Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Software Engineering

Planung und Entwicklung eines leicht weiterentwickelbaren Systems für Persönlichkeitstests

Nicolai Wolfrom

Matrikelnummer: 5344982

nicow03@zedat.fu-berlin.de

Betreuer/in:	Prof. Dr. Lutz Prechelt
Eingereicht bei:	Prof. Dr. Lutz Prechelt
Zweitgutachter/in:	Prof. Dr. Charlotte von Bernstorff

Berlin, 28. März 2022

Zusammenfassung

In dieser Arbeit wird ein System für Persönlichkeitstests geplant und entwickelt. Das System wird für einen Kunden erstellt, der mit Persönlichkeitstests arbeitet und auf diesem Gebiet forscht. Es existieren bereits ähnliche Software-Lösungen, die aber für die speziellen Anforderungen des Kunden nicht ausreichen. Daher bedarf es eines individuell entwickelten Systems.

Eine erste Version dieses Systems, welche die grundlegenden Anforderungen des Kunden erfüllt, wird in dieser Arbeit entwickelt. Da zukünftig aber noch weitere Anforderungen hinzukommen, liegt dabei ein besonderer Fokus auf der leichten Weiterentwickelbarkeit des Systems.

Die Entwicklung erfolgt recht agil, sodass es wiederholt Gespräche mit dem Kunden gibt. Dadurch fallen Änderungen an den Anforderungen frühzeitig auf und es kann leichter auf das Feedback des Kunden reagiert werden.

Das System besteht aus einem Python-Backend, dass mit dem Web-Framework Flask erstellt wird. Zum Anzeigen und Durchführen der Persönlichkeitstests wird die JavaScript-Bibliothek SurveyJS genutzt. Um das System möglichst leicht weiterentwickeln zu können, liegt bei der Entwicklung ein Fokus auf einer guten Modularisierung sowie einer hilfreichen und übersichtlichen Dokumentation.

Nach der Entwicklung wurde die Weiterentwickelbarkeit getestet, wobei das Ergebnis positiv ausfiel. Auch wäre der Kunde bereits in der Lage, Persönlichkeitstests mit dem System durchzuführen, da es bereits auf einem Server läuft und es Workshops zur Einarbeitung des Kunden gab.

Abstract

In this thesis a system for personality tests is planned and developed. The system is being created for a customer who works with personality tests and does research in this area. Similar software solutions already exist, but they are not sufficient for the special requirements of the customer. Therefore, an individually developed system is required.

A first version of this system, which meets the basic requirements of the customer, is developed in this thesis. However, since further requirements will be added in the future, a special focus is on the easy further development of the system.

The development is quite agile, so there are repeated discussions with the customer. As a result, changes to the requirements are noticed early on and it is easier to respond to customer feedback.

The system consists of a Python backend built with the Flask web framework. The JavaScript library SurveyJS is used to display and carry out the personality tests. In order to be able to further develop the system as easily as possible, the focus during development is on good modularization, as well as helpful and clear documentation.

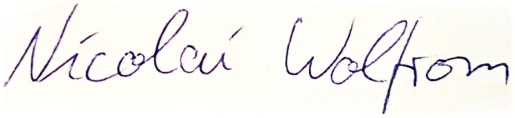
After the development, the further developability was tested, with the result being positive. The customer would also already be able to carry out personality tests with the system, since it is already running on a server and there were workshops to introduce the customer.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

28. März 2022

Nicolai Wolfrom



Inhaltsverzeichnis

1 Einleitung

In dieser Bachelorarbeit soll ein System für einen Kunden erstellt werden, der mit Persönlichkeitstests arbeitet und auf diesem Gebiet forscht. Die Persönlichkeitstests messen bei der getesteten Person die Ausprägung von Persönlichkeitsmerkmalen, die sich in verschiedene Kategorien einteilen lassen. Eine Reihe von auswertbaren Fragen lässt sich jeweils einer dieser Kategorien zuordnen. Diese auswertbaren Fragen werden auf einer Ordinalskala[1] beantwortet, wobei jede Antwort durch eine positive ganze Zahl dargestellt wird. Zusätzlich zu den auswertbaren Fragen werden auch noch weitere Fragen zu demographischen Informationen der getesteten Person gestellt.

Aktuell wird eine Software as a Service[2] Lösung genutzt, die den Anforderungen des Kunden aber nicht mehr gerecht wird. Mit dieser ist es möglich die Tests durchzuführen und die Antworten der Befragten als CSV-Datei zu exportieren. Eine Auswertung oder Analyse der Ergebnisse ist jedoch beispielsweise nur manuell möglich und wird aktuell mit Excel durchgeführt.

Der Kunde plant, Persönlichkeitstest im Rahmen von Studien einzusetzen und kommerziell anzubieten. Da dadurch neue Anforderungen an das System entstehen, die von der aktuellen Software as a Service Lösung nicht erfüllt werden, bedarf es einer besseren Lösung. Zu den neuen Anforderungen gehört zunächst die automatische Auswertung der Ergebnisse. Insbesondere bei einer größeren Anzahl an Nutzer*innen wäre die manuelle Auswertung der Persönlichkeitstests nicht nur aufwändig für den Kunden, sondern würde aus Sicht der Nutzer*innen auch zu lange dauern. Zudem sollte es nur autorisierten Nutzer*innen möglich sein, einen Persönlichkeitstest durchzuführen. Die Autorisierung erfolgt dabei über entsprechenden Zugangstoken[3], die bei Bedarf an Testgruppen ausgegeben werden können. Zusätzlich soll ein beispielhaftes Zertifikat mit den Ergebnissen generiert werden, das den Nutzer*innen nach dem Beantworten der Fragen und Auswerten der Antworten angezeigt wird.

Ziel der Bachelorarbeit ist es, eine erste Version eines Systems für die Durchführung von Persönlichkeitstests zu planen, entwickeln und deployen[4]. Dieses soll das Erstellen und Durchführen der Persönlichkeitstests, sowie die grundlegenden neuen Anforderungen erfüllen. Auch zukünftig werden noch neue Anforderungen hinzukommen, an die das System angepasst werden muss. Damit dies leicht möglich ist, liegt ein besonderer Fokus der Bachelorarbeit auf der Wahl guter Technologien, sowie auf der Entwicklung eines leicht wartbaren und weiterentwickelbaren Systems. Ob und inwiefern sich das System leicht weiterentwickeln lässt, wird dabei anhand eines praktischen Tests nach der Entwicklung überprüft.

An das System besteht im Rahmen der Bachelorarbeit aber nicht der Anspruch, dass alle Anforderungen, die während der Entwicklung oder in Gesprächen noch entstehen, umgesetzt werden. Wichtiger ist, dass das System am Ende als laufender

[1] Skala mit geordneten, diskreten Antwortmöglichkeiten; weitere Informationen: https://de.statista.com/statistik/lexikon/definition/99/ordinalskala/

[2] Gemietete(r) Server inklusive Software, wie z.B. Cloud-Basiertes Microsoft Office 365; weitere Informationen: https://www.computerwoche.de/a/was-ist-software-as-a-service,3332266

[3] Hashwerte, die Zugang zu etwas gewähren, indem sie Nutzer*innen identifizieren und autorisieren; weitere Informationen https://en.wikipedia.org/wiki/Access_token

[4] Software auf ein Server installieren, konfigurieren und starten; weitere Informationen: https://www.dev-insider.de/was-ist-deployment-a-1025926/

Server zur Verfügung steht, sodass es direkt von dem Kunden genutzt werden kann. Entsprechend soll das System zwar die wichtigsten Anforderungen umsetzen, beinhaltet aber z.B. nur ein minimalistisches Frontend-Design.

2 Vorgehensweise

Mit dem Ziel, einen groben Überblick über das System und dessen Anforderungen zu erhalten, finden zu Beginn ausführliche Gespräche mit einer Ansprechpartnerin des Kunden statt. Dabei geht es vor allem um fehlende Funktionalitäten der bestehenden Lösung, die Formate der Persönlichkeitstests und der zugehörigen Antworten sowie zukünftige Pläne für das System.

Anschließend beginnt die Planung und Entwicklung des Systems, wobei die Gespräche mit der Ansprechpartnerin fortgesetzt werden. In diesen werden neu aufgekommene Fragen zu den Anforderungen, der aktuelle Entwicklungsstand und zukünftige Anforderungen besprochen. Überschaubare Themen werden schriftlich und umfangreichere oder komplexe Themen telefonisch geklärt. Insbesondere in der Endphase der Bachelorarbeit finden die Gespräche mit Bildschirmübertragung statt, um mit Hilfe von Usability-Tests[5] die Benutzbarkeit des Systems zu testen oder die Ansprechpartnerin in das System einzuarbeiten.

Durch dieses Vorgehen bekommt die Entwicklung einen agilen Charakter. Es gibt dabei jedoch keine festen Sprints und eher unregelmäßige Gespräche, die auf Abruf vereinbart werden. Um diese zu erleichtern und Usability-Tests und Workshops[6] durchführen zu können, wird der Code schon frühzeitig mit einem Cloud-Service[7] verbunden und regelmäßig dort deployt. Somit ist es der Ansprechpartnerin stets möglich, einen relativ aktuellen Stand des Systems in einer Produktionsumgebung[8] zu betrachten.

Eine nachträgliche Anpassung einer Anforderung aus den Gesprächen ist beispielsweise, dass das Generieren eines Zertifikats nur mit einem gültigen Token möglich sein soll. Dafür soll ein Token seine übrigen Nutzungen, die ggf. auch unbegrenzt sein können, speichern. Der Token soll zeitlich begrenzt sein, sofern dies für diesen nicht anders eingestellt wurde. Eine neu hinzugekommene Anforderung ist, dass es zu Studienzwecken möglich sein soll, vor dem eigentlichen Persönlichkeitstest weitere Informationen von den Testpersonen abzufragen. Da diese Antworten nicht ausgewertet werden sollen, müssen diese Fragen als separate Tests vorliegen, was die Unterscheidung zwischen verschiedenen Testarten sowie die Verknüpfung von Testantworten erfordert. Zudem sollen die Token so konfigurierbar sein, dass ausgewählt werden kann, welcher der in der Datenbank gespeicherten Tests von den Testpersonen, die den Token nutzen, durchlaufen werden soll.

Nach dem initialen Erstellen des Grundaufbaus des Systems werden weitere Anforderungen in allen Komponenten parallel eingeführt. Dies bedeutet, dass zunächst nicht nur eine Komponente für eine Änderung angepasst wird, sondern dass alle Komponenten für eine Änderung parallel angepasst werden. Es wird also beispiels-

[5] Testet Benutzbarkeit von typischen Aufgaben einer Software; weitere Informationen: https://de.wikipedia.org/wiki/Usability-Test

[6] Gespräche, in denen Themen praktisch erarbeitet und erlernt werden; weitere Informationen: https://de.wikipedia.org/wiki/Workshop

[7] Dienst, zum Mieten von Servern; weitere Informationen: https://de.wikipedia.org/wiki/Cloud_Computing#Servicemodelle

[8] Auch Produktivumgebung; ist die Software-Umgebung, auf der das System vom Kunden genutzt wird; weitere Informationen https://de.wikipedia.org/wiki/Bereitstellungsumgebung#Unterscheidung

weise nicht zunächst das Frontend bis zur vollsten Zufriedenheit des Kunden optimiert, bevor mit Backendanpassungen begonnen wird.

Nach dem Umsetzen der wichtigsten Anforderungen sollen auch automatisierte Tests geschrieben werden. Abschließend wird die Dokumentation des Systems angelegt, zu der jeweils ein Handbuch für die Administration und Entwicklung gehört.

Während der Erstellung der automatischen Tests und der Dokumentation sowie in der Zeit danach gibt es neben Gesprächen zum System auch einen praktischen Test sowie einige Workshops. Diese finden mit Microsoft Teams statt, da der Ansprechpartnerin der Umgang mit Microsoft Teams bereits bekannt ist. Der praktische Test wird als Usability-Test durchgeführt, der testet, ob sich die Persönlichkeitstests auf der Webseite leicht genug durchführen lassen. Bei diesem wird ebenfalls die Erstellung eines neuen Tokens erklärt. Der Usability-Test findet mit zwei weiteren Stakeholdern[9] statt, nachdem alle wichtigen Anforderungen umgesetzt worden sind und sich das System als Ganzes betrachten lässt. Da hier die Entwicklung noch nicht abgeschlossen ist, könnten Änderungen noch problemlos vorgenommen werden, ohne dass man für diese auch die automatischen Tests oder die Dokumentation anpassen müsste. Die übrigen Workshops zielen darauf ab, der Ansprechpartnerin die Nutzung des Systems, sowie die genutzten Technologien näher zu bringen. Der erste geht um die Verbindung zur Datenbank und das Auslesen von Daten aus dieser. Im darauffolgenden wird das Thema der Datenbank noch weiter vertieft, in dem u.a. die groben Prinzipien von Datenbanken erklärt werden. Dabei werden auch wichtige Funktionen, wie das Exportieren von Daten, besprochen. Abschließen gibt es noch einen Workshop dazu, wie neue Persönlichkeitstests erstellt und hinzugefügt werden können und was dabei zu beachten ist.

Während der Entwicklung liegt ein Schwerpunkt auf der leichten Weiterentwickelbarkeit. Dafür ist es wichtig, dass neue Entwickler*innen den Code schnell verstehen können. Das kann z.B. erreicht werden, indem man sich an Konventionen hält oder Variablen, Funktionen, Klassen und Module bedeutsame Namen gibt [3]. Ebenfalls sollte das Projekt übersichtlich und verständlich strukturiert sein, sodass der für eine*n Entwickler*in relevante Code auch gefunden werden kann [5]. Die Strukturierung kann dabei u.a. durch Modularisierung und Refactorings verbessert werden [2]. Auch die Dokumentation des Projekts ist für neue Entwickler*innen entscheidend, da sie als Einstieg dient, eine erste Übersicht über das System verschafft und weitere Informationen verlinkt.

Für die Verständlichkeit des Codes ist insbesondere die Kombination aus treffender Benennung und Modularisierung oft sehr hilfreich. Indem beispielsweise ein komplexerer Codeabschnitt in eine Methode ausgelagert wird, kann die Verständlichkeit stark erhöht werden. Ein Beispiel dazu aus dem Code:

Code ohne Auslagerung:

```
if token is None or (token.creation_timestamp is not None and
   token.is_expired() or token.max_usage_count is not None and
   token.max_usage_count <= 0):
  abort(401, "Token not found or invalid.")
```

[9] Personen, die ein Interesse am Ergebnis haben; weitere Informationen: https://de.wikipedia.org/wiki/Stakeholder

Code mit Auslagerung:

```
if token is None or token.is_invalid():
  abort(401, "Token doesn't exist or is expired.")
```

Beide diese Ausschnitte haben die gleiche Bedingung, ihre Verständlichkeit ist aber sehr unterschiedlich. So ist es durch die Auslagerung nicht nötig zu wissen, welcher Attribute das Objekt token hat und wie sie genutzt werden müssen.

3 Technologie Stack

Zu Beginn der Entwicklung muss zunächst ausgewählt werden, welche Technologien für das System genutzt werden sollen. Dabei steht vor allem eine Survey-Technologie im Mittelpunkt. Diese kann eine Bibliothek, Framework, Programmiersprache oder ähnliches sein, die das Darstellen und Durchführen der Persönlichkeitstests, was die zentralen Funktionen des Systems sind, ermöglicht.

Die weiteren Technologien sollten an die Survey-Technologie angepasst ausgewählt werden, sodass sie die übrigen Anforderungen und ggf. zukünftige Anforderungen möglichst gut umsetzen lassen. Diese weiteren Technologien müssen für das Frontend, Backend, Code Organisation und Datenbank ausgesucht werden.

3.1 Survey

Die Survey-Technologie muss auch zukünftig möglichst einfach potenziell sehr verschieden aufgebaute Persönlichkeitstests durchführen können. Diese können sich verändern oder in verschiedenen Versionen vorliegen. Das sollte aber nicht dazu führen, dass Änderungen am System nötig sind. Stattdessen sollen Administrator*innen diese Änderungen vornehmen können, während das System läuft. Dementsprechend muss die Survey-Technologie diese Anforderungen unterstützen. Zudem muss es möglich sein das System so anzupassen und weiterzuentwickeln, dass es den Anforderungen des Kunden gerecht wird.

Da es grundsätzlich viele Technologien gibt, die diese Anforderungen an den Aufbau und die Durchführung der Persönlichkeitstests erfüllen, werden auch noch weitere Kriterien für die Auswahl mit einbezogen. Zu diesen gehören vor allem, welche Anforderungen an die Backend-Technologien entstehen oder wie die Persönlichkeitstests von Administrator*innen erstellt oder angepasst werden können. Einige mögliche Survey-Technologien, die verschiedene Ansätze verfolgen, können Tabelle 1 entnommen werden.

Diese Survey-Technologien unterstützen alle das Erstellen, Speichern und Durchführen von Persönlichkeitstests, unterscheiden sich aber vor allem hinsichtlich ihrer benötigten Backend-Technologie.

Sowohl LimeSurvey als auch formr sind bereits vollständige Applikationen, die als Webserver genutzt werden können. Einerseits können diese dadurch initial recht schnell genutzt werden, andererseits erschwert es aber auch die Umsetzungen weiterer Anforderungen. Falls man neue Anforderungen direkt in einen der beiden Dienste integrieren wollte, wäre dies auf Grund des Umfangs und der Komplexität der Applikationen vermutlich sehr aufwendig. Zudem müssten dann alle weiteren Technologien an diese Applikation anpasst werden. Das könnte behoben werden, indem LimeSurvey oder formr als eigener Webserver läuft, der von einem zweiten Webserver für die Erstellung und Durchführung der Surveys genutzt wird. Dadurch entstehen aber zwei zu verwaltende Webserver, was die Wartbarkeit und Weiterentwickelbarkeit verschlechtern würde. Es gäbe auch die Möglichkeit eine der beiden Applikationen zu mieten. Dies würde die Weiterentwickelbarkeit aber nicht vereinfachen, da Entwickler*innen trotzdem mit der Applikation und deren API vertraut sein müssen, an dieser aber keine Änderungen vornehmen könnten. Zusätzlich kann das Mieten

Technologie-Name	**Backend-Technologie Anforderungen**	**Survey-Erstellung**
surveyor [23] (Ruby-Framework)	Muss mit Ruby on Rails Backend genutzt werden	Surveys werden mit einer domänenspezifischen Sprache[10] (DSL) definiert
SurveyJS [22] (JavaScript-Bibliothek)	*Keine Anforderungen*	Surveys werden mit GUI auf Homepage von SurveyJS erstellt (oder mit Lizenz auch auf eigener Webseite)
django-survey [7] (Python-Framework Erweiterung)	Muss mit Python & Django-Backend genutzt werden	Surveys werden mit mitgelieferten GUI erstellt
LimeSurvey [13] (PHP-Applikation)	Beinhaltet eigenes PHP-Backend	Surveys werden mit mitgelieferten GUI erstellt
formr [9] (PHP-Applikation / Survey-Framework))	Beinhaltet eigenes PHP-Backend	Surveys werden durch Google-Spreadsheets mit bestimmtem Format definiert

Tabelle 1: Verschiedene Survey-Technologien, deren Anforderungen an das Backend und einer Beschreibung, wie Surveys erstellt werden.

teilweise hohe, regelmäßige Kosten verursachen, sodass sich beide Technologien nur bedingt als Survey-Technologie eignen [13, 9].

Die drei Alternativen surveyor, SurveyJS und django-survey sind alle keine fertigen Applikationen, sondern Frameworks oder Bibliotheken. Bei diesen treten also nicht die eben erwähnten Probleme auf, da sie darauf ausgelegt sind, in ein anderes System integriert zu werden.

Auf der Grundlage von Ruby on Rails bietet surveyor eine DSL an mit der es möglich ist, Persönlichkeitstests zu erstellen und anschließend durchzuführen. Für technisch versierte Administrator*innen wäre das Erstellen der Persönlichkeitstests mit Hilfe der DSL nach einer vorherigen Einführung vermutlich kein größeres Problem. Falls aber ein längerer Zeitraum vergeht und manche Inhalte der Einführung vergessen wurden, könnte die Nutzung der DSL schwieriger werden. Da anzunehmen ist, dass es größere Pausen zwischen dem Erstellen oder Ändern verschiedener Tests gibt, erscheint das Nutzen einer DSL langfristig nicht sinnvoll [23].

Sowohl mit SurveyJS als auch django-survey können Persönlichkeitstests über eine GUI erstellt werden. Diese ist bei django-survey direkt mit enthalten. Bei SurveyJS gibt es eine GUI zum Erstellen auf der Homepage von SurveyJS; der erstellte Persönlichkeitstest kann dann als JSON exportiert werden. Nach Erwerb einer entsprechenden Lizenz, könnte man die GUI auch direkt auf der eigenen Seite einbinden, da es aber voraussichtlich nicht häufig Änderungen an den Tests geben wird, ist das aktuell nicht nötig (Hinweis: Die SurveyJS Bibliothek selbst ist Open Source). Die Erstellung der Persönlichkeitstests ist also mit SurveyJS und django-survey recht ähnlich [22, 7].

Da django-survey auf dem Python-Framework Django aufbaut, muss mit diesem ein Django basiertes Backend genutzt werden. SurveyJS ist hingegen eine JaveScript-

Bibliothek und erfordert somit nur die Nutzung von JavaScript im Frontend. SurveyJS nutzt zum Anzeigen der Persönlichkeitstests den zuvor erstellten JSON-Export der Testerstellung und gibt die Antworten ebenfalls als JSON zurück. Eine API für diese JSON-Daten ist die einzige Anforderung an die Backend-Technologien, was ein großer Vorteil von SurveyJS ist. So können Backend-Technologien besser an die Anforderungen angepasst werden. Weitere Vorteile, die dazu führen, dass die Entscheidung der Survey-Technologie auf SurveyJS fällt, sind folgende: [22]

- Die Unterstützung einer großen Anzahl verschiedener Fragentypen
- Der Code wird aktiv gewartet und weiterentwickelt
- Es gibt direkte Unterstützung für verschiedene Frameworks wie Angular, jQuery, Knockout, React und Vue.js
- Relativ modernes Design
- Hohe Freiheit beim Aufbau, sodass Fragen
  - auf mehrere Seiten verteilt werden können,
  - optional oder obligatorisch sein können,
  - eingegebene Werte validieren können und
  - nur unter bestimmten Bedingungen angezeigt werden können

3.2 Frontend

Da SurveyJS eine JavaScript-Bibliothek ist, muss das Frontend webbasiert sein. Da es sich aber ohnehin um eine Webseite handelt, ist dies selbstverständlich. Grundsätzlich ist somit SurveyJS mit allen JavaScript-Framework kompatibel oder unterstützt diese sogar teilweise direkt, wodurch die weiteren Frontend-Technologien frei gewählt werden können.

Abgesehen von der Durchführung und Erstellung der Persönlichkeitstests gibt es im Rahmen der Bachelorarbeit nicht viele Anforderungen an das Frontend. Es müssen keine komplexen Seitenstrukturen, Übersichten oder ähnliches generiert werden. Es ist lediglich nötig einen Zugangstoken abzufragen, SurveyJS zu nutzen und ein generiertes Zertifikat als Ergebnis anzuzeigen. All diese Funktionen sind schon nativ von HTML, CSS und JavaScript unterstützt und brauchen keinen großen Implementierungsaufwand. Da das Frontend also entsprechend klein ist und auch kein Fokus auf dem Design liegt, ist hier ein Framework wie Angular oder React nicht nötig. Ein weiterer Vorteil daran kein Framework zu nutzen wäre, dass bei Bedarf einer größeren Weiterentwicklung am Frontend ein*e anderer*e Entwickler*in dafür ein gut geeignetes Framework auswählen könnte, ohne Rücksicht auf bestehende Technologien nehmen zu müssen.

3.3 Backend

Die einzige Anforderung des Frontends an das Backend ist, dass es eine entsprechende API bereitstellt, um unter anderem Informationen für SurveyJS zu bekommen

oder zu schicken. Damit wären grundsätzlich die meisten Backend-Web-Frameworks dafür geeignet. Damit die Auswahl etwas eingegrenzt wird, sollte zunächst die Programmiersprache, mit der man das Web-Framework nutzt, entschieden werden.

Um die Anforderung der leichten Weiterentwickelbarkeit auch hier umzusetzen, sollte es möglichst leicht sein, Entwickler*innen für die Programmiersprache des Backends zu finden. Wenn diese zu unbekannt ist, wäre es zu zeitaufwendig eine*n Entwickler*in für diese zu finden oder ein*e Entwickler*in müsste die Programmiersprache zunächst lernen. Dieses Problem kann umgangen werden, indem eine möglichst weit verbreitete Programmiersprache gewählt wird. Um zu messen, wie verbreitet eine Programmiersprache ist, können unterschiedliche Kriterien herangezogen werden, wie z.B. Umfragen, Menge des veröffentlichten Codes oder Trefferanzahlen in Suchmaschinen. Eine Übersicht der populärsten Programmiersprachen nach unterschiedlichen Kriterien kann Tabelle 2 entnommen werden.

Platzierung	**Stack Overflow Umfrage [20]**	**Github Top Languages [11]**	**PYPL-Index (Google Tutorial Suchen) [15]**	**Tiobe-Index (Trefferhäufigkeit größter Webseiten mit Suche) [24]**
1	**JavaScript**	**JavaScript**	**Python**	**Python**
2	HTML/CSS	**Python**	**Java**	C
3	SQL	**Java**	**JavaScript**	**Java**
4	**Python**	**TypeScript**	**C#**	C++
5	**Java**	**C#**	C/C++	**C#**

Tabelle 2: Die populärsten fünf Programmiersprachen, nach unterschiedlichen Quellen, mit unterschiedlichen Metriken, wobei typische Programmiersprachen für Webserver fett gedruckt sind.

Dabei fällt auf, dass Java, sowie insbesondere JavaScript und Python, besonders häufig genannt werden. Es ist also davon auszugehen, dass es auch zukünftig kein Problem sein sollte, Entwickler*innen für diese Sprachen zu finden. Jede dieser Programmiersprachen bietet viele verschiedene Web-Frameworks, sodass die genauere Auswahl der Programmiersprache primär von den weiteren Anforderungen an das System abhängt.

Für die Anforderungen, die innerhalb dieser Bachelorarbeit umgesetzt werden sollen, würden sich Java, Python oder JavaScript nahezu gleich gut eignen. Wenn man potenzielle zukünftige Anforderungen betrachtet, gibt es mehrere, bei denen Datenvisualisierung und -auswertung mit enthalten sein könnte. Beispielsweise könnten die nächsten Anforderungen die Visualisierung der Ergebnisse auf einem Zertifikat oder der durchschnittlichen Antworten einer Gruppe beinhalten. Dementsprechend sollte die Backend-Programmiersprache auch das Visualisieren und Auswerten von Daten unterstützen. Weiterhin könnte es zukünftig erforderlich sein, dass die Persönlichkeitstests auch KI-basiert ausgewertet werden. Es wäre also vorteilhaft, wenn die Backend-Programmiersprache auch dies unterstützt.

Da sich von den Programmiersprachen Java, Python und JavaScript insbesondere Python zur Datenverarbeitung eignet und bei Bedarf viele Machine-Learning-Frameworks anbietet, wird Python als Backend-Programmiersprache genutzt. Ebenfalls ist Python leicht zu lernen [17] und bietet eine gute Lesbarkeit des Codes [18]. So ist es auch für Nicht-Python-Programmierern*innen leichter, sich in den Code einzuarbeiten. Zu Beginn der Bachelorarbeit wurde die zu diesem Zeitpunkt aktuelle Python-Version 3.10.1 genutzt. Während der Entwicklung kam aber die neue Version 3.10.2 heraus, die u.a. auf Bugfixes ausgelegt ist, weswegen die ursprüngliche Version auf 3.10.2 geupgradet wurde.

Auch bezüglich Backend-Web-Frameworks bietet Python eine sehr große Auswahl an. Bei der Auswahl eines solchen Web-Frameworks gelten zunächst ähnliche Kriterien wie für die Backend-Programmiersprache; es sollte weit verbreitet und einfach zu nutzen sein. Tabelle 3 kann eine Übersicht über die meistgenutzten Python-Web-Frameworks nach unterschiedlichen Kriterien entnommen werden.

	Statista Umfrage (von Entwickler*innen zu: meist genutzte Frameworks) [21]	**Stack Overflow (Fragenanzahl mit Framework als Tag) [10]**	**Github (Sterneanzahl des Framework-Repository) [10]**
1	Flask	Django	Django
2	Django	Flask	Flask
3	FastAPI	Tornado	FastAPI

Tabelle 3: Top drei der meistgenutzten Python-Web-Frameworks nach unterschiedlichen Kriterien; in den Quellen genannte, nicht Python basierte Web-Frameworks werden hier ausgelassen.

Hier stechen vor allem Flask und Django heraus, die jeweils die ersten beiden Plätze bei allen Kategorien belegen.

Bei Flask handelt es sich um ein Microframework. Dies bedeutet, dass es nur wenige Zusatzfunktionen, neben dem Verarbeiten von HTTP-Anfragen, beinhaltet. So können weitere Frameworks, zum Beispiel für objektrelationale Abbildungen[11] (ORM), frei gewählt werden. Es werden also nur die wichtigsten Zusatzfunktionen mitgeliefert, wie z.B. eine Unterstützung für Unit-Tests. Dementsprechend eignet sich Flask vor allem für Applikationen die kleiner sind oder nur eine API bereitstellen sollen, da für diese ein Großteil der Zusatzfunktionen nicht benötigt wird [8].

Bei Django handelt es sich um ein Full-Stack-Framework. Anders als bei einem Microframework, sind bei einem Full-Stack-Framework viele Zusatzfunktionen enthalten, die man in seltenen Fällen brauchen könnte. Zu diesen gehören oft das backendseitige Generieren von HTML-Elementen, das Validieren von Eingaben oder ein mitgeliefertes ORM-Framework. Durch die große Anzahl der bereits integrierten Funktionen erlaubt Django vor allem am Anfang eine schnelle Entwicklung, da nicht zuerst entsprechende Bibliotheken oder Frameworks gesucht werden müssen. Hinzu kommt, dass keine eigene Projektstruktur entworfen werden muss, da auch diese von

[11] Darstellen von Tabellen als Klassen und Tabellenzeilen als Objekte; weitere Informationen: https://de.wikipedia.org/wiki/Objektrelationale_Abbildung

Django vorgegeben wird. Im Gegenzug kann man das Projekt aber nicht so frei wie mit Flask strukturieren, wo die Projektstruktur beliebig sein kann [6].

Sowohl Flask als auch Django haben viele Vorteile. Während Flask eher das Ziel verfolgt mit möglichst wenig Aufwand eine API erstellen zu können und dem oder der Entwickler*in dabei möglichst viel Freiraum zu lassen, zielt Django mehr darauf ab, eine schnelle Entwicklung für häufig größere Projekte zu ermöglichen. Da das System einen kleinen bis mittelgroßen Umfang hat und im Backend der Fokus auf einer API für das Frontend liegt, fiel die Entscheidung des Backend-Web-Frameworks auf Flask. Ein weiterer Grund dafür war, dass Flask einfach zu nutzen ist. Beispielsweise werden nur folgende fünf Codezeilen benötigt, um einen „Hello World“-Webserver zu erstellen:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return "Hello World!"
```

Infolgedessen wird für Flask auch keine große Einarbeitungszeit für neue Entwickler*innen benötigt, was die angestrebte Weiterentwickelbarkeit verbessert.

Neben Flask als Web-Framework muss auch noch ein Framework für die Datenbankinteraktion ausgewählt werden. Es wäre zwar möglich, diese direkt über einen Database-Adapter zu machen, dennoch ist es sinnvoller, dafür ein ORM-Framework zu nutzen. Einer der Hauptgründe dafür ist, dass das ORM-Framework unabhängig vom Datenbankmanagementsystem[12] genutzt werden kann, sodass dieses zukünftig problemlos geändert werden kann. Ein weiterer Grund für ein ORM-Framework wäre, dass es Fehler beim Erstellen von SQL-Queries vermeidet. Zu diesen gehören neben Syntaxfehlern auch SQL-Injection[13]. Ein Nachteil an ORM-Frameworks wäre, dass sich diese meist schlecht für komplexe SQL-Queries eignen, in diesem Fall könnte aber immer noch normaler SQL-Code genutzt werden.

Die Wahl des zu nutzenden ORM-Frameworks fällt dabei nicht sonderlich schwer. Es gibt in Python zwar grundsätzlich mehrere ORM-Frameworks, von diesen ist aber insbesondere SQLAlchemy [19] sehr verbreitet. Zudem ist SQLAlchemy das empfohlene ORM-Frameworks für Flask, das mit der Erweiterung Flask-SQLAlchemy direkt in eine Flask-Applikation eingebunden werden kann. Dementsprechend wird SQLAlchemy als ORM-Framework genutzt.

3.4 Code Organisation

Für die Versionsverwaltung des Codes wird git genutzt. Für manche Spezialgebiete gibt es zwar auch andere sinnvolle Versionsverwaltungs-Systeme, für ein normales Projekt wie dieses ist aber git grundsätzlich der Standard. Passend dazu wird Github

[12] Organisiert und strukturiert Daten, z.B. MySQL oder PostgreSQL; weitere Informationen: https://www.bigdata-insider.de/was-ist-ein-datenbankmanagementsystem-a-615034/

[13] Ausnutzen einer Sicherheitslücke zum Ausführen von SQL-Code; weitere Informationen: https://www.w3schools.com/sql/sql_injection.asp

als Plattform genutzt, um den Code zu veröffentlichen und anderen Entwickler*innen zugänglich zu machen.

Auch muss entschieden werden, welcher Cloud- Service genutzt wird. Da die aktuelle Software as a Service Fertiglösung nicht allen Anforderungen nachkommt und es keine passende Alternative gibt, wird ein anpassbareres Cloud-Service Modell benötigt. Ein solcher könnte Infrastructure as a Service oder Plattform as a Service sein. Beide stellen zu unterschiedlichen Bedingungen Server zur Verfügung. Infrastructure as a Service stellt nur virtualisierte Computerhardware bereit. Dadurch hat man zwar eine große Kontrolle über den Service, ist aber auch selbst dafür verantwortlich, die Plattform der Software auszuwählen, zu installieren und zu integrieren. Zur Plattform gehört dabei alles, was zum Ausführen der Software benötigt wird, also beispielsweise das Betriebssystem, die Programmiersprache sowie deren benötigte Frameworks und Bibliotheken. Plattform as a Service stellt im Gegensatz zu Infrastructure as a Service neben der benötigten Hardware auch die Plattform bereit. Hier muss also nur angegeben werden, welche Version einer Programmiersprache sowie welche Bibliotheken und Frameworks gebraucht werden.

Die zusätzliche Kontrolle, die man durch das Nutzen von Infrastructure as a Service bekommt, wird nicht benötigt und würde für einen Mehraufwand sorgen. Aus diesem Grund wird Plattform as a Service als Cloud-Service genutzt.

Als Plattform as a Service Anbieter wird Heroku [12] gewählt, wofür es mehrere Gründe gibt:

- Heroku ermöglicht das Deployment des Codes mittels git. Dabei verhält sich Heroku ähnlich wie Github und kann in git als Remote hinzugefügt werden. Um eine Änderung am Code auf dem Server vorzunehmen, muss diese lediglich mit git auf die Heroku-Remote gepusht werden, was ein sehr leichtes Deployment ermöglicht.

- Ein weiterer Grund für Heroku ist die umfangreiche Dokumentation, die es zur Nutzung von Heroku gibt. Diese ermöglicht einen schnellen und leichten Einstieg.

- Ebenfalls erfordert Heroku nicht viele Konfigurationen, beispielsweise ist eine Verschlüsslung für HTTP mittels HTTPS automatisch verfügbar. Nach dem Erstellen eines Service muss zur Konfiguration auch nur ein sogenanntes Buildpack ausgewählt werden, das zur Programmiersprache passt. Heroku bietet dafür direkt nutzbare Buildpacks für einige Programmiersprachen wie Python, Ruby, Java, etc. an, für andere Technologien können aber auch externe Repositories eingebunden werden.

- Für den Service wurde das Python-Buildpack genutzt. In diesem muss über eine runtime.txt Datei die zu nutzende Python-Version angegeben werden. Die benötigten externen Python-Pakete werden dabei automatisch aus der requirements.txt Datei ausgelesen, die immer aller benötigten Pakete auflisten sollte. Ein Vorteil dieser Konfigurationsweise ist, dass beiden Dateien mit im Repository des Codes liegen, sodass Entwickler*innen Änderungen an diesen direkt vornehmen können. Zudem ist das Nutzen einer requirements.txt Datei auch von einigen IDEs unterstützt und sehr verbreitet.

- Der Heroku-Dienst kann während der Entwicklung kostenfrei genutzt werden, sodass Änderungen am Code regelmäßig auf der Produktionsumgebung deployt werden können. So können Änderungen auf der Produktionsumgebung getestet und Usability-Tests leichter durchgeführt werden.

Zur Entwicklung wird PyCharm als IDE genutzt. Neben allen gängigen IDE Funktionalitäten bietet PyCharm viele Plugins, die etwa das Arbeiten mit der Dokumentation und Umgebungsvariablen vereinfachen. Zudem gibt es eine sehr umfangreiche Dokumentation zu verschiedensten Bereichen von PyCharm. Es können auch sogenannte „Run Configurations“ gespeichert werden, die es anderen Entwickler*innen mit nur wenigen Klicks ermöglichen, potenziell komplexe Programme fehlerfrei zu starten.

3.5 Datenbank

Abschließend muss auch noch ein Datenbankmanagementsystem ausgewählt werden. Da auch die Datenbank von Heroku bereitgestellt wird, ist das wichtigste Kriterium für das Datenbankmanagementsystem die Unterstützung von Heroku. In Heroku können alle Datenbankmanagementsysteme genutzt werden, die als Add-on einer Applikation hinzugefügt werden können. Grundsätzlich gibt es für alle gängigen Datenbankmanagementsysteme, wie PostgreSQL, MariaDB, MySQL oder MongoDB, ein Add-on. Von diesen wird aber nur PostgreSQL nativ von Heroku unterstützt, wohingegen die Alternativen von Drittanbietern sind. Da durch die native Unterstützung seltener Komplikationen auftreten und es eine bessere Unterstützung, z.B. durch Dokumentation, gibt, ist PostgreSQL hier die naheliegende Wahl. Zudem kann PostgreSQL JSON Daten speichern, die auch von SurveyJS genutzt werden. Weitere Gründe, weswegen PostgreSQL als Datenbankmanagementsystem genutzt wird, wären, dass es sehr weit verbreitet und Open Source ist. So kann es z.B. zukünftig nicht passieren, dass potenziell relevante Updates das Kaufen oder Mieten einer kostenpflichtigen Version erfordern [14].

4 Implementation

Nach der Auswahl des Technologie-Stacks, wird zunächst der Projektaufbau geplant und umgesetzt sowie die Entwicklungsumgebung eingerichtet. Im Anschluss an den Grundaufbau werden die Daten modelliert, die Webseiten und Funktionen dem System hinzugefügt, sowie Tests für dieses entwickelt. In diesem Abschnitt werden abschließend noch einige Schwierigkeiten der Entwicklung hervorgehoben.

4.1 Projektaufbau

Das Projekt beinhaltet zunächst die grundlegenden Bestandteile für git, Python und Heroku. Zu diesen gehören neben den Konfigurationsdateien der IDE, von git und Heroku auch die requirements.txt Datei, welche die benötigten Python-Pakete angibt, sowie ein src Ordner für den Python-Code und ein static Ordner für konstante Dateien des Frontends. Darüber hinaus gibt es einen test und doc Ordner für die Tests und Dokumentation des Projekts. Die Ordnerstruktur ist dabei frei wählbar, da Flask keine benötigte Projektstruktur vorgibt. Innerhalb des src Ordners liegt zunächst eine run.py Datei, die den Server startet sowie das test_system Modul, das den eigentlichen Code beinhaltet. Zusammenfassend sieht der grobe Aufbau wie folgt aus (hier wurden einige Dateien und Ordner ausgelassen):

```
├──.idea
├──doc
├──src
│   ├──test_system
│   │   run.py
│
├──static
├──test
│   .gitignore
│   Procfile
│   requirements.txt
│   runtime.txt
```

Neben der hier gezeigten Ordnerstruktur gibt es noch weitere Unterordner, zur besseren Organisation. Dabei liegt vor allem ein Fokus auf der Strukturierung des test_system Submodules. Für ein Webframework gibt es dafür zwei häufig genutzte Möglichkeiten:

1. Die für eine Route benötigten Dateien werden in einen Unterordner zusammenfasst. Dabei würde es beispielsweise pro Unterordner jeweils eine Datei für die Flask-Route und das ORM-Model geben.

2. Dateien werden nach ihrer Funktion gruppiert, sodass alle Flask-Routen in einem Ordner und alle ORM-Modelle in einem anderen Ordner liegen.

Da manche ORM-Modelle von mehreren Routen genutzt werden, wäre eine eindeutige Gruppierung nach der ersten Möglichkeit nicht möglich, weswegen die zweite genutzt wird. Dementsprechend werden die Routen in einem routes und ORM-Modelle in models Modul gespeichert. Die Geschäftslogik, die beschreibt wie Daten

in Modellen verarbeitet werden, liegt dabei mit im models Submodul. Des Weiteren gibt es noch ein managers Unterordner, in dem komplexere Aufgaben als eigene Module liegen, damit das models Modul nicht zu umfangreich wird. So beinhaltet das models Modul ausschließlich ORM-Modelle, deren Geschäftslogik sowie den datenbankbezogenen Code. Dieser Aufbau ist angelehnt an eine Model-View-Controller-Architektur[14], deren View-Code, der im static Ordner enthaltene Frontend-Code wäre. Die Controller wären wiederum im routes und die Modelle im models Submodul gespeichert.

4.2 Entwicklungsumgebung

Das System wurde in einer Windows-Umgebung entwickelt, ist aber so aufgebaut, dass es grundsätzlich auch mit Linux oder macOS kompatibel wäre, da Python unabhängig vom Betriebssystem genutzt werden kann. Zusätzlich beinhaltet oder verlinkt die Dokumentation bei Anleitungen auch entsprechende Schritte für alle Betriebssysteme, falls sich diese unterscheiden.

Für das System wird zudem eine virtuelle Python-Umgebung genutzt. Diese ermöglicht es, Python-Pakete mit bestimmten Versionen zu nutzen, unabhängig von den systemweit installierten Python-Paketen. So kann es nicht passieren, dass diese geändert werden müssen oder Komplikationen durch falsche Versionen entstehen. Ebenfalls vereinfacht dies das Nutzen der requirements.txt Datei, da diese nur mit den Python-Paketen der virtuellen Umgebung übereinstimmen muss.

Als Datenbankmanagementsystem wird neben der Produktionsumgebung auch lokal PostgreSQL genutzt. Hier würde zwar ggf. SQLite als leichtere Lösung erscheinen, dies könnte aber zu Kompatibilitätsproblemen führen. Beispielsweise nutzen mehrere Tabellen den Datentyp JSON, der von PostgreSQL direkt unterstützt wird, von SQLite hingegen nicht. So wäre das ORM-Framework zwar dazu in der Lage, mit dem Datenbankmanagementsystem zu interagieren, SQLite könnte aber die Tabelle mit dem JSON-Datentyp nicht verarbeiten. Um solche Probleme zu vermeiden, wird für die lokale Umgebung ebenfalls PostgreSQL genutzt, was auch der von Heroku empfohlene Weg ist. Dementsprechend wird auch für die Testumgebung, die für automatische Tests genutzt wird, PostgreSQL verwendet.

Während dieser Bachelorarbeit wird in git auf dem main Branch gearbeitet. Dies ist zwar ungewöhnlich, da während der Zeit aber nie mehr als eine Person am Code arbeitet und es kein automatisches Deployment gibt, war ein anderer Branch nicht nötig. Der Hauptgrund, warum kein automatisches Deployment des Systems genutzt wird ist, dass die Entscheidung eines Deployments eine bewusst getroffene Entscheidung des oder der Entwickler*in sein soll. Voraussichtlich wird sich das System auch nicht oft ändern, wodurch die Zeit und Arbeit, die durch ein automatisches Deployment gespart wird, minimal wäre. Falls der Bedarf entsteht, wäre es auch später noch sehr leicht möglich, Heroku mit Github zu verbinden, da dies von Heroku direkt unterstützt wird.

[14] Gruppiert Dateien nach Model, View und Controller, abhängig von deren Funktionen; weitere Informationen: lhttps://de.wikipedia.org/wiki/Model_View_Controller

4.3 Modellierung der Daten

Zum Modellieren der Daten wird das das ORM-Framework SQLAlchemy genutzt. Dieses ermöglicht, wie die meisten ORM-Frameworks, die Tabellen der Datenbank als ORM-Klassen zu definieren und auf die Daten ohne SQL-Queries zuzugreifen. Die Ergebnisse solcher Zugriffe werden dann als Objekte der entsprechenden ORM-Klassen zurückgegeben. Dementsprechend muss für jede Tabelle eine dazu passende ORM-Klasse erstellt werden.

Zunächst gibt es die Tabelle test. Diese beinhaltet die als JSON kodierten Tests, sowie deren Namen und Kategorien. Die Kategorie eines Tests gibt dabei an, wofür dieser genutzt werden kann. Es gibt folgende Kategorien:

- PERSONAL_DATA_TEST: Sammeln Personen bezogene Daten vor der Durchführung eines Persönlichkeitstests.
- PRE_COLLECT_TEST: Sind zwar Teil eines Persönlichkeitstests, werden aber nicht ausgewertet. Somit können sie beliebige Informationen abfragen, die z.B. für Studienzwecke gebraucht werden.
- EVALUABLE_TEST: Sind der eigentliche Persönlichkeitstest und werden ausgewertet.

Der Zugang zu einem Persönlichkeitstest erfolgt dabei über einen Token, der in einer weiteren token Tabelle gespeichert wird. Neben dem eigentlichen Zugangstoken beinhaltet diese noch Informationen zur Gültigkeit, wie beispielsweise eine Nutzungsanzahl, sowie die Namen der Tests, die mit diesem Token durchgeführt werden sollen.

Ein solcher Token kann dabei von einem oder einer Administrator*in hinzugefügt werden. Die Autorisierung dafür erfolgt über einen Nutzernamen und ein Passwort, die in einer user Tabelle gespeichert werden.

Die Antworten eines Persönlichkeitstests werden auf mehrere Tabellen verteilt gespeichert. Die Antwort eines PERSONAL_DATA_TESTS wird in der Tabelle person gespeichert, deren Spalten zu den abgefragten Informationen des Tests passen müssen. Antworten für einen Test einer anderen Kategorie werden in der Tabelle test_answer gespeichert. Neben den Antworten speichert diese Tabelle auch wann, für welchen Test und von welcher Person die Antworten abgegeben werden. Für EVALUABLE_TEST werden, neben den Daten in der test_answer Tabelle, die auswertbaren Antworten zusätzlich in einer evaluable_question_answer Tabelle abgespeichert. Diese Tabelle speichert zudem noch jeweils eine Referenz zu der Tabelle evaluable_question_name, die Namen und Kategorie einer Frage beinhaltet, und evaluable_test_answer. Die Tabelle evaluable_test_answer speichert dabei, welche test_answer sie erweitert sowie mit welchem Token die Antworten ausgewertet wurden.

Diese Beziehungen der Tabellen, können auch dem ER-Diagramm im Anhang entnommen werden (siehe Anhang A3). Die Ordnerstruktur, die sich daraus für das models Modul ergeben hat, ist wie folgt:

```
models
|    database.py
|
|---table_models
|    |    test.py
|    |    token.py
|    |    user.py
|    |
|    |---answers
|    |    |    person.py
|    |    |    test_answer.py
|    |    |
|    |    |---evaluable_answer
|    |    |    |    evaluable_question_answer.py
|    |    |    |    evaluable_question_name.py
|    |    |    |    evaluable_test_answer.py
```

Wobei database.py SQLAlchemy initialisiert und die Datenbankverbindung herstellt.

4.4 Webseiten & Funktionen

Das System beinhaltet vier Webseiten. Zunächst gibt es eine Seite, mit deren Hilfe die Administrator*innen einen neuen Token mit einer entsprechenden Konfiguration generieren lassen können. Die restlichen Seiten werden zum Durchlaufen eines Persönlichkeitstest genutzt.

Auf der Startseite kann ein Token eingegeben werden, um einen Persönlichkeitstest zu starten. Ist dieser valide, wird man auf die nächste Seite weitergeleitet, wo die im Token hinterlegten Tests durchlaufen werden. Nach Abschluss der Tests, gelangt man auf die letzte Seite, auf der einem eine schemenhafte PDF-Datei als Zertifikat generiert wird. Diese beinhaltet die angegebenen personenbezogenen Daten und die Ergebnisse der ausgewerteten Fragen.

Es gibt dabei aber keine Webseite zum Hinzufügen und Bearbeiten von Tests. Dies liegt zum einen daran, dass diese Funktion, im Gegensatz zur Tokenerstellung, voraussichtlich nur sehr selten gebraucht wird. Zum anderen besteht kein wirklicher Bedarf für diese Webseite. Zum Erstellen und Bearbeiten der Tests wird die GUI auf der Webseite von SurveyJS genutzt, da andernfalls eine kostenpflichtige Lizenz für diese erworben werden müsste, die sich voraussichtlich nicht für den Kunden lohnt. Die dort importierten und exportierten Tests liegen als JSON vor. Eine Webseite, die das Hinzufügen und Bearbeiten der Tests ermöglicht, müsste also die bestehende test Tabelle anzeigen können sowie neue Tests zu dieser hinzufügen oder das JSON bestehender Tests überschreiben können. Dies gehört aber zu den Grundfunktionen einer jeden Datenbank-Administrationssoftware. Da die Ansprechpartnerin aber ohnehin direkten Zugriff auf die Datenbank haben soll, kann sie die Tests direkt in der Datenbank hinzufügen und bearbeiten, wodurch eine dafür zuständige Webseite überflüssig wäre. Die einzige Voraussetzung dafür ist, dass die Ansprechpartnerin eine Datenbank-Administrationssoftware hat und mit dieser ausreichend umgehen kann. Hier wäre naheliegend, Excel dafür zu nutzen, da die Ansprechpartnerin be-

reits Erfahrung mit Excel hat. Dafür wäre aber einerseits ein Plugin nötig, dessen Installation komplex ist, und andererseits würde dies das Speichern von Änderungen deutlich erschweren. Die einfachere Lösung ist pgAdmin, was die Standardsoftware zur GUI basierten Verwaltung von PostgreSQL-Datenbanken ist. Diese ist für alle gängigen Betriebssysteme verfügbar und kann mit der Hilfe eines Installers installiert werden, was einfacher als die Installation eines Excel-Plugins ist. Dies hat u.a. den Vorteil, dass zukünftig die Ansprechpartnerin wichtige Updates selbst installiert kann. pgAdmin an sich ist zwar sehr umfangreich, die meisten der Funktionen werden aber nicht benötigt, weswegen dies kein größeres Problem darstellt. Es muss lediglich die Datenbank hinzugefügt, die Tabellen aufgerufen und Daten in diesen geändert werden. Zudem sind all diese Funktionen, das Speichern von Änderungen und das Exportieren von Tabellen im Benutzerhandbuch dokumentiert. Diese Funktionen sowie die Grundprinzipien von Datenbanken und der Nutzung von pgAdmin wurden der Ansprechpartnerin zusätzlich in den Workshops nähergebracht, sodass das Benutzerhandbuch nur zum Nachlesen dieser Tätigkeiten dient.

4.5 Tests

Zur leichteren Weiterentwickelbarkeit und der Vermeidung von Fehlern, soll das System auch über automatisierte Tests verfügen. Das dafür genutzte Test-Framework ist pytest [16], das sehr verbreitet und leicht zu nutzen ist. So werden beispielsweise von Python native assert Statements für Tests genutzt. Zudem lassen sich mit sogenannten fixtures leicht Variablen erstellen und initialisieren, die in mehreren Tests genutzt werden können. Diese können ebenfalls zum Initialisieren von Verbindungen und Sessions der Datenbank genutzt werden. Solche fixtures lassen sich in Tests nutzen, indem der Name dieser in der Testfunktion als Parametername angegeben wird. Ein Beispiel für einen solchen Test wäre folgender Code:

```
import pytest

def get_fruit():
    return "apple"

# test:

@pytest.fixture
def valid_fruits():
    return ["apple", "banana"]


def test_fruits(valid_fruits):
    assert get_fruit() in valid_fruits, "Invalid fruit!"
```

Listing 1: geändert von [16].

Da die Module des Systems nicht sehr umfangreich sind, liegt der Fokus der automatisierten Tests auf der korrekten Zusammenarbeit der einzelnen Komponenten. Solche Tests heißen Integrationstests und sollen sicherstellen, dass die Komponenten des Systems problemlos miteinander interagieren können. Um dies zu überprüfen,

testen die automatisierten Tests direkt die Routen, anstatt mit Unittests die Module oder Teile von diesen einzeln zu testen. So ist es möglich, mit wenigen Tests große Teile des Codes zu testen. Dabei werden insbesondere kritische Stellen getestet. Zu diesen gehören beispielsweise:

1. Werden übergebene Argumente korrekt verarbeitet?
2. Gibt es auf Anfragen die richtigen HTTP-Statuscodes, Antworten und Antwortformate?
3. Funktioniert die Interaktion (Erstellen, Lesen, Ändern und Löschen von Daten) mit der Datenbank richtig?

Damit wird unter anderem getestet, ob

1. nur autorisierte Nutzer*innen auf Persönlichkeitstests oder Zertifikate zugreifen können,
2. die Interaktion der Backend-API und des Frontend funktioniert, sodass das Backend damit vom Frontend genutzt werden kann und
3. Daten korrekt von der Datenbank gelesen und in die Datenbank geschrieben werden können.

Solche Integrationstests liegen dabei für alle Routen sowie einige statische Ressourcen vor. Außerdem gibt es auch noch Unit-Tests für vier Hilfsfunktionen deren Funktionalitäten essenziell sind, da sie genutzt werden, um Passwörter zu hashen und neue Zugangstoken zu generieren. Für die weiteren Module gibt es keine Unittests. Dies kommt hauptsächlich daher, dass die Integrationstests sehr ausführlich sind. So beinhalten sie für das routes Modul alle Tests, die auch die Unittests haben würden, hier wären weitere Unittests also redundant. Für das managers und models Modul könnten Unittests zwar sinnvoll sein, viele der dort enthaltenen Funktionen werden aber auch schon bei den Integrationstests getestet. Da diese dabei auch die Antworten des Servers auf deren Richtigkeit kontrollieren und diese Antworten u.a. mit den managers und models Modulen erstellt werden, würden dabei viele Fehler in diesen Modulen auffallen. Zudem haben die Integrationstest eine relativ hohe Testabdeckung. So werden 100% des routes Moduls abgedeckt, das für die Integrationstests zentral ist. Das gesamte test_system Modul hat dabei eine Testabdeckung von 94%, wobei alle Dateien durchlaufen werden. Diese niedrigere Testabdeckung kommt dabei von nicht testrelevantem Code, wie beispielsweise Datenbank-Daten, die nur initial in eine neue Datenbank hinzugefügt werden. Da normalerweise die Testdatenbank bei einem Testdurchlauf bereits initialisiert wurde, wird dieser Code also nicht ausgeführt. Falls man aber beispielsweise die Tests mit einer nicht initialisierten Datenbank laufen lässt, würde die Testabdeckung bei 96% liegen.

Abgesehen von eher unwichtigen Stellen, decken die Tests alle relevanten Codezeilen ab, was nicht überinterpretiert werden sollte. Dies sollte aber nicht überinterpretiert werden. Es handelt sich immer noch um größtenteils Integrationstests, weswegen man nicht davon ausgehen kann, dass der gesamte abgedeckte Code komplett

fehlerfrei ist. Dies kann man zwar auch bei anderen Testarten nicht, eine hohe Testabdeckung mit Unittests hätte aber grundsätzlich eine größere Aussage, da diese Tests in der Regel wesentlich detaillierter sind.

4.6 Schwierigkeiten

Bereits am Anfang, kam die neue Anforderung dazu, dass ein Persönlichkeitstest aus mehreren Teiltests bestehen kann, von denen nur einer ausgewertet werden soll. Auf dem Zertifikat sollen dementsprechend auch nur die ausgewerteten Tests enthalten sein. Diese Anforderung brauchte also zusätzliche Logik im System und eine angepasste Modellierung der Daten, da zwischen mehreren Test- und Antwortarten unterschieden werden musste. Da die Anforderung aber schon zum Beginn der Entwicklung in Gesprächen entstand, konnte diese direkt in der Planung berücksichtigt und relativ leicht umgesetzt werden.

Ebenfalls musste abgewogen werden, welche Funktionen alle Bereitgestellt werden und für welche Änderungen der Kunde eine*n Entwickler*in benötigt. Die am häufigsten genutzte Funktion, abgesehen vom Durchführen der Persönlichkeitstests, ist das Erstellen von Token, da auch dies regelmäßig passiert. Dementsprechend wurde für diese Funktion eine eigene Webseite angelegt. Einige Funktionen, die nur selten gebraucht werden, sind dabei über den direkten Zugriff auf die Datenbank mittels pgAdmin möglich, der bereits früher angesprochen wurde. Zu diesen gehören das Hinzufügen, Anzeigen, Ändern und Löschen von Tests und Token, um beispielsweise Testfragen zu ändern oder einem Token eine weitere Nutzung hinzuzufügen. Für detailliertere Änderungen, wie der Anpassung der Testauswertung oder des Aufbaus des Zertifikats, wäre aber aktuell ein*e Entwickler*in notwendig. Diese Änderungen kommen aber dabei voraussichtlich nur selten vor. Dies ermöglicht, dass relativ viele Funktionen zur Verfügung stehen, ohne einen großen Implementierungsaufwand zu haben.

Eine Schwierigkeit während der Entwicklung bestand in der Verbindung der Tests mit Flask und SQLAlchemy. Dabei hatte Flask ein sogenannten Testclient, der es ermöglicht, ohne laufenden Server, den Code zu testen und Anfragen zu schicken, was das Testen vereinfacht hat. Bei SQLAlchemy lag die Schwierigkeit vor allem auf der Verwaltung von Datenbanksessions in den Tests. Um Tests leichter erstellen zu können, sollen sich diese nicht selbst um die Datenbanksession kümmern müssen. Zusätzlich soll sich die Testdatenbank durch einen Test nicht verändern, um für nachfolgende Tests den richtigen Zustand zu haben. Dies konnte umgesetzt werden, indem die Datenbanksession mit der Hilfe einer pytest fixture erstellt wird. Diese kümmert sich dann sowohl um das Initialisieren der Session als auch um das Zurückrollen aller Änderungen an der Testdatenbank, die während des Tests vorgenommen wurden.

Eine kleinere Schwierigkeit mit dem ORM-Framework trat während des Testens auf, wenn die getesteten Routen Tabellenzeilen ändern, die als pytest fixture ebenfalls im Test genutzt werden. Für solche Änderungen müssen die Daten der fixtures, nach dem Aufruf der Route, manuell aktualisiert werden. Geschieht das nicht, werden veraltete Daten genutzt, was zunächst nicht direkt auffällt. Dies hatte für einen Bug in einem Test gesorgt, der sich aber leicht beheben ließ, nachdem er erkannt wurde.

Eine der zuletzt betrachteten Themen während der Entwicklung war die Daten-

bankmigratio[15], die es ermöglichen soll, Änderungen im Datenbankschema automatisch in bestehende Datenbanken zu übernehmen. Diese wird aber nicht nativ von SQLAlchemy unterstützt, sodass ein zusätzliches dafür zuständiges Paket als Erweiterung von SQLAlchemy genutzt werden müsste. Ein hierfür oft genutztes Paket ist alembic [1], dass von den gleichen Entwickler*innen wie SQLAlchemy stammt. Aktuell wird aber weder alembic noch ein anderes Paket für die Datenbankmigration genutzt. Das hängt u.a. damit zusammen, dass dessen Nutzung zum einen mehr Komplexität in das System bringt und zum anderen unabsichtlich irreversible Datenbankänderungen verursachen kann. Zudem ändert sich das Datenbankschema nicht häufig und die Datenbankmigration wäre vor allem für die Datenbank der Produktionsumgebung wichtig, die aber aktuell noch nicht aktiv genutzt wird. Es wäre aber durchaus denkbar, dass alambic zukünftig noch integriert wird.

[15] Automatisches Anpassen von Datenbankschemata bei Änderungen der ORM-Modelle; weitere Informationen: https://en.wikipedia.org/wiki/Schema_migration

5 Test der Weiterentwickelbarkeit

Nach der Entwicklung soll die Weiterentwickelbarkeit des Systems getestet werden. Als Kriterium für die Weiterentwickelbarkeit wurde definiert, wie leicht ein*e Entwickler*in gegebene Aufgaben umsetzen kann. Diese Testperson sollte von dem Kunden gefunden und ausgesucht werden, um dem Fall einer zukünftigen Weiterentwicklung möglichst nah zu kommen. Die Aufgaben werden dabei abhängig vom Wissenstand der Testperson sein und sollen grundsätzlich feststellen, ob sich die Testperson schnell in das System und die Code-Basis von diesem einarbeiten kann.

Die vom Kunden ausgewählte Testperson ist ein Bachelorstudent der Informatik, welcher gerade das vierte Fachsemester beendet hat. Bis zum Test hatte er zwar noch nicht mit Python gearbeitet, dafür hatte er aber beispielsweise Erfahrung in objektorientierter Programmierung, git, dem Umgang mit IDEs und den Grundlagen von HTML, JavaScript und CSS.

Nach dem Finden der Testperson, wurden die Aufgaben für sie entworfen. Diese bestanden zunächst darin, das System lokal für die Entwicklung einzurichten, zu nutzen und zu testen. Das Testen sollte dabei durch das Ausführen der automatisierten Tests erfolgen. Das Nutzen beinhaltete neben dem Ausführen der Applikation auch das Anlegen eines oder einer neuen Administrator*in mit gehashten Passwort in der Datenbank. Diese ersten Aufgaben sollten überprüfen, wie leicht der Einstieg in den Code ist. Nahezu jede Weiterentwicklung würde diese Aufgaben in der genannten oder einer ähnlichen Form beinhalten.

Die weiteren Aufgaben waren konkrete Anforderungen an das System, die zukünftig in ähnlicher Art entstehen könnten. Dabei sind diese aber so zugeschnitten, dass sie keine detaillierten Python-Kenntnisse benötigen. Die Anforderungen beinhalteten dabei das Hinzufügen eines Textes im Frontend, das Ändern eines bestimmten Wertes, sowie das Erstellen eines neuen Log-Eintrags für eine Route. Die genauen Aufgaben können dem Anhang entnommen werden (siehe Anhang A1). Zum Umsetzen der Anforderungen ist nur das Ändern oder Hinzufügen einzelner Codezeilen nötig. Dadurch besteht die Schwierigkeit eher darin, die Stelle im Code zu finden, die angepasst werden muss. Dabei spricht es für die leichte Wartbarkeit des Systems, wenn diese Stellen im Code leicht gefunden werden können [4].

Nach dem Erstellen der Aufgaben wurden diese an die Testperson weitergeleitet und geklärt, ob die Aufgabenstellung verstanden wurde. Die Testperson bekam anschließend Zugriff auf das Github-Repository, um die Aufgaben bearbeiten zu können. Nachdem die Testperson Zugriff auf den Code bekommen hatte, sollte sie nur Rückfragen stellen, wenn dies zwingend erforderlich ist. Die Testperson konnte die gestellten Aufgaben aber ohne Rückfragen lösen.

Zusätzlich sollte die Testperson für jede der Aufgaben folgende vier Fragen kurz beantworten:

- Wie lange hat die Teilaufgabe gedauert?
- Wie wurde die Aufgabe gelöst?
- Welche Probleme gab es?
- Wie konnten Probleme gelöst werden?

Die Antworten auf diese Fragen können dem Anhang entnommen werden (siehe Anhang A2).

Alle Aufgaben wurden von der Testperson erfolgreich umgesetzt. Insgesamt brauchte diese dafür nach eigenen Angaben 3,5 Stunden.

Ein kleineres, bei der Lösung der Aufgaben durch die Testperson aufgetretenes Problem war, dass während des Einrichtens der lokalen Entwicklungsumgebung eine Umgebungsvariable gesetzt werden muss. Wie das geschieht, war der Testperson unbekannt und wurde auch nicht in dem Entwicklerhandbuch erklärt oder verlinkt. Dies konnte die Testperson aber schnell lösen, indem sie auf Stack Overflow eine entsprechende Anleitung dazu fand. Nichtsdestotrotz erhöht das Recherchieren die benötigte Zeit zur Einrichtung der Entwicklungsumgebung. Um dies zukünftig für neue Entwickler*innen zu erleichtern, wurde ein entsprechender Artikel zum Anpassen von Umgebungsvariablen im Entwicklerhandbuch verlinkt.

Ein weiteres vermeintliches Problem war, dass das Anpassen einer Konfiguration vergleichsweise lange gedauert hätte. Dies kam daher, dass die benötigte Änderung nur den Wert einer Konstante anpassen musste, wofür die Testperson eine halbe Stunde gebraucht hat. Hierbei muss aber beachtet werden, dass die Testperson weder wusste, dass es sich um eine Konstante handelt, noch wo im Code der zu ändernde Wert steht und genutzt wird. Dementsprechend musste dies erst herausgefunden werden. Wenn man dabei auch noch beachtet, dass die Testperson noch nicht mit umfangreicheren Codebasen gearbeitet hat und dies ihr erster Kontakt zu Python-Code war, ist die gebrauchte halbe Stunde nicht wirklich als Problem anzusehen.

Das letzte von der Testperson genannte Problem, war dass sie noch nicht mit dem Logging in Python und Flask vertraut war. Dabei half aber die Dokumentation zu Python und Flask. Zudem wurden die Logging-Befehle anderer Routen als Orientierung genutzt, sodass auch dieses Problem leicht lösbar war und nicht auf eine schlechte Weiterentwickelbarkeit zurückzuführen ist.

6 Fazit

Das System ist am Ende dieser Arbeit so weit geplant und entwickelt, dass die grundlegenden Anforderungen umgesetzt wurden. Zu diesen gehören eine Autorisierung der Nutzer*innen, eine automatische Auswertung und das Generieren eines schemenhaften Zertifikats. Ebenfalls gibt es die Möglichkeit, studienrelevante Daten mit nicht ausgewerteten Tests abzufragen. Während der Entwicklung sind aber auch neue Anforderungen entstanden, die auf Grund ihres Umfangs nicht mehr innerhalb der Bachelorarbeit umgesetzt werden. Zu diesen gehört beispielsweise, dass sich für seine Gruppe, die den gleichen Token für den Persönlichkeitstest genutzt hat, Statistiken anzeigen lassen.

Es wurde eine agile Vorgehensweise genutzt, sodass Änderungen an den Anforderungen früh aufgefallen sind. So konnte z.B. die Anforderung einer möglichen zeitlichen Beschränkung der Token eingearbeitet werden, bevor automatisierte Tests geschrieben wurden, die eine Validierung von Token beinhalten. Andernfalls hätten diese automatisierten Tests noch angepasst werden müssen. Da das System bereits in der Entwicklung regelmäßig deployt wurde, wäre der aktuelle Stand von diesem theoretisch bereits für den Kunden nutzbar.

Mit dem Usability-Test wurde geprüft, ob das Durchführen der Persönlichkeitstests und das Erstellen der Zugangstoken aus Sicht der Kunden leicht genug möglich ist. Dabei entstanden zwar kleinere Änderungen, diese bezogen sich aber nur auf die angezeigten Texte und nicht auf Funktionalitäten, sodass der Usability-Test recht positiv ausgefallen ist.

Nach den Workshops war die Ansprechpartnerin auch in der Lage mit Hilfe des Benutzerhandbuchs und pgAdmin auf die Datenbank zuzugreifen, um etwa die übrigen Nutzungen eines Tokens anzupassen. Im letzten Workshop während der Bachelorarbeit hatte sie auch mit der zusätzlichen Hilfe der GUI auf der Webseite von SurveyJS einen ersten exemplarischen Test erstellt und dem System hinzugefügt. Dieser Workshop diente aber nur einem ersten Einblick in die Testerstellung. Allgemein ist die Testerstellung recht komplex, da für diese der Test in einem bestimmten Format erstellt und die richtige Kategorie des Tests in pgAdmin angegeben werden muss. Zudem ist die GUI zum Erstellen der Tests recht umfangreich.

Der Vorteil der Frontend-Technologie, viele Fragentypen zu unterstützen, stellte sich gegen Ende der Bachelorarbeit bereits als sehr nützlich heraus. Dort entstand während eines Workshops eine neue Anforderung. Da in manchen Fällen unbeschränkte Token ausgegeben werden könnten, soll bei diesen anfangs explizit drauf hingewiesen werden, dass der Token nicht weitergegeben oder unerlaubt genutzt werden darf. Das dies nicht getan wird, soll der oder die Nutzer*in zudem bestätigen. Das Umsetzen dieser Anforderung benötigte aber keinen Entwicklungsaufwand. Dies kam daher, dass SurveyJS neben einfachen Fragen auch die Möglichkeit bietet, HTML anzuzeigen und Unterschriften abzufragen. Damit ist es problemlos möglich in Tests einen Hinweistext anzeigen und unterschreiben zu lassen.

Neben dem Usability-Test fiel auch der Test der Weiterentwickelbarkeit gut aus. Die Testperson war dazu in der Laage alle Aufgaben umzusetzen, ohne dabei größere Probleme zu haben. Dies gelang, ohne dass die Testperson zuerst in das System eingearbeitet wurde oder Rückfragen während der Bearbeitung stellen konnte. Damit ist

die nichtfunktionale Anforderung der leichten Weiterentwickelbarkeit des Systems, soweit dies getestet werden kann, erfüllt.

Abschließend lässt sich festhalten, dass das System

- die wichtigsten Funktionen enthält,
- nutzungsbereit auf einem Server läuft,
- von der Ansprechpartnerin genutzt werden kann und
- dem Test zufolge leicht weiterentwickelbar ist.

Die Softwarestand zur Abgabe kann auf Github unter dem Commit-Hash b21aca9067db31916129eaddc8247ef55e653c69[16] gefunden werden.

[16] Der vollständige Link zum Github-Commit: https://github.com/Niwo1403/personality-test-system_bachelor-thesis/tree/b21aca9067db31916129eaddc8247ef55e653c69

Literatur

[1] *Alembic*. März 2022. URL: https://alembic.sqlalchemy.org/en/latest/.

[2] Italo Barros. *Developability*. März 2022. URL: https://towardsdatascience.com/cleaning-refactoring-and-modular-the-must-foundations-to-improve-your-python-code-and-carrer-65ef71cdb264.

[3] Brian Cline. *Developability*. März 2022. URL: https://www.brcline.com/blog/5-tips-write-maintainable-code.

[4] *Code Maintainability*. März 2022. URL: https://cloud.google.com/architecture/devops/devops-tech-code-maintainability.

[5] *Developability*. März 2022. URL: https://docs.teamscale.com/introduction/understanding-the-maintainability-of-your-code-base/#code-structure.

[6] *Django*. März 2022. URL: https://www.djangoproject.com/.

[7] *Django Survey*. Jan. 2022. URL: https://github.com/Pierre-Sassoulas/django-survey.

[8] *Flask*. the Pallets Projects. März 2022. URL: https://flask.palletsprojects.com/en/2.0.x/.

[9] *Formr*. Jan. 2022. URL: https://formr.org/.

[10] *Github Stack Overflow - Frameworks*. März 2022. URL: https://hotframeworks.com/.

[11] *Github Languages*. März 2022. URL: https://octoverse.github.com/#top-languages-over-the-years.

[12] *Heroku*. Feb. 2022. URL: https://devcenter.heroku.com/.

[13] *LimeSurvey*. Jan. 2022. URL: https://www.limesurvey.org/de/.

[14] *PostgreSQL*. März 2022. URL: https://www.postgresql.org/.

[15] *PYPL-Index*. März 2022. URL: https://pypl.github.io/PYPL.html.

[16] *Pytest*. März 2022. URL: https://docs.pytest.org/en/7.1.x/.

[17] *Python Dokumentation*. März 2022. URL: https://docs.python.org/3/tutorial/.

[18] *Python Readability*. März 2022. URL: https://docs.python-guide.org/writing/style/.

[19] *SQLAlchemy*. März 2022. URL: https://www.sqlalchemy.org/.

[20] *StackOverflow Survey*. März 2022. URL: https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages.

[21] *Statistica Frameworks*. März 2022. URL: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/.

[22] *SurveyJS*. Jan. 2022. URL: https://surveyjs.io/.

[23] *Surveyor*. Jan. 2022. URL: https://github.com/NUBIC/surveyor.

[24] *Tiobe-Index*. März 2022. URL: https://www.tiobe.com/tiobe-index/.

A Anhang

A.1 Testaufgaben

Testaufgaben

1. System einrichten & nutzen

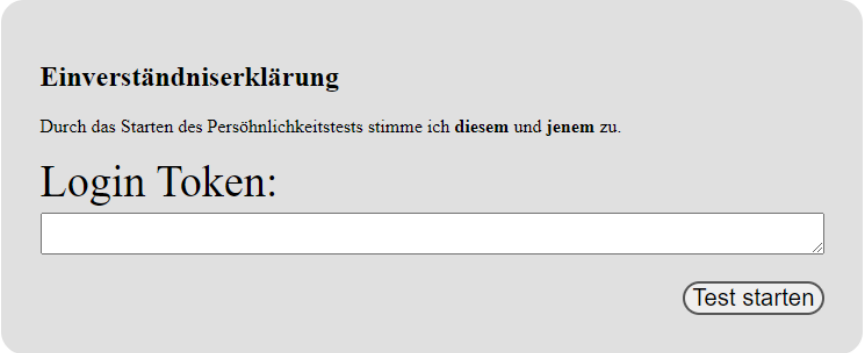
1.1. Zunächst sollte die lokale Entwicklungsumgebung eingerichtet werden.

1.2. Um zu testen, ob das Einrichten funktioniert hat, lass jeweils die Applikation & automatisierten Tests einmal lokal laufen.

1.3. Um einen ersten Blick in den Code, etc. zu bekommen, lege einen neuen Nutzer mit gehashten Passwort in der Datenbank an.

2. System weiterentwickeln

2.1. Auf der Startseite der Webseite soll zusätzlich ein beispielhafter Infotext zur "Einverständniserklärung" angezeigt werden. Dieser könnte z.B. wie folgt aussehen:



2.2. Die Gültigkeit eines zeitlich beschränkten Tokens soll ab jetzt nicht mehr 2 Wochen, sondern 3 Wochen sein.

2.3. Es soll ein passender Log Eintrag hinzugefügt werden, sodass geloggt wird, wenn ein*e Nutzer*in die Route GET /api/token-creator/ aufruft.

A.2 Feedback der Testperson

Aufgabe	1.1.	1.2.	1.3.
Wie lange hat die Teilaufgabe gedauert?	70 min	5 min	20 min
Wie wurde die Aufgabe gelöst?	Anhand des developer manuals habe die Entwicklungsumgebung Stück für Stück eingerichtet wie beschrieben.	Die vorgegebenen Run-Configurations in pyCharm ausgeführt.	Gehashtes Passwort mit util.py erstellt und per pgAdmin in den table user eingetragen. Wie das ging war in den manuals hinreichend erläutert.
Welche Probleme gab es?	Ich wusste nicht, wie genau ich PGUSER als environment variable definiere.	/	/
Wie konnten Probleme gelöst werden? – z.B. mit der Hilfe des *user manual*, *developer manual* oder Google	Eine kurze Suche auf Stackoverflow konnte das Problem lösen.	/	/

Aufgabe	2.1.	2.2.	2.3.
Wie lange hat die Teilaufgabe gedauert?	15 min	30 min	70 min
Wie wurde die Aufgabe gelöst?	Einfügen des Infotextes in index.html, Zeile 12	Ändern von TOKEN_PERIOD_OF_ VALIDITY_IN_DAYS in constants.py, Zeile 56 von 14 zu 21.	Einfügen eines Log-Befehls in token_creator.py, Zeile 14
Welche Probleme gab es?	/	Dass ich hier für eine so simple Änderung vergleichsweise lange gebraucht habe, liegt daran, dass ich mich recht lange durch den Code lesen musste, bis ich den Verweis auf die Variable in constants.py fand.	Da ich weder mit Python, noch mit Flask vertraut war, wusste ich zunächst nicht, wie das Logging hier funktioniert und wohin das log ausgegeben werden soll (auf der Konsole/ in eine log file)
Wie konnten Probleme gelöst werden? – z.B. mit der Hilfe des *user manual*, *developer manual* oder Google	/	/	Hilfe baten die offiziellen Dokumentationen unter flask.palletsprojects.com und docs.python.org. Die bereits an anderer Stelle im Code vorhandenen Log-Befehle waren ebenfalls bei der Orientierung hilfreich.

A.3 ER-Diagramm