HOCHSCHULE
ESSLINGEN

Weiterentwicklung der Privacy Assistant Software

Studienprojekt

im Studiengang
Softwaretechnik und Medieninformatik

vorgelegt von

Andreas Kolb & Maximilian Fink
Matr.-Nr.: 764390 & 764001

am 3. August 2023
an der Hochschule Esslingen

Erstprüfer: Prof. Dr. rer. nat. Tobias Heer
Zweitprüfer: Prof. Dr. Dominik Schoop

Kurzfassung

Dieser Bericht dokumentiert die Weiterentwicklung der Software „Privacy Assistant“ der Hochschule Esslingen, welche Datenlecks auf Websites erfassen kann. Basierend auf Django wurde das Projekt mit dem Ziel übernommen, die Benutzerfreundlichkeit und die Codequalität zu verbessern sowie neue Features einzubauen. Durchgeführte Maßnahmen umfassten die Containerisierung der Software, die Entwicklung eines neuen Frontends in React und die Umgestaltung der Backend-Struktur zu einer REST-API.

Schlagwörter: Privacy Assistant, Datenschutz, Privatsphäre, Scanner, Webseiten-Scraping, Python, Django, REST, Docker, React

Hinweis zur geschlechtergerechten Sprache

In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderweitige Geschlechteridentitäten werden dabei ausdrücklich mitgemeint, soweit dies für die Aussage erforderlich ist.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration / Continuous Delivery
CRUD	Create, Read, Update, Delete
DRY	Don’t Repeat Yourself
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JS	JavaScript
KISS	Keep It Simple and Stupid
o. D.	ohne Datum
PS	Privacy Assistant
URL	Uniform Resource Locator
UX	User Experience
REST	Representational State Transfer
vgl.	vergleiche
VS Code	Visual Studio Code

1 Einleitung

In der heutigen digitalen Welt spielt der Datenschutz eine zunehmend wichtige Rolle. Jeden Tag werden Unmengen von Daten in das Internet hochgeladen und geteilt, was dazu führt, dass es schwierig ist, den Überblick zu behalten und zu kontrollieren, welche persönlichen Daten öffentlich zugänglich sind. Dies gilt insbesondere für große Websites wie die der Hochschule Esslingen, auf der jeder Inhalte hochladen kann.

In diesem Kontext gewinnen Privacy Tools an großer Bedeutung. Bestehende Lösungen wie Macie von Amazon Web Services (AWS) stellen sicher, dass sensible Informationen in AWS S3 Buckets nicht unbeabsichtigt öffentlich gemacht werden und helfen dabei, Compliance-Anforderungen zu erfüllen, vgl. dazu Amazon Macie – AWS (2023).

Für das Web-Scraping gibt es hingegen noch keine umfassenden und stabilen Lösungen. Eine neuere Software auf diesem Gebiet ist PrivacyScore, welche sich selbst jedoch als Beta-Version und nicht vollständig zuverlässig beschreibt, vgl. dazu Welcome – PrivacyScore (o. D.). Daher bleibt das Bedürfnis nach einer zuverlässigen Lösung für das Web-Scraping bestehen.

1.1 Überblick über die Privacy Assistant Software

Der Privacy Assistant wurde im Rahmen von zwei vorangegangenen Studienprojekten gestartet. Das erste Projekt legte den Grundstein für das Tool, indem es grundlegende Funktionen wie das Scannen von Websites und die Erkennung persönlicher Daten implementierte. Das zweite Projekt war weniger erfolgreich und hinterließ das Tool in einem unaufgeräumten Zustand.

Der „Privacy Assistant“ ist eine Software, die dazu dient, Webseiten auf personenbezogene Daten zu überwachen. Ein Benutzer gibt dazu einen Uniform Resource Locator (URL) ein und der Privacy Assistant führt daraufhin in regelmäßigen Intervallen Scans dieser URL durch. In einem Scan durchsucht der Privacy Assistant die Hypertext Transfer Protocol (HTTP) Elemente nach E-Mail-Adressen, Telefonnummern, Personennamen und Bildern. Anschließend wird ein Bericht über die gefundenen Daten generiert. Dieser lässt sich auch als Excel-Datei herunterladen.

Der Privacy Assistant stellt daher ein wertvolles Hilfsmittel für Administratoren und Datenschutzbeauftragte dar. Er ermöglicht es, potenziell sensible Daten zu überwachen und sicherzustellen, dass Datenschutzbestimmungen eingehalten werden.

1.2 Ziel und Motivation des Projekts

Das Ziel dieses Studienprojekts ist es, die bestehende Software zu verstehen, weiterzuentwickeln und zu optimieren. So sollen neue Funktionen eingebaut werden, Fehler behoben werden und die Benutzererfahrung verbessert werden.

Ein langfristiges Ziel ist es, den Privacy Assistant als Open-Source-Projekt zu veröffentlichen, damit er auch außerhalb der Hochschule Esslingen verwendet und weiterentwickelt werden kann. Dies ermöglicht nicht nur eine breitere Nutzung, sondern eröffnet auch die Möglichkeit für externe Beiträge zur Weiterentwicklung und Verbesserung des Projekts.

1.3 Übersicht über den Bericht

Zunächst wird in diesem Bericht eine gründliche Analyse der Ausgangssituation durchgeführt. Es werden der aktuelle Zustand des Projekts und die identifizierten Probleme dargestellt, die es zu beheben gilt.

Anschließend erfolgt die Planung und Konzeption der Verbesserungen. In diesem Abschnitt wird dargelegt, wie die aufgedeckten Schwierigkeiten angegangen werden sollen und welche konkreten Schritte zur Umsetzung der Verbesserungen geplant sind.

Im darauffolgenden Kapitel wird die Umstrukturierung des Backends behandelt. Hier wird beschrieben, weshalb und wie das Backend neu strukturiert und optimiert wurde. Außerdem werden einige neuen Funktionalitäten beschrieben.

Die Entwicklung eines neuen Frontends wird danach vorgestellt. Dieser Abschnitt legt dar, wie das Frontend gestaltet und implementiert wurde, um eine benutzerfreundliche und professionelle Oberfläche zu gestalten. Außerdem wird die Integration der REST-API mit dem Frontend erläutert.

In der darauffolgenden Evaluation und im Vergleich werden die realisierten Verbesserungen gegenüber der Ausgangssituation gegenübergestellt.

Zum Abschluss folgt ein Fazit, in dem die Gesamtentwicklung des Projekts reflektiert wird. Auch werden zukünftige Entwicklungsmöglichkeiten aufzeigt.

2 Analyse der Ausgangssituation

In diesem Kapitel soll die Ausgangssituation analysiert werden. Dies legt den Grundstein für die weiteren Planungen und Entwicklungen.

2.1 Beschreibung der bestehenden Software und ihrer Funktionen

Im ersten Schritt werden Technologien und Tools betrachtet, die bei der bisherigen Entwicklung eingesetzt wurden.

Der Code des gesamten Projekts liegt im hochschuleigenen GitLab. Dieses wird für die Versionskontrolle und das Projektmanagement genutzt. Natürlich werden wir dies weiternutzen.

Als integrierte Entwicklungsumgebung (IDE) kam bislang PyCharm zum Einsatz. Für die Weiterentwicklung des Privacy Assistant haben wir uns dazu entschieden, zu Visual Studio Code (VS Code) zu wechseln. Dieses beherrschen wir bereits und es ist anders als PyCharm frei zugänglich, da es sich um eine Open-Source-Software handelt. Außerdem bietet es eine vielzahl von freizuganglichen und kostenlosen Extensions um die IDE zu erweitern und zu individualisieren.

Der Kern des Privacy Assistants ist in der Programmiersprache Python geschrieben. In Kombination mit Django, einem Web-Framework, das eine schnelle Entwicklung unterstützt, bildet Python das Fundament der Software. Das Frontend ist dabei mit Django Templates realisiert. Dies erlaubt eine einfache und schnelle Entwicklung für kleine Projekte.

Eine lange Liste an Python-Paketen wird benötigt, um die Software zu starten. Unter Anderem wird für das Scraping BeautifulSoup4 eingesetzt und Personen werden mit der Bibliothek spacy erkannt. Die Benutzeroberfläche bindet das Bootstrap-Framework ein, um einen einheitlichen Style zu schaffen.

Es gibt rund 30 Datenmodelle, die für die Funktion der Software notwendig sind. Hierfür wurde ein cleveres System entworfen, das wir zum besseren Verständnis visualisiert haben:

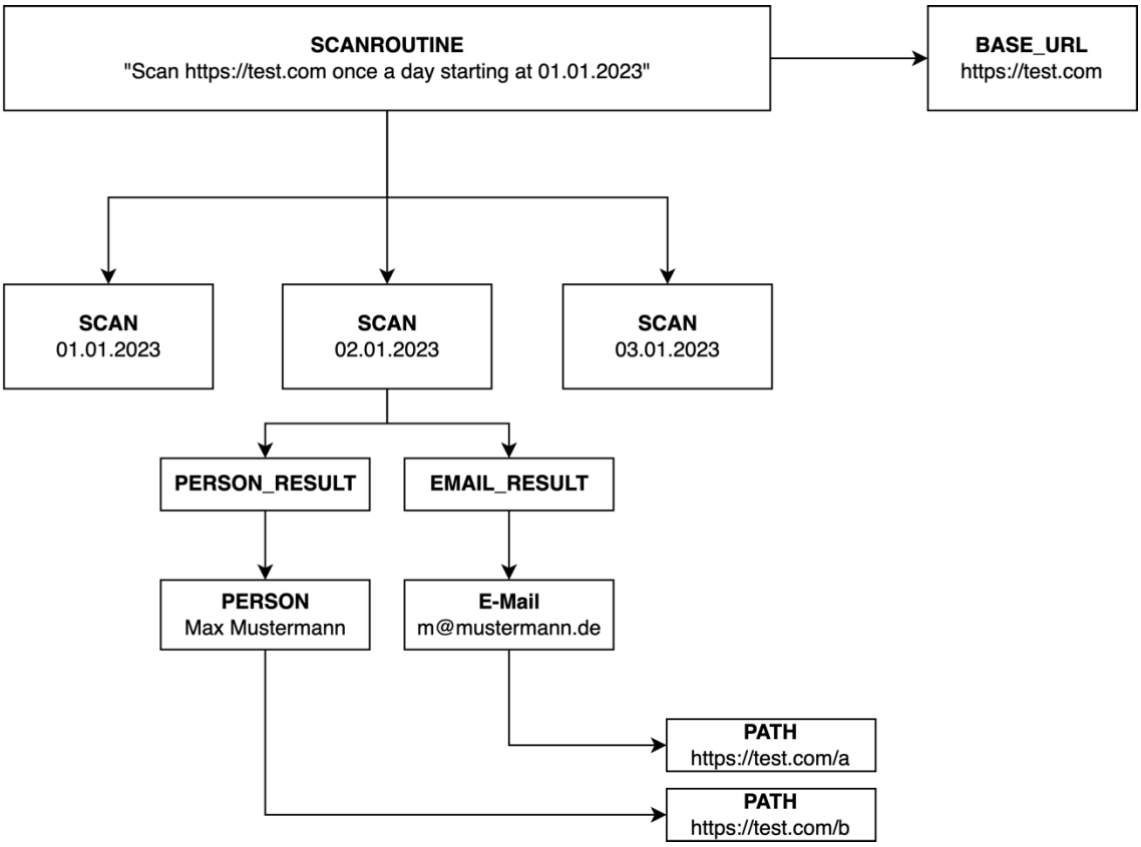

Abbildung 1: Zusammenhang der wichtigsten Datenmodelle

Auf oberster Ebene können vom Benutzer sogenannte Scanroutinen angelegt werden. Diese bestehen aus einer URL, einer Scanfrequenz und einem Startzeitpunkt. Die URL ist wiederrum ausgelagert in das Model BaseUrl. Abhängig von der Frequenz und dem Startzeitpunkt startet die Software den Scraper, der eine Seite scannt. Am Ende werden die Ergebnisse in die Datenbank geschrieben. Es entsteht ein neues Scan-Objekt, das durch die ID einer Scanroutine dieser zugeordnet ist. Gefundene Daten werden in die Tabellen PersonName, Email, PhoneNumber und Image geschrieben. Für jedes Ergebnis in jedem Scan wird ein Eintrag in der Tabelle PersonNameResult, bzw. EmailResult, PhoneNumberResult oder ImageResult angelegt. So wird möglichst wenig doppelt gespeichert. Die einzelnen Daten werden zudem einem Path verknpüft. So kann auch ausgewertet werden, auf welchen Unterseiten beispielweise eine Telefonnummer auftritt. Das können auch mehrere Pfade sein.

Sobald die Software gestartet ist, landet man auf einem leeren Dashboard:

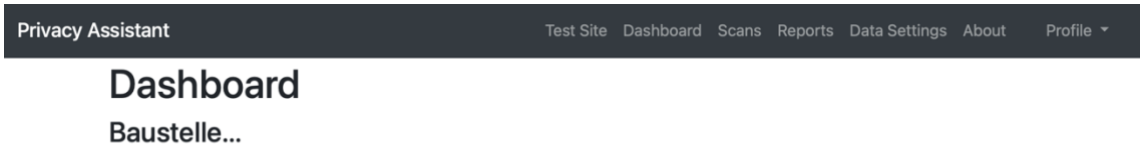

Abbildung 2: Altes Dashboard

Es gibt eine Test-Site, diese ist aber leer. Das Dashboard zeigt auch nur einen Text „Baustelle“ an. Unter Scans ist ein Formular für neue Scanroutinen implementiert, hier kann durch Eingabe einer URL, der Startzeit und der Anzahl der Scanwiederholungen, eine Scanroutine der Website angelegt werden, die unter der URL zu finden ist. Außerdem erhält der Nutzer einen Überblick über die bestehenden Scanroutinen, dabei kann er die Konfigurationen der Scanroutine sehen.

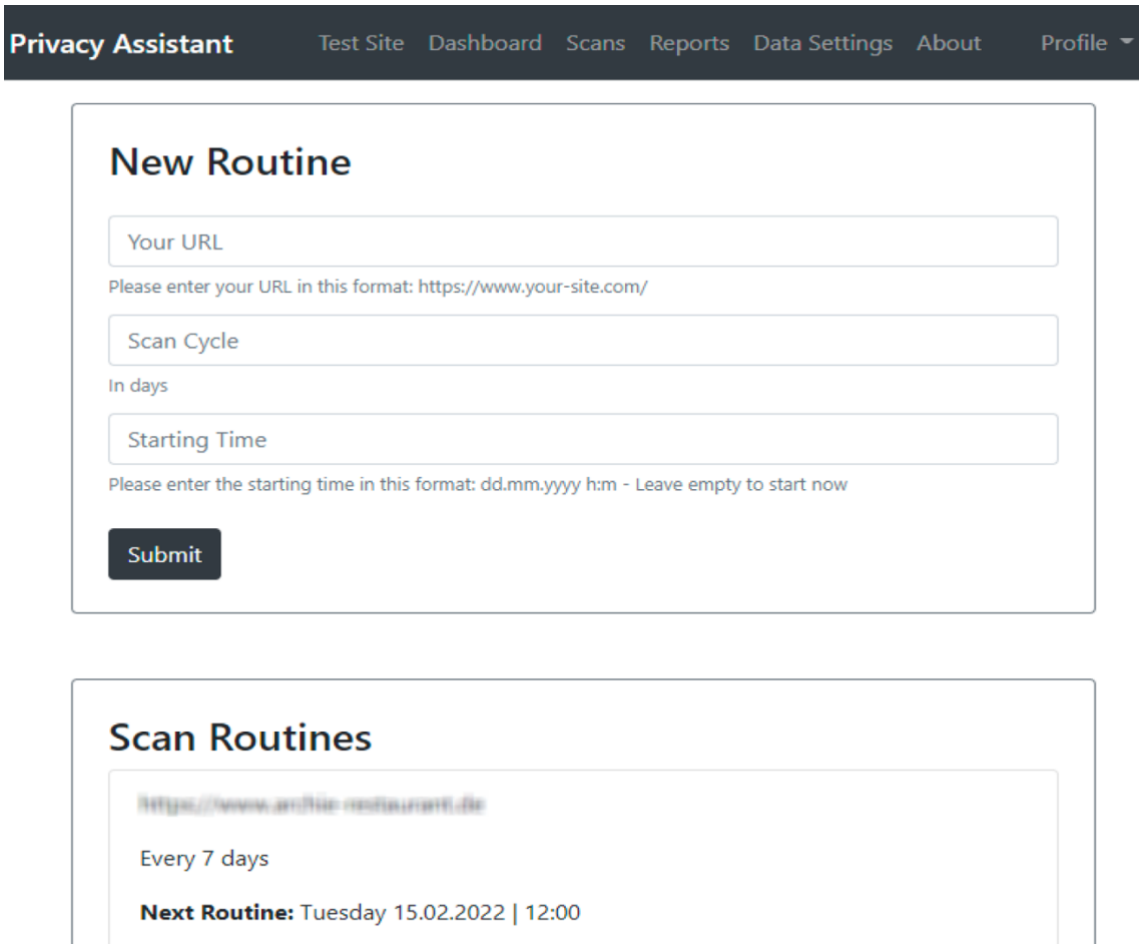

Abbildung 3: Alte Seite „Scans“

Durchgeführte Scans können im Abschnitt „Reports“ eingesehen werden. Die Umsetzung weiterer Seiten befindet sich noch in der Entwicklung.

Die grundlegenden Funktionen des Scans sind bereits einsatzfähig, aber viele Feinheiten und Details fehlen noch. Zum Beispiel sind einige Funktionen im Frontend bereits eingebaut, obwohl die entsprechenden Backend-Implementierungen noch fehlen. Dazu gehören die Berechnung von Statistiken, benutzerdefinierte Filter, Google-Filter und die Klassifizierung von Daten als fehlerhafte Ergebnisse.

2.2 Bewertung des vorhandenen Codes und der Struktur

Die Auswertung des bestehenden Codes sowie der Versuch, die Software zu starten, zeigten eine Reihe von Mängeln. Gemäß den dem Artikel „Top 10 mistakes that Django Developers make“, wie auf der Webseite von Toptal beschrieben, treffen neun der zehn zu, vgl. dazu Shurigin (2017).

Schon auf den ersten Blick ist die Qualität des Codes nicht zufriedenstellend. Der Code ist schwer zu lesen und zu verstehen. Funktionen und Views sind mehrere hundert Zeilen lang und es gibt nahezu keine Kommentare der vorherigen Entwickler im Code, was das Verständnis des Codes erheblich erschwert. Der folgende und einzige Kommentar einer sehr langen Funktion in der views.py zeigt dies sehr eindrücklich:

```
427     # Sorry to anyone who has to try to understand this
428     for image in images_db:
429         if any(image.image_url.image_url in image_from_grouped['image_url'] for image_from_grouped in
430                images_grouped):
431             [image_from_grouped['paths'].append({'path': image.image_url.path.path, 'id': image.image_url_id,
432                                                  'confidentiality': image.image_url.confidential}) for
433              image_from_grouped in images_grouped if image_from_grouped['image_url'] == image.image_url.image_url]
434         else:
435             image_group = {'image_url': image.image_url.image_url,
436                            'paths': [{'path': image.image_url.path.path, 'id': image.image_url_id,
437                                       'confidentiality': image.image_url.confidential}],
438                            'amount_found': image.amount_found,
439                            'id': image.image_url_id}
440             images_grouped.append(image_group)
441
442     for person in persons_db:
443         if any(person.name.name in person_from_grouped['person_name'] for person_from_grouped in persons_grouped):
444             [person_from_grouped['paths'].append(
445                 {'path': person.name.path.path, 'id': person.name_id, 'confidentiality': person.name.confidential})
446                 for
447                 person_from_grouped in persons_grouped if person_from_grouped['person_name'] == person.name.name]
448         else:
449             person_group = {'person_name': person.name.name,
450                             'paths': [{'path': person.name.path.path, 'id': person.name_id,
451                                        'confidentiality': person.name.confidential}],
452                             'amount_found': person.amount_found,
453                             'id': person.name_id}
454             persons_grouped.append(person_group)
455
456     for phoneNumber in phoneNumbers_db:
457         if any(phoneNumber.phone_num.phone_number in phoneNumber_from_grouped['phoneNumber'] for
458                phoneNumber_from_grouped in phoneNumbers_grouped):
459             [phoneNumber_from_grouped['paths'].append(
460                 {'path': phoneNumber.phone_num.path.path, 'id': phoneNumber.phone_num_id,
461                  'confidentiality': phoneNumber.phone_num.confidential}) for phoneNumber_from_grouped in
462                 phoneNumbers_grouped if
463                 phoneNumber_from_grouped['phoneNumber'] == phoneNumber.phone_num.phone_number]
464         else:
465             phoneNumber_group = {'phoneNumber': phoneNumber.phone_num.phone_number,
466                                  'paths': [{'path': phoneNumber.phone_num.path.path, 'id': phoneNumber.phone_num_id,
467                                             'confidentiality': phoneNumber.phone_num.confidential}],
468                                  'amount_found': phoneNumber.amount_found,
469                                  'id': phoneNumber.phone_num_id}
470             phoneNumbers_grouped.append(phoneNumber_group)
471
472     for email in emails_db:
```

Abbildung 4: Beispiel des bisherigen Codes

Der Code könnte in verständliche Unterfunktionen ausgelagert werden, sodass auch die fast identischen Wiederholungen im Code für verschiedene Datenarten überflüssig wären.

Eine Anleitung zum Start der Software ist nicht vorhanden. Es gibt zwar eine info.txt und eine commands.txt, doch beide Dateien enthalten wenige, unterschiedliche und auch gegensätzliche Informationen über die Ausführung und notwendige Installationen.

Auch die Abhängigkeiten der Software sind nicht klar definiert. Es fehlt eine requirements.txt-Datei, die eine genaue Aufzeichnung aller für das Projekt erforderlichen Python-Pakete und deren Versionen enthält. Mithilfe einer requirements.txt Datei könnten mit einem Befehl alle notwendigen Abhängigkeiten korrekt installiert werden. In diesem Fall führt es dazu, dass die Initialisierung und Ausführung der Software ohne die Datei sehr problematisch ist. Erst sehr mühsam konnten die benötigten Abhängigkeiten herausgefunden werden, die auch in den neuesten Versionen gar nicht mehr kompatibel mit der Software sind. Auf MacOS und Linux müssen außerdem noch weitere Developer Tools installiert werden, um dlib korrekt ausführen zu können.

Die Verwendung einer virtuellen Umgebung in Python (venv) hätte sehr geholfen. Dies ist eine auch von Toptal empfohlene Praxis, um die Paketabhängigkeiten und die Entwicklungsumgebung isoliert zu halten und Konflikte zwischen verschiedenen Projekten zu vermeiden, vgl. dazu Shurigin (2017).

Eine View ist eine Python-Funktion oder -Klasse, die eine HTTP-Anfrage verarbeitet und eine HTTP-Antwort zurückgibt. Die Views in der Software sind lang und nicht in kleine Methoden auf die Modelle heruntergebrochen. Auch dies ist einer der häufigsten Fehler laut Toptal.

Schließlich ist das Git-Repository unaufgeräumt und unorganisiert. Es gibt keine klare Struktur, keine Dokumentation und es fehlen notwendige Dateien. Außerdem gibt es keinen aktuellen main-Branch, der eine funktionierende Version der Software bereitstellt.

Ohne klare und umfassende Anweisungen ist es sehr schwierig, die Software korrekt zu installieren und zu starten, insbesondere für neue Entwickler oder Personen, die mit der Technologie nicht vertraut sind. Dies ist ein wichtiger Punkt, an welchem angesetzt, und verbessert werden soll.

3 Planung und Konzeption der Verbesserungen

Die identifizierten Probleme werden in diesem Kapitel herangezogen, um Verbesserungen und neue wichtige Features zu planen und konzipieren.

3.1 Ursprüngliche Zielsetzung

Noch bevor wir den Privacy Assistant erfolgreich installieren und ausführen konnten, waren wir bereits eifrig dabei, Ideen für neue Funktionen zu sammeln und sie als Tickets in GitLab einzupflegen.

Unsere ambitionierten Vorhaben reichten von der Benachrichtigung des Inhabers bei Datenschutzverletzungen bis hin zur Generierung von Berichten über diese Verletzungen und deren Häufigkeit, die beispielsweise wöchentlich an den Datenschutzbeauftragten gesendet werden könnten. Darüber hinaus wollten wir Funktionen zum Hinzufügen von URLs, zum Einstellen des Scanintervalls und zum Einsehen des Scanberichts implementierten. Fortschrittliche Funktionen wie Gesichtserkennung, etwa durch Amazon Rekognition, und Textanalyse mithilfe von Google Cloud Natural Language zur Identifizierung persönlicher Informationen waren ebenfalls Bestandteil unserer Ideensammlung. Nicht zuletzt wollten wir die Oberfläche attraktiver gestalten, den Verlauf von Datenschutzproblemen in einem Diagramm anzeigen und die Scan-Prozesse parallelisieren.

All diese Pläne basierten jedoch auf der Annahme, dass die bestehende Software sauber programmiert und gut dokumentiert wäre und dass wir daher in der Lage wären, schnell neue Funktionen einzubauen. Leider sollte sich diese Erwartung als zu optimistisch erweisen.

3.2 Startschwierigkeiten

Die ersten Hürden zeigten sich bereits bei der Installation des Privacy Assistants. Das Fehlen einer Installationsanleitung und einer requirements.txt-Datei machte diesen Prozess unerwartet schwierig. Wir mussten mühsam herausfinden, welche spezifischen Pakete benötigt wurden, um die Software auszuführen. Und selbst wenn wir diese identifiziert hatten, stießen wir auf Kompatibilitätsprobleme zwischen den aktuellen Versionen der erforderlichen Pakete und dem bestehenden Code. Darüber hinaus war es erforderlich, spezielle Developer Tools sowohl auf MacOS als auch auf Linux nachzuinstallieren.

Es schien, als würde jeder Schritt nur zu weiteren Fehlern führen. Auch der Versuch, die bisherigen Autoren des Projekts per E-Mail zu kontaktieren, war leider erfolglos und wir erhielten nie eine Antwort. Insgesamt hat es viel zu lange gedauert, die Software einfach nur zu starten, und das hat wertvolle Zeit gekostet. Und nach all der Arbeit war die Software nur gestartet und es war uns noch nicht klar, wie sie tatsächlich funktionierte.

Angesichts dieser enormen Anfangsschwierigkeiten mussten wir unsere Zielsetzung überdenken. Es wurde klar, dass die Software in ihrem derzeitigen Zustand unbrauchbar war. Sie war nicht für Open-Source geeignet und selbst kleine Änderungen waren extrem aufwendig und fehleranfällig. Es war für uns daher klar, dass umfangreiche Überarbeitungen und Strukturänderungen unternommen werden müssten, um den Privacy Assistant zu einem brauchbaren Projekt zu machen.

3.3 Neue Zielsetzung

Angesichts der unerwarteten Startschwierigkeiten haben wir unsere ursprüngliche Zielsetzung überarbeitet. Anstatt sofort mit dem Hinzufügen neuer Funktionen zu beginnen, erkannten wir die Notwendigkeit einer grundlegenden Überarbeitung.

Unser primäres neues Ziel war es nun, den Code professioneller zu gestalten. Dies beinhaltete die Trennung von Backend und Frontend, die über eine REST-API kommunizieren. Außerdem war es uns wichtig, eine gute Dokumentation anzulegen, um zukünftigen Entwicklern den Einstieg zu erleichtern und die Software nachhaltiger zu gestalten.

Nach Abschluss dieser grundlegenden Überarbeitungen könnten wir uns dann auf die Implementierung neuer Funktionen konzentrieren. Mit einer stabilen und gut strukturierten Codebasis sollte dies wesentlich einfacher und effizienter sein.

4 Umstrukturierung des Backends

Zur Aufteilung von Backend und Frontend gehören großflächige Änderungen im existierenden Backend des Privacy Assistants. Dabei musste fast der gesamte vorhandene Code grundlegend überarbeitet werden.

4.1 Gründe für die Umstrukturierung und Wahl einer REST-API

Die Neugestaltung des Backends und die Wahl einer REST-API wurden durch bestimmte Faktoren motiviert, die im Folgenden ausführlich beleuchtet werden.

4.1.1 Bedenken hinsichtlich der ursprünglichen Backend-Struktur

Der Privacy Assistant wurde ursprünglich unter Verwendung des Django Frameworks entwickelt, wobei das Frontend mit Django Templates, CSS und JavaScript realisiert wurde. Obwohl diese Struktur einen schnellen und einfachen Entwicklungsprozess ermöglicht, werden ihre Grenzen und Nachteile schnell deutlich.

Einer der bedeutendsten Nachteile ist die unzureichende Trennung von Backend und Frontend. Das Django-Templatesystem führt traditionell Backend und Frontend zusammen, was Wartung und Weiterentwicklung erschwert. Es ist damit nicht möglich, das Frontend unabhängig vom Backend zu entwickeln und zu aktualisieren.

Zudem wird durch die bestehende Struktur die Nutzung moderner JavaScript-Frameworks wie React, Vue.js oder Angular ausgeschlossen. Diese Frameworks sind optimal für Single-Page-Anwendungen ausgelegt und setzen eine REST-API auf der Backend-Seite voraus.

Darüber hinaus ist die Skalierbarkeit eingeschränkt. In der heutigen Softwarewelt der Microservices ist die Skalierbarkeit ein essenzieller Punkt der Softwarequalität. Bei einer Verknüpfung von Backend und Frontend in einem System, kann nicht nur ein Teilbereich skaliert werden. Im Falle des Privacy Assistants, einer Software, die umfangreiche Datenmengen sammelt und rechenintensive Backend-Prozesse besitzt, ist eine Trennung von Backend und Frontend sinnvoll, um eine effiziente, separate Skalierung zu ermöglichen.

Die Testbarkeit und Wartung leiden ebenfalls unter der bisherigen Architektur. Durch die enge Verknüpfung von Backend- und Frontend-Logik wird das Testen und Debuggen zur echten Herausforderung. Wie bereits in vorherigen Kapiteln beschrieben,

sind auch die vielen Startschwierigkeiten teilweise auf diese Architektur zurückzuführen.

Zuletzt ist die Integration neuer Funktionen in die bestehende Struktur unverhältnismäßig komplex, was bei einer Software wie dem Privacy Assistant problematisch ist.

Aus diesen Gründen wurde eine grundlegende Neustrukturierung notwendig.

4.1.2 Was ist eine REST-API

Eine REST (Representational State Transfer) API (Application Programming Interface) ist ein Architekturstil für Anwendungen. Sie definiert, wie Anwendungen miteinander kommunizieren und Informationen austauschen. REST basiert auf standardisierten Methoden und Statuscodes des HTTP-Protokolls und erlaubt den Zugriff auf Web-Ressourcen durch URLs, vgl. dazu RedHat (2023).

Bei einer REST-API handelt es sich in der Regel um eine Sammlung von Endpunkten oder URL-Routen, die unterschiedliche Funktionen erfüllen und spezifische Daten liefern. Jeder Endpunkt repräsentiert eine bestimmte Ressource, z.B. einen Benutzer, eine Scanroutine oder einen Scan beim Privacy Assistant. Auf diese Ressourcen kann dann mit den HTTP-Methoden zugegriffen werden – typischerweise GET zum Daten abrufen, POST um neue Daten hinzuzufügen, PUT/PATCH um bestehende Daten zu aktualisieren und DELETE um Daten zu löschen, vgl. dazu Häussler (2022).

Eine der wichtigsten Eigenschaften einer REST-API ist ihre Zustandslosigkeit. Das bedeutet, dass jede Anfrage unabhängig von anderen Anfragen ist und alle für die Ausführung notwendigen Informationen enthält. Dies erleichtert die Skalierung der Anwendung, da der Server keine Informationen über den Zustand der Clients speichern muss, vgl. dazu Häussler (2022).

Darüber hinaus sind REST-APIs sprachenunabhängig, was bedeutet, dass jede Anwendung, die HTTP-Anfragen senden und empfangen kann, mit einer REST-API kommunizieren kann. Dies eröffnet eine Vielzahl von Möglichkeiten für die Integration von Anwendungen in verschiedene Systeme und Plattformen. So kann das Backend weiterhin mit Django also in Python implementiert sein und das Frontend kann in in JavaScript mit dem Framework React entwickelt werden.

4.1.3 Vorteile einer REST-API

Die Einführung einer REST-API hat mehrere entscheidende Vorteile, die die zuvor genannten Bedenken in Bezug auf die ursprüngliche Backend-Struktur aus dem Weg räumen.

Die erste und vielleicht deutlichste Verbesserung ist die klare Trennung von Backend und Frontend. Durch die Einführung einer REST-API können beide Komponenten unabhängig voneinander entwickelt und aktualisiert werden, was die Modularität fördert und die Wiederverwendbarkeit des Codes erheblich verbessert. Im Betracht auf das Ziel, den Privacy Assistant später als Open-Source-Software zu veröffentlichen, ist dies ein wichtiger Schritt.

Zudem ermöglicht die REST-API die nahtlose Integration mit modernen JavaScript-Frameworks wie React, Vue.js oder Angular. Diese sind für Single-Page-Anwendungen optimiert und setzen eine REST-API auf der Backend-Seite voraus, was die bisherige Architektur nicht unterstützt. Ein Frontend mit einem dieser Frameworks wertet die Benutzererfahrung stark auf und lässt die Software mit gleicher Funktionalität professioneller wirken.

Ein weiterer Vorteil ist die verbesserte Skalierbarkeit. Mit einer REST-API kann das Backend unabhängig vom Frontend skaliert werden, was für eine datenintensive und rechenaufwändige Anwendung wie den Privacy Assistant von entscheidender Bedeutung ist, vgl. dazu Häussler (2022).

Darüber hinaus verbessert die Trennung von Backend und Frontend die Testbarkeit und Wartbarkeit der Software. Da die einzelnen Komponenten der Anwendung isoliert getestet und gewartet werden können, erhöht sich die Codequalität und die Fehlerbehebung wird vereinfacht. Dies Erleichtert die Weiterentwicklung und die Fehlersuche bei Problemen.

Mit einer REST-API wird auch die Integration neuer Funktionen in die Software erheblich erleichtert. Sie bietet eine klare und standardisierte Struktur für das Hinzufügen neuer Endpunkte und Funktionen.

Darüber hinaus ist die REST-API sprachenunabhängig und kann von jeder Anwendung genutzt werden, die HTTP-Anfragen senden und empfangen kann. Dies eröffnet eine größere Flexibilität und Offenheit gegenüber verschiedenen Technologien und Entwicklungsansätzen. So kann beispielsweise später auch einfach eine App für Mobilgeräte entwickelt werden, die die REST-API nutzt, vgl. dazu Häussler (2022).

Zusammengefasst stellen diese Vorteile die Einführung einer REST-API als sinnvolle und zukunftsorientierte Lösung für die Weiterentwicklung des Privacy Assistants dar.

4.2 Planung und Design der neuen Backend-Struktur

Nach der Entscheidung, das Backend umzustrukturieren und auf eine REST-API umzustellen, steht die Planung des neuen Designs an.

4.2.1 Erstellung der API-Spezifikation

Die Ausarbeitung einer API-Spezifikation bildet den Grundstein für die anschließende Implementierung. Dabei haben wir gemeinsam überlegt, welche Ressourcen und Endpunkte sinnvoll sind, wie das JSON-Format der Server-Antworten gestaltet sein sollte und welche HTTP-Statuscodes unter welchen Umständen zurückgegeben werden sollten. Diese Entscheidungen wurden in einer Notiz im Markdown-Format festgehalten, um eine strukturierte Dokumentation zu schaffen.

In dem folgenden Schritt haben wir versucht, die festgehaltene Spezifikation in einem standardisierten OpenAPI-Format zu dokumentieren. Mithilfe des Tools Swagger kann die API-Spezifikation im OpenAPI-Format übersichtlich und interaktiv als Website dargestellt werden.

In Anbetracht der Schwierigkeiten haben wir uns schließlich dazu entschlossen, stattdessen direkt mit der Implementierung der API zu beginnen. Die vollständige Spezifikation im OpenAPI-Format und die Darstellung mit Swagger ließ sich daraufhin automatisch aus dem Python-Code generieren.

4.2.2 Planung der notwendigen Services und Funktionen

Im Zuge der Umstrukturierung des Backends wurde besonderer Wert auf Modularität und Verständlichkeit gelegt. Dazu war eine sorgfältige Planung der benötigten Services und Funktionen vonnöten.

Die Backend-Struktur wurde in verschiedene Apps aufgeteilt, um die unterschiedlichen Funktionsbereiche klar voneinander abzugrenzen und so eine übersichtlichere Struktur zu schaffen. In der bisherigen Struktur war alles in einer einzigen App. Dadurch wurden Dateien sehr groß und unübersichtlich. Nun gibt es vier verschiedene Apps innerhalb des Projekts:

- **Scanroutine**: zuständig für Scanroutinen, BaseUrls und benutzerdefinierte Datenfilter
- **Scan**: zuständig für einzelne Scans innerhalb einer Scanroutine sowie deren Ergebnisse
- **Dashboard**: zuständig für die Berechnung von Statistiken, die auf dem Dashboard des Frontends angezeigt werden
- **Scraper**: zuständig für das Scrapen von Websites

Die meisten Funktionen aus dem Scraper wurden von der bisherigen Software übernommen. Die anderen drei Apps wurden komplett neu entwickelt, hier wurden nur die Datenmodelle übernommen. Anderenfalls hätte die Logik des Scrapers auch neu programmiert werden müssen.

Ein wesentlicher Punkt unserer Planung war die Wiederverwendbarkeit von Funktionen. Wir legten Wert darauf, Code-Blöcke nicht unnötig zu duplizieren, sondern sie auszulagern und wiederverwendbar zu gestalten. Dieses saubere Coden trägt zur Verständlichkeit der Software bei und erleichtert zukünftige Anpassungen und Erweiterungen. Dabei galt stets das Prinzip, Code nicht nur für die Maschine, sondern auch für Menschen lesbar zu gestalten, um eine nachhaltige und effiziente Weiterentwicklung zu ermöglichen.

4.3 Containerisierung mit Docker

Die Containerisierung der Anwendung und damit auch des Backends mittels Docker war spätestens nach der Vorlesung „Verteilte Systeme“ im vergangenen Semester von Beginn an ein zentrales Element und Ziel in unserer Entwicklung.

4.3.1 Vorteile der Verwendung von Docker

Docker bietet eine einfache und effiziente Möglichkeit, Anwendungen zu containerisieren und auszuführen, unabhängig von der zugrundeliegenden Infrastruktur. Dies erlaubt es, die gleiche Umgebung über alle Phasen der Softwareentwicklung hinweg zu nutzen – von der Entwicklung über das Testen bis hin zur Produktion. Doch das ist nicht alles, die Vorteile der Verwendung von Docker sind vielfältig. Erstens sorgt Docker für eine standardisierte Entwicklungsumgebung, in der alle notwendigen Abhängigkeiten und Konfigurationen bereits enthalten sind. Das bedeutet, dass neue Entwickler schnell und einfach in das Projekt einsteigen können, ohne lange mit der Einrichtung ihrer Entwicklungsumgebung verbringen zu müssen (also nicht so, wie wir es mussten…!). Zweitens erleichtert Docker die Bereitstellung und Skalierung der Anwendung. Docker-Container können auf jedem Host mit einer Docker-Installation ausgeführt werden, unabhängig vom Betriebssystem. Dies erleichtert die Bereitstellung der Anwendung auf verschiedenen Plattformen und in verschiedenen Umgebungen, vgl. dazu Joos (2019).

4.3.2 Containerisierung der Django-Anwendung

Zur Containerisierung unserer Django-Anwendung haben wir ein Dockerfile erstellt, das die notwendigen Befehle zur Erstellung des Docker-Images enthält. Dieses Dockerfile definiert das Basis-Image, in unserem Fall ein Python-Image, installiert die benötigten Abhängigkeiten und kopiert den Anwendungscode in den Container. Darüber hinaus legt das Dockerfile fest, welcher Befehl beim Start des Containers ausgeführt werden soll, in unserem Fall der Befehl zur Ausführung des Django-Servers. So gibt es keine Startschwierigkeiten mehr und auch jemand ohne eine aktive Python-

Installation oder Wissen über virtuelle Umgebungen in Python, kann die Anwendungen mit einem einzigen Befehl starten:

docker run -d -p 8000:8000 maximilianfink/privacyassistant-backend:v1.0

4.4 Implementierung der REST-API

Nach der intensiven Planung konnte die Implementierung der REST-API beginnen.

4.4.1 Vorgehen

Die Implementierung erforderte ein systematisches Vorgehen, um sicherzustellen, dass alle notwendigen Funktionen und Services korrekt umgesetzt wurden.

Der Prozess begann mit dem Verständnis des bestehenden Codes. Eine gründliche Analyse war notwendig, um die bestehenden Funktionen und Datenstrukturen zu verstehen, die in die neue Architektur übernommen werden sollten. Dies hat sich als sehr schwierig herausgestellt, da der Code sehr komplex, durcheinander, unstrukturiert und unkommentiert war. Dadurch hat dieser Schritt viel mehr Zeit als geplant benötigt und es wurde erneut deutlich, wie wichtig ein gut strukturierter und dokumentierter Code ist, damit ein Softwareprojekt langfristig Erfolg hat.

Nach dem Verständnis des ursprünglichen Codes wurde dieser refaktorisiert. Dazu gehörte insbesondere das Kürzen langer Methoden durch das Erstellen spezifischer Unterfunktionen, um die Lesbarkeit und Wartbarkeit des Codes zu verbessern. So konnten Methoden im Umfang von rund 300 Zeilen auf weniger als 50 Zeilen reduziert werden.

Anschließend wurde ein komplett neues Django-Projekt aufgesetzt. Das brachte uns den Vorteil, sauber von vorne beginnen zu können und nicht schon mit unnötigem und kaputtem Code anzufangen. Ebenso konnten so Fehler ausgegrenzt werden, die beim Start des alten Projekts immer und immer wieder auftraten. Im neuen Projekt wurden die notwendigen Verzeichnisse und Dateien initialisiert sowie die grundlegende Django-Umgebung eingerichtet. Zu diesem Zeitpunkt wurden auch die Datenmodelle aus dem alten Code in das neue Projekt kopiert, da sie die Grundlage für die neue API bildeten und für die Funktionalität des Scrapers notwendig waren. Diese wurden jedoch aufgeteilt in die vier verschiedenen Apps, um die einzelnen Apps in dem Projekt von Beginn an übersichtlich zu halten.

Ein wichtiger Schritt war das Schreiben der Serializer. Diese sind im Django REST Framework verantwortlich für die Umwandlung von komplexen Datentypen, wie Django-Modellen, in Python-native Datentypen. Diese Python-nativen Datentypen können dann leicht in JSON umgewandelt werden, um sie über die API zu versenden. Die Serializer waren ein guter Anfang, mussten jedoch später mehrfach umgeschrieben

werden, um spezielle Anwendungsfälle und Verknüpfungen der Datentypen in der API zu modellieren.

Anschließend wurden die Modelle in der admin.py registriert und entsprechende __str__() Funktionen wurden geschrieben, damit die Daten auch im Admin-Interface von Django unter /admin sauber administrierbar waren.

Im nächsten Schritt wurde die urls.py-Datei angelegt. Diese definiert alle Endpunkte der API und verbindet sie mit den entsprechenden Views. Die entsprechenden Routen wurden wie beschrieben bereits vorher definiert.

Die Erstellung der views.py Datei war ein entscheidender Teil der Implementierung der REST-API. Im Django REST Framework sind die Views dafür verantwortlich, Anfragen zu bearbeiten und die entsprechenden Antworten zu liefern. Bei der Erstellung der Views für die API wurde ein besonderer Schwerpunkt auf die Verwendung moderner Django-Praktiken wie Generic Views und Vererbung gelegt. Generische Views sind eine Funktion in Django, die es ermöglicht, wiederverwendbaren Code für übliche Create, Read, Update, Delete (CRUD) Operationen zu erstellen. Anstatt für jede einzelne CRUD-Operation eine spezifische View-Funktion zu schreiben, können generische Views verwendet werden, die diese Operationen bereits implementiert haben. Dadurch wird der Code sauberer, übersichtlicher und einfacher zu warten. In Verbindung mit den generischen Views wurde auch die Vererbung von get_queryset verwendet. In Django ist die Methode get_queryset dafür verantwortlich, die Datensätze abzurufen, die von einer View bearbeitet werden. So konnte gefiltert werden, dass nur Datensätze abgerufen werden, die der User-ID des aktuell eingeloggten Users entsprechen. Durch die Überschreibung dieser Methode in einer Basisklasse und die Vererbung dieser Basisklasse in den einzelnen Views konnten kurze Methoden und ein hohes Maß an Code-Wiederverwendung erreicht werden.

Die Scraper-Dateien, die bereits in der alten Software vorhanden waren, wurden analysiert und was benötigt war in die neue Scraper-App kopiert sowie deren Abhängigkeiten entsprechend angepasst. Dieser Schritt erforderte eine sorgfältige Prüfung und Anpassung, um sicherzustellen, dass der Scraper korrekt mit der restlichen Anwendung interagiert. Da die Scraping-Funktionen sehr komplex und unübersichtlich programmiert sind, war das auch ein zeitintensiver Teil der Implementierung.

Schließlich, nach all diesen Schritten, kam der Prozess des Bugfixing. In diesem Schritt wurden alle Routen ausgiebig getestet und die Probleme behoben.

4.4.2 Sicherheit und Authentifizierung via Token

Sicherheit und Authentifizierung sind zwei fundamentale Aspekte jeder modernen Webanwendung. Besonders wenn es sich um personenbezogene Daten wie in dieser

Anwendung handelt ist es äußerst wichtig, dass man keine fremden Daten einsehen, manipulieren oder löschen kann.

Für den Privacy Assistant wurde die Token-Authentifizierung gewählt, um diese Aspekte zu handhaben. Token-Authentifizierung ist ein Prozess, bei dem der Server einen Token an den Client ausgibt, nachdem der Client seine Identität erfolgreich überprüft hat. Dieser Token wird dann bei jedem nachfolgenden Request vom Client an den Server gesendet, um seine Identität zu bestätigen.

Die Token-Authentifizierung hat mehrere Vorteile, die sie zu einer idealen Wahl für den Privacy Assistant machen. Erstens muss der Server keine Information über den Client speichern, da alle notwendigen Informationen im Token enthalten sind. Dieses Prinzip wird auch als „stateless“ bezeichnet und ist ein zentrales Element des REST-Prinzips, das wir verfolgen. Zweitens erlaubt Token-Authentifizierung eine einfache Implementierung von Rechten und Berechtigungen. Jeder Token kann Informationen über die Rechte des Benutzers enthalten, so dass der Server leicht überprüfen kann, ob ein Client berechtigt ist, eine bestimmte Aktion auszuführen, vgl. dazu Entrust (o. D.).

Die Implementierung von Token-Authentifizierung in Django REST Framework ist einfach. Hierzu setzen wir auf die Pakete „djangorestframework“, „dj-rest-auth“ und „django-allauth“. Nach der erfolgreichen Anmeldung eines Benutzers erzeugt der Server einen einzigartigen Token, speichert ihn in der Datenbank in der „Token“-Tabelle und sendet ihn an den Client. Der Client speichert diesen Token und sendet ihn in dem Header „Authorization“ bei jedem nachfolgenden Request an den Server. Der Server liest den Token aus dem Header, verifiziert ihn und bestimmt die Identität des Benutzers und seine Berechtigungen.

4.4.3 API-Endpunkte

Im Laufe des Projekts sind einige Endpunkte aufgekommen. Schlussendlich sind es rund 30 Endpunkte, die die benötigten Funktionalitäten bereitstellen. Diese spielen in großen Teilen unsere Unterteilung in die vier logischen Apps in der Software wieder.

Unter GET /api/schema/ lässt sich ein OpenAPI File der Schnittstelle herunterladen:

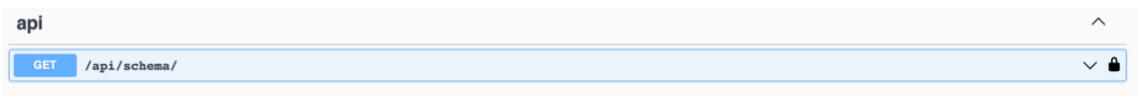

Abbildung 5: API Endpukt

Unter /login, /logout, /password und /register stehen verschiedene Endpunkte des Authentifizierungsprozesses zur Verfügung:

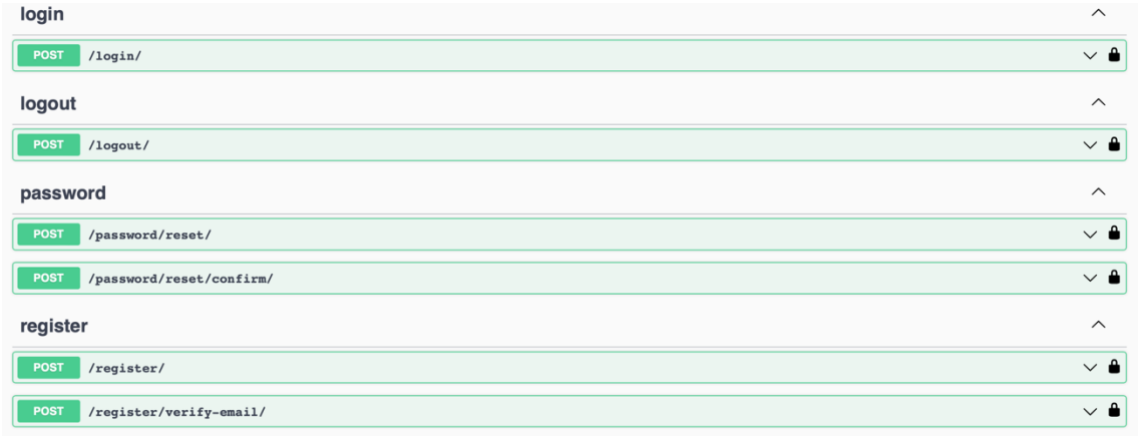

Abbildung 6: Endpunkte für die Authentifizierung

Unter /statistics können Statistiken für den aktuell eingeloggten Benutzer abgerufen werden:

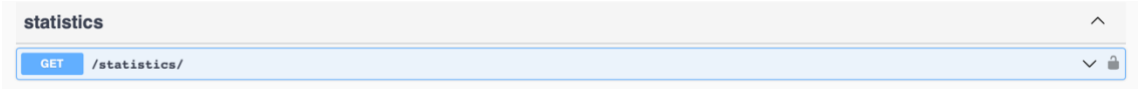

Abbildung 7: Endpunkt für Statistiken

Unter /scan gibt es viele Endpunkte, um einzelne Scans abzurufen, zu löschen oder Daten zu klassifizieren:

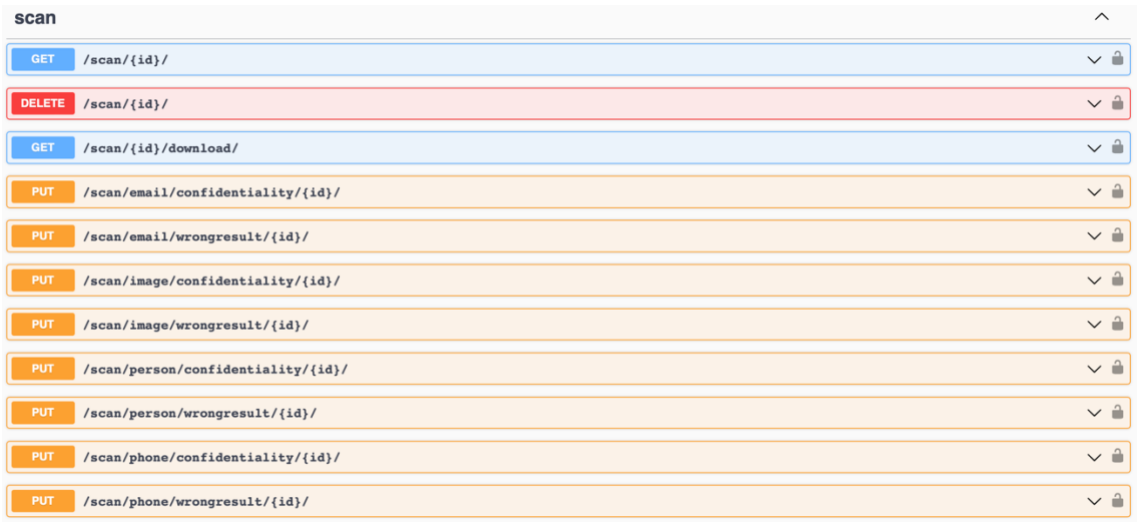

Abbildung 8: /scan Endpunkt

Unter /scanroutine können alle Operationen durchgeführt werden, die mit der Scanroutine zu tun haben:

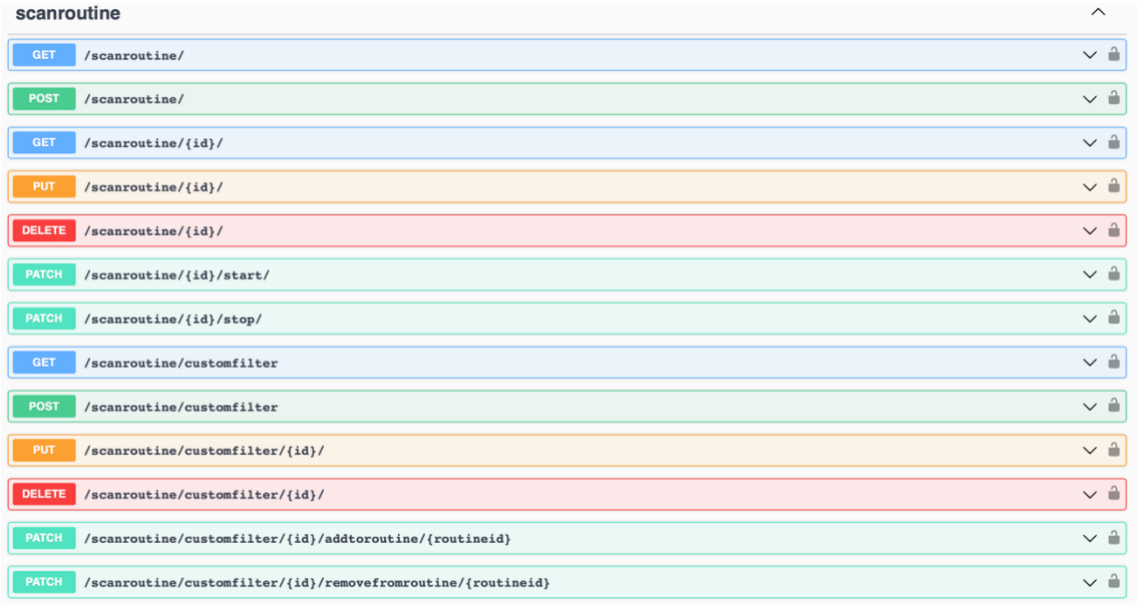

Abbildung 9: /scanroutine Endpunkt

In Summe lässt sich mit diesen Routen alles abbilden, was für das Frontend des Privacy Assistants notwendig ist.

4.5 Verbesserung der Codequalität

Ein großes Ziel von uns war die Verbesserung der Codequalität. Im Folgenden werden einige dieser Verbesserungen am Code erläutert, die wir am Backend vorgenommen haben.

Zunächst haben wir uns auf das Refactoring und die Verbesserung des bestehenden Codes konzentriert. Eine große Rolle spielte dabei das Verkürzen und Vereinfachen von Methoden, um diese schnell zu verstehen, besser zu testen und die Fehleranfälligkeit zu reduzieren. Wir haben die Prinzipien „Don’t repeat yourself“ (DRY) und „Keep it simple, stupid“ (KISS) verfolgt. Dies macht den Code nicht nur einfacher zu lesen und zu warten, sondern erleichtert auch das Erstellen und Durchführen von Tests, vgl. dazu generic.de (o. D.).

Um die Verständlichkeit des Codes weiter zu erhöhen, haben wir uns bemüht, sinnvolle Bezeichnungen für Variablen, Funktionen und Klassen zu verwenden. Wir haben auch den Code ausführlich kommentiert, um nächsten Entwicklern zu helfen, den Zweck und die Funktionsweise des Codes besser zu verstehen. Nur so können diese später den Privacy Assistant weiterentwickeln.

Wir haben Serializer in Django genutzt, um komplexe Datenstrukturen in Python in native Datentypen umzuwandeln. Dies hat dazu beigetragen, die Datenübertragung zwischen dem Backend und dem Frontend zu vereinfachen und zu standardisieren.

Für eine bessere Dokumentation der Endpunkte haben wir die @extend_schema-Dekoration von DRF-Spectacular verwendet. Diese erlauben es, zusätzliche Informationen zu den Endpunkten hinzuzufügen und erzeugen eine saubere und ausführliche Swagger-Dokumentation, die für die Entwicklung und das Debugging sehr hilfreich ist.

Die Aufteilung des Projekts in mehrere Django-Apps wie bereits erwähnt hat ebenfalls zu einer besseren Strukturierung des Codes beigetragen. Statt alles in einem Modul zu haben, haben wir nun verschiedene Module, die jeweils eine bestimmte Funktion erfüllen.

Zusätzlich haben wir auch die Nutzung einer requirements.txt-Datei eingeführt. Vorher gab es diese Datei nicht in unserem Projekt. Mit der requirements.txt-Datei können wir alle Python-Pakete, die unser Projekt benötigt, an einem zentralen Ort auflisten. Dies erleichtert die Installation der benötigten Pakete erheblich, da andere Entwickler oder Systeme einfach mit dem Befehl

pip3 install -r requirements.txt

alle notwendigen Abhängigkeiten installieren können. Darüber hinaus sorgt es für eine klarere Übersicht über die benötigten externen Ressourcen und macht die Projektumgebung reproduzierbar.

Dies war besonders relevant im Zusammenhang mit dem Einsatz von virtuellen Umgebungen. In der Python-Entwicklung sind virtuelle Umgebungen sehr hilfreich, um die Paketabhängigkeiten für verschiedene Projekte getrennt zu halten. Indem wir eine virtuelle Umgebung für unser Projekt erstellen, stellen wir sicher, dass die Installation oder Aktualisierung von Paketen für dieses Projekt keine Auswirkungen auf andere Python-Projekte auf demselben System hat. Darüber hinaus erlaubt es uns, genau zu kontrollieren, welche Versionen der Pakete wir für unser Projekt verwenden, was zur Vermeidung von Kompatibilitätsproblemen beiträgt. Dies war zu Beginn eine große Startschwierigkeit bei uns, die nun nicht mehr vorkommen kann.

Auch die Integration von Docker hat den Startprozess des Projekts stark vereinfacht und ist ein großer Schritt in Richtung eines produktionsreifen Systems. Mit Docker können wir sicherstellen, dass die Umgebung, in der das Backend läuft, überall gleich ist, unabhängig vom tatsächlichen Betriebssystem und den installierten Paketen. Dies hilft, „It works on my machine“-Probleme zu vermeiden.

Schließlich haben wir auch begonnen, erste Tests zu implementieren. Tests sind ein wesentlicher Bestandteil der Softwareentwicklung und helfen dabei, die Korrektheit des Codes zu gewährleisten und Regressionen zu vermeiden.

Zusammengefasst haben alle der oben genannten Verbesserungen sehr gut dazu beigetragen, die Codequalität erheblich zu steigern, die Entwicklung und das Testing zu

erleichtern, und haben unser Projekt näher an eine veröffentlichungsreife Software gebracht.

4.6 Testen und Validieren der neuen Backend-Struktur

Sobald die Implementierung von Funktionen abgeschlossen war, stand das Testen an.

4.6.1 Manuelle Tests mit Swagger

Unter Verwendung von Swagger haben wir umfangreiche manuelle Tests unserer API durchgeführt. Swagger ist ein sehr mächtiges Werkzeug, das uns eine interaktive Dokumentation unserer REST-API bietet. Es ermöglicht, die Funktionen der API direkt im Browser zu testen, was bei der Fehlersuche und -behebung sehr nützlich ist. Auch ist diese Möglichkeit wesentlich schneller, als automatisierte Tests zu schreiben, vgl. dazu Ionos (2022).

Wir haben jeden einzelnen Endpunkt unserer API intensiv mit Swagger getestet. Dabei haben wir verschiedene Anwendungsfälle berücksichtigt, um sicherzustellen, dass die API unter verschiedenen Bedingungen korrekt funktioniert. So haben wir zum Beispiel leere Eingaben, gültige Eingaben, ungültige Eingaben und falsch authentifizierte Eingaben abgesendet und überprüft, ob die entsprechenden Endpunkte die erwarteten Antworten liefern. Durch die unmittelbare Rückmeldung, die wir bei der Nutzung von Swagger erhalten, konnten wir Fehler schnell identifizieren und beheben. Swagger hat sich als unverzichtbares Werkzeug für die Durchführung der API-Tests erwiesen.

4.6.2 Automatisierte Tests

Automatisierte Tests bringen eine Reihe von Vorteilen mit sich. Sie sind effizient, lassen sich beliebig oft und jederzeit ausführen, sind zuverlässig und lassen sich auch in Continuous Integration und Continuous Delivery (CI/CD) Pipelines integrieren.

Das Django Framework bietet die Möglichkeit automatisierter Tests. Daher war unser ursprüngliches Ziel, eine hohe Testabdeckung zu erreichen. Aufgrund der Komplexität der Authentifizierung und der stark verknüpften Datenmodelle erwies sich die Erstellung automatisierter Tests allerdings als sehr aufwändig. In Anbetracht dessen haben wir uns dazu entschieden, einen repräsentativen automatisierten Test zu entwickeln. Dieser Test diente als beispielhafter Ausgangspunkt für mögliche weitere automatisierte Tests in der Zukunft.

Unsere Hauptaufmerksamkeit lag auf dem manuellen Testen. Während automatisierte Tests für die zukünftige Entwicklung wichtig sein können, war es in dieser Phase des Projekts sinnvoller, unsere Zeit auf das manuelle Testen zu konzentrieren, da uns dies Zeit ersparte.

4.7 Dokumentation des Backends

Die Dokumentation spielt eine fundamentale Rolle in der Softwareentwicklung. Sie der hilft weiteren Entwicklern und Anwendern, die Struktur und Funktionalität des Codes zu verstehen. Leider war in unserem Fall zu Beginn keine Dokumentation vorhanden, was das Verständnis des Backends und dessen Funktionalitäten erheblich erschwerte. Code, der nicht verstanden wird, ist praktisch nutzlos, denn er kann nicht gewartet, behoben oder weiterentwickelt werden. Daher haben wir uns entschlossen, dieses Problem anzugehen und eine gründliche Dokumentation zu erstellen. Eine gute Dokumentation stellt sicher, dass der Code auch von anderen Entwicklern verstanden und effizient genutzt werden kann.

Unsere Dokumentation umfasst verschiedene Aspekte. Sie beginnt bei der grundlegenden Beschreibung der Anwendung und ihrer Funktionen und geht bis hin zu spezifischen Details wie der Funktion von einzelnen Code-Teilen. Sie enthält auch Informationen über die Installation und Konfiguration der Anwendung, die genutzten Technologien und Bibliotheken sowie Anleitungen zur Weiterentwicklung.

4.7.1 Swagger Dokumentation

Ein wesentlicher Bestandteil unserer Backend-Dokumentation ist die Nutzung von Swagger, ein äußerst nützliches Tool zur Erstellung interaktiver API-Dokumentationen. Dieses wurde bereits im Kapitel zu manuellem Testen beschrieben.

Mit @extend_schema können wir die Eigenschaften jeder Route, wie HTTP-Methoden, Anforderungs- und Antwortmodelle, Statuscodes und mehr, präzise spezifizieren. Damit wird automatisch eine ausführliche und interaktive Dokumentation generiert, die all diese Informationen beinhaltet. Hier ist ein Abschnitt dieser Doku zu sehen:

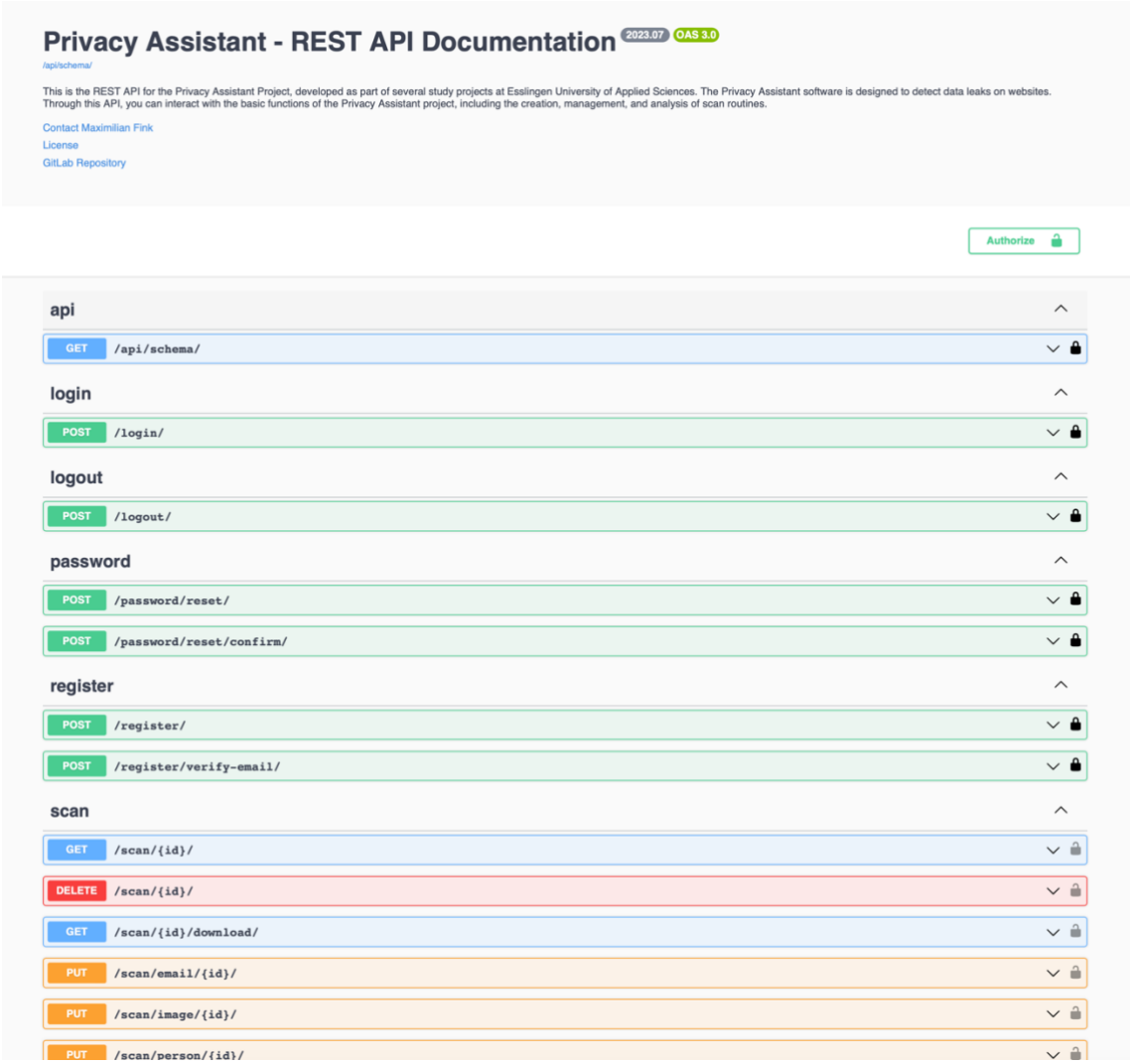

Abbildung 10: Übersicht Swagger-Dokumentation

Ein signifikanter Vorteil der Verwendung von Swagger ist seine Interaktivität. Mit der generierten Dokumentation können Entwickler die API direkt testen, indem sie Anfragen an die Endpunkte senden und die tatsächlichen Antworten der API sehen. Dies erleichtert nicht nur das Verständnis der API-Funktionalitäten, sondern hilft auch, mögliche Fehler oder Unstimmigkeiten zu identifizieren. Das sieht bei der Route GET /scanroutine/{id}/ beispielsweise so aus:

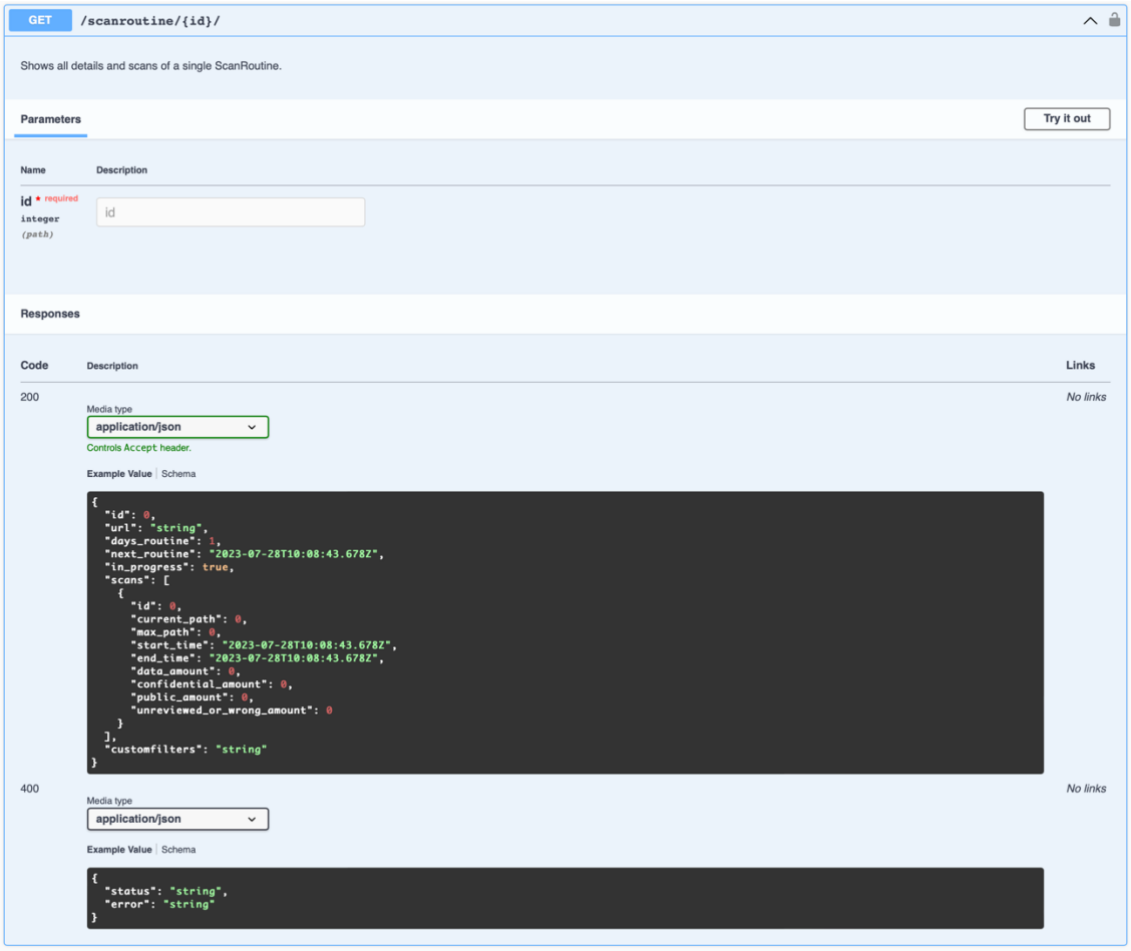

Abbildung 11: Interaktivität von Swagger am Beispiel der Route GET /scanroutine/{id}

Somit trägt die Nutzung von Swagger und @extend_schema erheblich zur Verbesserung der Qualität unserer Backend-Dokumentation bei.

4.7.2 Selbsterklärender Code

Um den Dokumentationsaspekt auch im Code selbst abzubilden, legen wir großen Wert auf selbsterklärenden Code. Die Erstellung kurzer und spezifischer Funktionen half dabei, die Komplexität zu reduzieren. Jede Funktion hat einen klar definierten Aufgabenbereich, was ihr Verständnis erleichtert. Gute Benennungspraktiken wurden für Variablen, Funktionen und Klassen angewendet. Ergänzend zum selbsterklärenden Code haben wir Kommentare im Code eingebaut. Schließlich haben wir Vererbung effektiv eingesetzt, um Wiederholungen zu vermeiden und die Code-Struktur zu optimieren. Durch die Definition allgemeiner Funktionalitäten in Basisklassen und deren spezifische Erweiterung in abgeleiteten Klassen, konnten wir eine hohe Konsistenz und Lesbarkeit erreichen.

4.7.3 README.md

Ein wichtiger Teil der Projektdokumentation ist eine README.md-Datei, die als Einführung und schnelle Referenz für das Projekt dient. Wir haben diese Datei angelegt und mit wichtigen Informationen rund um das Projekt gefüllt. Hier finden sich nicht nur allgemeine Informationen über das Projekt, sondern auch spezifische Anweisungen, wie man es einrichtet und startet.

Die README.md-Datei ist direkt im Hauptverzeichnis des GitLab-Repositorys und des Backend-Verzeichnisses, sowie des Frontend-Verzeichnisses platziert, sodass sie sofort sichtbar ist, wenn man das Repository besucht. Dies bietet einen guten Startpunkt für alle, die sich mit dem Projekt vertraut machen wollen.

4.8 Skalierbarkeit des Backends

In der heutigen, stark vernetzten und digitalen Welt ist die Skalierbarkeit von Software eine wesentliche Anforderung. Eine hoch skalierbare Anwendung kann sich flexibel an verändernde Lastsituationen anpassen und so eine gleichbleibend hohe Performance sicherstellen.

Unser Ansatz zur Gewährleistung der Skalierbarkeit besteht darin, die containerisierte Anwendung mit Docker und Kubernetes bereitzustellen. Docker ermöglicht es uns, die Anwendung in Container zu packen, die sich leicht reproduzieren und verteilen lassen. Kubernetes wiederum ist ein Orchestrierungswerkzeug, das die automatisierte Bereitstellung, Skalierung und Verwaltung von Anwendungen in Containern ermöglicht.

Durch die Trennung des Backends vom Frontend haben wir erreicht, dass die Datenbank, das Backend und das Frontend unabhängig voneinander skaliert werden können. Dies ermöglicht eine optimierte Nutzung von Ressourcen, da nur der Teil der Anwendung skaliert werden muss, der zusätzliche Kapazitäten benötigt. Im Moment ist eine SQLite-Datenbank im Einsatz. Diese kann jedoch mit wenigen Klicks auf eine separate PostgreSQL-Datenbank umgestellt werden, die dann einen eigenen Container darstellt.

4.9 Implementierung neuer Funktionalitäten

Neben der Umstrukturierung des Backends wurden auch einige neue Funktionalitäten implementiert. Auf einige soll im Folgenden eingegangen werden.

4.9.1 Stoppen von laufenden Scanroutinen

Aufgrund der potenziell zeitaufwändigen Scanroutinen, die sich, einmal gestartet, nicht mehr beenden lassen, entstand die Notwendigkeit, einen Prozess zu entwickeln, der den Scan unterbrechen kann. Besonders problematisch war diese Einschränkung, da während eines aktiven Scans die Löschung des Scans selbst nicht möglich war. Daher haben wir die Implementierung eines API-Endpunkts vorgenommen, der den Abbruch einer laufenden Scanroutine ermöglicht.

Die Umsetzung dieses Vorhabens stellte sich als komplex heraus, da es notwendig war, in die eigenständige Scraping-Routine einzugreifen. Da diese in einem separaten Thread läuft, lässt sie sich nicht direkt aus der API heraus steuern. Die Lösung bestand darin, dass der API-Endpunkt einen Eintrag in der Datenbank setzt, der signalisiert, dass der Scan gestoppt werden soll. Innerhalb der Scraping-Routine wurde die Schleife um eine Überprüfung erweitert, die zu Beginn jedes Durchlaufs erfolgt. Hierbei wird in der Datenbank geprüft, ob ein Abbruch des Scans gewünscht ist. Ist das der Fall, wird der Thread beendet und alle bisherigen Datenbankeinträge, die den Scan betreffen, werden gelöscht. Um zu verhindern, dass der Scan unmittelbar neu startet, wird das Datum der nächsten Scanroutine anhand des aktuellen Zeitpunkts und des festgelegten Scanintervalls neu berechnet.

4.9.2 Sofortiges starten von Scanroutinen

Mit der Implementierung der Funktion zum Abbrechen von Scans entstand die Notwendigkeit, ebenfalls eine manuelle Kontrolle über den Start der Scanroutinen zu ermöglichen. Zu diesem Zweck wurde ein weiterer API-Endpunkt eingerichtet. Bei einem korrekten Aufruf dieses Endpunkts wird eine neu geschriebene Funktion aufgerufen, die einen Thread mit dem gewünschten Scraping-Prozess sofort startet. So ist es nun möglich, Scanroutinen jederzeit zu starten, was einen erheblichen Zuwachs an Flexibilität und Kontrolle bedeutet.

4.9.3 Personalisierte Filter

Der Privacy Assistant besitzt die Fähigkeit, verschiedene Arten von personenbezogenen Daten zu identifizieren, darunter Telefonnummern, E-Mail-Adressen, Namen und Personen. Um die Anpassungsfähigkeit des Systems zu erhöhen und um weitere spezifische Datentypen, wie beispielsweise Matrikelnummern, zu berücksichtigen, haben wir die Implementierung personalisierter Filter vorbereitet. Die Idee hierbei ist, dass Nutzer in ihren Einstellungen Datenkategorien durch die Verwendung von regulären Ausdrücken individuell definieren können, die dann vom System erkannt werden.

Hierfür wurde ein Datentyp für Filter eingeführt. Dieser besteht aus einem Namen, einem regulären Ausdruck, dem Eigentümer des Filters sowie den ihm zugewiesenen Scanroutinen. Um die Wiederverwendung von Filtern zu ermöglichen, wurde eine n:n-Beziehung zwischen Filtern und Scanroutinen definiert. Das bedeutet, dass ein Filter mehreren Scanroutinen zugeordnet werden kann, andersrum aber auch eine Scanroutine mehrere Filter zugeordnet bekommen kann.

Die für diese Funktion notwendigen API-Endpunkte sind:

- GET /scanroutine/customfilter/ für das Auflisten der Filter eines Benutzers
- POST /scanroutine/customfilter/ zum Hinzufügen eines neuen Filters
- PATCH /scanroutine/customfilter/{id}/addtoroutine/{id} zum Zuordnen eines Filters zu einer Scanroutine
- PATCH /scanroutine/customfilter/{id}/removefromroutine/{id} zum Entfernen eines Filters von einer Scanroutine

Leider konnte die Integration der Filter in den Scraping-Prozess nicht innerhalb des gegebenen Zeitrahmens umgesetzt werden. Dieser ist leider sehr statisch codiert und nicht auf die flexible Erkennung einer variablen Anzahl von Datentypen ausgelegt. Hier wäre es sinnvoll, den gesamten Scraping-Prozess zu überarbeiten, um eine dynamische Anzahl an Datenarten zu scannen. Dabei kann auch der bisherige Code verbessert werden, der im Moment einige Duplikate enthält für die unterschiedlichen Datenarten.

4.9.4 Passwort zurücksetzen

Die vollständige Implementierung des Authentifizierungsprozesses beinhaltet auch die Funktion, das Passwort zu einem späteren Zeitpunkt zurücksetzen zu können. Diese Funktion war bisher nicht gegeben, konnte jedoch einfach implementiert werden, indem auf bereits existierende Django-Pakete zurückgegriffen wurde. Daher existiert nun ein API-Endpunkt, der das Zurücksetzen des Passworts eines Benutzers ermöglicht.

4.9.5 Löschfunktionalitäten

Die grundlegenden Funktionen einer CRUD REST-API umfassen nicht nur das Anlegen und Bearbeiten, sondern auch das Löschen von Daten. Ohne diese Möglichkeit würde die Datenbank immer weiter mit Daten gefüllt werden, ohne dass es eine Möglichkeit zur Bereinigung gibt. Daher wurden mehrere API-Endpunkte eingerichtet, um das Löschen einzelner Scans sowie gesamter Scanroutinen zu ermöglichen. Diese Funktionen sorgen für Flexibilität und Kontrolle über die Datenhaltung.

4.9.6 Mehr Möglichkeiten beim Anlegen einer Scanroutine

Im Laufe des Projektes wurden verschiedene Verbesserungen implementiert, die den Prozess des Anlegens einer Scanroutine vielseitiger und benutzerfreundlicher gestalten.

Zunächst wurde eine Funktion hinzugefügt, die automatisch ein BaseUrl-Modell erstellt und verknüpft, wenn dieses nicht bereits existiert. Das vereinfacht den Prozess und stellt sicher, dass stets eine korrekte Verknüpfung besteht und keine Duplikate von BaseUrls entstehen.

Darüber hinaus wurde die Möglichkeit eingeführt, ein Enddatum für eine Scanroutine festzulegen. Dies verhindert, dass eine Routine unbegrenzt weiterläuft.

Weiterhin wurde die Option hinzugefügt, einmalige Scans, sogenannte „onetime-Scans“, anzulegen. Diese Funktion ist besonders nützlich, wenn es gefordert ist, gezielte, einmalige Scans durchzuführen, ohne eine dauerhafte Routine einrichten zu müssen. Bisher war dies nicht möglich. Da auch kein Enddatum existierte, liefen Scanroutinen immer unendlich lange. Scanroutinen, die als Onetime-Scan gekennzeichnet sind, werden vom Watchdog ignoriert. Stattdessen wird beim Anlegen des Scans die für den manuellen Start eines Scans programmierte Funktion aufgerufen, die den Scan einmalig startet.

Um bei vielen Scanroutinen den Überblick zu behalten, wurde außerdem die Funktion implementiert, Routinen individuelle Namen zu geben.

4.9.7 Option zum Markieren von fehlerhaft erkannten Daten

Eine wichtige Ergänzung zur Funktionalität des Privacy Assistants ist die Möglichkeit, Daten als fehlerhaft erkannt zu markieren. Obwohl das entsprechende Feld bereits zuvor existiert hat, war es bislang funktionslos – das haben wir geändert.

Es wurden API-Endpunkte hinzugefügt, mit denen Daten als „wrong result“ klassifiziert werden können. Wenn Nutzer eine diese Kennzeichnung für erkannte Daten setzen, wird dies im Backend gespeichert und die entsprechenden Daten in der Benutzeroberfläche ausgegraut.

Darüber hinaus bietet diese Funktion das Potential für zukünftige Weiterentwicklungen: Diese als „falsch erkannt“ markierten Daten könnten in der Zukunft dazu genutzt werden, den Privacy Assistant durch maschinelles Lernen weiter zu verbessern. So könnte das System lernen, bestimmte Muster von Fehlern zu erkennen und diese in zukünftigen Scans zu vermeiden. Das allein stellt jedoch ein großes und eigenständiges Projekt dar, das komplexe Entwicklungen erfordert.

4.9.8 Statistiken

Eine neue Funktion im Frontend ist ein Dashboard, das verschiedene Statistiken und Fakten zur Benutzung des Tools anzeigt.

Um diese Funktion zu ermöglichen, wurde ein API-Endpunkt entwickelt, der die erforderlichen Berechnungen durchführt und die notwendigen Daten bereitstellt.

Einige Teile der Berechnungen werden bereits im Vorfeld bei der Abfrage einzelner Scans durchgeführt und als Teil der Scan-Daten ausgegeben sowie in der Datenbank zwischengespeichert. Diese Vorgehensweise stellt sicher, dass die Berechnungen verteilt und über die Zeit hinweg durchgeführt werden, anstatt alle Berechnungen auf einmal durchzuführen, wenn das Dashboard aufgerufen wird. Auf diese Weise bleibt die Antwortzeit des /statistics Endpunkts möglichst kurz.

Die Antwort des Endpunkts sieht beispielsweise so aus:

Response body


```
{
  "scans": {
    "total": 35,
    "active": 7,
    "deactivated": 13,
    "completed": 28,
    "not_evaluated": 28
  },
  "found_items": {
    "total": 875,
    "email": 66,
    "names": 139,
    "phone": 98,
    "images": 572
  },
  "next_scan": {
    "id": 24,
    "title": "24 - https://www.kleintierzuchtverein-steinhaldenfeld.de",
    "date": "2023-08-01T09:56:05.084000Z"
  }
}
```


Abbildung 12: Response des /statistics Endpunkt

5 Entwicklung des neuen Frontends

Das Frontend umfasst die grafische Benutzeroberfläche, über die der Benutzer mit der Anwendung interagiert, sowie die anschauliche Präsentation von Informationen und Daten. Es ist für das Rendering der Inhalte im Webbrowser zuständig. Dabei werden dem Benutzer die vom Backend geladenen Daten aufbereitet und auf eine strukturierte und anschauliche Weise dargestellt. Das Ziel besteht darin, dem Benutzer eine ansprechende und möglichst fehlerfreie, positive Benutzererfahrung zu bieten.

5.1 Entscheidung für React als Frontend-Framework für den Privacy Assistant

Die Entscheidung, React als Frontend-Framework für den Privacy Assistant einzusetzen, basierte auf mehreren Faktoren.

React ist im Jahr 2023 von der Stack Overflow Community auf Platz zwei der beliebtesten Web-Frameworks und Technologien hinter Node.js gewählt worden. Unter den professionellen Entwicklern liegt React auf Rang eins, vgl. dazu Stack Overflow (2023). Dies ist auch ein Grund, warum man im Internet viele Hilfestellungen und Tutorials zu jeglichen Themengebieten rund um React finden kann. Dies hilft dabei Fragestellungen und Probleme schnell zu lösen und Ideen für die Umsetzung von schwierigen Themen wie zum Beispiel die Authentifizierung und die Speicherung des Authentifizierungstokens frontendseitig zu lösen.

Außerdem bietet React viele Open-Source-Bibliotheken und fertige Komponenten an, welche einfach zu integrieren und dazu meistens noch anpassbar sind. Dies erleichterte die Implementierung von verschiedenen Funktionen innerhalb des Frontend des Privacy Assistant. Dazu zählen zum Beispiel die Verwendung von Axios, für das Anfragen der HTTP-Requests an das Backend, sowie die Verwendung von Chart.js zur übersichtlichen Visualisierung von Daten in einem Diagramm.

Die Dockerisierung einer React-Anwendung bietet viele Vorteile und ist ein weiterer Punkt, der für das Web-Framework React spricht. Es gestaltet sich unkompliziert, eine React-Anwendung in einem Docker-Container zu betreiben. Diese Vorgehensweise ermöglicht es, den Privacy Assistant reibungslos auf verschiedenen Betriebssystemen auszuführen und eine konsistente Umgebung sicherzustellen. Mit Docker-Containern können mögliche Kompatibilitätsprobleme vermieden werden, da die Anwendung in einer isolierten und portablen Umgebung läuft. Dadurch wird gewährleistet, dass der Privacy Assistant für jeden Benutzer eine konsistente und verlässliche Benutzererfahrung bietet, unabhängig von seinem Betriebssystem.

Ein entscheidender Faktor für die Wahl von React war auch die bereits vorhandene Erfahrung mit dem Web-Framework. Durch unsere vorherige Arbeit mit React in anderen Projekten wie dem Projekt Medieninformatik, ist die Syntax und Struktur bereits bekannt und vertraut. Die half uns dabei schnell in die Entwicklung einzusteigen und wichtige Themen wie die API-Anbindung schnell und strukturiert anzugehen.

5.2 Verwendung von TypeScript

Zum Beginn unseres Projekts haben wir entschieden TypeScript in Verwendung mit React für das Frontend zu benutzen. Auch ohne Vorerfahrung im Umgang mit TypeScript konnte der Umgang sich schnell angeeignet werden.

Die Verwendung von TypeScript im Vergleich zu herkömmlichem JavaScript bietet uns viele Vorteile.

Einer der Hauptvorteile liegt an der Funktionalität von TypeScript, statische Datentypen zu definieren. Festlegung von Datentypen für Variablen, Funktionen und Props verhindert potenzielle Typfehler bereits während des Code Schreibens, da die IDE diese bereits in diesem Prozess erkennt und anzeigt. Auch React selbst lässt ein Kompilieren des Codes mit solchen Fehlern nicht zu und verhindert so eine langwierige Fehlersuche. Somit entstehen nur in seltenet Fällen Laufzeitfehler aufgrund von Typfehlern und der Code ist deutlich robuster als mit herkömmlichen JavaScript.

Darüber hinaus verbessert TypeScript die Lesbarkeit und Verständlichkeit des Codes. Dies erleichtert einen möglichen neueinstieg oder die spätere Weiterarbeit am Projekt von anderen Entwicklern. Es kann also die Einarbeitungszeit deutlich verkürzt und beschleunigt werden.

Besonders geholfen hat uns TypeScript bei der Implementierung und Verwendung der API im Frontend, da die Datentypen aus der API-Dokumentation direkt im Frontend übernommen werden konnten. So war deutlich, welche Datentypen von einem Endpunkt erwartet werden und welche Datentypen als Antwort zurückkommen.

Schlussendlich kann schlussgefolgert werden, das TypeScript hilft die Qualität des Codes von Beginn an auf einem hohen Level zu halten, die Datentypen sorgen für eine gute Lesbarkeit und Wartbarkeit des Codes. Außerdem erleichtert es die Arbeit im Team von mehreren Entwicklern, weil an jeder Stelle des Codes die verwendeten Datentypen klar und durch die Hilfe einer IDE schnell ersichtlich sind.

5.3 Design- und Benutzererlebnisüberlegungen

Zu Beginn des Projekts wurden umfassende Gedanken hinsichtlich des Designs und des Benutzererlebnisses angestellt, die bei der Implementierung berücksichtigt wurden. Das

Ziel bestand darin, sich von einem Standard-Template, wie es früher verwendet wurde, zu lösen und ein individuelles Design zu schaffen. Hierbei lag besonderes Augenmerk auf einheitlichen Farben und einem hellen Erscheinungsbild.

Die Integration von Bootstrap spielte dabei eine entscheidende Rolle, da es eine große Palette von vordesignten und bewährten Komponenten bietet. Diese Entscheidung ermöglicht es später auch anderen Entwicklern das Design ohne große Mühen weiterzuführen.

Zur noch einfacheren Handhabung wurde zusätzlich zum klassischen Bootstrap auch die Bibliothek „react-bootstrap“ verwendet, die weitere Komponenten, mit integriertem Design auf der Basis von Bootstrap zu Verfügung stellt.

Um eine konsistente Benutzererfahrung zu schaffen und ein leicht erkennbares Design zu erzeugen, wurde ein roter Faden in der Farbgestaltung verfolgt. Dies gewährleistet, dass wichtige Komponenten auf allen Seiten einheitlich aussehen, es erleichtert den Nutzern die Handhabung der Software und das Zurechtfinden in der Anwendung wird unterstützt.

Für den Privacy Assistant wurden daher einheitliche Farben für die Hauptknöpfe, den Hintergrund und die Textfarbe definiert und konsequent in der gesamten Applikation verwendet. Dabei wurden Farbcodes sowohl von Bootstrap übernommen als auch als CSS-Variablen in der styles.css definiert und verwendet. Dadurch wird eine hohe Konsistenz in der Farbdarstellung gewährleistet und ermöglicht globale Anpassungen am Farbschema, die sich weitestgehend auf die gesamte Applikation auswirken. Dies Hilft, um den Wiedererkennungswert bei dem Benutzer zu erhöhen.

5.4 Implementierungsdetails und technische Herausforderungen

Die Implementierung des neuen Frontends für den Privacy Assistant beinhaltete die wichtige Aufgabe, die Funktionalität des alten Frontends abzudecken und bestenfalls zu verbessern. Hierzu wurden die einzelnen Seiten analysiert und übernommen. Dabei wurde darauf geachtet die Seiten übersichtlich aufzubauen und in kleinere, wiederverwendbare Komponenten aufzuteilen. Die Funktionalität wurde bestmöglich in diese kleineren Komponenten ausgelagert, um die Übersichtlichkeit des Codes zu wahren. Allerdings kam es zu Schwierigkeiten bei der Auslagerung der Filterfunktion in der „DetailedScanResult.tsx“, da sie viele Abhängigkeiten zu unterschiedlichen Komponenten besitzt.

Um die Navigation zwischen den Seiten zu realisieren, wurde die Logik mithilfe der Komponenten und Funktionen der „react-router“ und „react-router-dom“ Bibliotheken umgesetzt. Zusätzlich wurde eine eigene „PrivateRoute“-Komponente erstellt, die

sicherstellt, dass alle Seiten außer der Login- und Registrierungsseite nur zugänglich sind, wenn ein Benutzer eingeloggt und authentifiziert ist.

Ein eigenes Menü wurde entwickelt, das ein- und ausklappbar an der linken Seite platziert wurde und alle wichtigen Routen sowie die Möglichkeit zum Ausloggen beinhaltet.

Für die übersichtliche Darstellung der Ergebnisse der einzelnen Scans einer Scanroutine wurde die Bibliothek chart.js zusammen mit react-chartjs-2 verwendet.

Die API-Anfragen werden mithilfe von Axios gestellt und sind in eine eigene API-Klasse ausgelagert. Dabei wurden separate Instanzen für unautorisierte Anfragen und autorisierte Anfragen an die API verwendet. Nach erfolgreicher Authentifizierung erhält das Frontend einen Token, der bei allen weiteren autorisierten Anfragen an das Backend im Authorization-Header des Requests mitgesendet werden muss.

Bei der Verarbeitung und Speicherung der Daten gab es anfänglich Probleme. Die endgültige Lösung besteht darin, den Token im „Redux“-Store zu speichern und zusätzlich im Session Store zu persistieren, um ein manuelles neu laden der Seite ohne automatisches Ausloggen zu ermöglichen. Dafür wird ein „PersistGate“ in Verbindung mit einem „persistedReducer“ aus der „redux-persist“ Bibliothek verwendet.

Das Auslagern der API-Funktionen ermöglicht eine zentrale Bearbeitung und Konfiguration der API-Schnittstelle im Frontend und erweist sich als sehr nützlich, da Änderungen an der API nicht in jeder Komponente durchgeführt werden müssen, sondern an einem zentralen Ort durchgeführt werden können. Außerdem bietet dies Variante eine schnelle Einbindung von API-Abfragen in eine neue Komponente

Fehlermeldungen werden dem Nutzer teilweise als Hinweistext auf der Webseite selbst und zum Teil auch als Alert angezeigt, um eine transparente Benutzerinteraktion zu gewährleisten.

Insgesamt führt die sorgfältige Implementierung der technischen Details und das erfolgreiche Bewältigen der technischen Herausforderungen zu einem funktionalen, benutzerfreundlichen und zuverlässigen Frontend für den Privacy Assistant.

5.5 Wichtige und neue Seiten im Frontend

Im Folgenden wird auf wichtige Seiten in Bezug auf die Funktionalität des Privacy Assistant eingegangen.

5.5.1 Dashboard

Das Dashboard des Privacy Assistant ist eine zentrale Seite, die dem Benutzer eine übersichtliche Darstellung relevanter Statistiken und Funktionen bietet. Die Dashboard-

Seite wird dem Benutzer nach einem erfolgreichen Login angezeigt. Es ermöglicht den Benutzern, den Status ihrer Scanroutinen und die gefundenen personenbezogenen Daten auf einen Blick zu überprüfen.

Auf der Dashboard-Seite werden verschiedene Statistiken präsentiert, darunter die Anzahl der aktiven Scanroutinen, die Anzahl der abgeschlossenen Scans und die Anzahl der Scans, die noch ausgewertet werden müssen. Diese Informationen bieten den Benutzern einen schnellen Überblick über den aktuellen Zustand ihrer Scanroutinen und deren Ergebnissen.

Darüber hinaus zeigt das Dashboard den Benutzern den nächsten geplanten Scan, einschließlich des Titels der Scanroutine und des geplanten Datums. Dies ermöglicht es den Benutzern, den aktuellen Status seiner aktiven Scanroutinen genauer zu überwachen.

Ein weiteres wichtiges Feature des Dashboards ist die Anzeige der gefundenen personenbezogenen Daten. Es zeigt die Gesamtzahl der gefundenen Daten sowie die Anzahl der gefundenen Fotos, Namen, Telefonnummern und E-Mail-Adressen. Diese Funktion ermöglicht es den Benutzern, einen genauen Überblick über die gesammelten Daten zu behalten.

Zusätzlich bietet das Dashboard die Möglichkeit, eine neue Scanroutine zu erstellen. Durch das Ausfüllen eines einfachen Formulars können die Benutzer neue Scanroutinen erstellen und ihre Datenanalysen erweitern, ohne auf die separate Seite dafür wechseln zu müssen.

Insgesamt bietet das Dashboard eine benutzerfreundliche und informative Oberfläche, um den Benutzern einen klaren Einblick in den Status ihrer Scanroutinen zu geben.

Die folgende Abbildung zeigt das neue Dashboard:

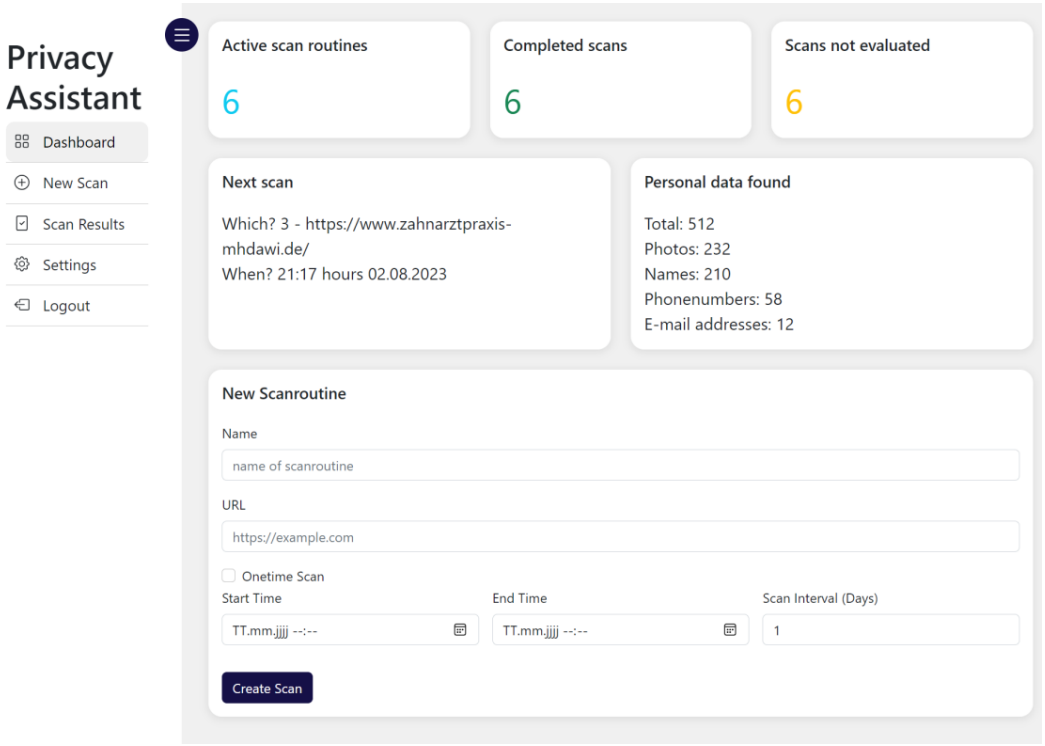

Abbildung 13: Beispielhafte Darstellung der Dashboard Seite

5.5.2 Settings

Die Settings Seite des Privacy Assistant bietet den Benutzern die Möglichkeit, benutzerdefinierte Filter zu verwalten und so später die Datenanalyse individuell anzupassen, wenn die Funktionalität der Filter auch auf den Scan Einfluss haben. Auf dieser Seite können Benutzer vorhandene Filter anzeigen, bearbeiten oder löschen sowie neue benutzerdefinierte Filter hinzufügen.

Die Settings-Seite stellt eine übersichtliche Benutzeroberfläche bereit, um bestehende Filter anzuzeigen und neue Filter hinzuzufügen. Benutzer können die Filternamen und Suchparameter einfach eingeben, um neue benutzerdefinierte Filter zu erstellen. Die Möglichkeit, Filter zu bearbeiten oder zu löschen, bietet eine flexible Möglichkeit, die Datenanalyse je nach Bedarf anzupassen.

Insgesamt trägt die Settings-Seite dazu bei, die Funktionalität des Privacy Assistant zu erweitern und den Benutzern in Zukunft eine personalisierte und benutzerfreundliche Datenverarbeitung zu ermöglichen. Das ist auf folgender Abbildung erkennbar:

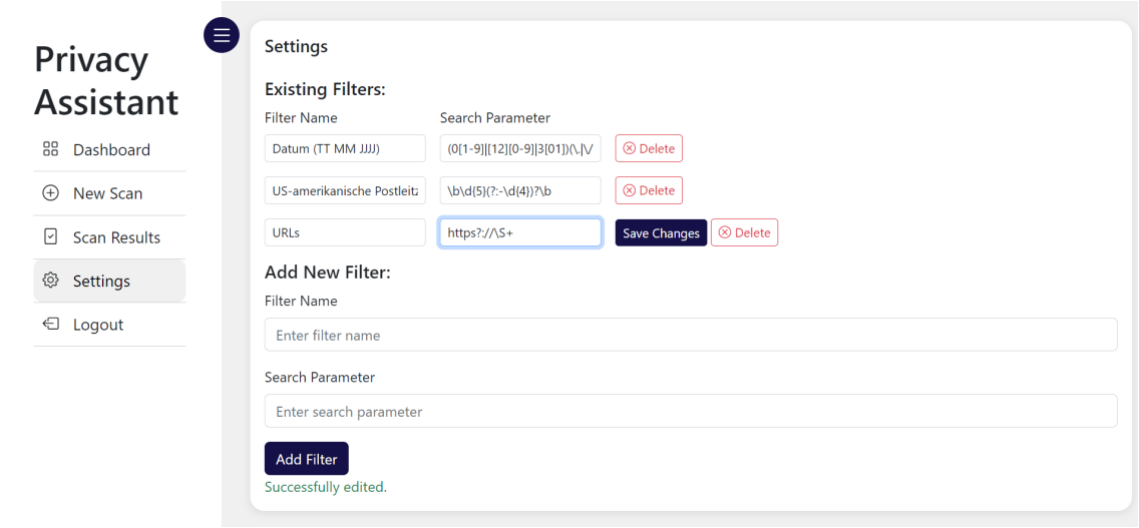

Abbildung 14: Beispielhafte Darstellung der Settings Seite

5.5.3 Neue Scanroutine anlegen

Die Seite, um eine neue Scanroutine im Privacy Assistant anzulegen ermöglicht es den Benutzern, neue Scanroutinen zu erstellen. Mit dieser Funktion können Benutzer benutzerdefinierte Scans konfigurieren, um bestimmte Websites auf personenbezogene Daten, Bilder, E-Mail-Adressen und Telefonnummern und Personennamen zu überprüfen.

Die Nutzer können den Namen der Scanroutine, die zu überprüfende URL und den Scanzeitplan festlegen. Das bedeutet, dass sie entscheiden können, ob der Scan einmalig durchgeführt werden soll oder in regelmäßigen Intervallen wiederholt wird. Wenn der Benutzer den „Onetime-Scan“-Modus auswählt, wird der Scan nur einmal ausgeführt. Dies ist nützlich, um gezielte Untersuchungen durchzuführen oder eine bestimmte Website zu überwachen.

Hingegen ermöglicht der regelmäßige Scan Benutzern, Websites kontinuierlich zu überwachen, um Änderungen oder neue personenbezogene Daten zu erkennen. Benutzer können den Scanzeitplan basierend auf ihren Präferenzen und Anforderungen anpassen.

Die Seite „New Scan“ sieht so aus:

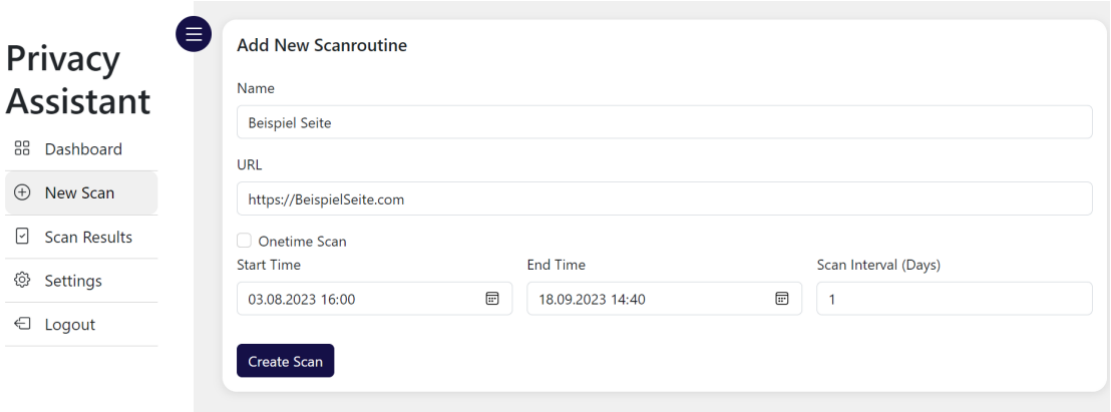

Abbildung 15: Beispielhafte Darstellung der „Neuen Scan anlegen“-Seite

5.5.4 Übersicht der Scanroutinen

Die Seite „Scanroutines“ im Privacy Assistant dient dazu, den Benutzern eine übersichtliche Darstellung aller aktiven und bestehenden Scanroutinen zu bieten.

Hier können die Benutzer schnell und einfach alle konfigurierten Scanroutinen einsehen und verwalten. Die Benutzeroberfläche ist benutzerfreundlich gestaltet und ermöglicht es den Benutzern, mühelos zwischen den Scanroutinen zu navigieren und verschiedene Aktionen durchzuführen, wie das Starten oder Stoppen von Scans, das Bearbeiten von Scanroutinen oder das Löschen nicht mehr benötigter Scanroutinen. Jede Scanroutine wird mit ihrem Namen, der zugehörigen URL und dem Zeitplan übersichtlich in der Liste dargestellt.

Benutzer haben außerdem die Möglichkeit, direkt auf die Ergebnisse jeder Scanroutine zuzugreifen, um sich die darin enthaltenen Scans anzeigen zu lassen. Das sieht so aus:

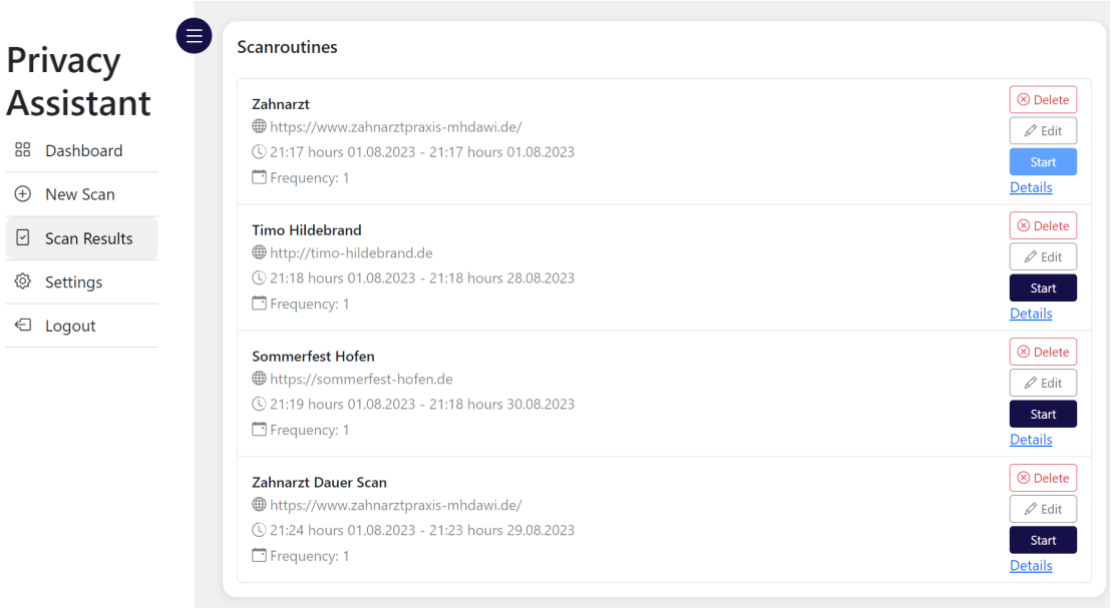

Abbildung 16: Beispielhafte Darstellung der „Scanroutines“-Seite

5.5.5 Detailansicht eines Scanergebnisses

Die Seite „Detailed Scan Result“ Seite im Privacy Assistant bietet eine umfassende und detaillierte Darstellung der Ergebnisse eines durchgeführten Scans. Ihr Hauptnutzen liegt darin, Benutzern die Möglichkeit zu geben, die erfassten Daten genau zu analysieren und Datenschutzverletzungen sowie potenzielle Risiken zu identifizieren und die gesammelten Ergebnisse zu bewerten.

Die Seite ist in verschiedene Registerkarten unterteilt, um die verschiedenen Arten von erfassten Daten übersichtlich darzustellen, darunter Fotos, Namen, E-Mail-Adressen und Telefonnummern. Benutzer haben die Möglichkeit, die Ergebnisse mithilfe von Filtern zu verfeinern und nur die für sie relevanten Informationen anzuzeigen.

Die Seite zeigt auch wichtige statistische Informationen an, wie die Gesamtzahl der gefundenen Pfade, die Anzahl der gefundenen Daten und die Dauer des Scans. Dies ermöglicht es den Benutzern, schnell einen Überblick über das Scanergebnis zu erhalten und den Datenschutzstatus zu beurteilen.

Des Weiteren bietet die Seite eine Funktion zum Herunterladen der Scanergebnisse, sodass Benutzer die Daten für weitere Analysen oder Sicherungszwecke speichern können.

Die Wichtigste Funktionalität der Seite besteht darin Scan-Ergebnisse auf der Basis Kategorien, „Confidential“, „Non-Confidential“ und „Wrong Result“, zu bewerten und so einen Überblick über die gesammelten Daten zu erhalten. Je nach Bewertung der einzelnen Ergebnisse kann der Benutzer handeln und gegebenen falls Daten aus seinen Webseiten entfernen. Im Frontend kann das dann wie folgt aussehen:

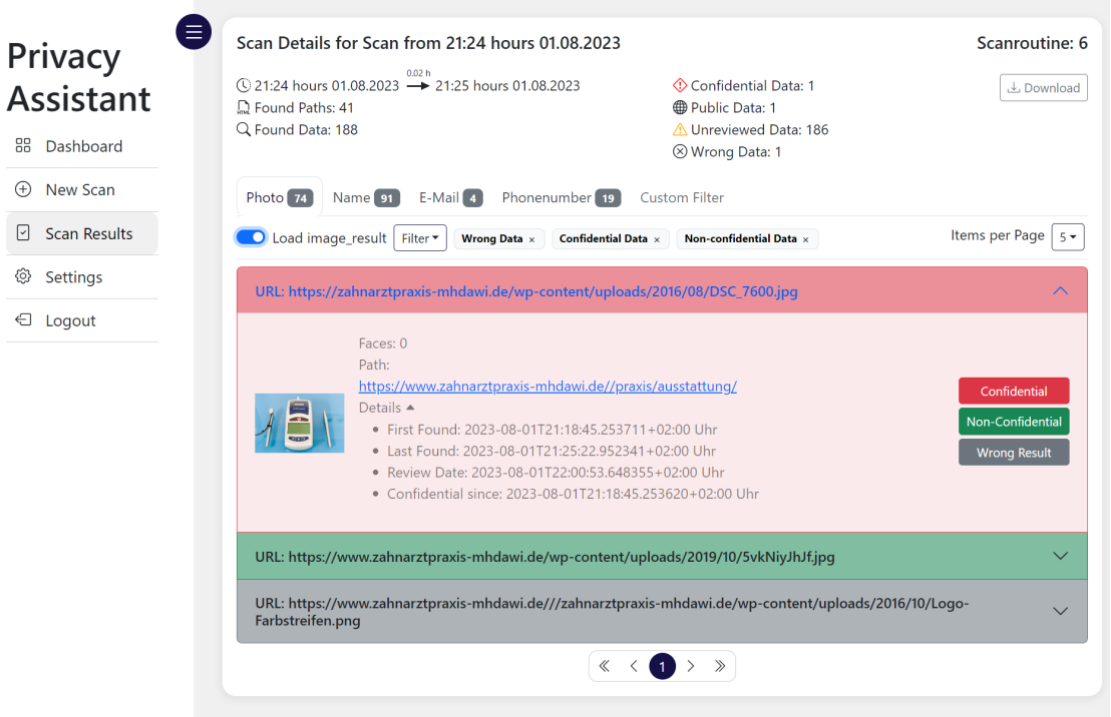

Abbildung 17: Beispielhafte Darstellung der Detail-Seite einer Scanroutine

6 Integration der REST-API mit dem Frontend

Die Integration der REST-API ins Frontend stellte sich als eine Herausforderung dar, da dieser Schritt erst spät im Projekt durchgeführt wurde. Während der Entwicklungsphase wurden das Frontend und das Backend weitestgehend getrennt voneinander entwickelt und auf verschiedenen Branches in GitLab verwaltet. Die Trennung der Entwicklung ermöglichte eine parallele Arbeit an beiden Teilen der Anwendung ohne codeseitige Überschneidungen. Dies ermöglichte eine gute Aufteilung der Arbeit. Allerdings führte diese Trennung dazu, dass das Zusammenfügen von Backend und Frontend in einem späten Stadium des Projekts eine Aufgabe war, die trotz ausführlicher Dokumentation der API im Voraus, viel Zeit und Anpassungen auf Seiten des Frontend und des Backend mit sich brachte.

6.1 Koordination von Frontend- und Backend-Entwicklung

Die Koordination zwischen der Frontend- und Backend-Entwicklung war ein wichtiger Bestandteil der Umsetzung. Regelmäßige wöchentliche Treffen spielten dabei eine wichtige Rolle, um das Team mit dem Betreuer abzustimmen, den Fortschritt zu besprechen und auftretende Probleme zu behandeln. In diesen Meetings konnte wertvolles Feedback erhalten, Ideen ausgetauscht und die nächsten Schritte besprochen werden

Intern im Team bestand auch ständiger Kontakt, um Abläufe und Details zu klären sowie sich auf die wöchentlichen Meetings vorzubereiten. Diese schnelle und terminunabhängige Kommunikation half, um Probleme schnell zu lösen.

Zu Beginn des Projekts wurde gemeinsam versucht, das Django-Projekt zum Laufen zu bringen. Die Herausforderungen, auf die wir dabei gestoßen sind, führten zu einzelnen und gemeinsamen Versuchen, den Code zum Funktionieren zu bringen. Es gab noch keine Trennung zwischen Frontend- und Backendentwicklung.

Nach der Trennung in Frontend und Backend verschob sich die Kommunikation mehr auf die Definition der Schnittstellen und Funktionen. Dabei wurde am Anfang großen Wert darauf gelegt die Schnittstellen klar zu definieren, um eine reibungslose Zusammenführung von Front und Backend zu ermöglichen. Diese Definition musste später nochmals angepasst und darauf reagiert werden. Während weniger gemeinsam gecodet wurde, konzentrierte sich das Team auf die Integration der Schnittstellen und die Zusammenarbeit an spezifischen Funktionen und Features der Applikation.

Insgesamt herrschte eine sehr gute Kommunikation im Team, die zu einer guten Koordination zwischen Frontend und Backend führte. Es wurde sich immer gegenseitig

geholfen und unterstützt. Die enge Zusammenarbeit und der offene Austausch von Ideen ermöglichten es uns, das Projekt erfolgreich umzusetzen.

6.2 Herausforderungen und Lösungen bei der Integration

Zu Beginn des Projekts wurde eine klare Definition der REST-API festgelegt. An dieser Definition orientierten sich die API-Anfragen umfangend und ermöglichte es die Entwicklung im Frontend auf der Basis der erwarteten Datenstrukturen und Schnittstellen voranzutreiben und mit Beispiel JSON-Daten zu testen.

Auf der Seite der Backendentwicklung musste mit der Zeit festgestellt werden, dass durch die Verwendung von altem Code, der wiederverwendet werden sollte, einige Definitionen in der API nachträglich geändert werden mussten. Diese Abwandlung führte dazu die Kommunikation und die Zusammenarbeit Frontend und Backend übergreifend zu erhöhen und die Änderungen gemeinsam auf beiden Seiten zusammen zu führen. Mithilfe von vielen manuellen Tests musste getestet werden, dass die Funktionalität im Frontend, sowie im Backend trotz der Änderungen an den wichtigen Schnittstellen gewährleistet ist. So konnten frühzeitig Bugs gefunden und korrigiert werden.

Schlussendlich stellte sich die Authentifizierung als größte Herausforderung dar. Die Schwierigkeit bestand darin, den vom Backend erhaltene Token im Frontend richtig zu speichern. Bei den darauffolgenden Anfragen an das Backend über die API muss dieser an der richtigen Stelle in der Anfrage enthalten sein, um die Identifikation zu gewährleisten.

Nachdem dieser Meilenstein geschafft war, konnten die restlichen API-Endpunkte ohne größere Komplikationen und deutlich schneller angebunden werden.

7 Vergleich und Fazit

In diesem Kapitel werden die erreichten Fortschritte der Privacy Assistant Software reflektiert. Hierbei steht insbesondere die Gegenüberstellung des Ausgangszustands und des Endergebnisses im Fokus.

7.1 Fazit und Vergleich mit der Ausgangssituation

Die Überarbeitungen und Weiterentwicklungen des Privacy Assistants in unserem Studienprojekt haben zu Verbesserungen auf verschiedenen Ebenen geführt. Im Vergleich zur Ausgangssituation ist die Software nun in einem deutlich professionelleren und ansprechenderen Zustand.

Die grafische Benutzeroberfläche des Privacy Assistants wurde vollständig erneuert. Mit dem Wechsel zum bekannten Open-Source-Framework React konnte die User Experience (UX) und das visuelle Erscheinungsbild der Anwendung signifikant verbessert werden. Zudem ermöglicht die Wahl dieses Frameworks eine einfache und effiziente Erweiterung der Software.

Im Backend ist der Privacy Assistant nun deutlich strukturierter und sauberer programmiert. Durch die Implementierung einer REST-API und die Verwendung von Docker sind nicht nur die Startprozesse vereinfacht, sondern auch die Skalierbarkeit und die Modularität der Software verbessert worden. Dies bildet die Grundlage für eine einfache Erweiterung und Pflege der Software.

Darüber hinaus haben wir eine Reihe neuer Funktionalitäten eingeführt. So haben wir Funktionen zur Generierung von Statistiken und Auswertungen implementiert oder auch die Möglichkeiten, Scanroutinen manuell zu starten oder zu stoppen. Darüber hinaus kann der Benutzer nun Filter verwalten, Routinen ändern und Daten löschen. Zudem haben wir die Optionen beim Anlegen von Scans erweitert und eine Funktion für einmalige Scans hinzugefügt. Auch die Klassifizierung von falsch erkannten Daten wurde als neues Feature integriert.

Schon auf der Startseite sieht man die Verbesserungen auf den ersten Blick:

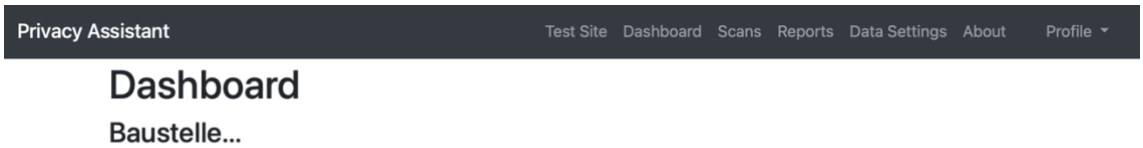

Abbildung 18: Dashboard vor unserem Projekt

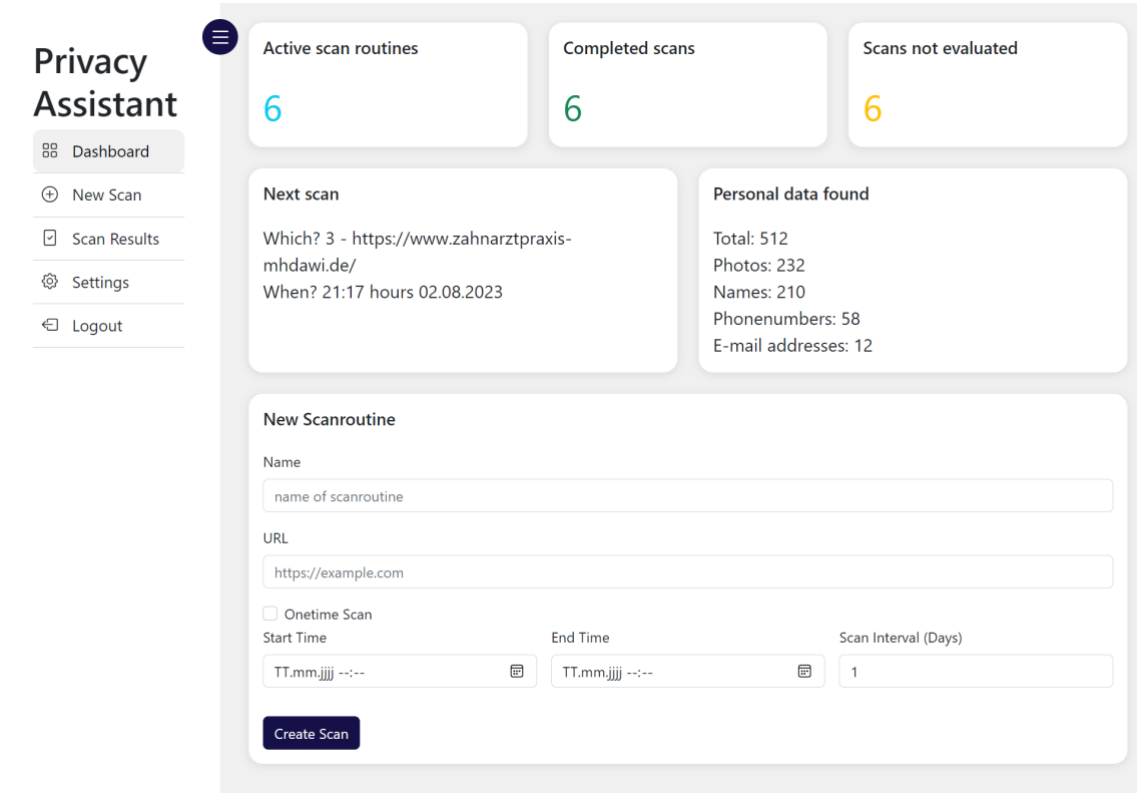

Abbildung 19: Dashboard nach unserem Projekt

Zusammenfassend lässt sich sagen, dass unser Projekt zwar nicht die ursprünglich angedachten Ziele erreicht hat, jedoch zu einer erheblichen Verbesserung gegenüber der ursprünglichen Software geführt hat, sowohl in Bezug auf die Benutzerfreundlichkeit und die Codequalität als auch in Bezug auf die Funktionalität.

7.2 Lernerfahrungen und erzielte Kompetenzen

Im Verlauf des Studienprojekts konnten wir eine Fülle von Erfahrungen sammeln. Eine der grundlegendsten Erkenntnisse war die Bedeutung sauberen Codings und einer ordentlichen Dokumentation. Durch das Fehlen dieser Aspekte in der ursprünglichen Software hat uns dies verdeutlicht, wie ungemein wichtig dies in der Softwareentwicklung ist.

Darüber hinaus hat uns das Projekt einen wertvollen Einblick in den Prozess der Weiterentwicklung bestehender Software gegeben. Im Gegensatz zu unseren bisherigen Projekten im Studium, die wir stets von Grund auf neu begonnen haben, mussten wir uns diesmal in ein bestehendes Projekt einarbeiten. Dies ist eine neue und wertvolle Erfahrung, die als Softwareentwickler sicherlich von Nutzen sein wird.

Auf technischer Ebene konnten wir uns in einer Vielzahl von Technologien einarbeiten oder Erfahrung sammeln. Dazu gehören das Arbeiten mit REST-APIs, der Einsatz von

Docker, die Programmierung mit Python und Django, sowie die Entwicklung von Frontends mit React.

Neben diesen technischen Aspekten konnten wir auch unsere Fähigkeiten in Bezug auf die Zusammenarbeit in einem Entwicklungsteam weiterentwickeln. Dies umfasst die Absprachen untereinander, das effiziente Arbeiten mit Git und das Verwalten von Branches.

8 Ausblick und Schlussworte

In diesem abschließenden Kapitel werden die gewonnenen Erkenntnisse zusammengefasst und Möglichkeiten für zukünftige Verbesserungen beleuchtet.

8.1 Zusammenfassung der Erkenntnisse

Im Verlauf dieses Projekts haben wir wertvolle Erfahrungen gesammelt und Einsichten gewonnen. Ursprünglich gingen wir mit anderen Zielen an das Projekt heran, wurden jedoch aufgrund der Startschwierigkeiten und des suboptimalen Zustands der bestehenden Software dazu gezwungen, unsere Ziele zu überarbeiten. Dies führte zu einer umfassenden Neustrukturierung und Überarbeitung des Projekts, die es uns ermöglichte, eine professionellere Basis zu legen, die zukünftige Weiterentwicklung und möglicherweise sogar eine Veröffentlichung unterstützt.

Unsere Bewertung des Projekterfolgs ist daher schlussendlich sehr positiv und wir sind stolz auf das erreichte Ergebnis. Trotz einiger schwieriger Phasen, in denen unser Fortschritt langsamer war als erwartet, konnten wir gegen Ende des Projekts erhebliche Fortschritte erzielen. Dank intensiver Arbeit und Investition von viel Zeit, gelang es uns, unsere überarbeiteten Ziele erfolgreich zu erreichen.

8.2 Potenzielle zukünftige Verbesserungen und Erweiterungen

Die Entwicklung einer Software ist ein kontinuierlicher Prozess und bietet stets Raum für Verbesserungen und Erweiterungen. Auch beim Privacy Assistant gibt es viele Ideen, wie das Tool weiterentwickelt werden kann.

Eine große Verbesserung wäre die Modularisierung des gesamten Scraping-Prozesses. Diese Änderung würde die Integration neuer Datenarten vereinfachen und den Code schlanker und hochwertiger machen. Dabei wäre ein umfassender Umbau des bestehenden Scraping-Prozesses nötig, aber es würde letztlich auch die Implementierung von benutzerdefinierten Filtern vereinfachen.

Ein weiterer spannender Ansatz ist die Integration von Machine Learning-Techniken. Die Software könnte aus Fehlern lernen, speziell aus den als falsch klassifizierten Ergebnissen, und somit die Genauigkeit der Scans kontinuierlich verbessern. Auch dies ist jedoch ein sehr umfassendes Thema und vermutlich ein Studienprojekt für sich.

Auch wäre eine Funktion, die Benutzer per E-Mail über neue Scanergebnisse informiert, ein sehr nützliches Feature.

Bevor der Privacy Assistant schließlich veröffentlicht werden kann, sollten Stresstests mit großem Datenvolumen durchgeführt werden. Es ist für diese Software entscheidend, wie sie sich verhält, wenn große Websites über längere Zeiträume gescannt werden.

All dies sind nur einige der Ideen – die Möglichkeiten sind fast unbegrenzt und bieten Raum für viele weitere Studienprojekte.

8.3 Schlussbemerkungen

Abschließend möchten wir unser Studienprojekt mit den Gedanken eines jeden Programmierers zusammenfassen: „Ich hasse Programmieren. Ich hasse Programmieren. Ich hasse Programmieren. Es funktioniert! Ich liebe Programmieren!“ Dieser Kreislauf spiegelt sich auch in diesem Projekt wider – gespickt mit Misserfolgen, Rückschritten und schließlich Triumphen. Trotz aller Herausforderungen hat uns dieses Projekt gezeigt, dass man mit Ausdauer und Teamarbeit jede Codezeile zähmen kann. Nun freuen wir uns auf die nächsten Kapitel im Leben des Privacy Assistants und hoffen, dass die Fehlermeldungen weniger und die Erfolgserlebnisse mehr werden. Wir sind gespannt, wie sich diese Software in Zukunft entwickeln wird.

Anhang

Der gesamte Code des Privacy Assistants liegt im hochschuleigenen GitLab unter folgender URL (nur aus dem Hochschulnetz erreichbar):

https://gitlab.hs-esslingen.de/it-sec-research-projects/PrivacyAssistant

Literaturverzeichnis

DRY vs KISS – Clean Code Prinzipien – generic.de. (2023). https://www.generic.de/blog/dry-vs-kiss-clean-code-prinzipien

Erkennung und Schutz sensibler Daten - Amazon Macie - AWS. (o. D.). Amazon Web Services, Inc. https://aws.amazon.com/de/macie/

Hartung, D. (2022). *Studienprojekt Daniel Hartung*.

Häussler, M. (2022). *Distributed Systems - Vorlesung* [Vorlesungsfolien; PowerPoint]. Vorlesung Distributed Systems, Esslingen, Deutschland.

Ionos. (2022). Swagger: Mehr Komfort bei der API-Entwicklung. *IONOS Digital Guide*. https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-swagger/

Joos, T. (2019, 5. Juli). Die 10 wichtigsten Vor- und Nachteile von Docker-Containern. *IP-Insider*. https://www.ip-insider.de/die-10-wichtigsten-vor-und-nachteile-von-docker-containern-a-844230/

Shurigin, A. (2017). Top 10 mistakes that Django Developers make. *Toptal Engineering Blog*. https://www.toptal.com/django/django-top-10-mistakes

Stack Overflow. (2023). *2023 Developer Survey*. Abgerufen am 1. August 2023, von https://survey.stackoverflow.co/2023/

Was ist eine REST-API? (2023, 24. Juli). RedHat. Abgerufen am 1. August 2023, von https://www.redhat.com/de/topics/api/what-is-a-rest-api

Was ist Token-Based Authentication? | Entrust. (o. D.). https://www.entrust.com/de/resources/faq/what-is-token-based-authentication

Welcome - PrivacyScore. (o. D.). https://privacyscore.org/

Ehrenwörtliche Erklärung

Hiermit versichern wir, dass wir den vorliegenden Bericht selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebene Literatur und Hilfsmittel verwendet habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Name:	Fink	Vorname:	Maximilian
Matrikel-Nr.:	764001	Studiengang:	SWB

Name:	Kolb	Vorname:	Andreas
Matrikel-Nr.:	764390	Studiengang:	SWB

Stuttgart, 03.08.2023	
Ort, Datum	Unterschrift

Stuttgart, 03.08.2023	
Ort, Datum	Unterschrift