon = document.createElement("img");
                DynamicTemplateIcon.classList.add("IndividualResultsIcon");
                DynamicTemplateIcon.src = "/static/TwitterLogo.png";
                DynamicTemplateIcon.alt = `Twitter`;

                //create content from main page css
                const DynamicTemplateContent = document.createElement("div");
                DynamicTemplateContent.classList.add("IndividualResultsContent");


                //create Clickable URL for the content
                const ProfileURL = document.createElement("a");
                ProfileURL.href = UserData.URL;
                ProfileURL.target = "_blank";
                ProfileURL.textContent = `@${UserData.username}`;

                // text for the name and the url at the end
                const NameAndUsername = document.createElement("p");
                NameAndUsername.textContent = `${UserData.name} - `;
                NameAndUsername.appendChild(ProfileURL);

                //create confirm button from main page css
                const DynamicTemplateButton = document.createElement("button");
                DynamicTemplateButton.classList.add("IndividualResultConfirmButton");
                DynamicTemplateButton.textContent = "Confirm";

                //add all to template div
                DynamicTemplateContent.appendChild(NameAndUsername);
                DynamicTemplateResults.appendChild(DynamicTemplateIcon);
                DynamicTemplateResults.appendChild(DynamicTemplateContent);
                DynamicTemplateResults.appendChild(DynamicTemplateButton);

                //append to results list for front end
                resultsContainer.appendChild(DynamicTemplateResults);

                //end add to dynamic template

            } else {
                resultsContainer.innerHTML = "<p>No results found.</p>";
            }
```

Figure 17: JavaScript code to call to an API with user data, and dynamically present data in a template

Once the user has identified positive results, they can confirm these. These are then displayed with a dynamic score that is calculated using JavaScript, dependent on weightings given to each different type of information. The score is displayed in a traffic light-style colour system, aiding a user's understanding of risk, incorporating the 'match between system and real world' usability heuristic as described by Jakob Nielsen (Nielsen. J, 1994). This score is also compared to a user's stored score from a previous scan if they are a returning user, to ensure that they can monitor how they are improving their digital footprint, encouraging use of the application multiple times.



Check Your Score

69 /100

Your score has increased by 12, keep scanning and work at it!

Check Your Score

94 /100

Your score has decreased by 21, keep it up!

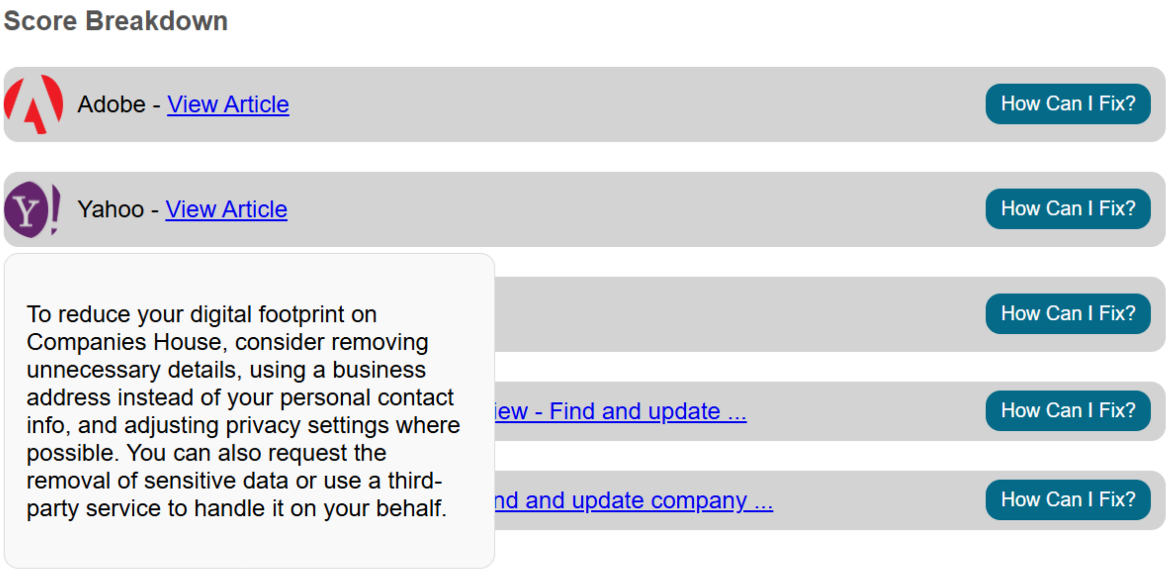
Check Your Score

98 /100

Your score has decreased by 3, keep it up!

Figure 18: score system with traffic light-style colour system

The user is also presented with a score breakdown, showing only the confirmed positive entries. Each result is accompanied in a template with a button which, when interacted with, displays a tailored hint stating how the user can work to improve their digital footprint or be more secure online. These hints are loaded in dynamically, with the information being retrieved from a static database.



To reduce your digital footprint on Companies House, consider removing unnecessary details, using a business address instead of your personal contact info, and adjusting privacy settings where possible. You can also request the removal of sensitive data or use a third-party service to handle it on your behalf.

Figure 19: score breakdown section with custom hints presented to the user

A settings menu was also implemented to give the user easy access to customisable options. The menu consists of a help section, including FAQs and advice for a user, an account settings menu, for account management and deletion following GDPR, and an option to revisit the personal information data form, allowing the users to delete and edit information about themselves. Whilst the initial settings menu and the overlay preventing the interaction with the home page are loaded statically with HTML and CSS, the individual settings pages that can be selected within the menu are dynamically injected into the DOM using JavaScript.

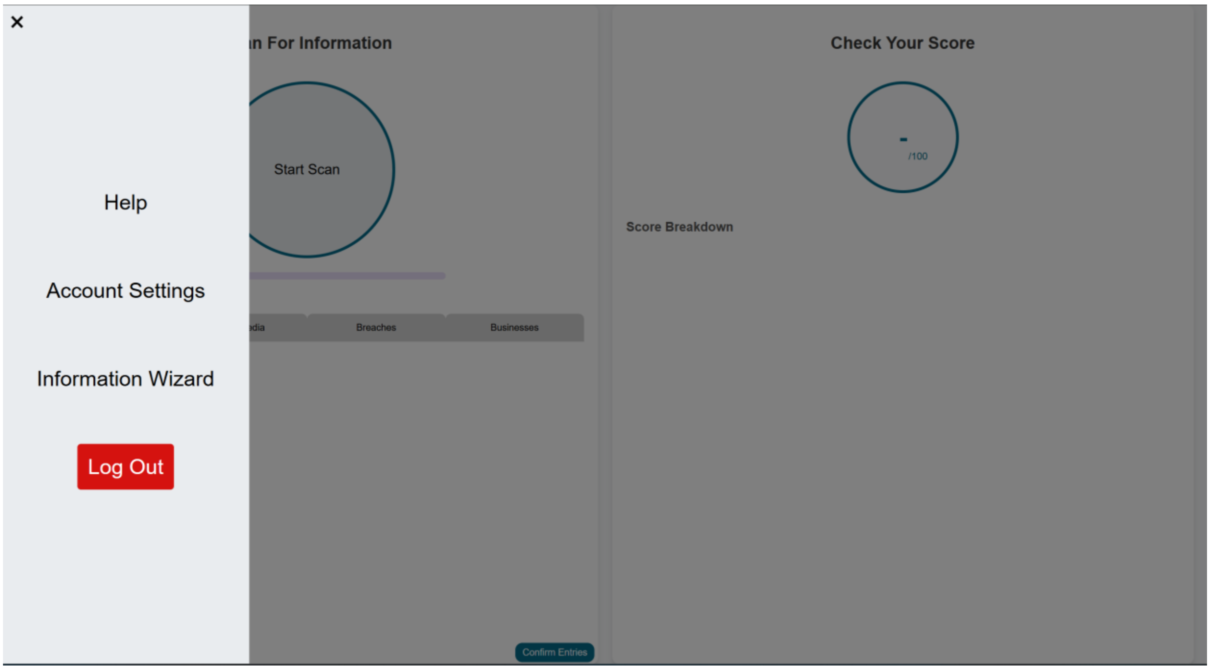
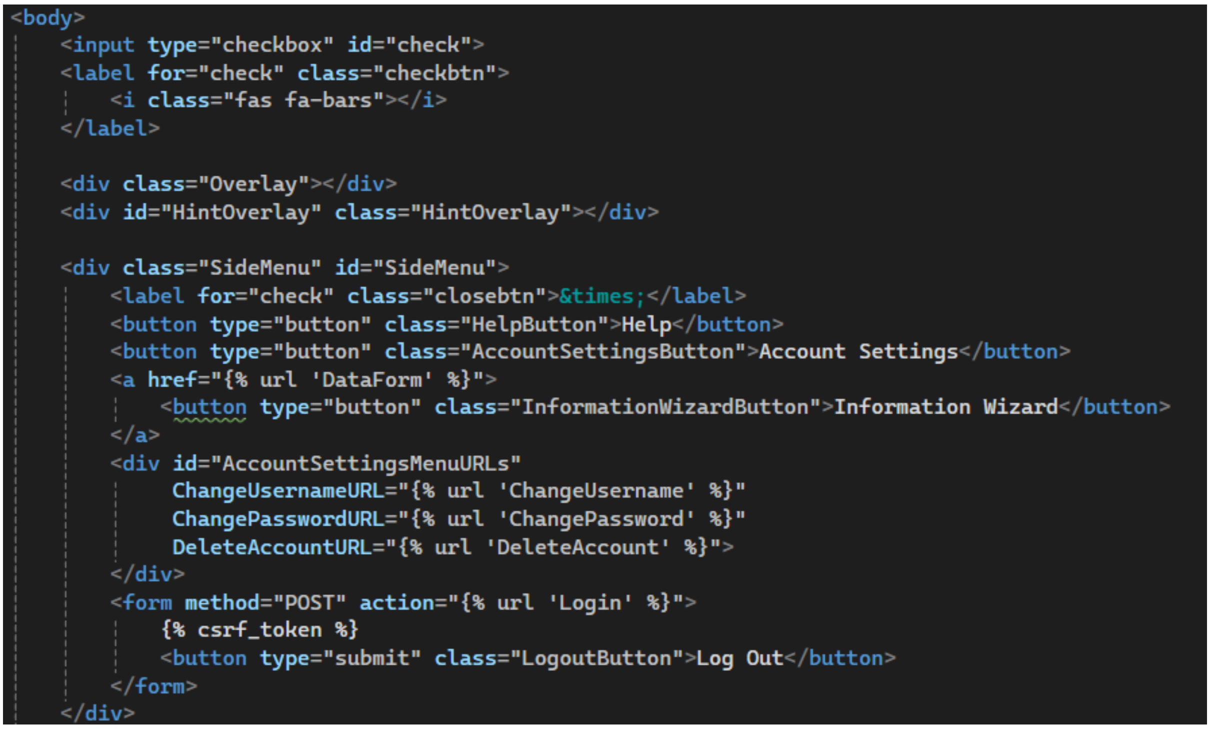


Figure 20: Settings menu accessible from main page



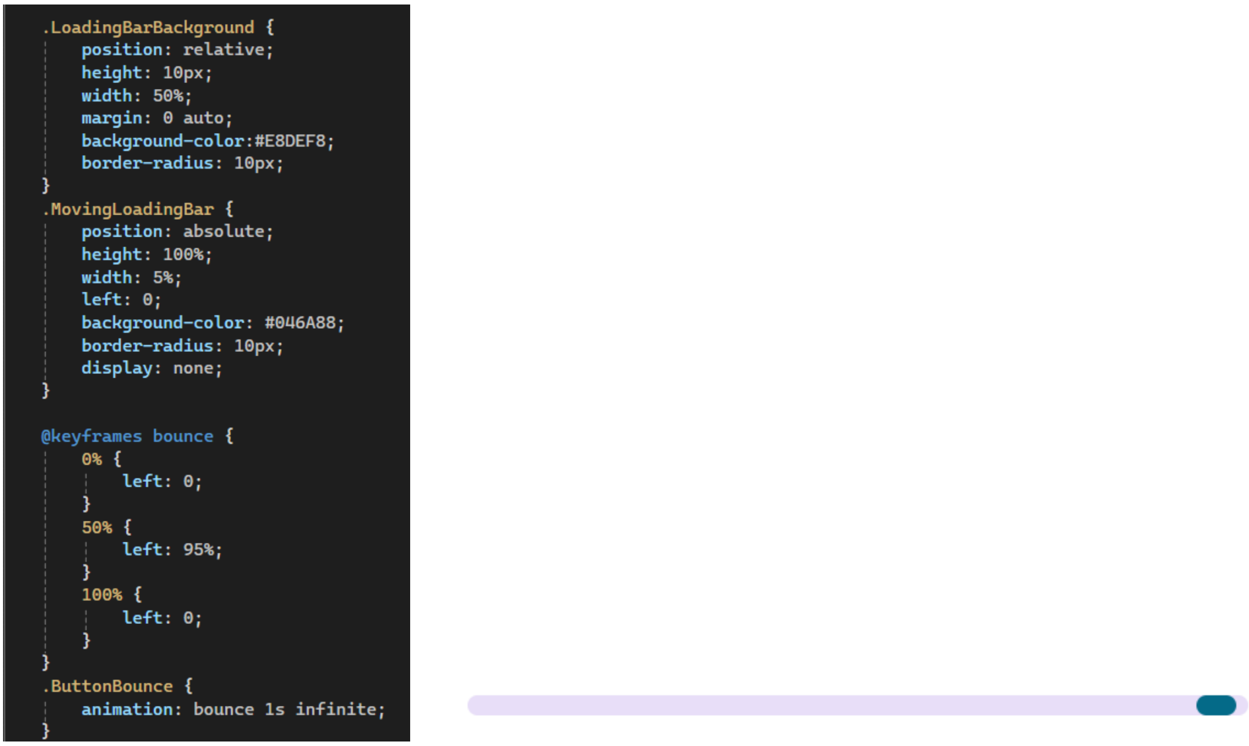
```
<body>
    <input type="checkbox" id="check">
    <label for="check" class="checkbtn">
        <i class="fas fa-bars"></i>
    </label>

    <div class="Overlay"></div>
    <div id="HintOverlay" class="HintOverlay"></div>

    <div class="SideMenu" id="SideMenu">
        <label for="check" class="closebtn">&times;</label>
        <button type="button" class="HelpButton">Help</button>
        <button type="button" class="AccountSettingsButton">Account Settings</button>
        <a href="{% url 'DataForm' %}">
            <button type="button" class="InformationWizardButton">Information Wizard</button>
        </a>
        <div id="AccountSettingsMenuURLs"
             ChangeUsernameURL="{% url 'ChangeUsername' %}"
             ChangePasswordURL="{% url 'ChangePassword' %}"
             DeleteAccountURL="{% url 'DeleteAccount' %}">
        </div>
        <form method="POST" action="{% url 'Login' %}">
            {% csrf_token %}
            <button type="submit" class="LogoutButton">Log Out</button>
        </form>
    </div>
```

Figure 21: Initial settings menu and main page overlay code

To avoid frustration points for the user, feedback needed to be displayed. Django's built-in error message functionality ensured that clear error messaging was displayed for the user. This was essential for functionality such as the registration page, where a password policy needed to be followed, and if there was an error, the user was informed of the action they needed to take.

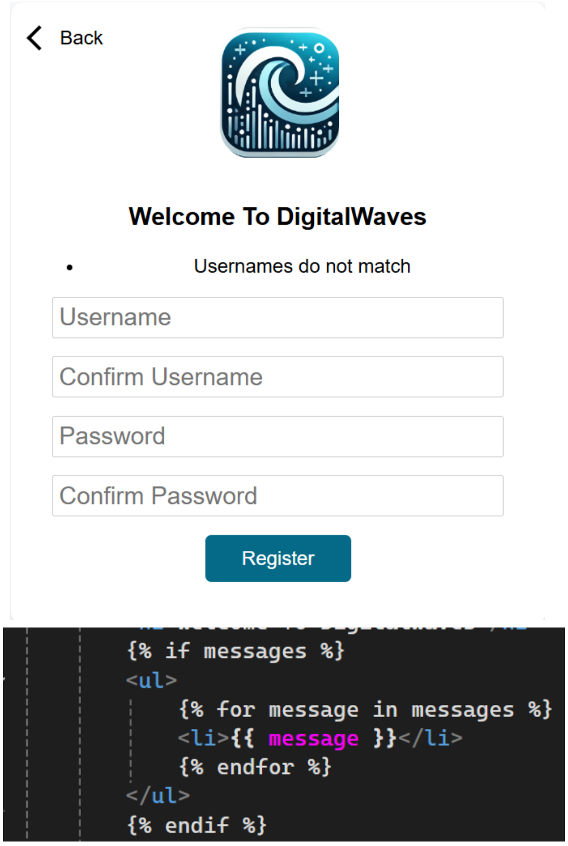
User feedback was also implemented when the user initiates a scan, with a loading bar being added that was dynamically triggered to move when background processing took place, to ensure the user was not frustrated, showing close alignment with Nielsen's usability heuristics (Nielsen. J, 1994).

The application also responds to user interaction whilst using the scan results template. Confirming an entry as positive dynamically alters the button colour to green, easily signifying to the user that they have selected that entry.



```css
.LoadingBarBackground {
    position: relative;
    height: 10px;
    width: 50%;
    margin: 0 auto;
    background-color:#E8DEF8;
    border-radius: 10px;
}
.MovingLoadingBar {
    position: absolute;
    height: 100%;
    width: 5%;
    left: 0;
    background-color: #046A88;
    border-radius: 10px;
    display: none;
}

@keyframes bounce {
    0% {
        left: 0;
    }
    50% {
        left: 95%;
    }
    100% {
        left: 0;
    }
}
.ButtonBounce {
    animation: bounce 1s infinite;
}
```

Figure 22: Scan loading bar and code for user feedback



- Usernames do not match

```
{% if messages %}
<ul>
    {% for message in messages %}
    <li>{{ message }}</li>
    {% endfor %}
</ul>
{% endif %}
```

Figure 23: Error message feedback on registration page

5.7: Backend implementation

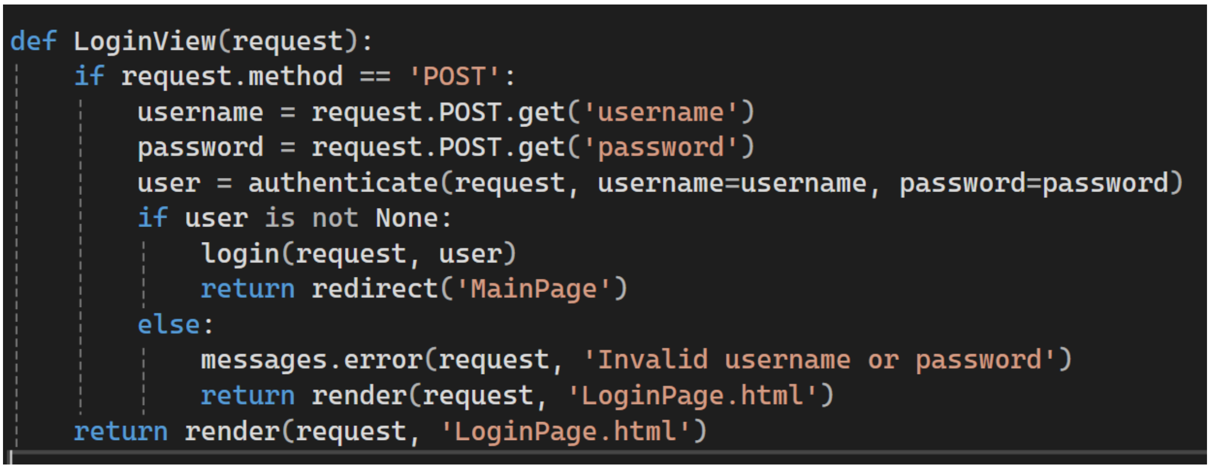
Django's backend framework was used to enforce a modular design on the backend logic. Distinct sections were required for each area of backend functionality due to the Model-Template-View (MTV) style framework, defined as:

- Models – Definition of database structures
- Views – handling of logic and processes user requests such as loading HTML pages
- Templates – renders HTML sent client-side
- URLs – defined URL paths to each view

The logical flow of the backend ensured a modular and scalable approach to development, underpinning the non-monolithic approach.

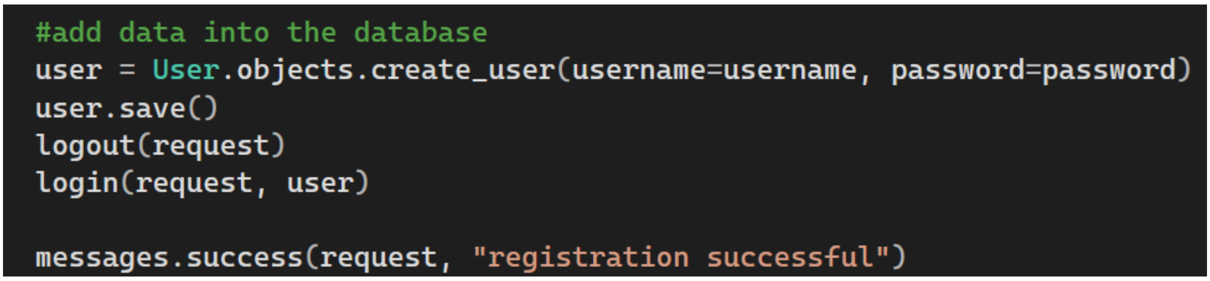
User authentication and security was paramount for implementation of the login and accounts system. Django's built-in authentication framework was used to achieve this, with pre-made databases for accounts and login, implementing PBKDF2 password hashing (Django, 2024) to enforce security by never storing the password as plaintext.

The framework also allows for validation, using built-in account authentication to log a user in and out. The authentication was also useful in ensuring an unauthenticated user could not access pages unauthorised, by locking each view to a logged in access only status.



```
def LoginView(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('MainPage')
        else:
            messages.error(request, 'Invalid username or password')
            return render(request, 'LoginPage.html')
    return render(request, 'LoginPage.html')
```

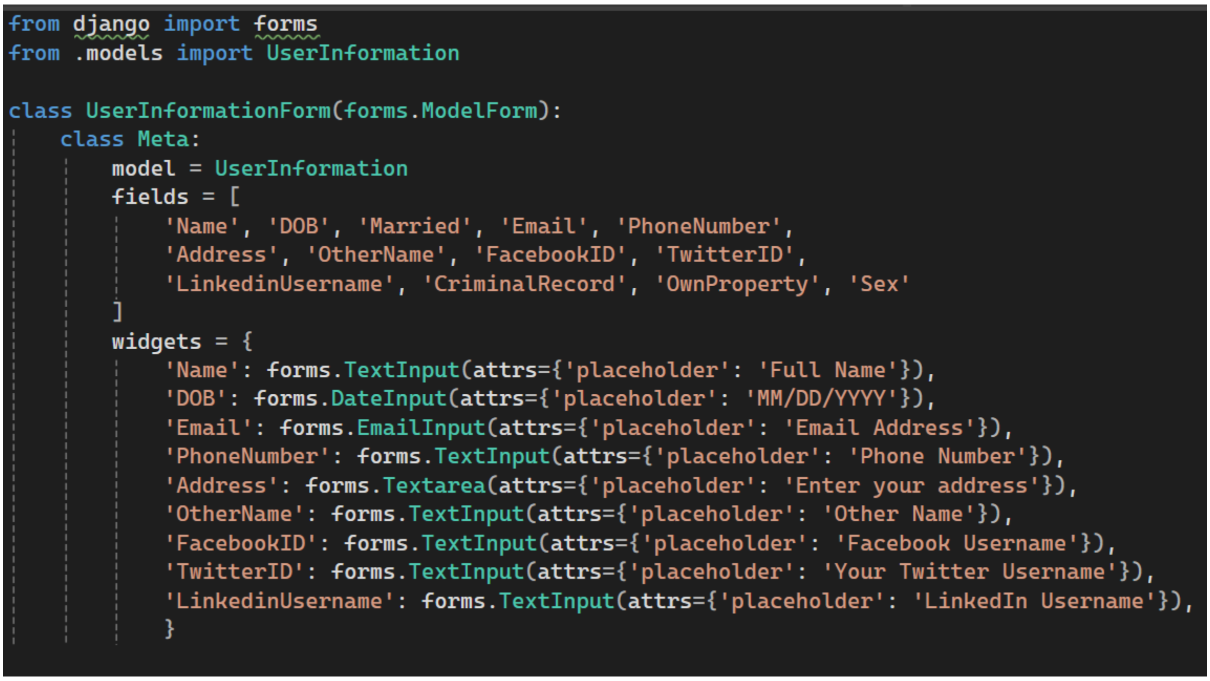
Figure 24: User login logic



```
#add data into the database
user = User.objects.create_user(username=username, password=password)
user.save()
logout(request)
login(request, user)

messages.success(request, "registration successful")
```

Figure 25: Logic to save a user account to the database

To retrieve information from a user input in the frontend, Django processes forms. The clear structure of the form allows for most data validation to be pre-set prior to it being entered to the database, keeping data sanitation concise. By using the built-in ‘request’ feature, data can be retrieved from the front-end HTML form, processed, sanitised, and sent to the database using a view for the logic. This keeps the data processing modular, ensuring that alterations can be made easily to each section, rather than a single structure being used, also enabling scalability.



```
from django import forms
from .models import UserInformation

class UserInformationForm(forms.ModelForm):
    class Meta:
        model = UserInformation
        fields = [
            'Name', 'DOB', 'Married', 'Email', 'PhoneNumber',
            'Address', 'OtherName', 'FacebookID', 'TwitterID',
            'LinkedinUsername', 'CriminalRecord', 'OwnProperty', 'Sex'
        ]
        widgets = {
            'Name': forms.TextInput(attrs={'placeholder': 'Full Name'}),
            'DOB': forms.DateInput(attrs={'placeholder': 'MM/DD/YYYY'}),
            'Email': forms.EmailInput(attrs={'placeholder': 'Email Address'}),
            'PhoneNumber': forms.TextInput(attrs={'placeholder': 'Phone Number'}),
            'Address': forms.Textarea(attrs={'placeholder': 'Enter your address'}),
            'OtherName': forms.TextInput(attrs={'placeholder': 'Other Name'}),
            'FacebookID': forms.TextInput(attrs={'placeholder': 'Facebook Username'}),
            'TwitterID': forms.TextInput(attrs={'placeholder': 'Your Twitter Username'}),
            'LinkedinUsername': forms.TextInput(attrs={'placeholder': 'LinkedIn Username'}),
            }
```

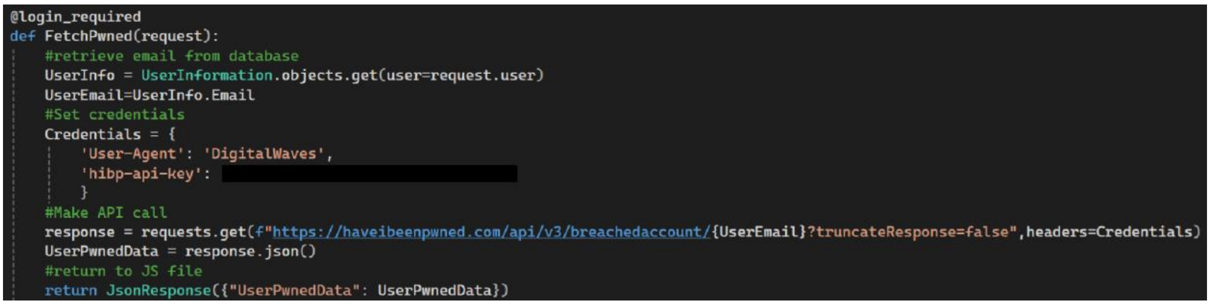
Figure 26: Django form for user information form

Django views were used to code the backend logic, with each piece of logic being inside a function for modularity and ensuring redundant code was not an issue. Each view handles a request and returns a response, which can be rendering a template, redirecting the user to a URL, or fetching data.

Interaction with data sources via APIs for personal information retrieval was vital functionality for the application. APIs allow for interaction with third-party databases via HTTP GET requests, returning data in a JSON format. This data could then be sanitised and useful data abstracted, which could then be sent to the frontend for presentation. Once an API was selected from those identified in the design phase, an API key could be obtained and an HTTP GET request formulated. Each API required different information as a parameter, hence each view had to also interact with the local database storing the users information to retrieve this parameter.

The APIs queried by the application are:

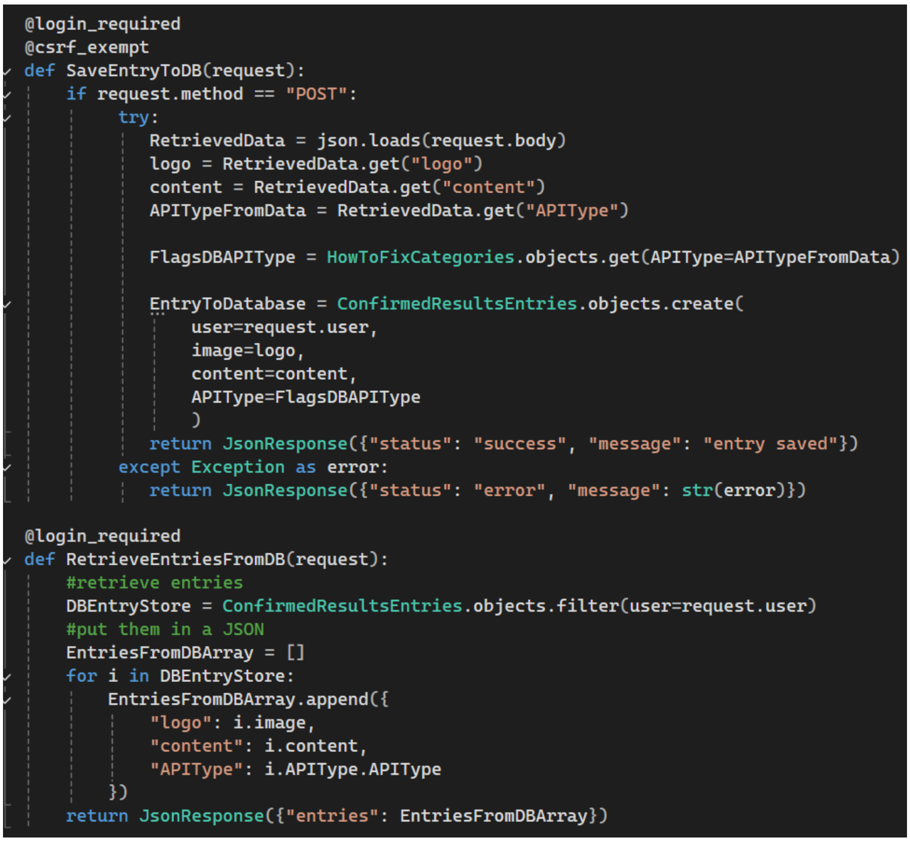
- HaveIBeenPwned
- Reddit
- GitHub
- X (Twitter)
- News
- Google CSE: custom search engines for Facebook, Companies House, LinkedIn



```
@login_required
def FetchPwned(request):
    #retrieve email from database
    UserInfo = UserInformation.objects.get(user=request.user)
    UserEmail=UserInfo.Email
    #Set credentials
    Credentials = {
        'User-Agent': 'DigitalWaves',
        'hibp-api-key': [redacted]
        }
    #Make API call
    response = requests.get(f"https://haveibeenpwned.com/api/v3/breachedaccount/{UserEmail}?truncateResponse=false",headers=Credentials)
    UserPwnedData = response.json()
    #return to JS file
    return JsonResponse({"UserPwnedData": UserPwnedData})
```

Figure 27: API GET request interaction logic for 'HaveIBeenPwned' API

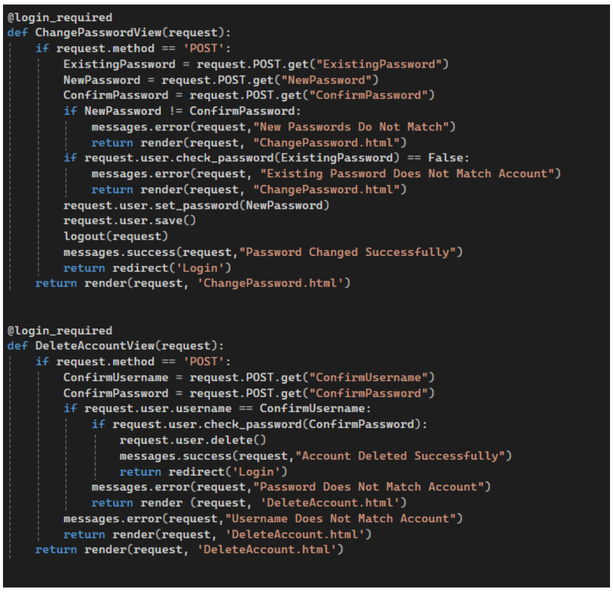
Views also had to be implemented to handle a user confirming an entry from the returned API responses. This logic interacted with the frontend JavaScript and the backend database for entry storage, to ensure that when a user clicked 'confirm', the entry was processed by the view, and the correct data was stored in the database. This was critical in enabling other logic to do the opposite process and retrieve the confirmed entries for each user from the database, allowing for the confirmed results to be displayed to the user and their score calculated.



```
@login_required
@csrf_exempt
def SaveEntryToDB(request):
    if request.method == "POST":
        try:
            RetrievedData = json.loads(request.body)
            logo = RetrievedData.get("logo")
            content = RetrievedData.get("content")
            APITypeFromData = RetrievedData.get("APIType")

            FlagsDBAPIType = HowToFixCategories.objects.get(APIType=APITypeFromData)

            EntryToDatabase = ConfirmedResultsEntries.objects.create(
                user=request.user,
                image=logo,
                content=content,
                APIType=FlagsDBAPIType
                )
            return JsonResponse({"status": "success", "message": "entry saved"})
        except Exception as error:
            return JsonResponse({"status": "error", "message": str(error)})

@login_required
def RetrieveEntriesFromDB(request):
    #retrieve entries
    DBEntryStore = ConfirmedResultsEntries.objects.filter(user=request.user)
    #put them in a JSON
    EntriesFromDBArray = []
    for i in DBEntryStore:
        EntriesFromDBArray.append({
            "logo": i.image,
            "content": i.content,
            "APIType": i.APIType.APIType
        })
    return JsonResponse({"entries": EntriesFromDBArray})
```

Figure 28: logic for saving a user confirmed entry to the database and retrieving a user's entries

Logic had to be implemented using views to allow the user to delete their data, abiding by the right to be forgotten following GDPR (European Union, 2016). Django's built-in accounts system was used to allow users to delete their account. Custom logic was also implemented to allow users to delete any personal data marked as 'optional' within the data form, as well as editing any of the stored data.



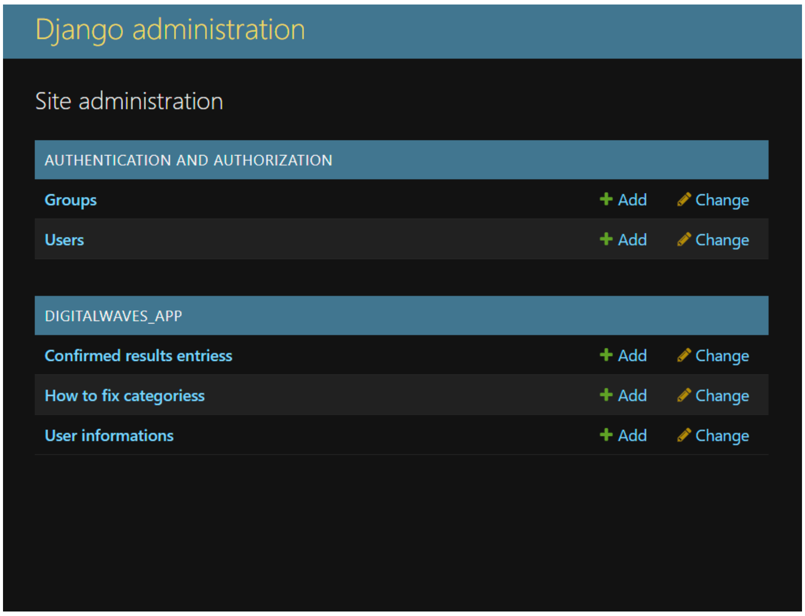
```
@login_required
def ChangePasswordView(request):
    if request.method == 'POST':
        ExistingPassword = request.POST.get("ExistingPassword")
        NewPassword = request.POST.get("NewPassword")
        ConfirmPassword = request.POST.get("ConfirmPassword")
        if NewPassword != ConfirmPassword:
            messages.error(request,"New Passwords Do Not Match")
            return render(request, "ChangePassword.html")
        if request.user.check_password(ExistingPassword) == False:
            messages.error(request, "Existing Password Does Not Match Account")
            return render(request, "ChangePassword.html")
        request.user.set_password(NewPassword)
        request.user.save()
        logout(request)
        messages.success(request,"Password Changed Successfully")
        return redirect('Login')
    return render(request, 'ChangePassword.html')


@login_required
def DeleteAccountView(request):
    if request.method == 'POST':
        ConfirmUsername = request.POST.get("ConfirmUsername")
        ConfirmPassword = request.POST.get("ConfirmPassword")
        if request.user.username == ConfirmUsername:
            if request.user.check_password(ConfirmPassword):
                request.user.delete()
                messages.success(request,"Account Deleted Successfully")
                return redirect('Login')
            messages.error(request,"Password Does Not Match Account")
            return render (request, 'DeleteAccount.html')
        messages.error(request,"Username Does Not Match Account")
        return render(request, 'DeleteAccount.html')
    return render(request, 'DeleteAccount.html')
```

Figure 29: Logic to allow user to change password and delete account

5.8: Database Storage

Django's Object Relational Mapping (ORM) database system was used for local storage, with custom tables being added to suit the requirements of the application, being written in SQLite due to its widespread support and ease of use.

Django also allows for a backend-style view of the database storage of the application though an administrative portal. Once a superuser is set up on the application, credentials can be used to access the admin view, where the databases in use can be seen and their contents. This is an useful tool, as it provides a GUI to interact with the databases for an admin, instead of using SQL for tasks.



Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	✎ Change
Users	+ Add	✎ Change

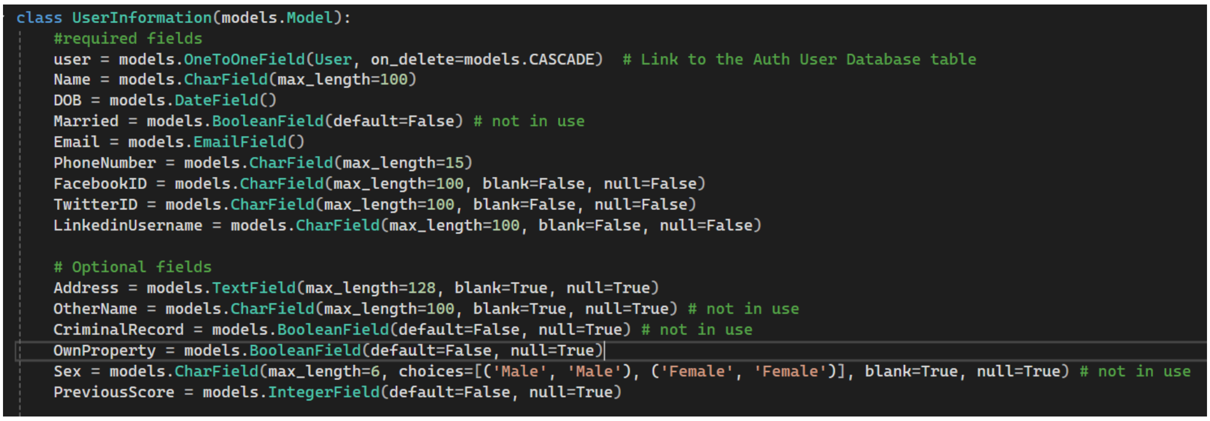
DIGITALWAVES_APP		
Confirmed results entriess	+ Add	✎ Change
How to fix categoriess	+ Add	✎ Change
User informations	+ Add	✎ Change

Figure 30: Django administrator site view, to interact with databases through a GUI

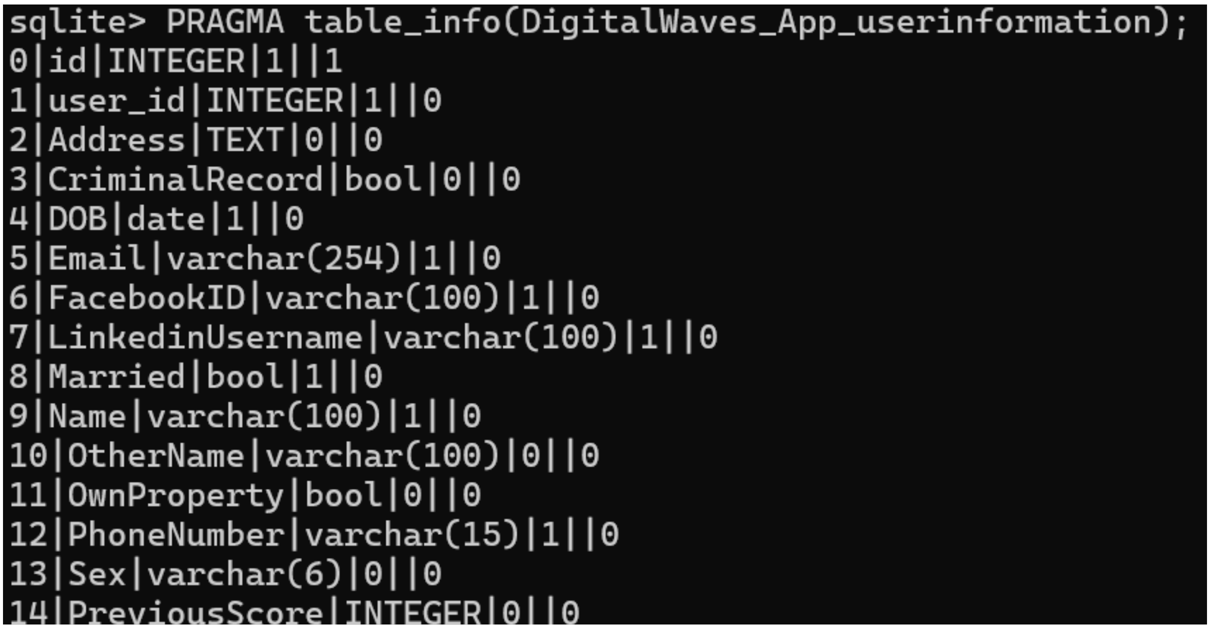
The standard Databases included by the Django framework included an accounts system, so this was utilised, along with the primary key the framework generates upon account creation. This was critical in ensuring that the custom databases could be linked to an account, observing Normal Form database laws.

Django uses models to define new databases, written in Python classes. Each table to be added to the database needed to be defined, and each field needed a data type defined, as well as any restrictions such as string length or whether the field was required.

The main table added was the user information table, which interacted with the data form to store a user's personal information which would be used as parameters for the API requests. This table used some mandatory and some optional fields, parameters which carried over to the data form automatically due to Django's framework handling this. Each record was tied to a user account using the account's primary key as a foreign key, and was coded to 'cascade delete', meaning the record for the user would delete if the user deleted their account. The fields in the database were also able to be edited



```
class UserInformation(models.Model):
    #required fields
    user = models.OneToOneField(User, on_delete=models.CASCADE)  # Link to the Auth User Database table
    Name = models.CharField(max_length=100)
    DOB = models.DateField()
    Married = models.BooleanField(default=False) # not in use
    Email = models.EmailField()
    PhoneNumber = models.CharField(max_length=15)
    FacebookID = models.CharField(max_length=100, blank=False, null=False)
    TwitterID = models.CharField(max_length=100, blank=False, null=False)
    LinkedinUsername = models.CharField(max_length=100, blank=False, null=False)

    # Optional fields
    Address = models.TextField(max_length=128, blank=True, null=True)
    OtherName = models.CharField(max_length=100, blank=True, null=True) # not in use
    CriminalRecord = models.BooleanField(default=False, null=True) # not in use
    OwnProperty = models.BooleanField(default=False, null=True)
    Sex = models.CharField(max_length=6, choices=[('Male', 'Male'), ('Female', 'Female')], blank=True, null=True) # not in use
    PreviousScore = models.IntegerField(default=False, null=True)
```

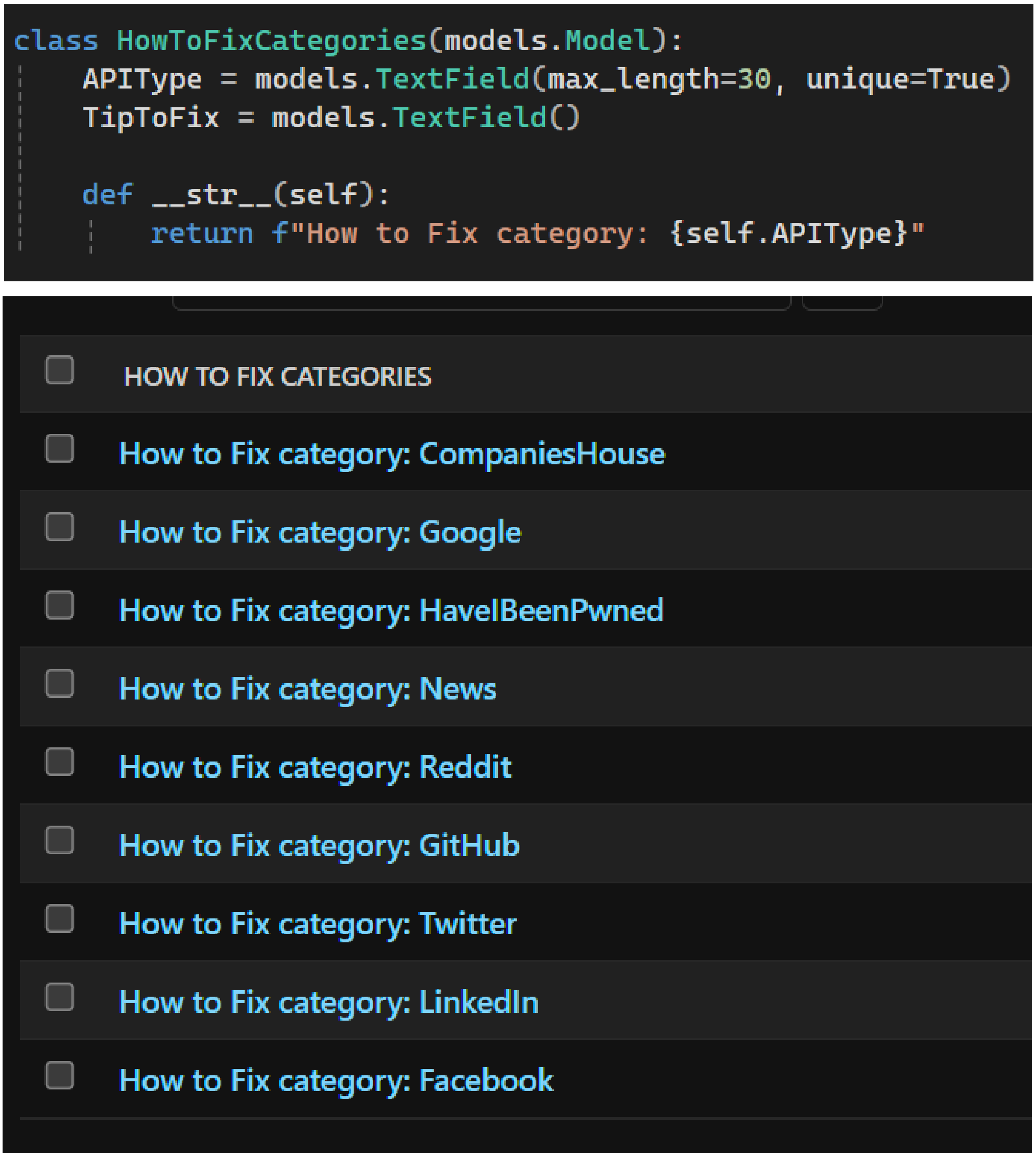


```
sqlite> PRAGMA table_info(DigitalWaves_App_userinformation);
0|id|INTEGER|1||1
1|user_id|INTEGER|1||0
2|Address|TEXT|0||0
3|CriminalRecord|bool|0||0
4|DOB|date|1||0
5|Email|varchar(254)|1||0
6|FacebookID|varchar(100)|1||0
7|LinkedinUsername|varchar(100)|1||0
8|Married|bool|1||0
9|Name|varchar(100)|1||0
10|OtherName|varchar(100)|0||0
11|OwnProperty|bool|0||0
12|PhoneNumber|varchar(15)|1||0
13|Sex|varchar(6)|0||0
14|PreviousScore|INTEGER|0||0
```

Figure 31: User information Model and table fields shown in SQL

after creation, which was used effectively when implementing storing the users previous scan score. An extra field was added to the database to do this, meaning a user's current and previous score could be compared to show improvement.

A "hints" database was also modelled, to store the advice which changed dependent on the type of data the confirmed entry was. This was a critical piece of implementation, as it allowed for the dynamic displaying of hints to the user of confirmed entries, which



```
class HowToFixCategories(models.Model):
    APIType = models.TextField(max_length=30, unique=True)
    TipToFix = models.TextField()

    def __str__(self):
        return f"How to Fix category: {self.APIType}"
```

□ HOW TO FIX CATEGORIES

□ How to Fix category: CompaniesHouse

□ How to Fix category: Google

□ How to Fix category: HaveIBeenPwned

□ How to Fix category: News

□ How to Fix category: Reddit

□ How to Fix category: GitHub

□ How to Fix category: Twitter

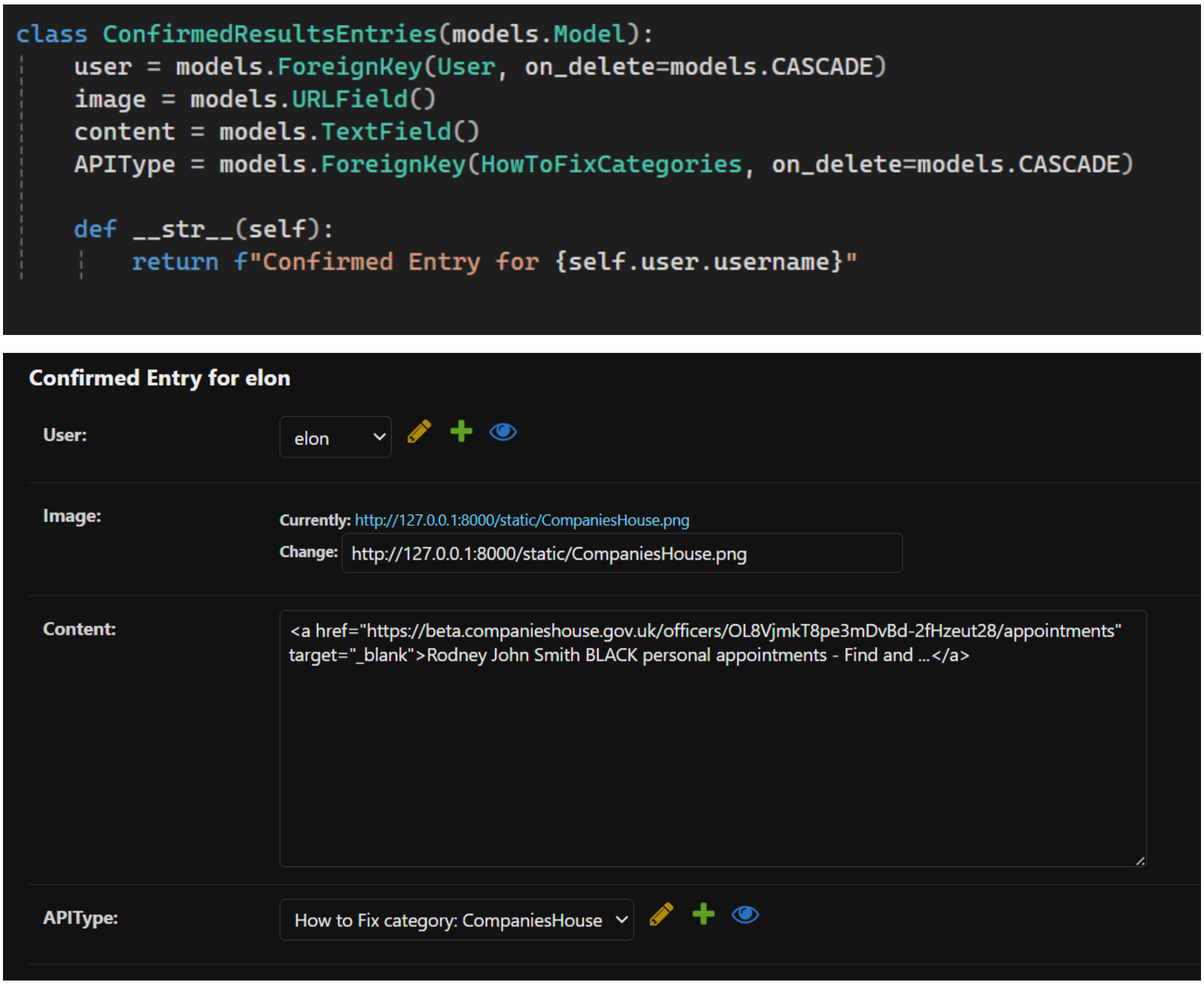
□ How to Fix category: LinkedIn

□ How to Fix category: Facebook

Figure 32: Hints system model and records shown in Django Admin view

showed them how to remove an entry to improve their digital footprint. This database was static and did not depend on a user's information, and hence did not need to be linked via a foreign key.

A recent scan database was also needed to ensure the user's confirmed entries could be stored, and loaded into the score section of the application along with the appropriate advice. This database used the same foreign key logic as the user information database, ensuring each entry was linked to a user with an account, ensuring unique entries and enforcing accountability.



```
class ConfirmedResultsEntries(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    image = models.URLField()
    content = models.TextField()
    APIType = models.ForeignKey(HowToFixCategories, on_delete=models.CASCADE)

    def __str__(self):
        return f"Confirmed Entry for {self.user.username}"
```

Confirmed Entry for elon

User:	elon
Image:	**Currently:** http://127.0.0.1:8000/static/CompaniesHouse.png **Change:** http://127.0.0.1:8000/static/CompaniesHouse.png
Content:	<a href="https://beta.companieshouse.gov.uk/officers/OL8VjmkT8pe3mDvBd-2fHzeut28/appointments" target="_blank">Rodney John Smith BLACK personal appointments - Find and ...</a>
APIType:	How to Fix category: CompaniesHouse

Figure 33: User-confirmed results database model and fields shown from Django admin view

Chapter 6: Testing and Evaluation

Testing and Evaluation are vital parts of the software development life cycle. Successful testing will ensure the application functions as intended and meets the initial functional and non-functional requirements. Comprehensive testing helps to catch bugs that a developer might overlook and handle edge cases, enforcing a positive UX and improving reliability of the application. Testing is particularly vital for applications such as DigitalWaves, where sensitive personal information is being handled, as bugs about security can lead to consequences such as GDPR breaches and/or a loss of user trust.

Both in-house Unit testing and User acceptance testing were considered for this project, to ensure a comprehensive testing regime was completed and a solid final product was delivered.

6.1: Unit Testing

A rigorous unit testing plan is vital to catching bugs and ensuring the application functions as designed. This means that every aspect of the system needs to be tested as much as possible to catch any unexpected outcomes. This structured method to bug fixing was critical to finding errors in the application, as an organic approach to this leads to areas and use cases not being tested accidentally, due to a developer working on the project for considerable periods and becoming naturally accustomed to how the application works in its current state, not how it should work as outlined in the requirements.

A test plan was devised for each major part of the application, with a focus on areas where the user inputted information, such as the login and accounts system. In these cases, not only does it need to be tested to ensure the product is functional, but boundary, normal, abnormal and extreme cases need to be tested to ensure that errors are caught and handled appropriately.

Login system:

Test	Expected outcome	Actual Outcome	Changes/comments
Login with valid credentials	User redirected to home page	User redirected to home page	
Login with incorrect Password	User shown error message	User shown error message	
Login with incorrect Username	User shown error message	User shown error message	
Login with empty Username field	User shown error message	User shown error message	
Login with empty Password field	User shown error message	User shown error message	

Accounts system:

Test	Expected outcome	Actual Outcome	Changes/comments
Register with valid credentials	User redirected to data form	User redirected to data form	
Register with mismatched usernames	Error message: usernames must match	Error message: usernames must match	
Register with existing username	Error message: User already exists	Error message: User already exists	
Register with mismatched passwords	Error message: passwords do not match	Error message: passwords do not match	
Register with password less than 8 characters – "1234567"	Error message: password must be more than 8 characters	Error message: password must be more than 8 characters	
Register with password with no capital letter – "password"	Error message: password must contain a capital letter	User redirected to data form	Check for capital letter needs to be added to account creation
Register with password without number - "passwordpassword"	Error message: password must contain a number	Error message: password must contain a number	
Register with password without special character – "password1"	Error message: password must contain a special character	Error message: password must contain a special character	
Register with empty fields	Error message: this field is required	Error message: this field is required	
Navigate back to login page	User redirected to login page	User redirected to login page	

Data forms system:

Test	Expected outcome	Actual Outcome	Changes/comments
Valid data submitted in required fields	User redirected to home page	User redirected to home page	
Submitted with empty required fields	Error message: this field is required	Error message: this field is required	
Invalid date format – DD/MM/YYYY	Error message: invalid date format	Form foes not submit. No error message shown to user	Error message functionality needs to be added to data form

Invalid email address format	Error message: invalid email format	Error message: invalid email format	Error message shown for ‘no @ in email address’ but not for no ‘.’ Suffix
Invalid phone number – not 11 numbers	Error message: invalid phone number format	User redirected to home page	
User tries to access data form unauthenticated	User redirected to login page	User redirected to login page	

Delete Accounts system:

Test	Expected outcome	Actual Outcome	Changes/comments
User clicks back button	User returned to home page	No back button present	Back button must be added
User enters valid information	Account deleted – User redirected to login page	Account deleted – User redirected to login page	
User does not enter username or password	Error message: fill in required fields	Error message: fill in required fields	
User enters incorrect username	Error message: incorrect username	Error message: incorrect username	
User enters incorrect password	Error message: incorrect password	Error message: incorrect password	
User tries to access page unauthenticated	User redirected to login page	User redirected to login page	

Change Username system:

Test	Expected outcome	Actual Outcome	Changes/comments
User clicks back button	User returned to home page	No back button present	Back button must be added Messages from login page also appear here – ‘registration successful’. Must be addressed
User enters valid information	Username changed in database – user redirected to login screen	Username changed in database – user redirected to login screen	
User does not enter current	Error message: fill in required fields	Error message: fill in required fields	

username or new username			
User enters incorrect current username	Error message: incorrect username	Error message: incorrect username	
User enters new username same as current	Error message: new username cannot be the same as old	Error message: new username cannot be the same as old	
User enters new username already in use	Error message: username already in use	Error message: username already in use	
User tries to access page unauthenticated	User redirected to login page	User redirected to login page	

Change Password system:

Test	Expected outcome	Actual Outcome	Changes/comments
User clicks back button	User returned to home page	No back button present	Back button must be added
User enters valid information	Password changed in database – user redirected to login screen	Password changed in database – user redirected to login screen	
User does not enter current password, new password or confirm password	Error message: please fill in existing fields	Error message: please fill in existing fields	
User enters incorrect current password	Error message: incorrect password	Error message: incorrect password	Current password field is not coded as password type – visible text not dots
User enters mismatched passwords	Error message: passwords must match	Error message: passwords must match	
Register with password less than 8 characters – “1234567”	Error message: password must be at least 8 characters	Password changed successfully	Implement check
Register with password with no capital letter – “password”	Error message: password must contain a capital letter	Password changed successfully	Implement check

User enters password without number - “Passwordpassword”	Error message: password must contain a number	Password changed successfully	Implement check
User enters password without special character – “Password1”	Error message: password must contain a special character	Password changed successfully	Implement check
User enters new password same as old password	Error message: New password must be different	Password changed successfully	Implement check
User tries to access page unauthenticated	User redirected to login page	User redirected to login page	

Delete Optional Information system:

Test	Expected outcome	Actual Outcome	Changes/comments
User clicks ‘delete optional information’ whilst having optional information stored	Success message: optional information deleted	Delete failed: please try again	Fix delete optional information
User clicks ‘delete optional information’ whilst having no optional information stored	Error caught: User has no optional information stored	Delete failed: please try again	Implement error catch
User tries to click button unauthenticated	Action is not carried out	Action is not carried out	

Settings menu system:

Test	Expected outcome	Actual Outcome	Changes/comments
User Clicks the settings icon on home page (hamburger icon)	Main settings menu loads in on left side of window	Main settings menu loads in on left side of window	
User clicks ‘Help’	Help menu loaded in in settings menu	Help menu loaded in in settings menu	FAQs stretch to the edge of width – implement padding
User scrolls through FAQs	Scroll view functions correctly, text wraps correctly	Scroll view functions correctly, text wraps correctly	
User clicks ‘Back’ from help menu	Main settings menu loaded in in settings menu	Main settings menu loaded in in settings menu	

User clicks ‘Account Settings’	Account settings menu loaded in in settings menu	Account settings menu loaded in in settings menu	
User clicks ‘Change Username’ in account settings	User redirected to change username page	User redirected to change username page	
User clicks ‘Change Password in account settings	User redirected to change password page	User redirected to change password page	
User clicks ‘Delete Account in account settings	User redirected to Delete Account page	User redirected to Delete Account page	
User clicks ‘Back’ from account settings menu	Main settings menu loaded in in settings menu	Main settings menu loaded in in settings menu	
User clicks ‘information wizard’	User redirected to data form page	User redirected to data form page	
User clicks ‘Log out’	User is unauthenticated and redirected to the login page	User is unauthenticated and redirected to the login page	Investigate cookies storing login credentials
User clicks ‘x’ on the settings menu	Settings menu disappears	Settings menu disappears	
User clicks on home page whilst settings menu is open	Home page is not interactive	Home page is not interactive	
User closes the settings menu and opens it again	No matter the page that was open, the settings page opens again on the main settings menu	The page that was previously open is loaded in again	Change so main settings menu is loaded in

Scan and results system:

Test	Expected outcome	Actual Outcome	Changes/comments
User clicks ‘start scan’	Scan loading bar bounces from side to side. Requests are sent to APIs for data. API data is loaded into templates and displayed to the user	Scan loading bar bounces from side to side. Requests are sent to APIs for data. API data is loaded into templates and displayed to the user	

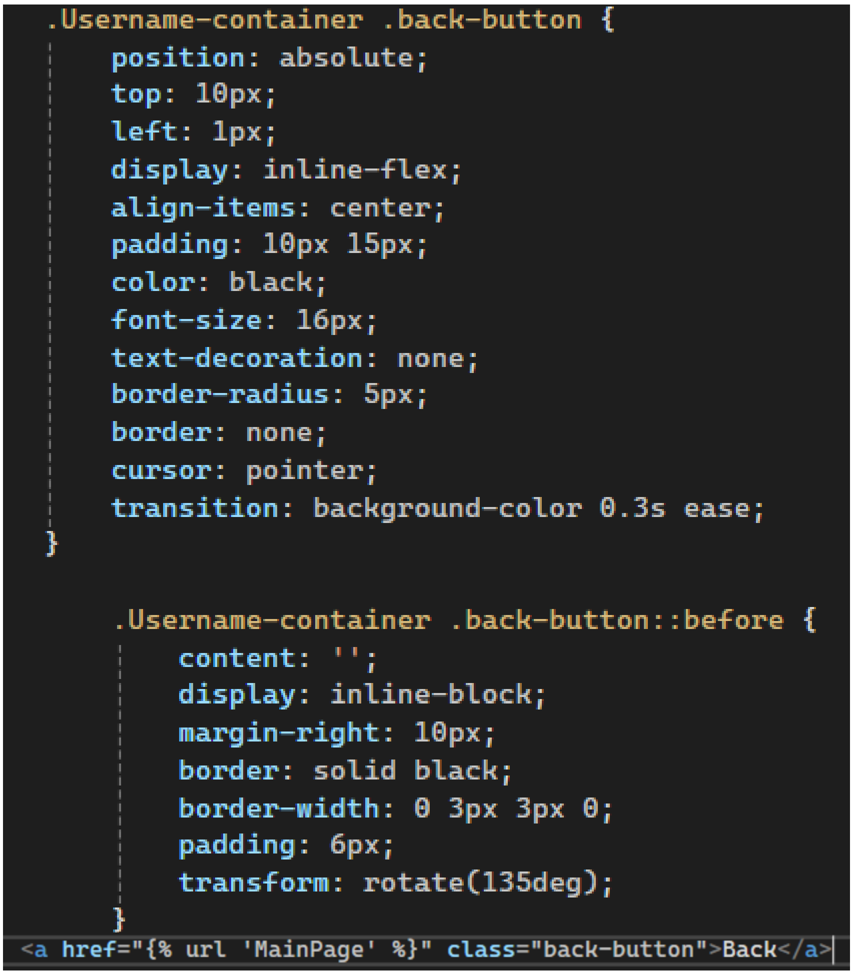
User clicks on ‘News’ tab	News tab is highlighted. Entries from the news API are visible	News tab is highlighted. Entries from the news API are visible	
User clicks on ‘Social Media’ tab	Social Media tab is highlighted. Entries from the Social Media APIs are visible	Social Media tab is highlighted. Entries from the Social Media APIs are visible	
User clicks on ‘Breaches’ tab	Breaches tab is highlighted. Entries from the HaveIBeenPwned API are visible	Breaches tab is highlighted. Entries from the HaveIBeenPwned API are visible	
User clicks on the ‘Businesses’ tab	Businesses tab is highlighted. Entries from the Companies house CSE are visible	Businesses tab is highlighted. Entries from the Companies house CSE are visible	
User Scrolls through entries on all tabs	Scroll views on all tabs allow the entries to wrap correctly and not extend past the bottom or sides of the page	Scroll views on all tabs allow the entries to wrap correctly and not extend past the bottom or sides of the page	
User clicks the link on an entry	The link pertaining to that entry is opened in a new tab	The link pertaining to that entry is opened in a new tab	
User clicks ‘confirm’ on an entry	‘confirm’ button should change to ‘confirmed’ and colour should change to green	‘confirm’ button should change to ‘confirmed’ and colour should change to green	

Score and hints system:

Test	Expected outcome	Actual Outcome	Changes/comments
User clicks ‘Confirm Entries’	Confirmed entries are stored in a database. The score is calculated based on the confirmed entries	Confirmed entries are stored in a database. The score is calculated based on the confirmed entries	
Score colours	The score representation should change	The score representation changes colour	

	colour based on a traffic light system – depending on how good the users score is	based on a traffic light system – depending on how good the users score is	
Comparison to previous score – first scan	No comparison should be shown	A comparison against 0 is shown	A check should be added for a 0/first score
Comparison to previous score – new score is worse	The difference should be shown to the user, with a sentence in red	The difference should be shown to the user, with a sentence in red	
Comparison to previous score – new score is better	The difference should be shown to the user, with a sentence in green	The difference should be shown to the user, with a sentence in green	
User scrolls though score breakdown entries	The scroll view should wrap the entries correctly. They should not extend beyond the size of the window	Scroll view extends beyond the bottom of the window	Fix scroll view parameters
User clicks 'how can I fix?' on an entry	A hint is retrieved from the database and displayed next to the entry in a box	Hint boxes appear at the top of the scroll view instead of next to the entries if user is scrolled down further than the length of the initial view size	Fix hint box constraints
User clicks anywhere on the page but the hint when the hint is open	The hint should close. No other part of the page is interactive when the hint is open	The hint closes. No other part of the page is interactive when the hint is open	
User initiates a new scan	Scan data is overwritten	News results section confirm buttons stay confirmed	Refresh news scan results
User refreshes the page	The scan is cleared	The scan is cleared	

Unit testing was instrumental in identifying bugs, contributing to a solid final product, particularly in areas that had been quickly generated from templates used in other areas of the application and had hence been less rigorously scrutinized, such as the ‘change username’ and ‘change password’ pages, which were generated from the login page template. However, as the login page did not have a back button, one was not included in these pages, meaning that the user was unable to return to the main page. This was quickly rectified include this feature.



```
.Username-container .back-button {
    position: absolute;
    top: 10px;
    left: 1px;
    display: inline-flex;
    align-items: center;
    padding: 10px 15px;
    color: black;
    font-size: 16px;
    text-decoration: none;
    border-radius: 5px;
    border: none;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

    .Username-container .back-button::before {
        content: '';
        display: inline-block;
        margin-right: 10px;
        border: solid black;
        border-width: 0 3px 3px 0;
        padding: 6px;
        transform: rotate(135deg);
    }
<a href="{% url 'MainPage' %}" class="back-button">Back</a>
```

Figure 34: back button HTML and CSS additions to change username, password and delete account pages

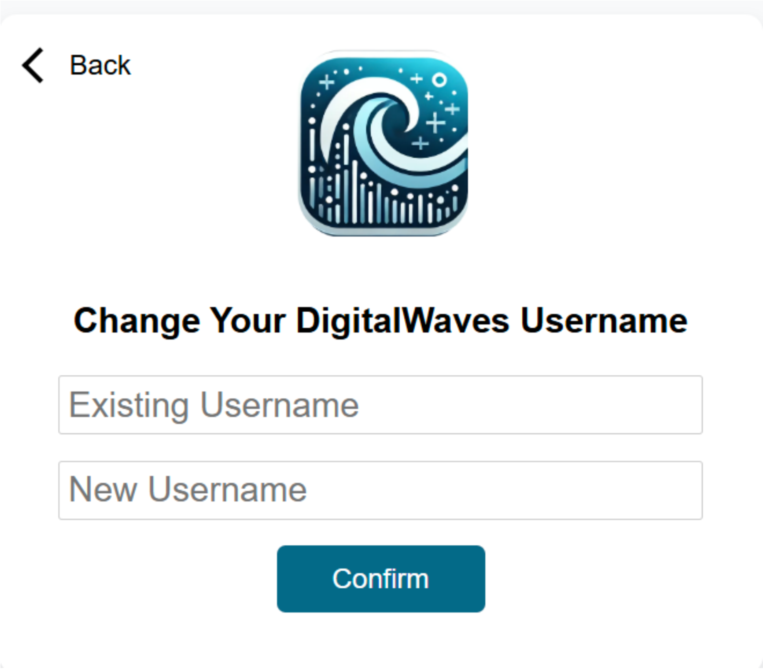
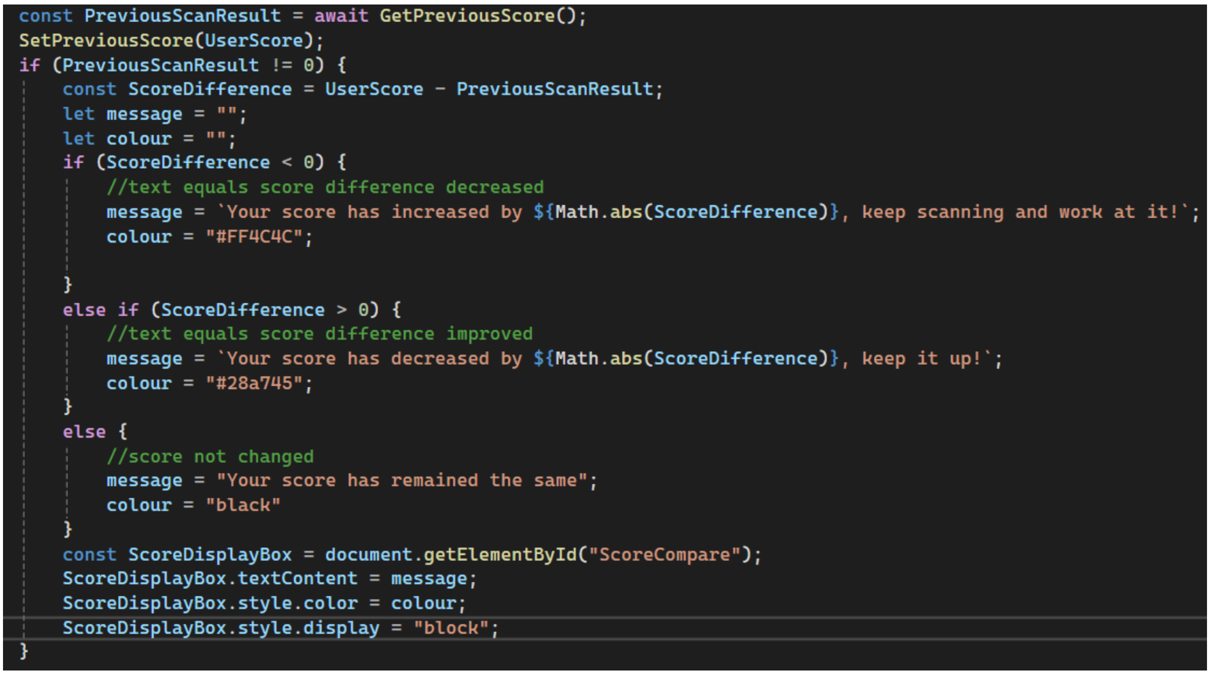


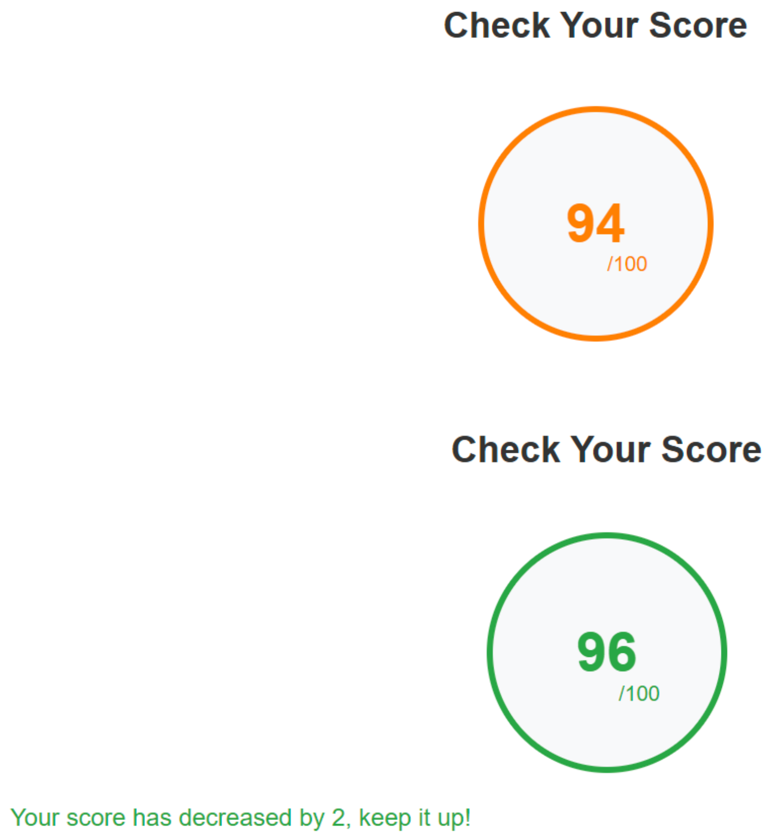
Figure 35: back button added to change username page

It was also identified that if the user was using the system for the first time, there was no error handling in place for comparing against a previous scan score. Since the previous score defaulted to 0 in the database, the logic compared their score with 0 and presented an inaccurate improvement message to the user. This was fixed in the logic by adding an error handling mechanism for the previously scanned score being 0, indicating a first-time scan.



```
const PreviousScanResult = await GetPreviousScore();
SetPreviousScore(UserScore);
if (PreviousScanResult != 0) {
    const ScoreDifference = UserScore - PreviousScanResult;
    let message = "";
    let colour = "";
    if (ScoreDifference < 0) {
        //text equals score difference decreased
        message = `Your score has increased by ${Math.abs(ScoreDifference)}, keep scanning and work at it!`;
        colour = "#FF4C4C";

    }
    else if (ScoreDifference > 0) {
        //text equals score difference improved
        message = `Your score has decreased by ${Math.abs(ScoreDifference)}, keep it up!`;
        colour = "#28a745";
    }
    else {
        //score not changed
        message = "Your score has remained the same";
        colour = "black"
    }
    const ScoreDisplayBox = document.getElementById("ScoreCompare");
    ScoreDisplayBox.textContent = message;
    ScoreDisplayBox.style.color = colour;
    ScoreDisplayBox.style.display = "block";
}
```

Figure 36: error handling code for the stored score being 0



Check Your Score

94
/100

Check Your Score

96
/100

Your score has decreased by 2, keep it up!

Figure 37: A first-time score and a following scan, showing no comparative message for the first scan

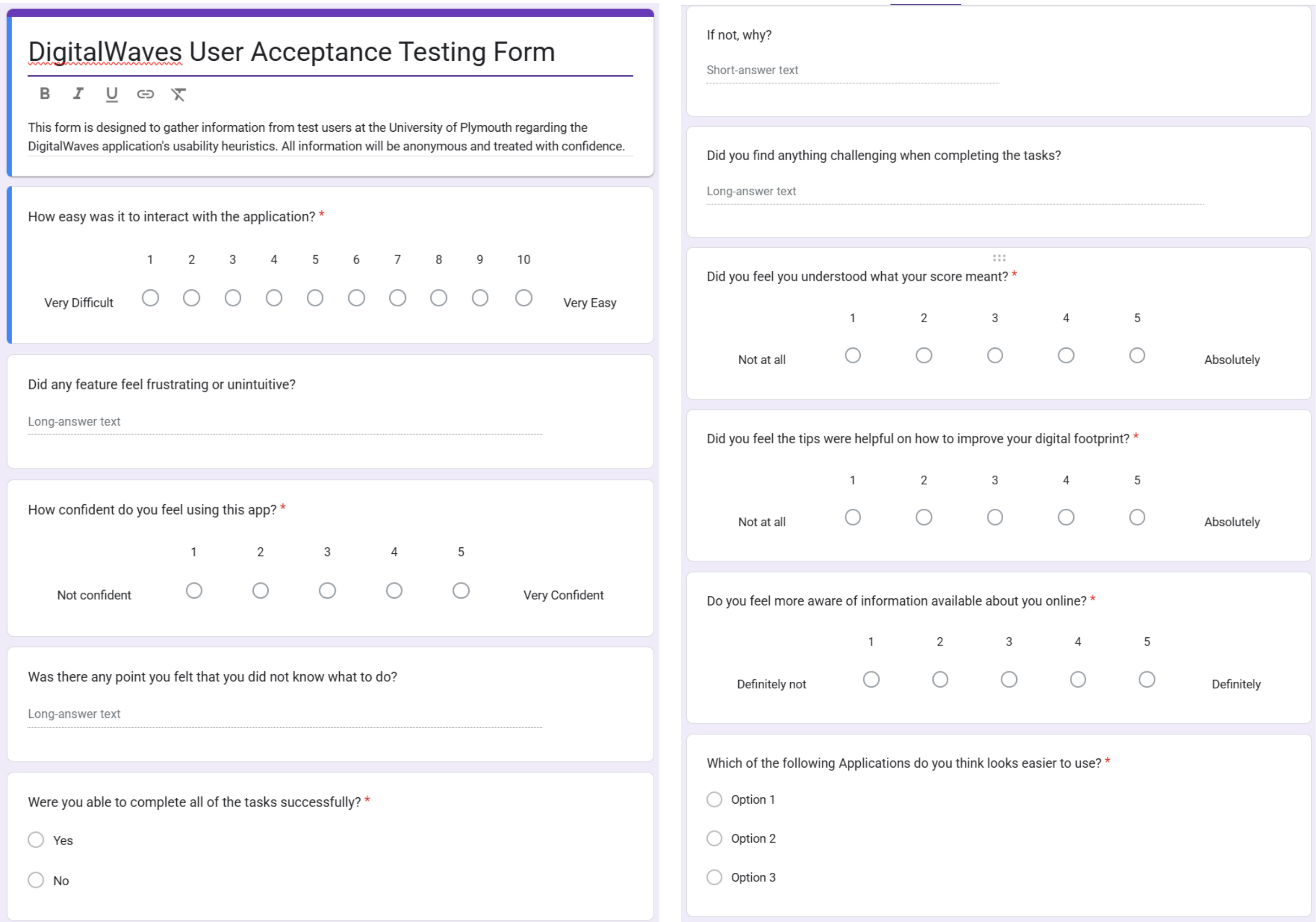
6.2: User Acceptance Testing

User Acceptance Testing (UAT) proved to be a vital process for verification of usability and success of features with live testing. This comprised of 8 users of the target demographic – non technical students aged 20-22 who had not been involved in the development process, ensuring a fair test.

Each user was provided with the application, as well as a basic list of tasks to complete:

1. Register for an account (or log in if you already have one) and fill in the data sheet
2. Scan your digital footprint
3. Explore the results – confirm some entries as being yours
4. Explore your score – how could you improve?
5. Delete your account

Finally, each user was asked to complete a survey, to obtain insight to their experience with the application:



DigitalWaves User Acceptance Testing Form

This form is designed to gather information from test users at the University of Plymouth regarding the DigitalWaves application's usability heuristics. All information will be anonymous and treated with confidence.

How easy was it to interact with the application? *

	1	2	3	4	5	6	7	8	9	10	
Very Difficult	○	○	○	○	○	○	○	○	○	○	Very Easy

Did any feature feel frustrating or unintuitive?

Long-answer text

How confident do you feel using this app? *

	1	2	3	4	5	
Not confident	○	○	○	○	○	Very Confident

Was there any point you felt that you did not know what to do?

Long-answer text

Were you able to complete all of the tasks successfully? *

○ Yes

○ No

If not, why?

Short-answer text

Did you find anything challenging when completing the tasks?

Long-answer text

Did you feel you understood what your score meant? *

	1	2	3	4	5	
Not at all	○	○	○	○	○	Absolutely

Did you feel the tips were helpful on how to improve your digital footprint? *

	1	2	3	4	5	
Not at all	○	○	○	○	○	Absolutely

Do you feel more aware of information available about you online? *

	1	2	3	4	5	
Definitely not	○	○	○	○	○	Definitely

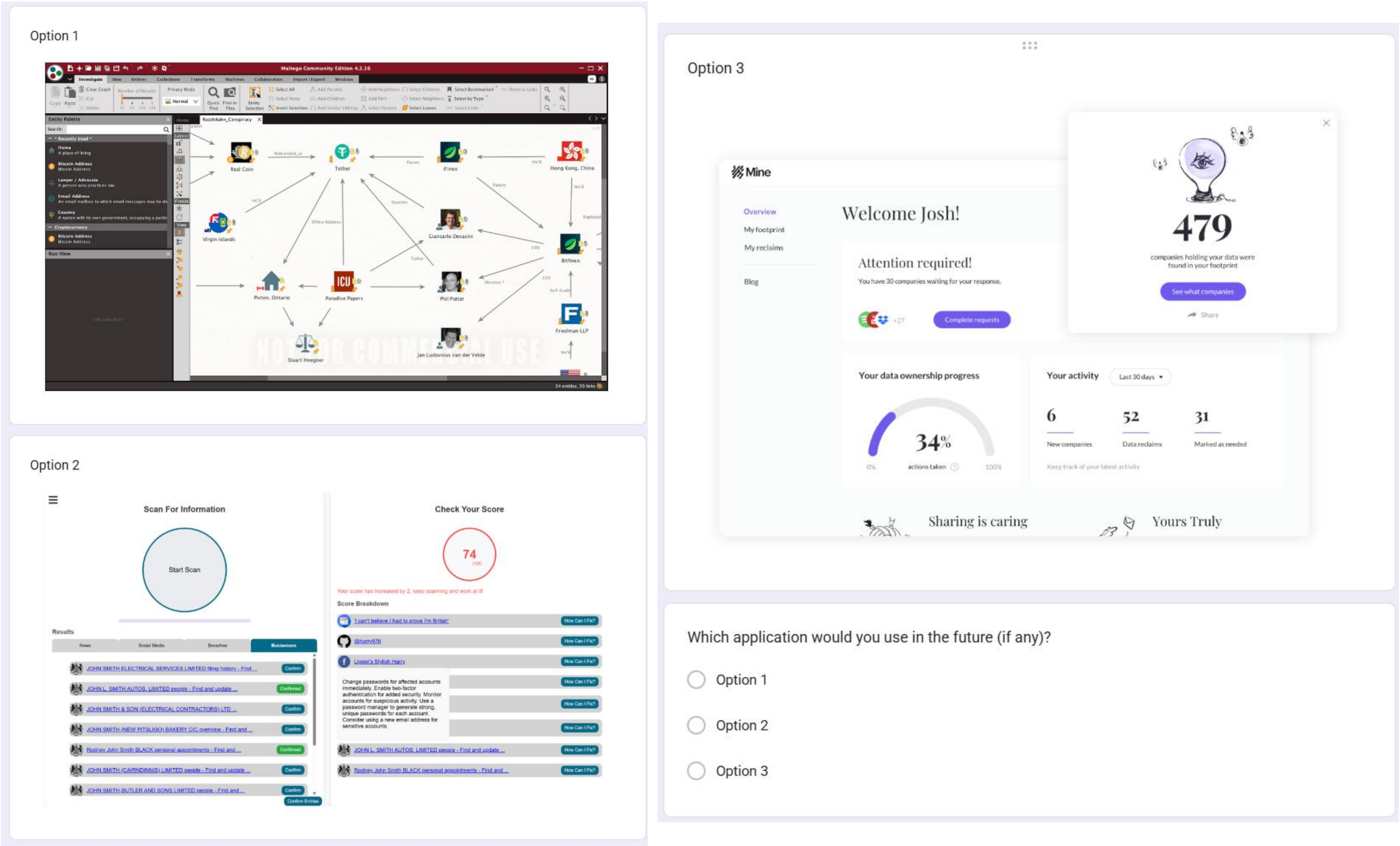
Which of the following Applications do you think looks easier to use? *

○ Option 1

○ Option 2

○ Option 3

Figure 38: User acceptance testing feedback form



Option 1

Option 2

Change passwords for affected accounts immediately. Enable two-factor authentication for added security. Monitor accounts for suspicious activity. Use a password manager to generate strong, unique passwords for each account. Consider using a new email address for sensitive accounts.

Option 3

Which application would you use in the future (if any)?

○ Option 1

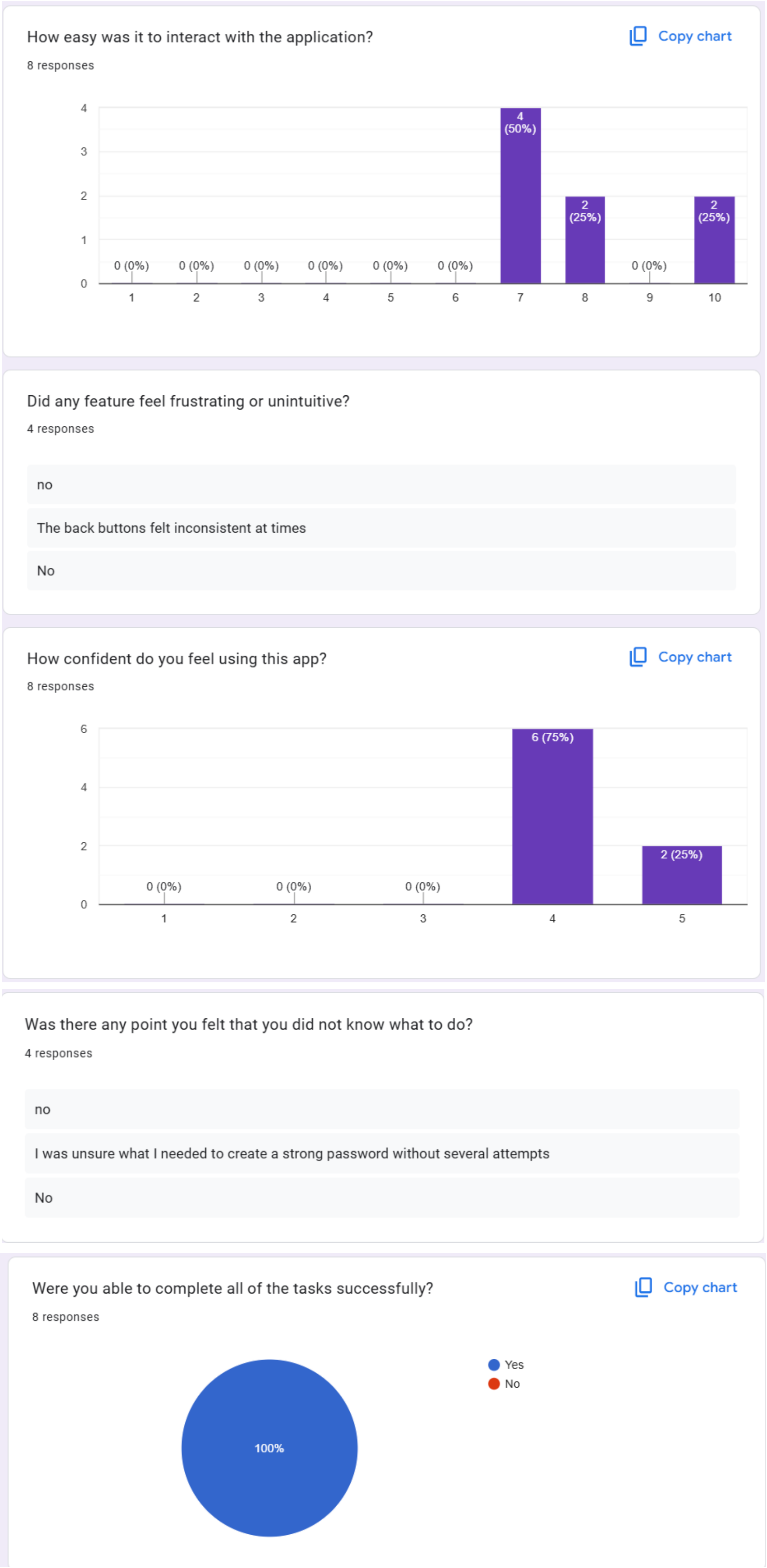
○ Option 2

○ Option 3

Figure 39: User Acceptance Testing feedback form

The results from the form were then compiled into statistics for critical analysis.

The results of UAT were positive overall, with all users being able to complete the list of tasks. The statistics demonstrated that the application had successfully met its requirements of being easy to interact with for non-technical people, with users feeling it was very easy to interact with. All users also reported feeling confident navigating the application and knowing what to do at all times.



How easy was it to interact with the application?

8 responses

Did any feature feel frustrating or unintuitive?

4 responses

no

The back buttons felt inconsistent at times

No

How confident do you feel using this app?

8 responses

Was there any point you felt that you did not know what to do?

4 responses

no

I was unsure what I needed to create a strong password without several attempts

No

Were you able to complete all of the tasks successfully?

8 responses

Figure 40: Ease of user interaction statistics

It was also clear that the requirement of making users more aware of their digital footprint had been met, with 75% of users feeling 'definitely' more aware of the information available about them online. 75% of users also felt that the tips on confirmed entries were 'absolutely' useful, displaying clearly that the overall requirements of the application had been met, by ensuring that not only were the users more aware of their digital footprint, but also more equipped to deal with the threat that they were aware of.

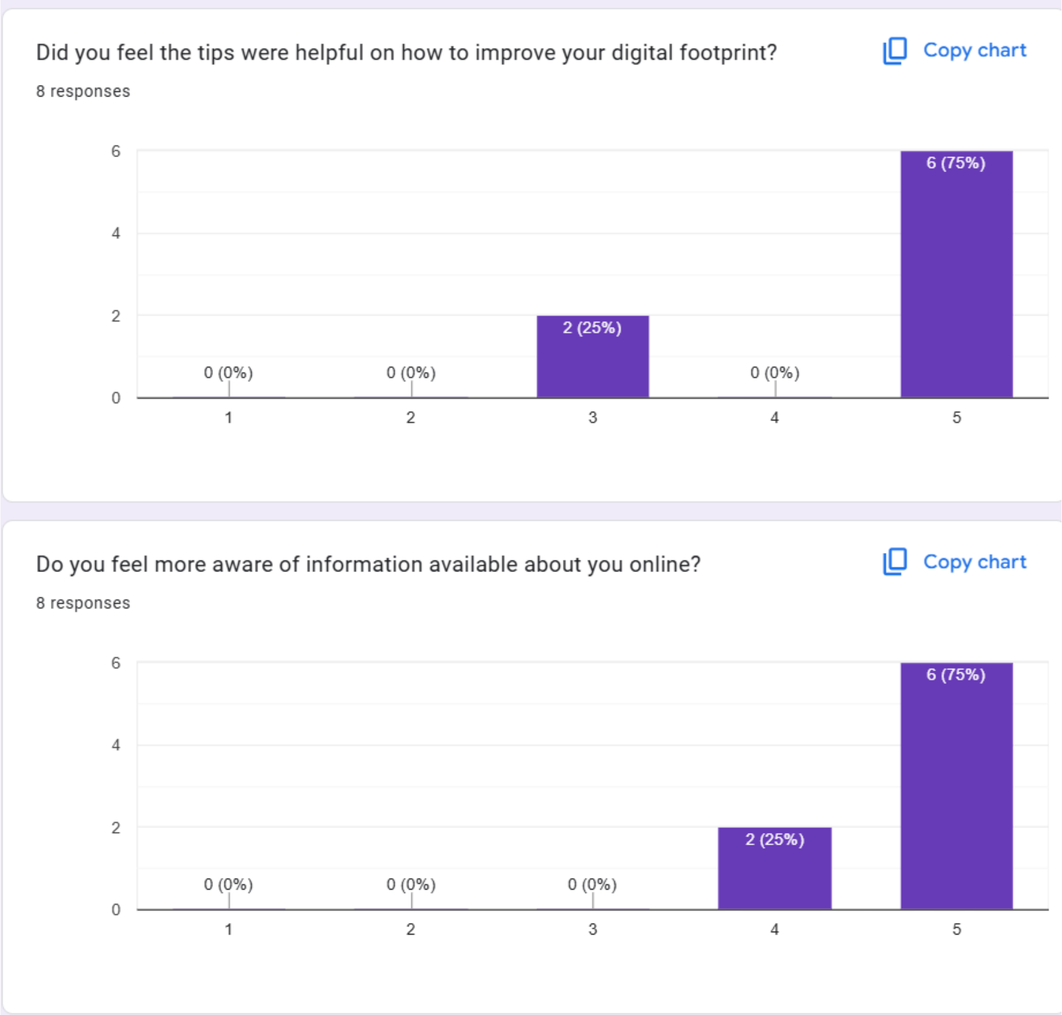
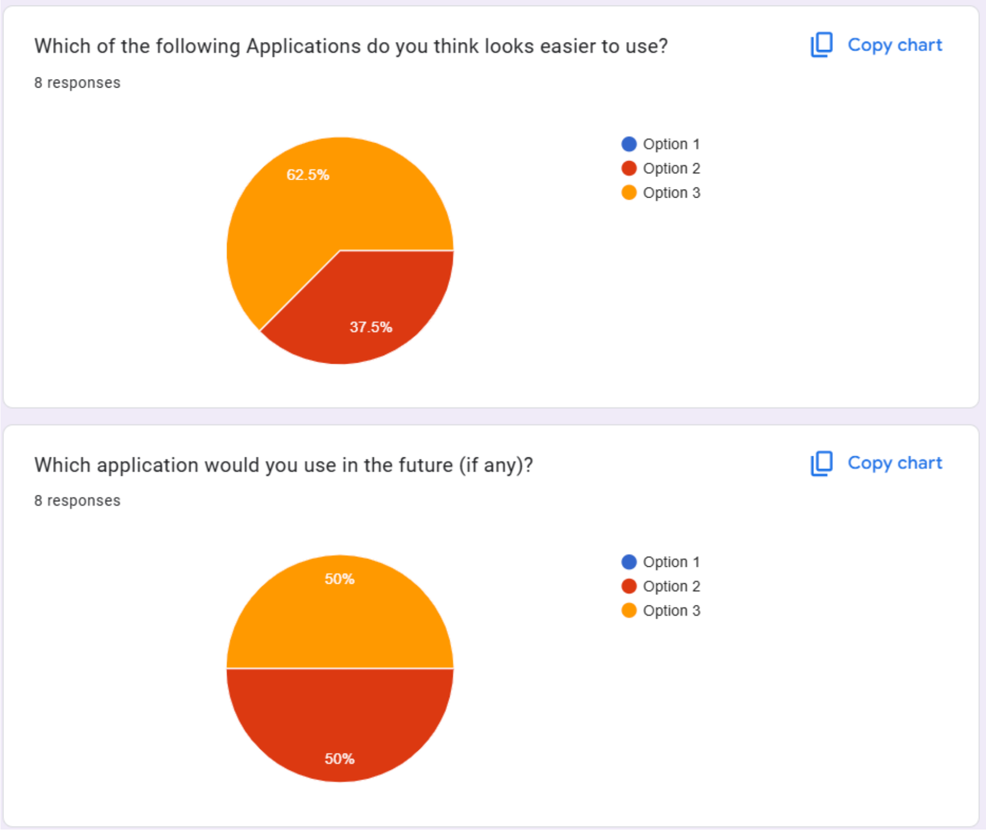


Figure 41: User Acceptance Testing statistics on improvement of user awareness of their digital footprint

Users were also presented with screenshots of 3 digital footprint tracking applications – Maltego, DigitalWaves and MIME. They were then asked which application looks easiest to use, and which application they would use in the future. These statistics were very valuable, as it demonstrated the comparison between the application and the current state of the art. Furthermore, it also demonstrated whether the user would use the application in the future, fulfilling non-function requirement 5, focusing on education through consistent use.

Results were split between DigitalWaves and MIME, demonstrating that the complicated UI of Maltego is not appealing to a non-technical user. 37.5% of users felt that DigitalWaves was the easiest to use, demonstrating that further improvements could be made to further align the product with usability heuristics. However, 50% of users stated that they would use DigitalWaves in the future, demonstrating clearly that whilst improvements could be made in the future prior to commercialisation, the application has fulfilled its requirement of being able to be used consistently.



Which of the following Applications do you think looks easier to use?

8 responses

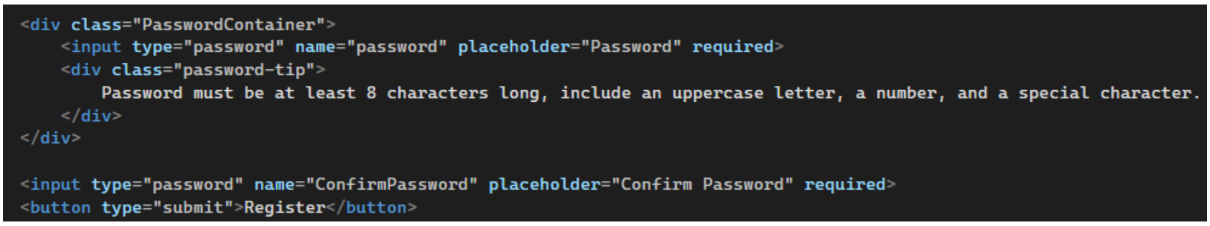
Which application would you use in the future (if any)?

8 responses

Figure 42: Application comparison statistics

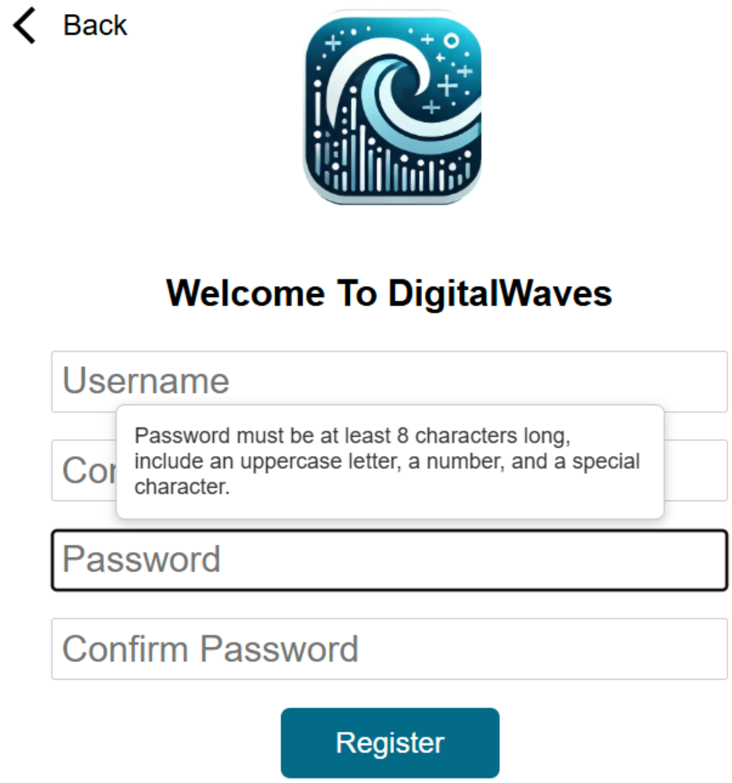
Further understanding was also gained following the UAT, when it was discussed with some users how MIME works – by accessing your email account and automating the digital footprint removal service. The users to whom this was disclosed to felt very uncomfortable at this and stated that they would not use MIME.

Valuable insights were drawn on what could be improved. A major frustration point observed ubiquitously across the users was that they struggled to create an account due to the password policy not being displayed before an error message being displayed. This frustration was concerning, as it could lead to a user not even creating an account before becoming frustrated and giving up. This was remedied by ensuring the password policy is displayed to the users on the account creation page.



```
<div class="PasswordContainer">
    <input type="password" name="password" placeholder="Password" required>
    <div class="password-tip">
        Password must be at least 8 characters long, include an uppercase letter, a number, and a special character.
    </div>
</div>

<input type="password" name="ConfirmPassword" placeholder="Confirm Password" required>
<button type="submit">Register</button>
```

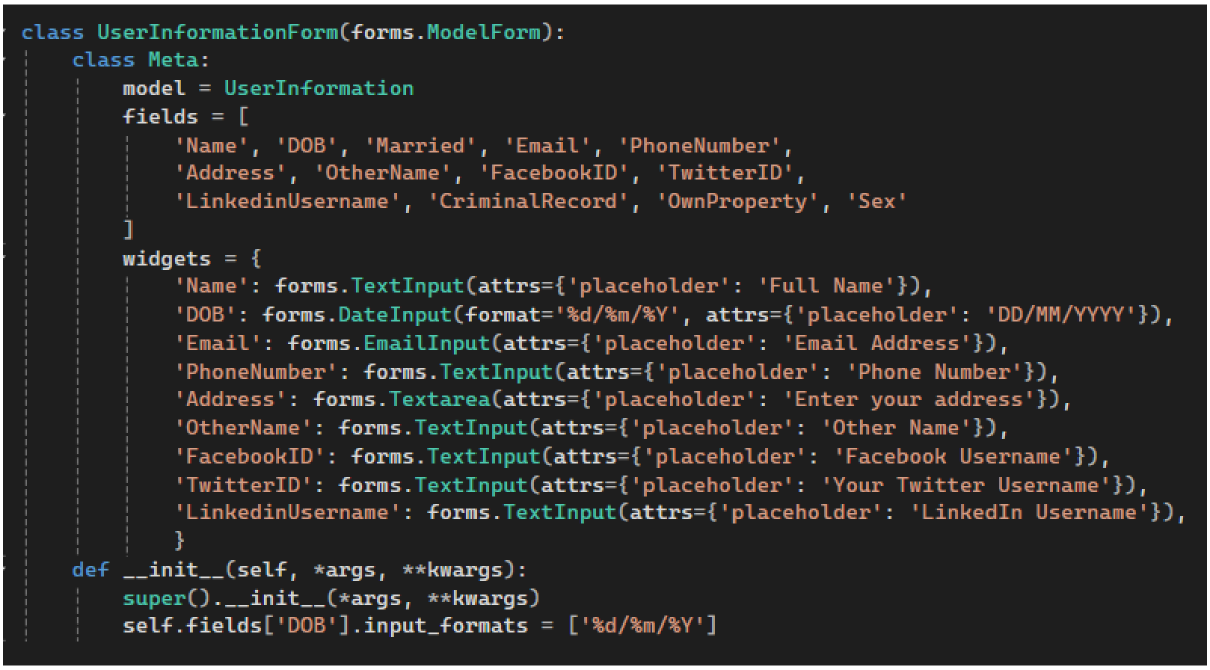
Figure 43: updated registration form code to show the password policy



Password must be at least 8 characters long, include an uppercase letter, a number, and a special character.

Figure 44: Registration form showing password policy when password box is highlighted

Minor issues were also encountered by users on the data information page. When entering their date of birth, users naturally entered the date in a UK format (DD/MM/YYYY), however, because of the use of Django’s forms, the date needed to be entered in the US format (MM/DD/YYYY). Even though this was displayed as the hint, users still became very confused, especially as the data form page did not provide an error informing the user of this. This oversight of needs for the target demographic was addressed by altering the date format within the Django Form using a custom field.



```
class UserInformationForm(forms.ModelForm):
    class Meta:
        model = UserInformation
        fields = [
            'Name', 'DOB', 'Married', 'Email', 'PhoneNumber',
            'Address', 'OtherName', 'FacebookID', 'TwitterID',
            'LinkedinUsername', 'CriminalRecord', 'OwnProperty', 'Sex'
        ]
        widgets = {
            'Name': forms.TextInput(attrs={'placeholder': 'Full Name'}),
            'DOB': forms.DateInput(format='%d/%m/%Y', attrs={'placeholder': 'DD/MM/YYYY'}),
            'Email': forms.EmailInput(attrs={'placeholder': 'Email Address'}),
            'PhoneNumber': forms.TextInput(attrs={'placeholder': 'Phone Number'}),
            'Address': forms.Textarea(attrs={'placeholder': 'Enter your address'}),
            'OtherName': forms.TextInput(attrs={'placeholder': 'Other Name'}),
            'FacebookID': forms.TextInput(attrs={'placeholder': 'Facebook Username'}),
            'TwitterID': forms.TextInput(attrs={'placeholder': 'Your Twitter Username'}),
            'LinkedinUsername': forms.TextInput(attrs={'placeholder': 'LinkedIn Username'}),
            }
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.fields['DOB'].input_formats = ['%d/%m/%Y']
```

Figure 45: Updated user information form to allow for a custom date format input

Chapter 7: Project Conclusions

7.1: conclusion

The initial aims of the project were to create a tool that focuses on allowing non-technical users to monitor and track their digital footprint, with an aim specifically on helping them to improve by securing aspects of their digital footprint, or removing it entirely. The project culminated in a functional and easy-to-use web application that allows the user to track their digital footprint, scanning for information using third party APIs, and receive advice on how to sanitise their digital footprint.

The project aimed to fill a gap in the market, with existing tooling focusing on the technical side of OSINT and its application to investigations. The project had a specific focus on ensuring the final product was accessible to those who had less technical ability, and would become frustrated at a complicated product, leading them to ignore the threats that a large and exposed digital footprint poses to a user. Significant pre-implementation research allowed for this to be incorporated successfully. Designs reflected a simple and easy-to-use interface that users would recognise and become comfortable with more quickly, adhering closely to HCI principles and pre-existing design conventions. The positive user testing outcomes display that the HCI-focussed approach was successful, with 100% of users not finding the application frustrating or unintuitive to use, demonstrating a positive UX.

Security needed to be a high priority throughout the implementation of the project, with the product handling sensitive personal information about a user. If security was not implemented correctly, this would have impacted the product significantly, with user trust falling. GDPR also had to be adhered to, with user data being stored locally wherever possible. User account protection was critical, with a login and registration system being implemented utilising password hashing for additional security. This ensured protection of user data, and accountability, as scans and personal data were attributable to an account.

Education of users and repeat use of the application was also a key priority. The product allows users to use the tailored hints system, providing actionable advice on how to improve their digital footprint. If the user has used the application before, the product also displays a comparative score, enabling the user to track their progress and see if they have improved. This education-focused approach encourages users to take advantage of the advice to improve, and then return to the application to see how they could improve further.

Whilst the project overall was a success, some limitations occurred. Upon further research during development, it was discovered that some APIs that were to be implemented were very limited, and some did not work in a way that would provide useful information to a user. The Facebook API was an example of this, with the API only returning user-related data to an application once a user has authenticated their Facebook account through the application, meaning the user would have to login to their account to see information about their account, rendering it pointless. To solve this issue, a google custom search engine was implemented to search for only Facebook-related links, with the users information as a parameter, allowing the product to progress as planned. This same method was also used when similar issues were encountered with the LinkedIn API.

Another issue that was apparent throughout the testing of the application was the volume of false positives that occurred. The amount of information on the internet leads to a large amount of the links presented to a user not being relevant to them. A more complicated application may be able to filter this information more effectively, however this would make the UI more complicated, which was out of scope of this project. This introduces the need for a balance, of how many results are shown to the user, however, the less results shown to a user, the higher the chance is that a positive entry is missed and not shown to the user.

Future development was considered heavily throughout the project. With this being a full-stack development project in limited time, more depth could be added in some areas. The addition of more APIs and data sources would lead to a more comprehensive version of the product. This would be relatively simple to implement due to the modular design of the application, with each API having separate logic.

Addition of Large Language Models (LLMs) could also be considered, leading to a much more in depth and personalised hints system, with each hint being specifically tailored to that entry, providing more depth and additional context to a user. OpenAI's API could be used, also allowing the hints system database to be removed, making the application more lightweight.

With the focus of the product being on education, the application could also be sold or licensed to universities or digital safety organisations, to help more users to sanitise their digital footprint.

The project delivered a successful functioning product that solves a real-world problem, allowing users to address their digital footprint securely and clearly. Comprehensive UAT demonstrated the products effectiveness and demonstrated promising features for additional development and commercial deployment.

7.2: Personal reflection

This project presented an excellent learning opportunity, allowing me to identify a cyber security problem, and then enhance my software engineering and cyber security skills to solve it.

The project allowed me to tackle full-stack development as a sole developer for the first time, which enabled me to have complete understanding of the entire system, and learn how to balance functionality between the sections of the system. This approach enabled me to effectively balance the user-friendly UI/UX with the complex backend logic that needed to be implemented to create the functionality the application needed. This was particularly rewarding when I was able to witness users effectively engage with the application during user testing, proving that the project that I had solely developed had been a success.

The project also enabled me to tackle an unfamiliar tech stack, using Python's Django framework for the first time, providing a valuable learning opportunity into an industry-standard framework.

The continued use of agile software development throughout the project was also a positive achievement. Whilst I have used the agile development framework before, implementing it on an individual project allowed for an appreciation for iterative development, even when working alone. Sole development can often lead to scope creep and an organic-style of project management, however use of agile software development prevented this. The early delivery of the MVP ensured that the client was happy with the proof of concept prototype of the project, and the following sprints allowed for the iterative approach taken to this project. The regular client meetings also ensured that progress was always being made, and that blockers could be addressed and the project could pivot if necessary.

Throughout developing the product, I came across many challenges that were overcome. The initial learning curve for the Django backend framework was particularly notable, as its modular approach was something I was unfamiliar with. Whilst this took considerable time at the beginning of the project, it provided significant benefit in the implementation that followed, allowing for easy implementation of a non-monolithic project, allowing quick changes if necessary and offering time-saving built-in features such as the secure accounts system.

Another challenge that I faced was the complexity of the significant amount of dynamic elements that had to be implemented into the main page. Each entry into both the results and score section needed to be sanitised and added to a template, and then dynamically generated in HTML. This complexity allowed me to learn more about how each element of the stack interacts with one another during implementation of a web application, further improving my technical confidence in full-stack development applied to a cyber security problem.

If I were to complete this project again from scratch, there are things I would change to improve the final product. Further investigation before the implementation phase could have demonstrated that some APIs may not have functioned how I expected, and hence this blocker could have been avoided and the time spent better in implementing other data sources that could have been more useful.

The frontend complexity was not something I had anticipated, so considerable time was spent on the dynamic elements, utilising HTML, CSS and JavaScript to achieve the desired outcome. Had I been aware of this, and taking into account how helpful the Django backend framework was, I would have conducted research into a frontend framework, such as React. This could have made the frontend less difficult to create, and sped up development, allowing for the time to be spent implementing more data sources or LLM implementation.

Overall, this project has provided me with significant opportunity to hone my skills in software development, both in a technical and non-technical context. The exposure to full-stack development alone provided vital insight into the complexity of a web application. The use of agile within the software development life cycle allowed for scope creep to be managed, and constant progress to be made in an iterative process following MVP delivery. This project also helped to further refine my interest in cyber security tooling, helping to recognise the major issue of the trade-off between security and HCI friendly tools, and the progress needed to narrow this gap further in the cyber security field. I look forward to continuing to develop security-focused technologies that will make a difference for users.

References:

Cabinet Office. (2009). *The cost of cyber crime: Summary report*. Detica in partnership with the Office of Cyber Security and Information Assurance. Available at: https://assets.publishing.service.gov.uk/media/5a78e882e5274a2acd18ab84/THE-COST-OF-CYBER-CRIME-SUMMARY-FINAL.pdf

Gantz, J. F., & Reinsel, D. (2011). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the history of the Northwest. *IDC iView*. International Data Corporation. Available at: https://www.idc.com/research/viewtoc.jsp?containerId=2326

Akin, J. (2019). *Identity Theft Statistics*. [online] Experian.com. Available at: https://www.experian.com/blogs/ask-experian/identity-theft-statistics/.

European Union (2016). *General Data Protection Regulation (GDPR)*. [online] General Data Protection Regulation (GDPR). Available at: https://gdpr-info.eu/.

UK Government (2018). *Data Protection Act 2018*. [online] Legislation.gov.uk. Available at: https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted.

Kenneally, E. and Dittrich, D. (2012). The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research. *SSRN Electronic Journal*. [online] doi:https://doi.org/10.2139/ssrn.2445102.

GOV.UK (2024). *Cyber security breaches survey 2024*. [online] GOV.UK. Available at: https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2024/cyber-security-breaches-survey-2024.

Herley, C. (n.d.). *So Long, And No Thanks for the Externalities: The Rational Rejection of Security Advice by Users*. [online] Available at: https://www.nspw.org/papers/2009/nspw2009-herley.pdf.

Beautement, A., Sasse, M. and Wonham, M. (n.d.). *The Compliance Budget: Managing Security Behaviour in Organisations*. [online] Available at: https://www.nspw.org/papers/2008/nspw2008-beautement.pdf.

ResearchGate. (n.d.). *Transforming the 'Weakest Link' — a Human/Computer Interaction Approach to Usable and Effective Security*. [online] Available at: https://www.researchgate.net/publication/2404434_Transforming_the_.

Nielsen, J. (1994). *Usability Engineering*. [online] *Google Books*. Morgan Kaufmann. Available at: https://books.google.co.uk/books?hl=en&lr=&id=95As2OF67f0C&oi=fnd&pg=PR9&dq=Nielsen.

Django (2024). *Password management in Django | Django documentation*. [online] Django Project. Available at: https://docs.djangoproject.com/en/5.1/topics/auth/passwords/.

Django (2024). *The Web framework for perfectionists with deadlines | Django*. [online] Djangoproject.com. Available at: https://www.djangoproject.com/.

https://digital.ai/. (n.d.). *17th State of Agile Report | Press Releases | Digital.ai*. [online] Available at: https://digital.ai/press-releases/17th-state-of-agile-report-71-use-agile-in-their-sdlc-small-organizations-report-strong-business-benefits-medium-and-larger-sized-companies-continue-to-experience-barriers-in-successfully-scaling-a/.