FH Westküste
Wirtschaft und Technik

Datenverarbeitung und Visualisierung von Umrichter-Testbench-Daten

Jon Feddersen
22122017

Bachelorarbeit im Studiengang Elektrotechnik und

Informationstechnik bei

Prof. Dr. rer. nat. Kristina Schädler

Semesteranschrift
Ochsendrift 31
25853 Drelsdorf

Studienfach
Elektrotechnik und
Informationstechnik

Abgabetermin: 27.10.2025

Fachsemesterzahl: 9

Sperrvermerk

Diese Arbeit enthält vertrauliche Daten und Informationen des Unternehmens, in dem die Bachelor-/Masterarbeit angefertigt wurde. Sie darf Dritten deshalb nicht zugänglich gemacht werden.

Die für die Prüfung notwendigen Exemplare verbleiben beim Prüfungsamt und beim betreuenden Hochschullehrer.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

DUTs Devices-Under-Test
XML Extensible Markup Language
DTD Document Type Definition
API Application Programming Interface
DOM Document Object Model
SAX Simple API for XML
FSW Functional-Switching
OCP Over-Current-Protection
DSCP Dynamic-Short-Circuit-Protection
BIT Burn-In-Test
OTP Overtemperature-Protection-Test
XSLT Extensible Stylesheet Language Transformations
XSD XML Schema Definition
XXE XML External Entity
SQL Structured Query Language
ORM Object-Relational Mapping
SVG Scalable Vector Graphics

1. Einleitung

„Data is the new oil. Like oil, data is valuable, but if unrefined it cannot really be used.“ Clive Humby (2006)

Die Aufbereitung von Daten, sowohl in ihrer Struktur als auch in ihrer Darstellung, ist entscheidend, um technische Prozesse zu verstehen und effizient mit großen Datenmengen arbeiten zu können. Das Erfassen, Aufbereiten und Interpretieren von Messdaten ist heute aus der Industrie und Wissenschaft nicht mehr wegzudenken. In nahezu allen technischen Bereichen werden Daten erhoben, sei es in der Forschung, bei Qualitätsprüfungen oder in industriellen Testverfahren. Damit diese Daten nutzbar sind, müssen sie verständlich aufbereitet und übersichtlich dargestellt werden.

Die Verarbeitung und Visualisierung großer Datenmengen erfolgt heutzutage meist mithilfe spezialisierter Softwarelösungen. Viele dieser Systeme sind jedoch allgemein gehalten und nur begrenzt auf spezifische technische Anwendungen anpassbar. Das kann sich negativ auf Effizienz, Benutzerfreundlichkeit und Aussagekraft der Ergebnisse auswirken. Besonders in Bereichen, in denen die Datensätze komplex und individuell strukturiert sind, wie bei der Prüfung von Leistungselektronik aus Windenergieanlagen, besteht Bedarf an speziell angepassten Softwarelösungen.

Ziel dieser Arbeit ist es, eine Web-Applikation zu entwickeln, die automatisch erzeugte XML-Berichte eines Umrichter-Teststandes einliest, die enthaltenen Mess- und Gerätedaten in einer Datenbank speichert und diese anschließend grafisch aufbereitet. Auf diese Weise sollen die Daten besser ausgewertet und sowohl für interne Analysen als auch für externe Berichte genutzt werden können. Dabei liegt der Fokus auf einer übersichtlichen Benutzeroberfläche, einer modularen Erweiterbarkeit und einer einfachen Integration in die bestehende Systemlandschaft des Unternehmens.

Die Arbeit entstand in Zusammenarbeit mit einem Serviceanbieter für die Instandhaltung von Windenergieanlagen. Der zugrunde liegende Teststand wird zur Überprüfung von Umrichtern eingesetzt, die aus bestehenden Anlagen stammen. Mithilfe spezieller Prüfverfahren wird bestimmt, ob diese Geräte nach der Instandsetzung wiederverwendet werden können. Die dabei entstehenden XML-Dateien enthalten sämtliche Messwerte, Parameter und Prüfstandsinformationen und bilden die Grundlage für die zu entwickelnde Software.

Für die Umsetzung wurde ein iteratives Vorgehen gewählt. Das bedeutet, dass die Anwendung schrittweise entwickelt und nach jedem Zwischenschritt getestet und verbessert wird. Dieses Vorgehen orientiert sich an Prinzipien agiler Softwareentwicklung, um flexibel auf mögliche Anpassungen reagieren zu können. So entsteht eine funktionierende Anwendung, die sich im Laufe der Entwicklung immer weiter verfeinern lässt.

Die Arbeit gliedert sich in sechs Kapitel. Nach dieser Einleitung werden in Kapitel 2 die theoretischen und technologischen Grundlagen behandelt, die für die Entwicklung der Anwendung relevant sind. In Kapitel 3 folgen die Analyse der vorhandenen Strukturen und der Entwurf der

Systemarchitektur sowie eine Beschreibung der funktionalen und nicht-funktionalen Anforderungen. Kapitel 4 beschreibt die Implementierung der Applikation, während Kapitel 5 die Integration in die bestehende Systemumgebung und die Testdurchführung erläutert. Kapitel 6 fasst die Ergebnisse zusammen und gibt einen Ausblick auf mögliche Erweiterungen.

2. Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen für die Entwicklung der Software sowie das notwendige Verständnis des Teststandes und seiner Abläufe behandelt.

2.1. Überblick über den Umrichter-Prüfstand

In diesem Abschnitt wird der Umrichter-Teststand, von dem die zu verarbeitenden Datensätze stammen, beschrieben, da dies für das generelle Verständnis der einzulesenden Datensatzstruktur unerlässlich ist. Die genaue Bezeichnung des Teststandes „USTB DWT Test Bench (XCT0006-1)“ wird im Folgenden als Test-Bench oder Teststand benannt. Diese Art Test-Bench wird im Allgemeinen für die End-of-Line-Prüfung von unterschiedlichen Umrichtern nach ihrer Herstellung genutzt, um die Produktqualität und -funktionalität sicherzustellen. [1]

In dem hier vorliegenden Fall wird der Teststand verwendet, um die aus dem Feld kommenden Umrichter auf ihre weitere Nutzungstauglichkeit zu testen. Die weitere Nutzungstauglichkeit wird ermittelt, indem die Messwerte mit Mittelwerten, die von mehreren fabrikneuen Umrichtern stammen, verglichen werden. Diese Messwerte müssen sich in einem vorher definierten Toleranzbereich befinden, um weiter im Feld verwendet zu werden.

Die Umrichter werden in der gegebenen Fachliteratur zur Test-Bench als Devices-Under-Test (DUTs) bezeichnet. Diese Bezeichnung kommt auch in den Berichten auf dem Teststand vor, daher hat der Autor diese Abkürzung übernommen.

2.1.1. Aufbau des Teststandes

Die Test-Bench besteht aus mehreren Komponenten, die sich im Testraum in unterschiedlichen Schaltschränken befinden. Der Teststand besteht aus folgenden Hauptkomponenten:

- Das Netzteil wandelt die 400-V-Netzspannung in eine isolierte Gleichspannung für den Zwischenkreis um. Das Netzteil liefert maximal 80 kW mit 1200 V DC oder 800 V DC. Welche Werte verwendet werden, kann vor Teststart bestimmt werden. In Abbildung 1 wird das Netzteil als PSU bezeichnet, was für „Power Supply Unit“ steht.
- Das Elektronik-Rack, auf dem die Mess- und Steuerkomponenten befestigt sind, ist ein wichtiger Bestandteil des Teststandes. Hier befindet sich auch der (XCS2100) System-Controller, der das ganze System mit dem PC, auf dem die Test-Bench-Software läuft, via Ethernet verbindet. In Abbildung 1 mit ER bezeichnet, für „Electronic Rack“.
- Der Testmatrix-Schrank, in dem die Sammelschienen für den Stromanschluss und die Schützen sitzen. In Abbildung 1 mit TM bezeichnet, für „Test Matrix cabine“.
- Der Schrank mit dem Kühlungssystem, da die Umrichter während des Betriebes mit Wasser oder Luft gekühlt werden müssen. In Abbildung 1 mit „Cool1“ bezeichnet.

- Der Carrier, auf dem die Umrichter befestigt werden, ist speziell für bestimmte Umrichter konstruiert. Dieser wird speziell für bestimmte Umrichter konstruiert. In Abbildung 1 wird der Carrier als Carrier1 bezeichnet.

Neben den Hauptkomponenten befinden sich außerhalb des Sicherheitsbereiches, der während des Betriebes nicht betreten werden darf, ein PC mit einer Software zum Steuern der Testeinrichtung sowie eine Betriebsanzeige und ein Notaus. [1]

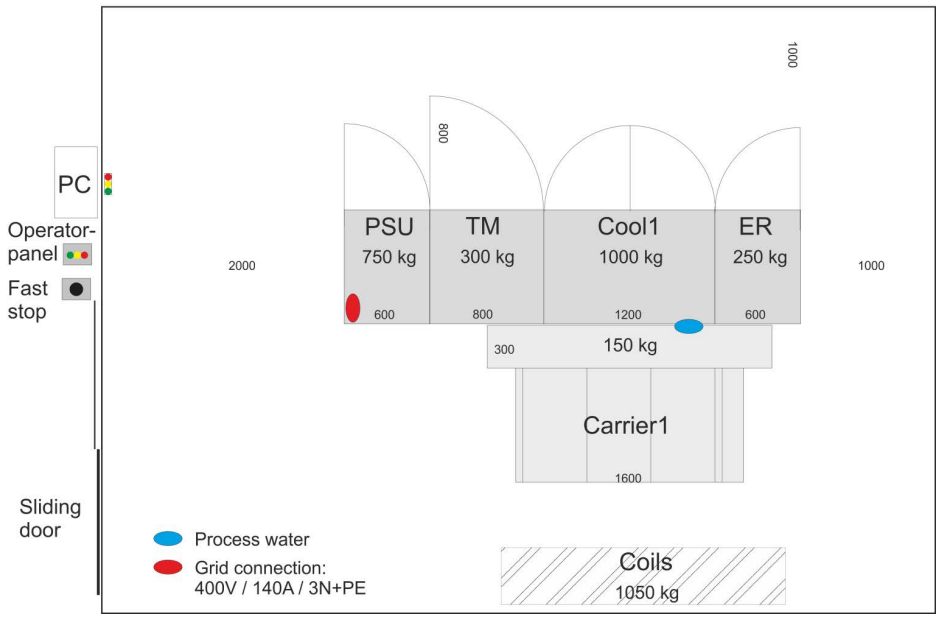

Abbildung 1: Aufbau des Teststandes
Quelle: [1, S. 7]

2.1.2. Testmodule

Es gibt mehrere Testmodule, die auf dem Teststand laufen und verschiedene Funktionen der Umrichter testen. Einige der Funktionen eines DUT können mit dem gleichen Modus eines Testmoduls überprüft werden, indem die entsprechenden Parameter ausgewählt werden. Jeder Test bzw. jeses Testmodule ist autonom und kann mehrmals ausgeführt werden, auch mit unterschiedlichen Parametern.
Im Folgenden wird eine kurze Beschreibung der Funktionen der für diese Arbeit relevanten Testmodule gegeben:

Driver-Consumption-Test: Der Driver-Consumption-Test überprüft den Stromverbrauch des Treibers im Leerlauf und während Pulssprüngen.

Pulse-Test: Der Impulstest verfügt über drei Funktionsmodi.

- Im Modus Functional-Switching (FSW) kann überprüft werden, ob die Halbleiter, die sich in den DUTs befinden, generell schalten.

- Im Modus Over-Current-Protection (OCP) kann die Überstromüberwachung weiche Kurzschlüsse überprüfen. Ein weicher Kurzschluss ist, wenn der Stromfluss nicht sofort und vollkommen unterbrochen wird.
- Im Modus Dynamic-Short-Circuit-Protection (DSCP) wird ein harter Kurzschluss, also ein vollständiger und sofortiger Kurzschluss, überprüft.

Power-Test: Mit diesem Test werden zwei verschiedene Funktionen getestet werden:

- Burn-In-Test (BIT) dient dazu, die DUTs zyklisch zu betreiben und so reale Betriebszustände zu simulieren. Zudem kann mit Hilfe dieser Funktion die Kühltemperatur überprüft werden, um die korrekte Wärmeübertragung der Halbleiter sicherzustellen.
- Während des Overtemperature-Protection-Test (OTP) wird ein DUT, ähnlich wie beim BIT, nur mit reduzierter Kühlung betrieben, bis die maximal zulässige Kühlkörpertemperatur erreicht ist und die Temperaturschutzschaltung auslöst. [1]

2.1.3. Testablauf

Die Schrittfolge des Testablaufes mit Montage der DUTs ist klar festgelegt und vor jedem Durchlauf gleich. Die Montage läuft wie folgt ab:

1. Die Umrichter werden auf dem Carrier befestigt. Es werden meist 3 dieser Geräte gleichzeitig getestet, Abweichungen sind je nach Bauform der DUTs möglich. Die Reihenfolge der elektrischen Phasen ist bei Draufsicht des Carriers von links nach rechts U-V-W. Dies ist für das Layout des Berichtes relevant.
2. Der Kühlkreislauf wird angeschlossen, je nach Umrichtertyp Lüftungs- oder Wasserkühlung.
3. Die DUTs werden mit den AC- und DC-Link-Kontakten verbunden, um das Gerät zu betreiben und die DC-Spannung wieder abzuleiten.
4. Das Anbringen von Signalkabeln zwischen DUT und Merkurbox. Die Merkurbox dient als Schnittstelle zwischen DUTs und Testbench.
5. Das Montieren von VCE-Klemmen an die AC-Kontakte der DUTs zum Controller des Teststandes.

Vor Beginn des Testablaufs benötigt die Test-Bench noch weitere Informationen zu den DUTs, Testsequenzen und der Hardware-Topologie. Diese Daten können über das Scannen von vorliegenden QR- oder Barcodes eingefügt werden. Alternativ können diese auch über ein Eingabefenster im Teststandprogramm eingetragen werden. Je nach Topologiekonfiguration kann auch ein zweiter Träger mit DUTs gescannt werden, Dies kommt bei dem Test, den das Unternehmen durchführt, nicht vor. [1]

Das eigentliche Testen mit den verschiedenen Testmodulen wird autonom durchgeführt. Im Unternehmen läuft der Test regulär wie folgt ab:

1. Durchführung des Driver-Consumption-Testes, um die Stromversorgung der Treiberplatine auf den Umrichter zu testen.
2. Durchführung des-Pulse-Testes im Modus Functional-Switching, dies prüft die Umrichter.
3. Durchführung des Power-Testes, in den Berichten auch XPower-Test genannt. Dieser Test simuliert die Belastung der Umrichter im Feld und überprüft, ob sich die Werte in einem bestimmten Toleranzbereich befinden.

Nach dem Durchlaufen eines Tests wird automatisch eine Extensible Markup Language (XML)-Struktur mit den erhobenen Messdaten und allen vorher bestimmten Einstellungen in Adminreport generiert. Diese Struktur wird manuell in eine separate XML-Datei eingefügt und so gespeichert. Bei der Demontage nach dem Testdurchlauf werden alle Schritte in umgekehrter Reihenfolge durchgeführt.

2.2. Verarbeitung von XML-Daten

Dieses Kapitel behandelt einige grundlegende und für diese Arbeit relevante Aspekte von dem Dokumententyp XML. Da die Messwerte und Berichte, die der Teststand generiert, in diesem Format vorliegen, ist es relevant, dieses Kapitel zu behandeln.

Bei XML handelt es sich um eine Auszeichnungssprache, also eine formale Sprache, die verwendet wird, um die Struktur und Darstellung von Daten oder Texten zu beschreiben [2]. XML wurde entwickelt, um Informationen in einem maschinenlesbaren und strukturierten Format zu speichern und zu übermitteln. Sie wird hauptsächlich in Bereichen wie Webdiensten, Datenbanken und Konfigurationsdateien eingesetzt. XML ermöglicht die hierarchische Organisation von Informationen in einem strukturierten Aufbau und kann sowohl von Menschen als auch von Maschinen interpretiert werden. [3, S. 4]

Das Grundkonzept hinter XML war, eine universell einsetzbare und erweiterbare Sprache zu erschaffen, die von verschiedenen Systemen unabhängig von deren fundamentalen Technologieansatz genutzt werden kann. Hierbei wäre das angestrebte Ziel, Daten in einem einheitlichen Standard zwischen verschiedenen Anwendungen und Plattformen zu speichern und auszutauschen. zu kommen. [3, S. 3–5]

2.2.1. XML-Strukturaufbau

Eine XML-Datei beginnt mit einem Prolog, der die XML-Version und die verwendete Zeichencodierung definiert. In Abbildung 2 ist ein häufig genutzter Prolog dargestellt, der auch in den Teststand-Berichten zum Einsatz kommt. Die erste Zeile des Prologs ist die sogenannte XML-Deklaration. Die XML-Deklaration enthält häufig die Attribute „version“ und „encoding“, jedoch ist nur das Attribut „version“ Pflicht. Werden auch die anderen notiert, müssen sie in der angegebenen Reihenfolge deklariert werden. Attribut „version“: Mit „version“ wird die verwendete XML-Version angegeben. Das Attribut „encoding“ gibt die im Dokument verwendete Zeichenkodierung an, d. h., mit welcher Codierung die Datei gespeichert wird. Fehlt die Angabe, wird als

Vorgabe UTF-8 (8-Bit Unicode Transformation Format) verwendet. Neben der XML-Deklaration können im Prolog auch noch Verarbeitungsanweisungen und Verweise auf eine Document Type Definition (DTD) deklariert werden. Diese sind jedoch optional und für diese Arbeit nicht weiter relevant. [4, S. 8, 9]

```
<?xml version="1.0" encoding="UTF-8"?>
    <root>
    ....
    </root>
```

Abbildung 2: XML Prolog Beispielcode
Quelle: eigene Darstellung

Der Hauptteil eines XML-Dokuments besteht aus einer Reihe von Elementen, die durch Tags markiert sind. Für jedes Element gibt es ein Starttag und ein Endtag, welche das Element beginnen und beenden. Ein Starttag kann beispielsweise so aussehen: „<NamedesTags>“ So aus, dann würde der dazugehörige Endtag „</NamedesTags>“ so aussehen. Der entscheidende Unterschied ist hierbei der Schrägstrich beim Endtag. Der Name des Elementes wird durch den Inhalt der Keiler- und Großer-Zeichen bestimmt. Bei diesem Beispiel wäre der Name „Namedes-Tags“. Elemente haben einen Inhalt, der aus Text, weiteren Elementen oder aus beidem bestehen kann. Wenn Elemente andere Elemente beinhalten, werden sie als Elternelemente und die enthaltenen Elemente oft als Kindelemente bezeichnet. Diese Eigenschaft der Elemente sorgt dafür, dass XML-Dateien einer hierarchischen Baumstruktur folgen. Hierbei wird das oberste Element als Wurzelelement bezeichnet, im Englischen „root“. [4, S. 10–14]

```
<buch><!-- Das Element "Buch" enth lt 2 Kinderelement-->
  <titel>XML-Grundlagen</titel>
  <autor>Max Mustermann</autor>
</buch>
```

Abbildung 3: XML-Elemente Beispielcode
Quelle: eigene Darstellung

Jedes Element kann neben Inhalt auch mit beliebig vielen Attributen ausgestattet sein, die zusätzliche Informationen enthalten. Attribute werden im Starttag eines Elements definiert. Diese bestehen immer aus einem Attributnamen und einem Wert. Der Wert wird dabei mit Anführungszeichen deklariert, wie in Abbildung 4 gezeigt. [4, S. 10–14]

```
<buch genre="Lehrbuch">ES</buch>
    <!--Hier ist das Attribut "genre"  mit dem Inhalt "Lehrbuch"-->
```

Abbildung 4: XML-Attribut Beispielcode
Quelle: eigene Darstellung

Kommentare werden mit den Tags „<!–“ und „–>“ eingefügt und dienen der Dokumentation oder dem Hinweis auf bestimmte Teile des Codes [4, S. 10–14]. Dies ist für die automatisch generierten Berichte irrelevant, jedoch für das Beschreiben der Beispiele hilfreich. In Abbildung 3 und Abbildung 4 werden diese zur Beschreibung verwendet. Kommentare werden beim Parsen des Dokuments ignoriert. Parsen und Parser werden im nachfolgenden Kapitel behandelt.

2.2.2. Verarbeiten von XML-Dateien

In diesem Abschnitt wird eine Zusammenfassung der grundlegenden Methoden und Techniken gegeben, um XML-Dateien mithilfe verschiedener Tools unabhängig von der verwendeten Programmiersprache zu verarbeiten. Es wird beschrieben, wie man XML-Dateien analysiert, modifiziert und überprüft, um sie für verschiedene Zwecke einsatzbereit zu machen. Am Ende des Abschnittes wird jedoch etwas genauer auf Methoden, die in Python genutzt werden, eingegangen.

2.2.2.1. XML-Parser

Der erste Schritt beim Verarbeiten einer XML-Datei ist das Parsen. Hierbei wird die XML-Datei in ein Programm geladen und in ein Format umgewandelt, das dieses interpretieren kann. XML-Parser prüfen hierbei auch die XML-Daten auf Korrektheit, also ob die Vorlagen eingehalten werden und das Dokument vollständig ist. Dabei wird zwischen nicht-validierenden und validierenden Parsern unterschieden. Der Unterschied besteht darin, dass validierende Parser neben der korrekten Schachtelung und Bezeichnung der Strukturelemente wie die nicht-validierende Parser auch noch auf eine Vorgabe einer Dokumenttypdefinition oder eines Schemas prüfen. [4, S. 10]

Parser werden verwendet, um einer Applikation über eine Application Programming Interface (API) eine Schnittstelle zu einem XML-Dokument zu geben. Bei APIs wird in diesem Bereich zwischen zwei Grundtypen unterschieden: [4, S. 405]

Baumbasiert
Die baumbasierten APIs lesen über den XML-Parser das XML-Dokument ein, parst es und erzeugt ein Modell als Baum von Knoten im Arbeitsspeicher. Auf Grund der im XML-Dokument vorkommenden Informationseinheiten wird in verschiedene Knotentypen unterschieden. Das generierte Modell dient der Applikation für die weitere anwendungsspezifische Verarbeitung. Ein Beispiel für eine baumbasierte API ist Document Object Model (DOM).

DOM ist ein objektorientiertes Modell, das die Struktur eines XML-Dokuments abbildet. In der Baumstruktur wird das gesamte Dokument abgebildet, indem jedes Element, jedes Attribut und jeder Text bzw. Inhalt als Knoten gilt. Das DOM hat den Vorteil, dass es das XML-Dokument vollständig im Arbeitsspeicher darstellt, was das Durchsuchen und Bearbeiten des Dokuments erleichtert. Zudem bietet es eine einfache Schnittstelle, um XML-Daten zu erreichen und zu

verändern. Allerdings benötigt diese Herangehensweise viel Speicherplatz, da es das gesamte Dokument im Arbeitsspeicher ablegt, was bei umfangreichen XML-Dokumenten ein Problem darstellen kann. [4, S. 413, 414]

Ereignisbasiert

Die ereignisbasierten APIs lesen XML-Dokumente sequenziell von Beginn an. ein und meldet während des Lesens jedes Ereignis durch sogenannte Callbacks an die aufrufende Applikation zurück. Ein Ereignis ist ein Signal, das Änderungen in dem Markup-Status anzeigt. Das bedeutet, Ereignisse traten bei Element-Tags, Zeichendaten, Kommentaren und Verarbeitungsanweisungen sowie bei den Grenzen des Dokumentes auf. Der Parser sendet hierbei durch die Callbacks eine Mitteilung an die aufrufende Applikation, welche Ereignisse eingetreten sind. Das Programm, das den Parser aufgerufen hat, muss nun das Ereignis interpretieren und entsprechend reagieren. Das XML-Dokument wird hierbei nicht vollständig im Arbeitsspeicher gespeichert, sondern in Teilen gelesen und bearbeitet. Deshalb eignen sich ereignisbasierte APIs besonders gut für die Verarbeitung großer XML-Dateien, da dies für eine geringe Arbeitsspeicherbelastung sorgt. Trotz der Effizienz hinsichtlich des Speicherverbrauchs sind ereignisbasierte APIs für die Datenmanipulation nicht sonderlich gut geeignet. Durch das teilweise Einlesen wird nämlich keine umfassende Dokumentstruktur wie beim vollständigen Aufbauen erzeugt. Die baumbasierten APIs, welches die Manipulation bzw. Bearbeitung erschwert. Ein Beispiel für eine bedeutende ereignisbasierte API ist Simple API for XML (SAX). Diese API arbeitet nach dem oben beschriebenen Prinzip und wurde ursprünglich als Java-API entwickelt. Inzwischen gibt es SAX auch für weitere Sprachen wie z. B. C++, Perl und Python. [4, S. 405]

2.2.2.2. Adressierung mit XPath

Neben dem Einlesen der XML-Datei muss zum Verwenden der Daten in dem XML-Dokument navigiert werden, um bestimmte Knoten zu adressieren. Aus diesem Grund wurde die Abfragesprache XML Path Language, kurz XPath, entwickelt. XPath wird hauptsächlich in der Transformationssprache Extensible Stylesheet Language Transformations (XSLT) eingesetzt, welche XML-Dateien in andere Datenformate umwandeln soll. Zudem wird XPath in anderen Programmiersprachen wie z. B. JavaScript, C oder einigen Pythonbibliotheken für die Adressierung von Bestandteilen des XML-Baumes verwendet.

Im Zusammenhang mit XML und XPath wird oft der Begriff „Knoten“ verwendet. Knoten können Teile des XML-Dokumentes beschreiben, aber auch über den Wurzelknoten das gesamte Dokument. Knoten können alle im XML-Dokument vorhandenen strukturellen Teile sein. Daher kann zwischen den verschiedenen Typen von Knoten im XML-Dokument unterschieden werden:

- Wurzelknoten: Auch Dokumentknoten oder root genannt, sind der Ursprung des XML-Dokuments.
- Elementknoten: ein beliebiges Element.
- Textknoten: ein Text, welcher einem Element untergeordnet ist.

- Attributknoten: ein beliebiges Attribut eines Elements.
- Kommentarknoten: ein beliebiger Kommentar.
- Namensraumknoten: eine beliebige Namensraumangabe eines Elements oder Attributs.
- Verarbeitungsanweisungsknoten: ein XML-Verarbeitungshinweis.

Über diesen Knoten kann man durch die XML-Struktur navigieren, aber um die Knoten zu adressieren, werden Achsen benötigt. Achsen spezifiziert die Beziehung zwischen den Knoten, um so die gewünschten Knoten zu selektieren. Es gibt Achsen, welche nur einen Knoten adressieren, so wie Achsen, welche mehrere Knoten gleichzeitig auswählen. In der folgenden Tabelle 1 werden die verschiedenen Achsen benannt und beschrieben. Für alle Achsen, die keine Kurz-Notation enthalten, muss im Code der Name ganz ausgeschrieben werden. [5]

Name	**Kurz-Notation**	**selektierte Knoten**
/	/	Wurzelknoten
child	(nicht notwendig)	direkt untergeordnete Knoten (Kindknoten)
self	.	aktuelle Knoten (Kontextknoten)
parent	..	direkt übergeordneter Knoten (Elternknoten)
descendant	.//	alle untergeordnete Knoten
descendant-or-self		alle untergeordnete Knoten sowie der aktuelle Knoten
ancestor		alle übergeordnete Knoten
ancestor-or-self		alle übergeordneten Knoten sowie der aktuelle Knoten
following		alle nachfolgende Knoten (ohne Kindknoten)
following-sibling		alle nachfolgende Knoten (ohne Kindknoten), die den gleichen Elternknoten haben
preceding		alle vorangehende Knoten (ohne alle Elternknoten)
preceding-sibling		alle vorangehende Knoten (ohne alle Elternknoten), die den gleichen Elternknoten haben
attribute	@	Attributknoten
namespace		Namensraumknoten

Tabelle 1: Übersicht über XPath-Achsen

Quelle: eigene Darstellung nach Tabelle aus [5]

Um die Selektierung der Knoten noch weiter einzugrenzen, können sogenannte Prädikate verwendet werden. Prädikate befinden sich immer in eckigen Klammern und in diesen können verschiedene Operatoren genutzt werden. Dazu zählen die mathematischen Operatoren + (Addition), – (Subtraktion), × (Multiplikation), div (Division) und mod (Modulo, Divisionsrest), aber auch die logischen Operatoren and (und) und or (oder) sowie die Vergleichsoperatoren = (gleich) und

!= (ungleich), < (kleiner als), (kleiner-gleich), > (größer als) und (größer-gleich). Mit diesen Operatoren und Aneinanderreihungen von diesen kann eine Vielzahl von Sucheingrenzungen geschaffen werden, welche die Genauigkeit der Suchen stark erhöhen und variabel gestalten. [5]

Ein XPath-Ausdruck, also ein Path, welcher zur Adressierung von einem oder mehreren Knoten verwendet wird, besteht aus einem oder mehreren Lokalisierungsschritten. Ein Lokalisierungsschritt besteht aus weiteren 3 Teilen: der Achse, einem Knotentest und bei Bedarf aus einem oder mehreren Prädikaten. Der Knotentest ist hierbei nichts anderes als der Name des Knotens. Für die korrekte Form des Lokalisierungsschrittes muss der Achsenname durch zwei Doppelpunkte vom Knotentest getrennt werden. Bei Kurz-Notationen können die Doppelpunkte jedoch weggelassen werden, wenn anstelle des Knotentests bzw. dem Knotennamen das Wildcard-Zeichen in einem Lokalisierungsschritt verwendet wird, werden unabhängig vom Namen und von den Achsen alle Knoten selektiert, welche sich unter dem letzten Lokalisierungsschritt befinden. Die Lokalisierungsschritte werden mithilfe des Schrägstrichs „/“ voneinander getrennt. [5] In der folgenden Abbildung 5 ist ein Beispiel für einen XPath-Ausdruck. Dieser XPath würde in einem XML-Dokument mit Büchern das erste Buch auswählen, welches vor 1900 erschienen ist. Geht dann zum nächsten Geschwisterelement weiter und gibt den Titel zurück.

```
1 //buch[jahr<1900]/following-sibling::buch[1]/titel
```

Abbildung 5: Beispiel XPath-Ausdruck
Quelle: eigene Darstellung

2.2.2.3. Pythonbibliotheken für XML-Verarbeitung

Für die Bearbeitung von XML-Dokumenten gibt es in Python einige Bibliotheken. Diese basieren auf den obengenannten Ansätzen, sind jedoch in dem Umfang deutlich kleiner gehalten und bieten weniger Funktionen. Im Folgenden werden einige wichtige Bibliotheken genannt und kurz erläutert:

xml.etree.ElementTree
Die Standardbibliothek für XML in Python. Sie ist einfach zu benutzen und es sind keine Zusatzinstallationen neben Python notwendig. Gut geeignet, um kleine bis mittlere XML-Dateien zu verarbeiten. Unterstützt grundlegendes Parsen, Schreiben und einfaches XPath, aber bietet keine XSLT oder komplexe Features für XML-Dokumente. [6]

xml.dom.minidom
Eine DOM-basierte Implementierung in Python. Ermöglicht den Zugriff auf XML über das klassische Document Object Model. Bietet feine Kontrolle, wirkt aber oft sperrig und weniger performant. [7]

xml.sax
Ein eventbasiertes XML-Parsing. Liest XML zeilenweise und löst Ereignisse aus, wenn Elemen-

te gefunden werden. Sehr speichersparend und ideal für sehr große XML-Dateien, aber auch komplizierter in der Anwendung. [8]

lxml
Die leistungsfähigste XML/HTML-Bibliothek in Python. Basiert auf den C-Bibliotheken libxml2 und libxslt und ist daher sehr schnell. Unterstützt XPath, XSLT, Validierung und kann sowohl sauberes XML als auch „kaputtes“ HTML verarbeiten. Der wird oft als heutiger De-facto-Standard für komplexe XML-Verarbeitung in Python bezeichnet. [9]

BeautifulSoup
Eigentlich für HTML gedacht, funktioniert aber auch mit XML. Sehr tolerant gegenüber fehlerhaftem Code, was es beim Web-Scraping nützlich macht. Langsamer als lxml, aber sehr einfach in der Benutzung. [10]

xmlschema
Spezialisiert auf XML Schema Definition (XSD). XSD ist eine Sprache, die die Struktur und den Inhalt von XML-Dokumenten definiert. XML-Schema kann XML-Dokumente validieren und direkt in Python-Objekte umwandeln. Gut geeignet, wenn sicherstellen werden muss, dass XML-Daten einer bestimmten Struktur entsprechen. [11]

defusedxml
Eine sichere Variante der Standardbibliotheken. Schützt vor bekannten XML-Sicherheitslücken wie „Billion Laughs“ oder XML External Entity (XXE). Billion Laughs beschreibt das Einsetzen von stark verschachtelten und sich wiederholenden XML-Strukturen, die, wenn Sie in den Parser geladen werden, wodurch es zu Speicher- oder CPU-Überlastung führen kann. Mit XXE können Befehle in der XML-Struktur genutzt werden, welche zu Sicherheitslücken beim XML-Parsing führen. So können Angreifer externe Entitäten einbinden, um Daten auszulesen oder sogar Befehle auszuführen. Wichtig, wenn XML aus unsicheren Quellen kommt. [12]

untangle
Sehr einfache Bibliothek, die XML in Python-Objekte übersetzt. Damit können XML-Strukturen fast wie normale Attribute angesprochen werden. Gut für kleine Projekte, aber eingeschränkt im Funktionsumfang. [13]

2.3. Flask-Grundlagen und Funktionsweise

In diesem Unterkapitel wird das Webframework Flask, das in der Web-Applikation genutzt wird, behandelt. Ein grundlegender Überblick wird gegeben und sein wichtiger Funktionsablauf wird erläutert.

2.3.1. Überblick

Flask ist ein schlankes Webframework für Python, das auf WSGI (Web Server Gateway Interface) basiert. Das Web Server Gateway Interface (WSGI) ist eine Schnittstelle, die es Webservern ermöglicht, mit Python-Webanwendungen oder -Frameworks zu kommunizieren. Der Begriff

„leichtgewichtig“ bedeutet in diesem Zusammenhang, dass der Framework-Kern mit Bedacht minimalistisch entworfen wurde und viele Funktionen nur durch Erweiterungen hinzugefügt werden können. In der Standardkonfiguration setzt sich Flask aus zwei zentralen Komponenten zusammen: [14]

- Werkzeug: Ist eine Sammlung von Funktionen zum Erstellen von WSGI-Anwendungen, die HTTP-Requests und -Responses abstrahiert und die Schnittstelle zwischen dem Webserver und der Python-Anwendung ermöglicht. Außerdem umfasst sie Funktionen zum Aufspüren und Beheben von Fehlern (Debugging), zum Zuordnen von URLs zu Funktionen oder Aktionen in einer Webanwendung (URL-Routing) sowie Hilfsfunktionen für die Arbeit mit HTTP (HTTP-Utilities).
- Jinja2: eine Template-Engine, die es ermöglicht, HTML-Dateien mit Platzhaltern, Schleifen, Bedingungen und weiteren Syntaxelementen zu erstellen, um sie dynamisch in der Anwendung zu rendern. Sie erlaubt so die dynamische Erstellung von HTML-Seiten (oder anderen Textdateien), indem Python-Daten in Platzhalter innerhalb einer Vorlage (Template) eingefügt werden.

2.3.2. Grundprinzip der Funktionsweise

Die beiden obengenannten zentralen Komponenten bilden die Grundlagen für die Funktionsweise von Flask.

Für die Darstellung im Browser werden die Seiten der Applikation serverseitig dynamisch erstellt und an den Client gesendet, indem diese Template-Vorlagen genutzt werden. Die HTTP-Anfragen (Requests) des Clients werden über die WSGI-Schnittstelle, die Werkzeug bereitstellt, entgegengenommen, von der Anwendung verarbeitet und als HTTP-Antworten (Responses) an den Webserver zurückgeschickt. Hierbei kümmert sich Jinja2 um die dynamische Erstellung der HTML-Seiten, während Werkzeug die Kommunikation zwischen Webserver und Anwendung regelt. Die Auslieferung von dynamisch generierten Webseiten wird durch das Zusammenwirken dieser beiden Komponenten ermöglicht.

Um die Funktionsweise von Flask zu veranschaulichen, wird im Folgenden ein typischer Ablauf einer Webanfrage kurz beschrieben: [15], [16]

1. **Erhalt einer HTTP-Anfrage**
   Ein Webclient (wie ein Browser) schickt eine Anfrage an den Webserver, der sie über die WSGI-Schnittstelle an die Flask-Anwendung weiterleitet.
2. **Routing**
   Flask untersucht die URL und weist sie einer passenden View-Funktion zu. Dies erfolgt über sogenannte Routen, die mit dem Dekorator @app.route() festgelegt werden.

3. **Bearbeitung der Anfrage**
   Die Geschäftslogik wird in der zugeordneten Funktion ausgeführt; dazu gehört unter anderem das Auslesen von Parametern, der Zugriff auf eine Datenbank oder die Aufbereitung von Daten.
4. **Template Rendering (optional)**
   Falls die Antwort HTML-basiert ist, wird ein Jinja2-Template mit dynamischen Daten gefüllt und gerendert.
5. **Erzeugung der HTTP-Antwort**
   Flask gibt die fertige Antwort (z. B. HTML, JSON oder Redirect) an den Client zurück.

2.3.3. Blueprints in Flask

Flask ermöglicht es, über interne Funktionen und Klassen sogenannte Blueprints zu erstellen und zu nutzen. Blueprints sind eine strukturierte Möglichkeit, Webanwendungen modular aufzubauen. Ein Blueprint stellt ein logisches Teilmodul der Anwendung dar und kann eigene Routen, Templates, statische Dateien und Logik enthalten. Dadurch lassen sich umfangreiche Projekte in klar abgegrenzte Funktionsbereiche unterteilen, was die Übersichtlichkeit und Wartbarkeit der Anwendung erhöht. Jeder Blueprint wird in einer separaten Datei definiert und anschließend in der Hauptapplikation registriert. Dieses Vorgehen ermöglicht es, neue Funktionsmodule ohne Eingriff in bestehende Komponenten zu ergänzen. Somit befördert Blueprint das Unterteilen in modulare Softwarearchitektur, insbesondere die geringe Kopplung und hohe Kohäsion zwischen den Modulen.[17]

2.4. Datenbankentwurf und Normalisierung

In diesen Unterkapiteln werden das grundlegende Wissen zu Datenbanken vermittelt und der Erstellungsablauf erläutert. Außerdem werden zwei Methoden zum Einbinden in die Applikation aufgezeigt.

Datenbanken sind strukturierte Zusammenstellungen von Daten, die elektronisch gespeichert und verwaltet werden. Ihr Hauptziel ist es, große Datenmengen strukturiert zu speichern, den Zugriff zu optimieren und die Integrität der Daten zu gewährleisten. Datenbanken haben im Vergleich zu anderen Dateisystemen Mechanismen, die mehreren Benutzern die parallele Nutzung ermöglichen, sowie redundante Datenhaltung vermeiden und die effiziente Abfragen über spezielle Sprachen wie die Structured Query Language (SQL) ermöglichen. [18, S. 6]

In der Folge werden die Hauptanforderungen an Datenbanken bzw. Datenbankmanagementsysteme kurz zusammengefasst: [19, S. 7],[18, S. 6]

- Datenunabhängigkeit: Speicherung unabhängig von Programmen und Plattformen
- Benutzerfreundlichkeit: Einfache Sprachen und grafische Oberflächen
- Mehrfachzugriff: Gleichzeitiger Zugriff für autorisierte Benutzer

- Flexibilität: Wahlfreier Zugriff und fortlaufende Verarbeitung
- Effizienz: Kurze Zeiten für Abfragen, Änderungen und Ergänzungen
- Datenschutz: Zugriff nach Benutzergruppen beschränkt
- Datensicherheit: Schutz vor Fehlern und Ausfällen
- Datenintegrität: Vollständige, korrekte und widerspruchsfreie Speicherung
- Redundanzfreiheit: Daten nur einmal speichern, Redundanz vermeiden

2.4.1. Datenbankstruktur

Die Struktur der Datenbank legt fest, wie das Datenbanksystem organisiert ist und wie die Datenelemente angeordnet sind. Das Grundprinzip relationaler Datenbanken sind im Grunde Tabellen und die Beziehungen zwischen diesen. Die Hauptbestandteile sind: [20, S. 35]

- Tabellen – das grundlegende Element, welches Daten in Zeilen (Tupeln) und Spalten (Attributen) strukturiert.
- Ein Datensatz in einer Tabelle wird durch Primärschlüssel (Primary Keys) eindeutig identifiziert.
- Schlüssel aus anderen Tabellen (Foreign Keys): Tabellenverknüpfung, um die referenzielle Integrität zu gewährleisten.
- Indizes: Datenstrukturen, die das Beschleunigen von Abfragen ermöglichen.
- Views: Sie sind virtuelle Tabellen und beruhen auf den Ergebnissen von Abfragen.
- Constraints: Vorgaben zur Gewährleistung der Datenintegrität (z. B. Wertebereiche, Pflichtfelder oder Dopplungen).

2.4.2. Datenintegrität und Normalisierung

Ein methodischer Ansatz zur Reduzierung von Redundanzen und Anomalien ist die Normalisierung. Der Prozess erfolgt schrittweise durch die Normalformen, die durch Indizes (1NF, 2NF, 3NF, BCNF) definiert sind. Alle Normalformen haben spezifische Arten von Datenanomalien zum Ziel: [20, S. 38–41]

- 1. Normalform (1NF): Beseitigung mehrfacher Werte innerhalb einer Zelle.
- 2. Normalform (2NF): Beseitigung partieller Abhängigkeiten.
- 3. Normalform (3NF): Beseitigung transitiver Abhängigkeiten.

Die Gewährleistung, dass Daten korrekt, konsistent und vollständig sind, fällt unter das Konzept der Datenintegrität. Sie wird erzielt durch: [20, S. 38–41]

- Entity-Integrität (eindeutige Primärschlüssel).
- Referentielle Integrität (gültige Fremdschlüsselverweise).
- Domänenintegrität (gültige Wertebereiche und Datentypen).
- Entity-Integrität (eindeutige Primärschlüssel).
- Referentielle Integrität (gültige Fremdschlüsselverweise).
- Integrität der Domäne (gültige Wertebereiche und Datentypen).

2.4.3. Vorgehen zur Erstellung der Datenbank

Bei der Erstellung einer Datenbank sollte nach folgendem Schema vorgegangen werden. Bei diesem Schema gilt es, sowohl technische als auch konzeptionelle Aspekte zu berücksichtigen: [19, S. 9–11]

1. Anforderungsanalyse

   Zuerst wird im Hinblick auf das Geschäftsumfeld bestimmt, welchen konkreten Zweck die Datenbank erfüllen soll. Hierbei ist die Zweckbestimmung entscheidend, da sie festlegt, welche Daten als relevant gelten. Der Konzeptvorschlag, der das Projektziel definiert und die Vorgehensweise umreißt, ist das Ergebnis.

   - Festlegung der fachlichen und technischen Voraussetzungen.
   - Bestimmung der relevanten Datenquellen und -formate.
   - Definition von Integritäts- und Sicherheitsanforderungen.

2. Konzeptionelles Datenmodell

   Es soll das geschäftliche Umfeld betrachtet werden. Die bestehenden Objekte (z. B. Reports mit Materialnummer, Datum, Version), deren Attribute sowie die Beziehungen und Einschränkungen zwischen diesen Objekten. Das Entity-Relationship-Modell (ERM) nach Chen oder das PrecisedERM (PERM) werden häufig zur Modellierung verwendet. Der konzeptionelle Entwurf ist die Grundlage für die nächste Phase und dient als Diskussionsbasis.

   - Die Entwicklung eines Entity-Relationship-Modells (ERM), das die reale Welt in Entitäten, Attribute und Beziehungen abbildet.
   - Die Einbeziehung von Kardinalitäten (1:1, 1:n, n:m).

3. Logisches Datenmodell

   Der konzeptionelle Entwurf wird in einen logischen Entwurf umgewandelt, der die fachlichen Konzepte in ein datenbanktechnisches Format überführt. In der Regel kommt das Relationenmodell zum Einsatz, welches die Daten in Form von Tabellen organisiert. Transformationsregeln garantieren, dass Beziehungen und Integritätsbedingungen richtig umgesetzt werden.

   - Transformation des konzeptionellen Modells in ein relationales Schema.
   - Festlegung von Tabellen, Spalten (Attributen), Primärschlüsseln und Fremdschlüsseln.
   - Definition von Datentypen und Zellkonfigurationen (z. B. NOT NULL, UNIQUE, CHECK).

4. Physisches Datenmodell

   Eine physische Datenbankstruktur wird durch SQL erstellt, basierend auf dem logischen Entwurf. Die konkrete Festlegung von Tabellen, Indizes, Constraints usw. erfolgt dabei durch das implementierte System, welches immer wieder getestet und zusammen mit den Nutzerinnen auf fachliche Richtigkeit überprüft wird.

   - Realisierung des logischen Modells in einer spezifischen Datenbankmanagementsoftware (z. B. MySQL, MariaDB oder Microsoft SQL Server).
   - Die Verbesserung der Speicherstrukturen, der Indexierung und der Partitionierung.

5. Implementierung und Prüfung

   Nach der erfolgreichen Umsetzung wird das System vom Kunden gemäß einem vorher festgelegten Abnahmeplan freigegeben. Ein Wartungsplan kümmert sich anschließend um die Betreuung, schult die Endbenutzer und überwacht die IT-Umgebung kontinuierlich.

   - Der Aufbau der Tabellen und Relationen erfolgt entsprechend dem Datenbankschema.
   - Die Testdaten implementieren.
   - Die Kontrolle der Funktionalität und Leistung.

2.4.4. Methoden zum Datenbankzugriff in Flask

In diesem Abschnitt werden die maßgeblichen Methoden für die Arbeit mit SQL-Datenbanken in Flask vorgestellt, ihre Charakteristika erläutert und ein Vergleich gezogen. Hier werden zwei der meistgenutzten Ansätze vorgestellt und Vor- und Nachteilen aufgezeigt.

Direkter SQL-Zugriff via DB-Treiber

Bei dieser Methode wird direkt mit einem Datenbanktreiber (z. B. psycopg2, mariaDB oder mysql-connector) gearbeitet. SQL-Statements werden als Strings formuliert und über einen Cursor ausgeführt. Ein Cursor ist ein Steuerobjekt, das SQL-Befehle ausführt und Ergebnisse verwaltet.

Diese Methode eignet sich vorwiegend für kleine Anwendungen oder für hochoptimierte Spezialfälle, bei denen abstrakte Ein- und Auslesen-Befehle vorgenommen werden. Ein typischer Ablauf für diese Methode läuft wie folgt ab: [21], [22], [23]

1. Der Verbindungsaufbau zu einer Datenbank.
2. Ausführung eines SQL-Statements durch einen Cursor (z. B. SELECT, INSERT, UPDATE, DELETE).
3. Abrufen der Resultate und Überführen in eine geeignete Datenstruktur (z. B. eine Liste oder Dictionary).
4. Schließen der Verbindung nach fertiger Ausführung.

In der folgenden Tabelle 2 werden einige Vor- und Nachteile der Methode aufgeführt.

Vorteile	**Nachteile**
Maximale Kontrolle über das ausgeführte SQL	Gefahr für SQL-Injections, wenn Parameter nicht sicher gebunden werden
Kein Overhead durch zusätzliche Abstraktionsschichten	Manuelle Fehleranfälligkeit (z. B. bei Verbindungshandling oder Transaktionen)
Nützlich bei komplexen, datenbankspezifischen Abfragen, die ein ORM möglicherweise schwer modellieren kann	Schwer zu skalieren bei komplexer Domänenlogik
—	Geringe Lesbarkeit, insbesondere bei vielen Joins oder Abhängigkeiten

Tabelle 2: Vor- und Nachteile des direkten SQL-Zugriffs in Flask

Quelle: eigene Darstellung

Object-Relational Mapping

Object-Relational Mapping (ORM) eine Methode in der Softwareentwicklung, die es ermöglicht, Objekte aus einer objektorientierten Programmiersprache auf Tabellen in einer relationalen Datenbank abzubilden, um die Interaktion mit Datenbanken zu erleichtern. Entwickler müssen keine SQL-Abfragen mehr manuell erstellen. Sie interagieren einfach mit Objekten in ihrer Programmiersprache und das ORM-Tool wandelt dies in die passenden Datenbankoperationen um. [24], [25]

Eine der am meisten verwendeten Python-Bibliotheken für diese Methode ist SQLAlchemy. SQLAlchemy ist ein umfangreiches Toolkit und ORM für relationale Datenbanken. In Flask nutzt man normalerweise die Erweiterung Flask-SQLAlchemy, die SQLAlchemy in die Kontextstruktur von Flask integriert und den typischen Boilerplate-Aufwand reduziert. [26]

Mit Flask-SQLAlchemy definiert man Modelle als Python-Klassen, die automatisch auf Datenbanktabellen abgebildet werden (Declarative Mapping). Die Objekte dieser Klassen repräsentieren Datensätze, und Operationen daran (z. B. session.add(), session.commit()) generieren passende SQL-Befehle.[26]

In der folgenden Tabelle 3 werden einige Vor- und Nachteile der Methode bzw. von SQLAlchemy aufgeführt. Neben SQLAlchemy gibt es noch einige andere Python-Bibliotheken, die ORM nutzen z. B. SQLModel. Sie ermöglicht es, ein Modell zu definieren, das sowohl als ORM-Klasse als auch als Validierungs- / Datentypklasse (Pydantic) dient[27].

Vorteile	**Nachteile**
ORM-Abstraktion: Arbeiten mit Objekten statt SQL	Leicht höhere Komplexität als direkter SQL-Zugriff
Automatisches Erstellen und Aktualisieren von Tabellen	Performance bei sehr großen Datenmengen leicht geringer
Integriert gut in Flask	—
Kompatibel mit vielen Datenbanken (SQLite, MySQL, PostgreSQL, etc.)	—

Tabelle 3: Vor- und Nachteile der Verwendung von Flask-SQLAlchemy

Quelle: eigene Darstellung

2.5. Grundlagen der Datenvisualisierung

In diesem Unterkapitel der Arbeit wird die Art der Graphen zum Visualisieren der Messdaten erläutert und Anforderungen an die Umsetzung formuliert, so wie mögliche kurzer Überblick zu Bibliotheken für die Umsetzung durchgeführt.

2.5.1. Visualisierung von Zeitreihen

Für das Darstellen der zu visualisierenden Testergebnisse sollen nach Claus O. Wilke Liniendiagramme genutzt werden. Da die Messwerte der dazustellenden Testmodule anhand des Zeitverlaufs zugeordnet werden, haben die Daten somit eine inhärente Reihenfolge.

Das bedeutet, die Messwerte können nach dieser Zeitreihe geordnet werden und so Vor- und Nachfolger definiert werden, um zusätzliche Informationen zu erfassen. Die Messungen können nicht einfach vertauscht werden, ohne die zeitliche Information zu verlieren.

Liniendiagramme eignen sich für solche Daten, bei denen die Reihenfolge einen Teil der Informationen enthält, sehr gut. Zum Beispiel bei Zeitverläufen, Messreihen oder Funktionswerten für eine gute Darstellung. Die Linie im Diagramm verbindet die Punkte in dieser Reihenfolge und zeigt dadurch den Verlauf oder Trend über die Zeit bzw. andere Dimensionen auf. [28]

2.5.2. Anforderungen an Liniendiagramme

Im Folgenden werden Anforderungen für das Erstellen von gut dargestellten Grafen nach Claus O. Wilke beschrieben. Diese Anforderungen sollen bei dem erstellten Graphen und der Auswahl der Bibliotheken für die Erstellung im späteren Arbeitsverlauf berücksichtigt werden: [28]

Geeignete Skalierung der Achsen

Eine korrekte und sinnvolle Skalierung der Achsen ist entscheidend, um die Daten realistisch darzustellen. Die Achsen sollten gleichmäßig skaliert und klar beschriftet sein. Ungleichmäßige Skalierungen oder willkürliche Achsenschnitte können die visuelle Wahrnehmung von Trends verfälschen und sollten vermieden werden.

Klarheit und Lesbarkeit

Liniendiagramme müssen so gestaltet sein, dass die dargestellten Informationen schnell und eindeutig erfassbar sind. Dazu gehören klar beschriftete Achsentitel, Einheiten und Legenden. Linien und Beschriftungen sollten gut lesbar sein, ohne sich gegenseitig zu überlagern. Eine angemessene Linienbreite und Schriftgröße trägt wesentlich zur Verständlichkeit bei.

Konsistente und sparsame Liniengestaltung

Eine konsistente visuelle Gestaltung unterstützt die Vergleichbarkeit zwischen mehreren Linien. Unterschiedliche Linienarten, etwa durch Farbe oder Strichmuster, sollten nur eingesetzt werden, wenn sie zur Unterscheidung notwendig sind. Gleiche Bedeutungen müssen stets durch dieselbe visuelle Kodierung wiedergegeben werden. Eine übermäßige Verwendung von Markern oder Symbolen kann die Lesbarkeit beeinträchtigen und sollte vermieden werden.

Farben gezielt und barrierefrei einsetzen

Farben dienen der Unterscheidung und Hervorhebung, sollten jedoch bewusst und zurückhaltend eingesetzt werden. Die gewählten Farbtöne müssen ausreichend kontrastreich und auch für Personen mit Farbsehschwächen unterscheidbar sein. Wenn bestimmte Linien besonders hervorgehoben werden sollen, sollte dies dezent und mit klarer inhaltlicher Begründung geschehen, um den Fokus der Betrachtenden zu lenken, ohne die Gesamtwirkung zu stören.

Visueller Überlastung

Überflüssige grafische Elemente wie 3D-Effekte, Schatten, starke Gitterlinien oder dekorative Symbole sollten vermieden werden, da sie die Wahrnehmung der eigentlichen Daten stören. Ein reduziertes, funktionales Design lenkt den Blick auf die wesentlichen Inhalte und erhöht die Lesbarkeit des Diagramms.

Korrekte Darstellung und Kontext

Ein Liniendiagramm sollte den dargestellten Sachverhalt klar erkennbar machen. Dazu gehören ein prägnanter Titel, eine verständliche Legende und gegebenenfalls ein Untertitel oder eine

erläuternde Beschriftung. Werden mehrere Diagramme zum Vergleich gezeigt, ist eine einheitliche Achsenskalierung erforderlich, um falsche Eindrücke zu vermeiden und die Vergleichbarkeit sicherzustellen.

Behandlung überlappender Linien

Wenn mehrere Linien in einem Diagramm dargestellt werden, kann es zu Überlappungen kommen, die die Lesbarkeit beeinträchtigen. In solchen Fällen empfiehlt Wilke den Einsatz von Transparenz, dünneren Linien oder unterschiedlichen Stricharten. Diese Maßnahmen ermöglichen es, auch bei komplexen Darstellungen den Überblick zu behalten.

Verständliche Hervorhebung wichtiger Daten

Besonders relevante Trends, Ereignisse oder Datenpunkte dürfen hervorgehoben werden, beispielsweise durch eine andere Linienfarbe oder eine beschriftete Markierung. Solche Hervorhebungen sollten jedoch gezielt und sparsam erfolgen, um den Blick der Betrachtenden zu lenken, ohne die übrigen Daten in den Hintergrund zu drängen.

2.5.3. Graphenerstellung mit JS

In diesem Abschnitt werden einige JavaScript-Bibliotheken für das Erstellen von Graphen In Kurzfassung erläutern, um einen groben Überblick zu erhalten. Die endgültige Auswahl wird in Kapitel 3 genauer beschrieben.

Chart.js Eine einfache Bibliothek für Standarddiagramme wie Balken, Linien oder Kreisdiagramme. Schnell eingebunden, gute Standardoptik, ideal für kleinere Projekte oder schnelle Visualisierungen. [29]

D3.js Eine mächtige Low-Level-Bibliothek, die mit Scalable Vector Graphics (SVG), Canvas und HTML arbeitet. Extrem flexibel für individuelle und interaktive Visualisierungen, aber mit einer steilen Lernkurve. [30]

Plotly.js Bietet viele interaktive Diagramme (Zoom, Hover, Export als PNG). Unterstützt auch 3D- und wissenschaftliche Charts. Gut geeignet für Dashboards und Datenanalyse. [31]

ECharts Eine leistungsstarke Open-Source-Bibliothek von Apache. Unterstützt viele Diagrammtypen (z. B. Heatmaps, Maps, Candlesticks) mit Animationen und Interaktivität. [32]

vis.js Spezialisiert auf Netzwerk- und Zeitachsenvisualisierungen. Eignet sich Besonders geeignet für Knoten-Graphen, Beziehungen oder Prozessdiagramme mit Interaktivität. [33]

Recharts Eine React-spezifische Bibliothek, die auf D3.js basiert. Bietet eine einfache Komponenten-API für gängige Diagramme. Ideal, wenn eine Web-Applikation mit React entwickelt wird. [34]

2.6. Anforderungen an modulare Softwareentwicklung (nach ISO/IEC 9126)

Die Norm ISO/IEC 9126 beschreibt ein Qualitätsmodell für Softwareprodukte, das mehrere Haupt- und Untermerkmale definiert. Diese Merkmale lassen sich auf die modulare Softwareentwicklung übertragen, da sie zentrale Eigenschaften wie Wartbarkeit, Erweiterbarkeit und Austauschbarkeit quantifizierbar machen. Eine modulare Architektur erfüllt diese Qualitätsziele, wenn Sie folgende Eigenschaften grundlegend erfüllt:[35]

- Funktionalität: Jedes Modul soll die vorgesehenen Aufgaben korrekt und zweckmäßig ausführen. Eine hohe Funktionalität setzt voraus, dass Module die geforderten Spezifikationen erfüllen, interoperabel mit anderen Komponenten sind und relevante Standards einhalten. *Untermerkmale: Eignung, Korrektheit, Interoperabilität, Konformität, Sicherheit.*
- Zuverlässigkeit: Module sollen auch unter unerwarteten Bedingungen stabil arbeiten und definierte Wiederherstellungsmechanismen besitzen. Eine hohe Zuverlässigkeit gewährleistet, dass Fehlfunktionen lokal begrenzt bleiben und das Gesamtsystem funktionsfähig bleibt. *Untermerkmale: Reife, Fehlertoleranz, Wiederherstellbarkeit.*
- Benutzbarkeit: Module und Schnittstellen sollen verständlich, erlernbar und bedienbar sein. Diese Anforderung gilt sowohl für Benutzerschnittstellen als auch für APIs, um eine konsistente Integration und Nutzung zu ermöglichen. *Untermerkmale: Verständlichkeit, Erlernbarkeit, Bedienbarkeit.*
- Effizienz: Module sollen vorhandene Ressourcen optimal nutzen und geforderte Leistungswerte einhalten. Eine effiziente Implementierung trägt zu einem stabilen Laufzeitverhalten und einer guten Skalierbarkeit des Gesamtsystems bei. *Untermerkmale: Zeitverhalten, Ressourcenverhalten.*
- Wartbarkeit: Module sollen leicht analysierbar, anpassbar und testbar sein. Änderungen müssen möglichst lokal vorgenommen werden können, ohne unbeabsichtigte Auswirkungen auf andere Systemkomponenten zu verursachen. *Untermerkmale: Analysierbarkeit, Änderbarkeit, Stabilität, Testbarkeit.*
- Portabilität: Module sollen an unterschiedliche Zielumgebungen anpassbar und bei gleichbleibender Schnittstelle austauschbar sein. Dadurch wird die Wiederverwendung und langfristige Nutzbarkeit der Software verbessert. *Untermerkmale: Anpassbarkeit, Installierbarkeit, Konformität, Austauschbarkeit.*

Die Berücksichtigung dieser Qualitätsmerkmale bei der Entwicklung modularer Systeme stellt sicher, dass Software langfristig wartbar, erweiterbar und zuverlässig bleibt. Das Qualitätsmodell der ISO/IEC 9126 bietet damit eine strukturierte Grundlage zur Bewertung und Verbesserung der architektonischen Qualität modularer Anwendungen. [35]

3. Analyse und Konzeption

In diesem Kapitel werden die grundlegenden Anforderungen und konzeptionellen Überlegungen für die Entwicklung der Webapplikation beschrieben. Ziel ist es, eine fundierte Basis für die spätere Implementierung zu schaffen, indem sowohl die funktionalen als auch die nicht-funktionalen Anforderungen analysiert und geeignete technische Konzepte erarbeitet werden. Dazu werden zunächst die Ausgangssituation und die zu verarbeitenden Datenquellen untersucht. Die erarbeiteten Konzepte bilden die Grundlage für das Design und die Realisierung der Software im weiteren Verlauf der Arbeit.

3.1. Definition der Anforderungen

In diesem Abschnitt werden die funktionalen und nicht-funktionalen Anforderungen der zu entwickelnden Web-Applikation beschrieben. Sie wurden in Abstimmung mit den späteren Nutzerinnen und Nutzern sowie den betreuenden Personen im Unternehmen festgelegt. Die Anforderungen bilden die Grundlage für den Entwurf und die Umsetzung der Software und dienen dazu, deren Zielsetzung, Funktionsumfang und technische Rahmenbedingungen klar zu definieren.

3.1.1. Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben die vorgesehenen Funktionen und Abläufe der Applikation. Sie definieren, was die Anwendung leisten soll und wie die Nutzerinnen und Nutzer mit ihr interagieren können.

1. Navigation und Benutzerführung
   Die Applikation soll eine einfache und übersichtliche Navigation ermöglichen. Der Wechsel zwischen den Hauptfunktionen erfolgt über eine feststehende Menüleiste im oberen Bereich der Benutzeroberfläche. Dadurch können die Bereiche Upload, Berichte und Analyse direkt aufgerufen werden.

2. Einlesen von XML-Dateien
   Das Einlesen der automatisch generierten XML-Berichte erfolgt über eine Upload-Seite. Die Anwendung soll sowohl ältere als auch neuere XML-Strukturen verarbeiten können, da der Teststand unterschiedliche Versionen erzeugt.

3. Datenverarbeitung und Speicherung
   Nach dem Upload sollen die XML-Dateien automatisch geparst, validiert und in die Datenbank überführt werden. Doppelte Einträge sollen erkannt und vermieden werden.

4. Anzeige der Berichte
   Die gespeicherten Berichte sollen tabellarisch dargestellt werden. Jede Zeile repräsentiert einen Testbericht und zeigt die wichtigsten Informationen wie Datum, Seriennummer, Materialnummer und Testergebnis an.

5. Filter- und Suchfunktionen
   Die Berichtstabelle soll über Filteroptionen verfügen, um gezielt nach bestimmten Kriterien (z. B. Datum, Seriennummer, Testmodul oder Ergebnisstatus) zu suchen.

6. Grafische Darstellung der Testdaten
   Die Anwendung soll die Messergebnisse der einzelnen Module grafisch darstellen. Die Diagramme sind nach Modulen sortiert und enthalten eine klare Achsenbeschriftung sowie Legenden. Das Layout der Diagramme soll sich an den Graphen des Teststandprogramms orientiert, um den Nutzerinnen und Nutzern die Interpretation zu erleichtern.

7. Systemmeldungen und Fehlermanagement
   Nach erfolgreichen oder fehlerhaften Aktionen (z. B. Upload, Datenbankeintrag, Filterabfrage) sollen entsprechende Systemmeldungen angezeigt werden, um den Benutzerstatus transparent zu machen.

8. Exportfunktionen
   Die Anwendung soll die Möglichkeit bieten, ausgewählte Graphen in ein externes Format zu exportieren. Dadurch können die Graphen auch außerhalb der Anwendung weiterverarbeitet werden.

3.1.2. Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen beschreiben die Qualitätsmerkmale der Software. Sie legen fest, unter welchen Bedingungen die Applikation betrieben werden soll und welche Eigenschaften sie erfüllen muss.

1. Plattform und Laufzeitumgebung
   Die Web-Applikation wird auf einem unternehmenseigenen Apache2-Server betrieben. Sie basiert auf Python und dem Webframework Flask.

2. Performance
   Das System muss auch bei größeren XML-Dateien zuverlässig arbeiten. Hierbei wird von XML-Daten mit einer vielzahl von Testmodulen ausgegangen. Eine aktive Laufzeitoptimierung ist nicht erforderlich, solange der Arbeitsfluss nicht beeinträchtigt wird.

3. Usability
   Die Benutzeroberfläche soll intuitiv bedienbar sein und eine klare Struktur aufweisen. Eingabefehler sollen vermieden und, wenn möglich, automatisch erkannt werden.

4. Modularität und Wartbarkeit
   Die Applikation ist modular aufgebaut, um eine einfache Pflege und spätere Erweiterung zu ermöglichen. Änderungen an einzelnen Komponenten sollen keine Anpassungen an anderen Modulen erfordern.

5. Sicherheit und Datenintegrität
   XML-Dateien müssen vor dem Einlesen auf korrekte Struktur und mögliche Sicherheitsrisiken geprüft werden (z. B. Schutz vor fehlerhaften oder manipulierten Dateien).

6. Fehler- und Ausnahmebehandlung
   Unerwartete Systemfehler sollen abgefangen und im Log gespeichert werden. Für Benutzerinnen und Benutzer werden stattdessen verständliche Fehlermeldungen ausgegeben.
7. Skalierbarkeit
   Das System soll bei Bedarf mit minimalem Aufwand um zusätzliche Funktionen, Testmodule oder Datenquellen erweitert werden können.
8. Kompatibilität
   Die Applikation soll auf allen gängigen Browsern lauffähig sein (z. B. Chrome, Edge, Firefox).
9. Dokumentation und Nachvollziehbarkeit
   Der Code soll übersichtlich dokumentiert werden. Wichtige Abläufe (Upload, Datenverarbeitung, Visualisierung) werden nachvollziehbar beschrieben.

3.1.3. Anforderungen an die Datenbank

Die Datenbank ist ein zentraler Bestandteil der Anwendung. Sie speichert alle relevanten Testdaten, Parameter und Informationen aus den XML-Berichten.

1. Datenbanksystem
   Die Datenbank basiert auf dem relationalen Datenbanksystem MariaDB.
2. Vollständige Abbildung der XML-Daten
   Sie muss in der Lage sein, sämtliche in den XML-Berichten enthaltenen Daten vollständig abzubilden, um die Wiederherstellung eines gesamten Testberichts theoretisch zu ermöglichen.
3. Vermeidung von Datendopplungen
   Datendopplungen sollen vermieden werden. Häufig vorkommende Informationen (z. B. Teststandname, Versionsnummern, DUT-Typen) werden ausgelagert und über Fremdschlüssel referenziert.
4. Strukturierung von Gerätedaten
   Für DUT-Typen und Seriennummern werden separate Informationstabellen angelegt, um Redundanzen zu vermeiden und die Datenstruktur klar zu halten.
5. Erweiterbarkeit der Datenbankstruktur
   Die Datenbankstruktur soll so gestaltet sein, dass zukünftige Teststände oder zusätzliche Testmodule problemlos integriert werden können.
6. Versionierung und Migration
   Änderungen an der Datenbankstruktur werden verwaltet, um eine konsistente Weiterentwicklung der Datenbank sicherzustellen.

7. Datenvalidierung und Konsistenzprüfung
   Beim Einfügen neuer Datensätze sollen Validierungen sicherstellen, dass die Werte logisch und formal korrekt sind (z. B. keine Nullwerte bei Pflichtfeldern, korrekte Datentypen).

Die beschriebenen Anforderungen bilden den Rahmen für die Entwicklung der Web-Applikation. Während die funktionalen Anforderungen die konkreten Aufgaben und Abläufe definieren, legen die nicht-funktionalen Anforderungen fest, wie die Anwendung qualitativ umgesetzt werden soll. Zusammen mit den Datenbankanforderungen bilden sie die Grundlage für den folgenden Entwurf der Systemarchitektur und die spätere Implementierung.

3.2. Grundlegender Ablauf des Programmes

In dem folgenden Unterkapitel wird der geplante Ablauf der Applikation grundlegend beschrieben. Hierzu wird ein Ablaufdiagramm zur visuellen Unterstützung verwendet.

Abbildung 6 zeigt den geplanten Ablauf der Datenverarbeitung innerhalb der entwickelten Web-Applikation. Der Prozess beginnt mit dem Einlesen der XML-Dateien, die die Prüf- und Messdaten enthalten. Anschließend erfolgt eine Validierung der Datenstruktur und des Formats, um sicherzustellen, dass die XML-Dateien den definierten Spezifikationen entsprechen. Bei fehlerhaften oder unvollständigen Dateien wird der Prozess abgebrochen. Sind die Daten gültig, werden sie mit den vorhandenen Datenbankeinträgen abgeglichen. Abhängig vom Ergebnis werden entweder fehlende Datensätze ergänzt oder neue Einträge vollständig eingefügt. Dadurch wird sichergestellt, dass die Datenbank stets konsistente und aktuelle Informationen enthält.

Nach dem erfolgreichen Datenimport erfolgt die Auswahl eines Datensatzes, dessen Inhalte grafisch dargestellt werden. Optional kann der erzeugte Graph als PNG-Datei gespeichert werden.

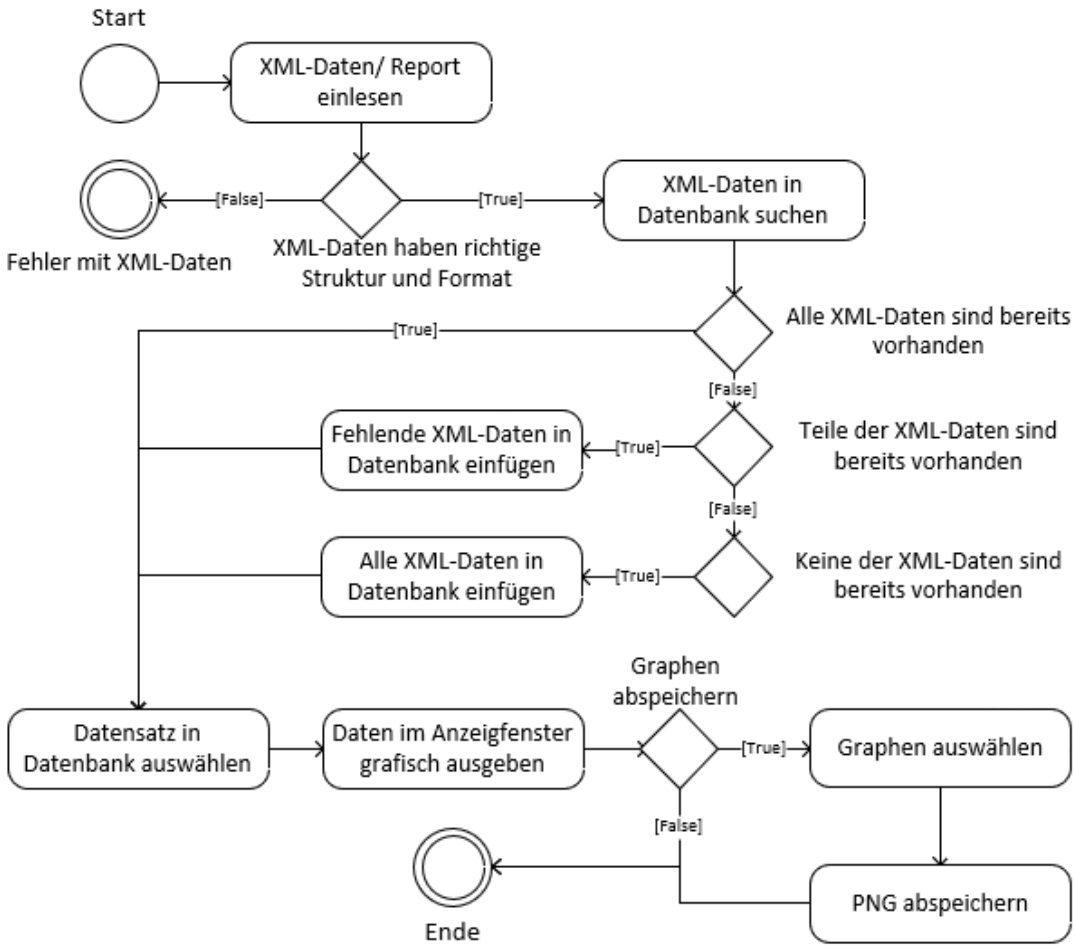

Abbildung 6: Grundlegedes Ablaufdiagramm der Nutzung der Web-Applikation
Quelle: Eigene Darstellung mit Microsoft Visio

3.3. Analyse der generierten XML-Berichte und bestehenden Strukturen

In diesem Unterkapitel wird der Aufbau der automatisch generierten Testberichte aus dem Teststand und der vorhandenen Ausgabe- und Speicherstruktur erläutert. Dies ist relevant für die Erstellung der Datenbank und das Verständnis der Ein- und Auslesefunktionen der Web-Applikation.

3.3.1. XML-Berichtsstrukturschema

Die vom Teststand automatisch generierten Berichte folgen einem konstanten Strukturschema, welches sich bis zu einer gewissen Ebene der XM-Struktur in jedem Bericht wiederholt. Wie im Kapitel „Grundlagen“ beschrieben, kann ein Element Attribute, andere Elemente und einen Inhaltswert beinhalten.

Das Stammelement heißt in allen automatisch generierten Berichten „test“ und besitzt immer das Attribut „id“. Die Zahl in diesem Attribut beschreibt den Typ des DUTs, die in diesem Test geprüft wurden.

Das Stammelement hat die Unter- bzw. Kinderelemente „info“, „testbench“, „string“ und dreimal „testmodule“. Diese Elemente haben wiederum alle weiter Unterelemente und Attribute. In der folgenden Abbildung 7 ist die unveränderliche Struktur eines Testberichtes abgebildet, welcher erfolgreich durchgeführt wurde.

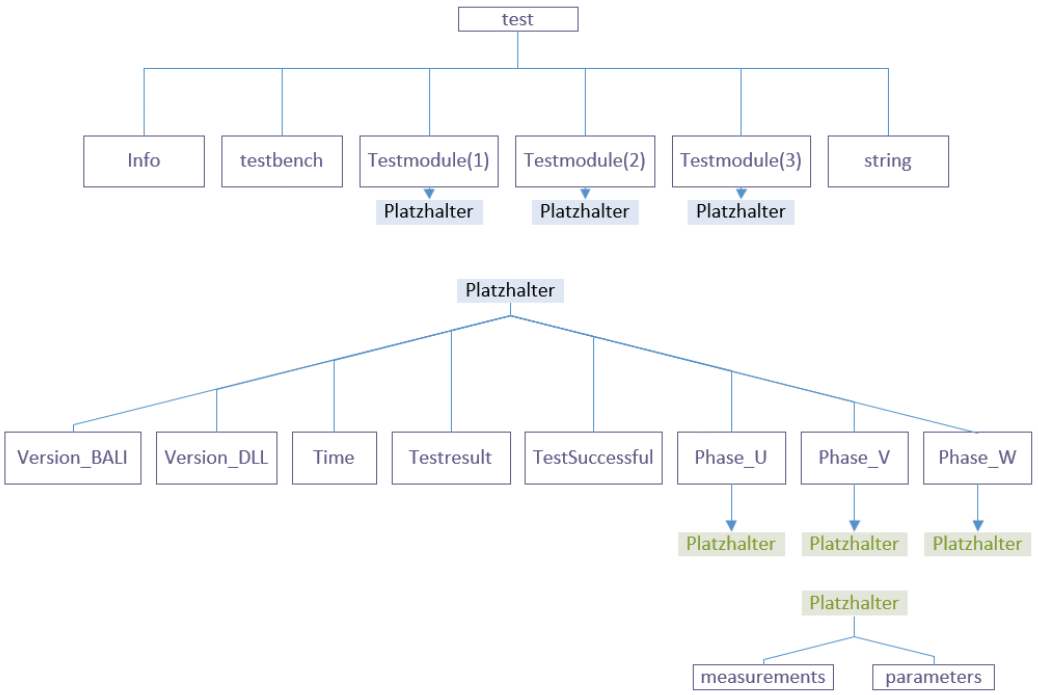

Abbildung 7: Aufbau unveränderlicher XML-Berichtstruktur
Quelle: Eigene Darstellung mit Microsoft Visio

Hierbei sollte angemerkt werden, dass in allen Testberichten die Elemente, die kein Unterelement besitzen, gleich aufgebaut sind. Sie besitzen mindestens das Attribut „name“ und ihre Elementenbezeichnung ist immer nach dem Datentyp ihres Inhalts benannt. Für ein besseres Verständnis siehe Abbildung 8.

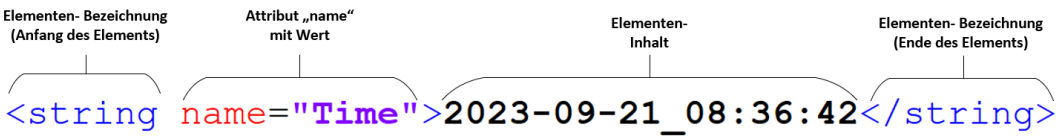

Abbildung 8: Elementaufbau aus XML-Bericht
Quelle: Eigene Darstellung mit Microsoft Visio

Jedes der Elemente „testmodule“ hat ein Attribut „name“, durch welches Sie unterschieden werden können. Das Element „string“ hat ein Namensattribut mit der Bezeichnung „test sequence status“. Der Inhalt dieses Elementes gibt an, ob der Test erfolgreich abgeschlossen wurde oder ob es einen Fehler bzw. eine Grenzüberschreitung von Messwerten gab. Die hier genannten Eigenschaften sind in der Abbildung 9 zu erkennen.

```
<?xml version="1.0" encoding="UTF-8"?>
    <test id="124">
    <info></info>
    <testbench></testbench>
    <testmodule name="DriverConsumptionTest"></testmodule>
    <testmodule name="PulseTest␣FSW␣L"></testmodule>
    <testmodule name="XPowerTest"></testmodule>
    <string name="test␣sequence␣status">
        Test sequence successfully finished.</string>
    </test>
```

Abbildung 9: Direkte Kinderelemente unter Stammelement
Quelle: Eigene Darstellung

Das Element „info“ enthält in seinen Unterelementen die allgemeinen Testinformationen, wie die Startzeit des Tests, die Typen-ID, die Konfigurationsbezeichnung des Teststandes, die Bezeichnung des angeschlossenen Carriers und die Seriennummern des DUTs. Diese Elemente haben keine Unterelemente und sind alle mit einem Namensattribut und einem Inhalt definiert. Für die genauen Bezeichnungen und Struktur siehe Abbildung 10.

```
<info>
    <string name="Time">2023-09-21_08:36:33</string>
    <string name="Material_number">124</string>
    <string name="Configuration">L_3DUT_3P1Q</string>
    <string name="ID_Carrier_left">Carrier 1</string>
    <string name="ID_DUT_R">10-29550</string>
    <string name="ID_DUT_S">10-29396</string>
        <string name="ID_DUT_T">10-29482</string>
</info>
```

Abbildung 10: XML-Strukturbeispiel Info-Element
Quelle: Eigene Darstellung

Die Teststandbezeichnung und die Versionen der Hard- und Software werden im Element „testbench“ gespeichert. Diese Elemente enthalten auch keine weiteren Unterelemente und sind alle mit einem Namensattribut und einem Inhalt versehen. Die genauen Bezeichnungen und die Struktur sind in Abbildung 11 zu sehen.

```
<testbench>
    <string name="Testbench_name">USTB_DtWind</string>
    <string name="Testbench_HW_revision">V1.5</string>
    <string name="Version_GUI">V1.4.1</string>
        <string name="Version_controller">V1.3.2</string>
</testbench>
```

Abbildung 11: XML-Strukturbeispiel Testbench-Element
Quelle: Eigene Darstellung

Die Testmodule-Elemente sind alle nach dem gleichen Schema aufgebaut. Die ersten Unterelemente enthalten Grundinformationen zu dem Status des Testmoduls, der Tages- und Uhrzeit des Tests und welche Versionen davon verwendet wurden. Danach befinden sich noch drei weitere Elemente im Element „module“. Diese Elemente heißen „Phase U“, „Phase V“ und „Phase W“. Sie besitzen kein Namensattribut, siehe Abbildung 12.

```
<testmodule name="DriverConsumptionTest">
    <integer name="TestSuccessful" min="1" max="1">1</integer>
    <string name="Testresult">
        Test finished successfully!</string>
    <string name="Time">2023-09-21_08:36:42</string>
    <string name="Version_DLL">V1.2.4</string>
    <string name="Version_BALI">V1.2.4</string>
    <Phase_U>...</Phase_U>
    <Phase_V>...</Phase_V>
    <Phase_W>...</Phase_W>
</testmodule>
```

Abbildung 12: XML-Strukturbeispiel Testmodulheader
Quelle: Eigene Darstellung

Diese Phasenelemente enthalten die Parameter und die Messdaten für die DUTs. Jede Phase enthält die Messwerte für ein DUT. Somit besitzen alle Module unter dem Element „Phase U“ die Messwerte und Parameter für den ersten DUT, dessen Seriennummer unter dem Element mit dem Namensattribut „ID DUT R“ in Abbildung 10 zu finden ist.

In dem Element „parameters“, welches in jedem Phasenelement vorkommt, befinden sich die voreingestellten Rahmbedingungen und Anforderungen für das Testmodul. Diese enthalten alle ein Namensattribut und einen Inhalt.

Die Elemente unter dem Eintrag „measurements“ enthalten die Messdaten der Testmodule. Sie haben oft noch extra Attribute wie „min“ oder „max“, welche die Toleranzgrenzen beschreiben. Diese Toleranzgrenzen zeigen, in welchem Bereich der jeweilige Elementinhalt angesiedelt sein muss, um kein positives bzw. fehlerfreies Ergebnis zu erzeugen. Die meisten Attributswerte finden sich schon in den Parametern wieder und sind somit doppelt im Bericht zu finden. Nur bei

den Floatblock-Elementen, also Elementen mit der Bezeichnung Floatblock, sind besondere Attribute zu finden, welche nicht gedoppelt vorkommen. Diese Elemente umfassen zusätzlich die Attribute „size“ und „duration“, welche die Anzahl der in dem Inhalt enthaltenen Floatwerte und die Dauer der Werteaufnahme angeben Dies ist in Abbildung 13 zu erkennen. Diese sind für die grafische Darstellung besonders relevant.

```
<floatblock name="Time" size="698" duration="959.952" unit="s">
0.71365,2.1370,3.5323,...</floatblock>
```

Abbildung 13: Beispiel XPowertest Floatblock-Elemente
Quelle: Eigene Darstellung

Auf der Phasenebene treten keine Veränderungen in der XML-Struktur auf, lediglich die Elemente unter den Phasenelementen von Bericht zu Bericht variieren etwas. Die beträchtliche Ausnahme von dieser Regel bilden die Berichte, bei denen ein Fehler aufgetreten ist oder sogar der Testvorgang abgebrochen wurde. Bei diesen Berichten hört die XML-Struktur bei den fehlgeschlagenen Testmodulen schon auf. Der Testbericht ist aber in sich geschlossen. Das bedeutet, die XML-Struktur ist vollkommen und kann von einem XML-Parser eingelesen werden. Es befinden sich nur weniger Testmodul-Elemente in dem Bericht und im Element für den Testsequenzstatus ist folgender Inhalt enthalten „Test sequence finished with errors“. Zudem wird bei dem Testmodul, bei dem der Fehler aufgetreten ist, unter dem Element mit dem Namen „Testresult“ ein Fehlercode und seine Bedeutung ausgegeben. In Abbildung 14 ist ein Beispiel für einen Bericht, welcher beim ersten Testmodul einen Fehler ausgegeben hatte. Bei einem Ausfall eines der zu testenden DUTs gibt es die Besonderheit, dass die anderen noch funktionsfähigen DUTs keine weiteren Datenaufnahmen mehr machen können. Denn das System benötigt alle drei Phasen, um den Test durchzuführen. Die XML-Struktur ist in sich geschlossen, aber es werden keine Daten von den anderen DUTs mehr aufgenommen.

```
<testmodule name="DriverConsumptionTest">
    <integer name="TestSuccessful" min="1" max="1">0</integer>
    <string name="Testresult">Idrv not within limits on Phase U
0x30010019 !
0x30010013</string>
    <string name="Time">2021-03-10_09:10:37</string>
    <string name="Version_DLL">V1.2.4</string>
    <string name="Version_BALI">V1.2.4</string>
    <Phase_U>...</Phase_U>
    <Phase_V>...</Phase_V>
        <Phase_W>...</Phase_W>
</testmodule>
```

Abbildung 14: Beispiel Fehler bei DriverConsumptionTest
Quelle: Eigene Darstellung

3.3.2. Einbindungskonzept der XML-Daten in die Applikation

Die XML-Daten sollen über die Bibliothek `lxml` eingebunden werden. Die Wahl für das Nutzen dieser Bibliothek wird in folgenden Stichpunkten erläutert: [9]

1. Hohe Performance
   Durch die Implementierung in C ist *lxml* deutlich schneller als rein in Python geschriebene Parser.
2. Kompatibel zu ElementTree
   lxml ist API-kompatibel zur Standardbibliothek *xml.etree.ElementTree*, was die Migration erleichtert.
3. XPath- und XSLT-Unterstützung
   `lxml` ermöglicht die Nutzung XML-Technologien wie XPath zur gezielten Abfrage von Knoten sowie XSLT zur Transformation von XML-Daten in andere Formate (z. B. HTML oder strukturierte Textausgaben).
4. Schema-Validierung
   Das Modul unterstützt die Validierung von XML-Dokumenten anhand von XSD- oder DTD-Schemata. Was die Absicherung beim Einlesen stark erhöhen kann.
5. Namespace-Unterstützung
   Ermöglicht den Umgang mit standardisierten XML-Formaten, was als Sicherung beim Parsen genutzt werden kann.

Aufgrund dieser genannten Eigenschaften wurde *lxml* für das Einbinden ausgewählt.

3.4. Datenbankdesign und Strukturkonzeption

In diesem Unterkapitel soll der Verlauf der Erstellung der Datenbank beschrieben und nachvollzogen werden. Es wurde in den Anforderungen festgelegt, dass MariaDB genutzt werden soll. MariaDB ist ein kostenloses, relationales Open-Source-Datenbankmanagementsystem, das aus einer Abspaltung von MySQL entwickelt wurde. MySQLs früherer Hauptentwickler Michael Widenius hat das Projekt ins Leben gerufen. [36]

3.4.1. Betrachtung der Anforderungen

Für das genaue Interpretieren und Umsetzen der Anforderungen wurde festgelegt, dass die Ersteller/-in der Datenbank erst ein Konzept nach den festgelegten Anforderungen erarbeiten und umsetzen. Diese Umsetzung wird dann im Laufe der weiteren Erstellung der Applikation, falls nötig, angepasst.

3.4.2. Konzeptionelles Datenmodell

Für das Erstellen der Konzeption des Datenbankmodelles wurde die XML-Struktur analysiert, um die Kenntnisse und Muster aus Unterkapitel 3.3 abzuleiten. Aus diesen Erkenntnissen wurde dann ein ER-Modell erstellt, siehe Abbildung 15. Im Folgenden wird das ER-Modell kurt erläutert.

Durch das Durchführen eines Testes wird ein Bericht erstellt. Dieser Bericht enthält einige Attribute, welche Testinformationen, darunter die Seriennummern, das Datum und die Uhrzeit des Tests, die Materialnummer (DUT-ID), die Konfigurations-Version und die Carrier-Bezeichnung enthalten. Zudem besitzt der Bericht ein Unterelement „Testbenchinformationen“, das die Hard- und Softwareversionen des Teststandes enthält. In der Unterentität Testmodelle befinden sich Versions-, Zeit- und Ergebnisinformationen sowie drei Datensätze, welche die Testparameter und die Messdaten enthalten. Diese Datensätze können den Phasen des Stromnetzes des Teststandes und somit den DUTs zugeordnet werden. Zusätzlich zu dem Aufbau wurden hier schon Schlüsselattribute hinzugefügt, die später die primären Schlüssel der Datenbanktabellen dienen sollen, orange markiert.

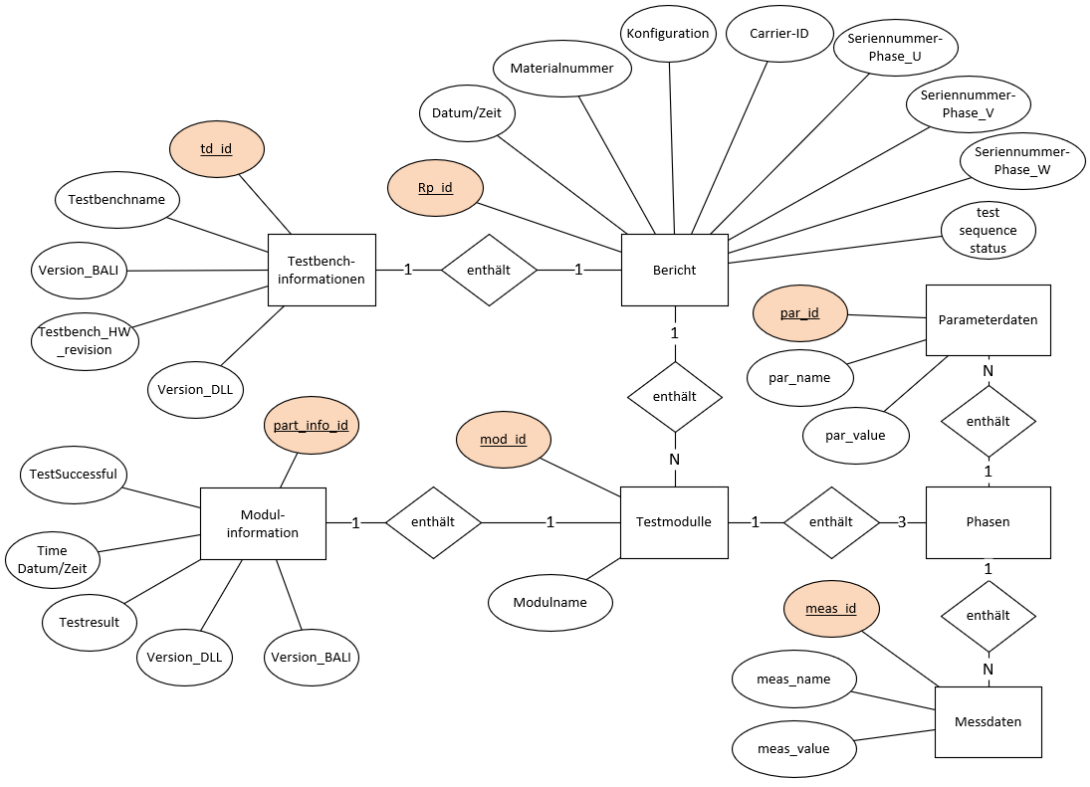

Abbildung 15: ER-Modell Überlegung der Datenbankstruktur
Quelle: Eigene Darstellung mit Microsoft Visio

Aus diesem ER-Model wurde die Datenbanktabelle und ihre Verbindungen über Fremdschlüssel abgeleitet. In Abbildung 16 werden die Datenbanktabellen, ihre genauen Namensbezeichnungen der Tabelle, der Datentyp in der Datenbank und ihr Inhalt sowie ihre Verbindungen abgebildet.

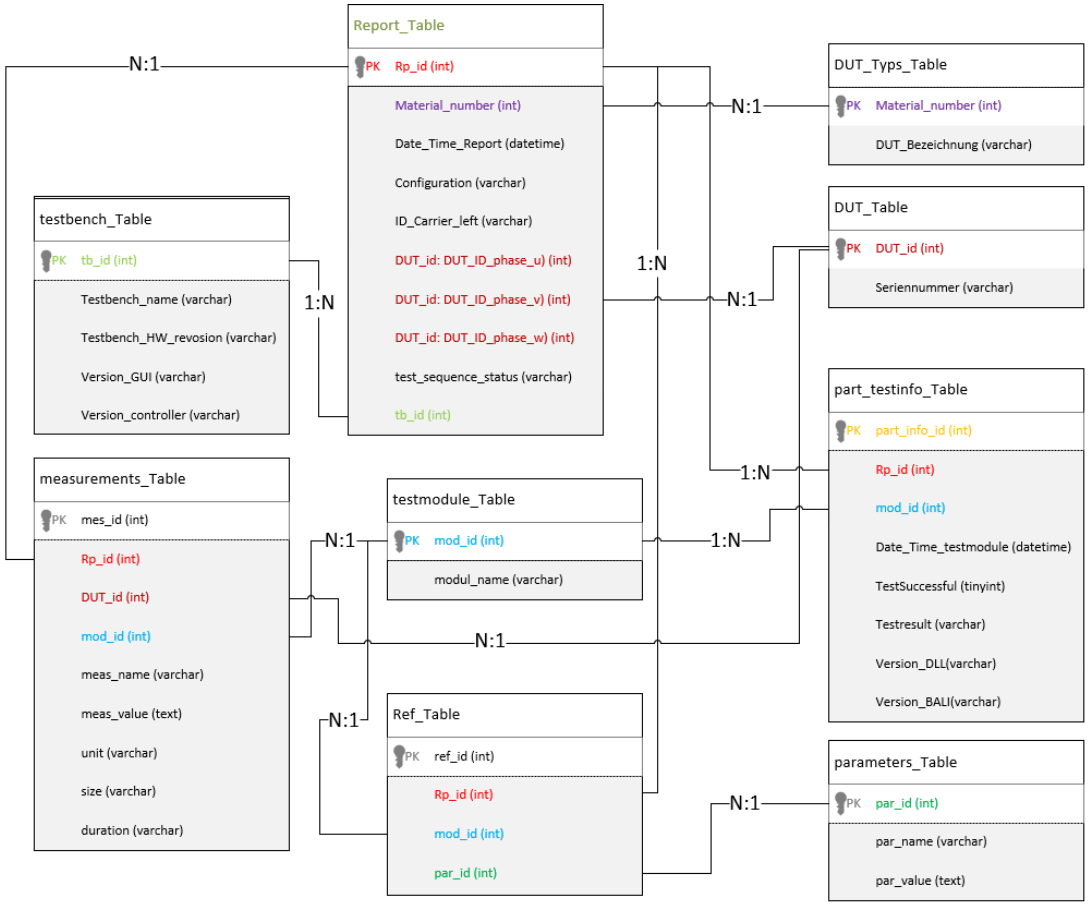

Abbildung 16: Darstellung der Datenbanktabellen und ihrer Verbindungen
Quelle: Eigene Darstellung mit Microsoft Visio

Zur persistenten Datenspeicherung wird das ORM-Framework *Flask-SQLAlchemy* verwendet, das auf *SQLAlchemy* basiert. Es ermöglicht die objektorientierte Abbildung relationaler Datenbanken und vereinfacht den Zugriff auf Daten über Python-Klassen. Dadurch wird eine saubere Trennung von Anwendungslogik und Datenzugriffsschicht erreicht.

3.5. Grundkonzept des Benutzeroberflächen-Designs

In diesem Abschnitt soll das Grundkonzept der Benutzeroberfläche beschrieben und dargestellt werden.

Die Benutzeroberfläche wird in drei Seiten aufgeteilt, welche sich mit dem Einlesen der Daten, dem darstellen und auswählen zu visualisierenden Daten und dem Ausgeben des Ergebnisses befasst. Für das Navigieren zwischen den Seiten soll sich auf jeder Seite ein Navigationsmenü im oberen Bereich der Seite befinden. Zudem sollen alle Seiten einen Bereich für Systemnachrichten wie Bestätigung oder Fehlermeldungen und ein Label für die Seitenüberschrift besitzen.

Diese soll direkt unter dem Navigationsmenü angesiedelt werden, um die Grundstruktur identisch aufzubauen. In den nachfolgenden Text werden die Konzepte der Benutzeroberflächen und ein

grundlegendes Benutzungskonzept der einzelnen Seiten anhand der Abbildungen 17, 18 und 19 beschrieben.

Für das Einlesen der XML-Daten soll eine Eingabebox genutzt werden, in die die XML-Struktur kopiert wird. Die Eingaben sollen über einen Button unter der Eingabebox bestätigt werden. Bei einem erfolgreichen Einlesen und Einfügen in die Datenbank soll eine Bestätigungsnachricht im Bereich für Systemnachrichten erscheinen. Bei einem Fehler soll in diesem Bereich eine Fehlermeldung mit möglicher Lösung erscheinen. Der Aufbau dieser Seite ist in der Abbildung 17 dargestellt.

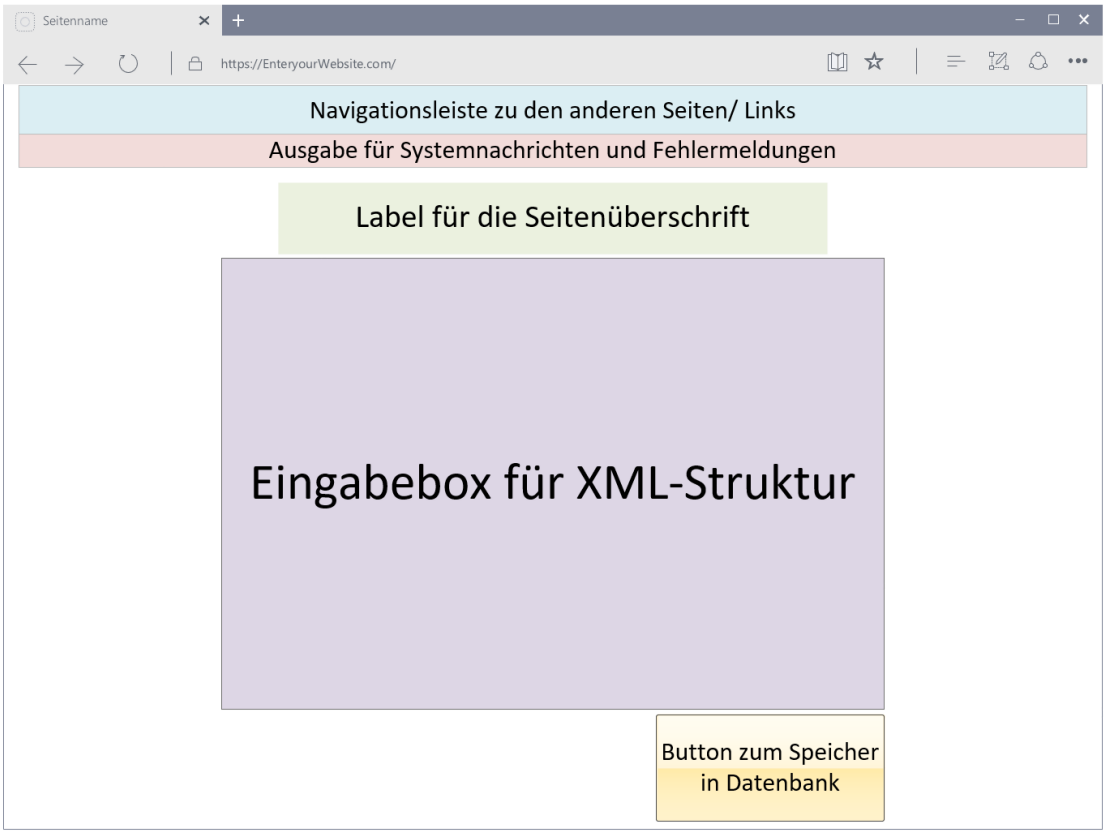

Abbildung 17: Benutzeroberflächenentwurf der Seite zum Einlesen der XML-Struktur
Quelle: Eigene Darstellung mit Microsoft Visio

Die Seite für das Darstellen und Auswählen der zu visualisierenden Daten soll im oberen Abschnitt der Seite einen Bereich für Filtereinstellungen besitzen, mit möglichen Filteroptionen wie Datum oder Seriennummer. Neben der Filtereinstellung sollen zwei Buttons für das Bestätigen und Zurücksetzen der Filtereinstellungen sein. Bei einem Fehler bei den Filtereingaben soll im Bereich für Systemnachrichten eine Fehlermeldung mit möglicher Lösung erscheinen. Im unteren Bereich der Seite soll sich eine Tabelle, welche die eingelesenen Berichte in der Datenbank anzeigt. Diese soll durch die Filtereinstellungen angepasst werden. Hierbei muss auf eine Anzeige Möglichkeit für längere Tabellenstrukturen berücksichtigt werden, um die Benutzung effizient zu halten. Das Das Grundkonzept ist in Abbildung 18 dargestellt.

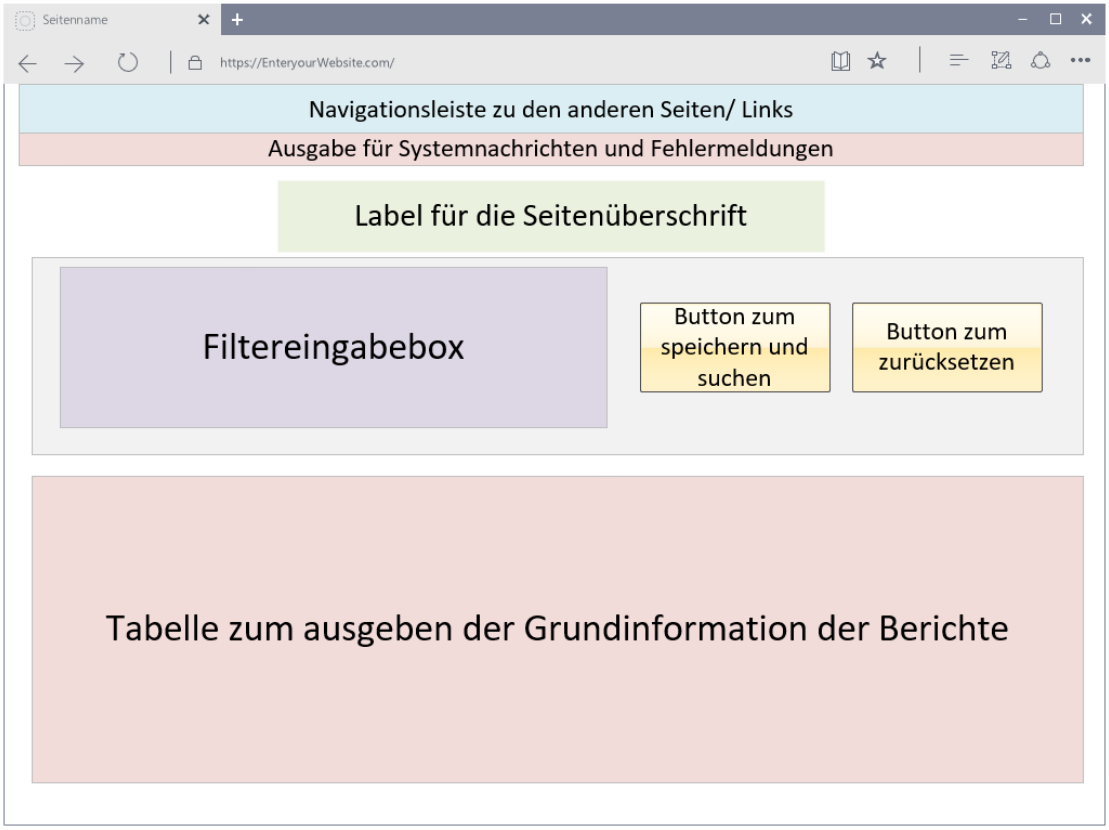

Abbildung 18: Benutzeroberflächenentwurf der Seite zum Ausgeben der Berichtstabelle
Quelle: Eigene Darstellung mit Microsoft Visio

Für das Ausgeben der ausgewählten Berichtsdaten soll im oberen Abschnitt der Seite eine Instanz eingefügt werden, die die Information zu den ausgewählten Berichtsdaten anzeigen. Im unteren Bereich soll ein Label für die Seriennummer der ausgewählten Berichtsdaten eingefügt werden. Darunter werden die enthaltenen Module mit Labeln benannt, und unter diese Modulnamen werden die Graphen und Werte für die visuelle Darstellung der Testdaten eingefügt. Die Anzahl der Darstellungen hängt von den in den Bericht Daten enthaltenden Modulen ab. Die Anzahl und der Aufbau des Unteren Bereiches ist variabel. Der grundlegende Aufbau dieser Seite ist in der Abbildung 19 dargestellt.

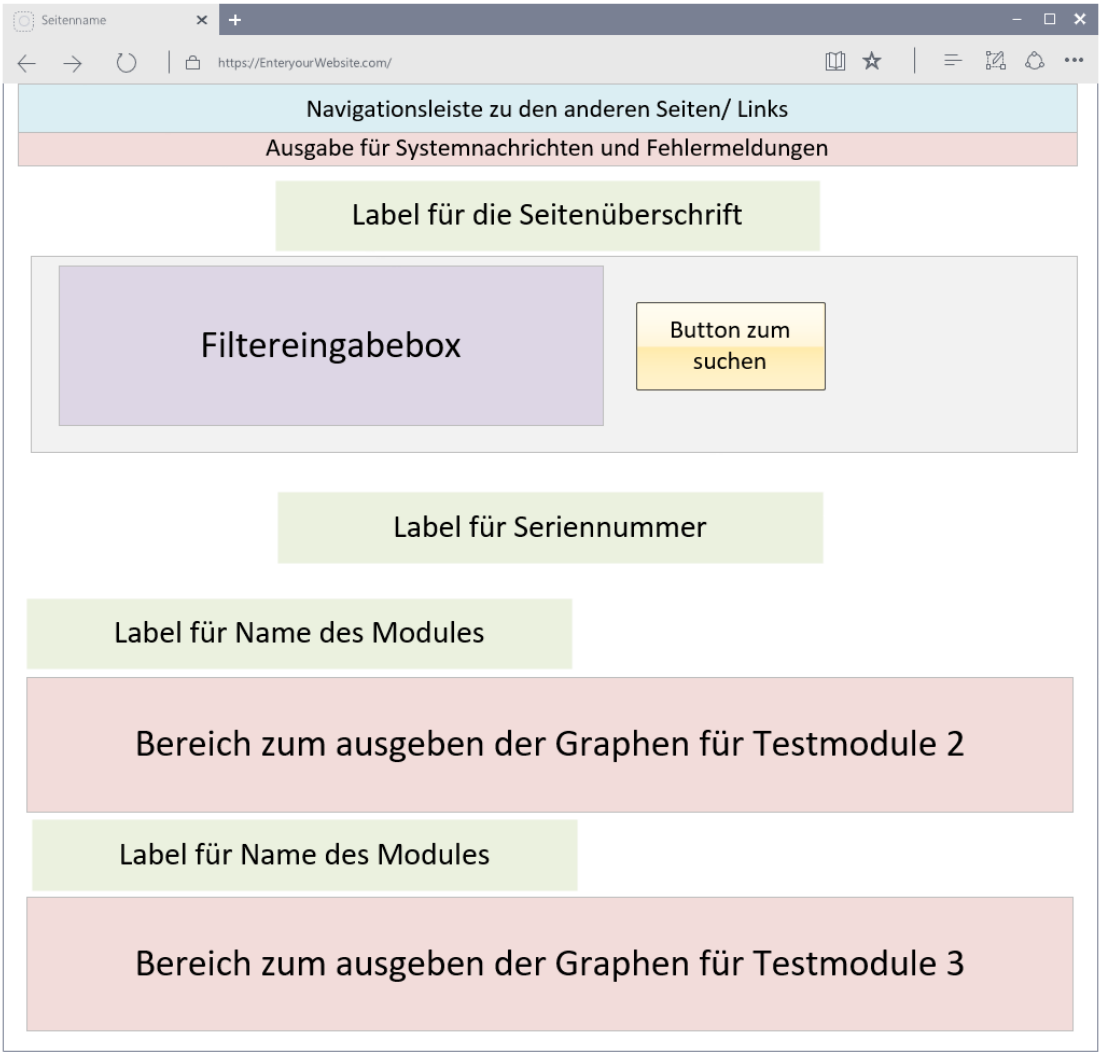

Abbildung 19: Benutzeroberflächenentwurf der Seite zum Ausgeben der Graphen
Quelle: Eigene Darstellung mit Microsoft Visio

3.6. Entwurf der Applikationsarchitektur

Der Entwurf der Applikationsarchitektur bildet die konzeptionelle Grundlage für die technische Umsetzung der entwickelten Web-Applikation. Das Ziel ist es, eine Struktur zu schaffen, die modular, wartbar und erweiterbar ist und die gleichzeitig den funktionalen Anforderungen gerecht wird und sich in die bestehende Systemlandschaft integrieren lässt. Die Architektur der Anwendung ist schichten- und komponentenorientiert aufgebaut und nutzt das Webframework Flask, welches eine klare Trennung zwischen der Präsentations-, Logik- und Datenhaltungsschicht ermöglicht.

3.6.1. Architekturübersicht

Die Applikation soll als serverbasierte Webanwendung im Client-Server-Modell funktionieren. Das bedeutet: Während der Server die Datenverarbeitung, -speicherung und -bereitstellung über-

nimmt, ermöglicht der Client über den Webbrowser die Darstellung und Interaktion. Die Anwendung ist in mehrere Schichten gegliedert, zu denen man die einzelnen Programmteile zuordnen kann. Die Anwendung wird in folgende Schichten gegliedert:

- Präsentationsschicht (Frontend): Bereitstellung der Benutzeroberfläche über HTML-Templates, CSS und JavaScript.
- Applikationslogik (Backend): Implementierung der Geschäftslogik, Steuerung des Datenflusses und Verarbeitung der XML-Dateien.
- Datenhaltungsschicht: persistente Speicherung der extrahierten Mess- und Gerätedaten in einer relationalen Datenbank mittels ORM.

Ein schematisches Architekturdiagramm dieser Struktur ist in Abbildung 20 dargestellt.

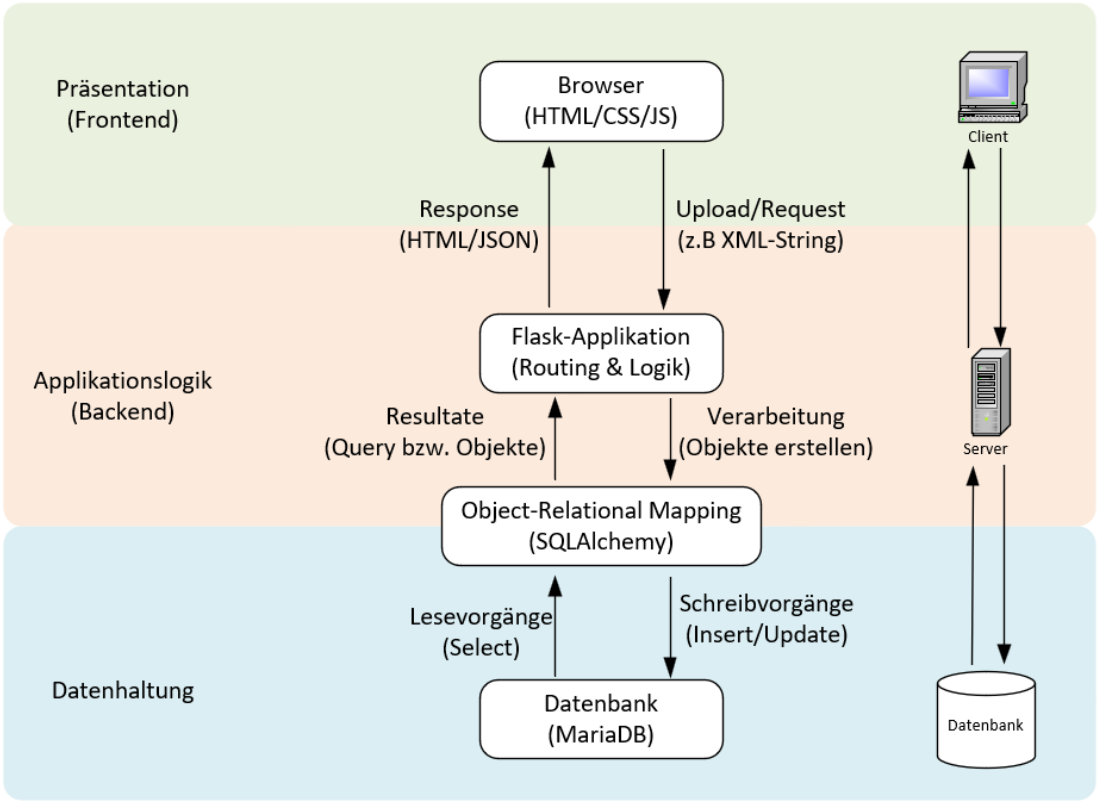

Abbildung 20: Schematisches Architekturdiagramm der Applikation
Quelle: Eigene Darstellung mit Microsoft Visio

Der Entwurf der Ordnerstruktur für die Applikation wird in Abbildung 21 dargestellt.

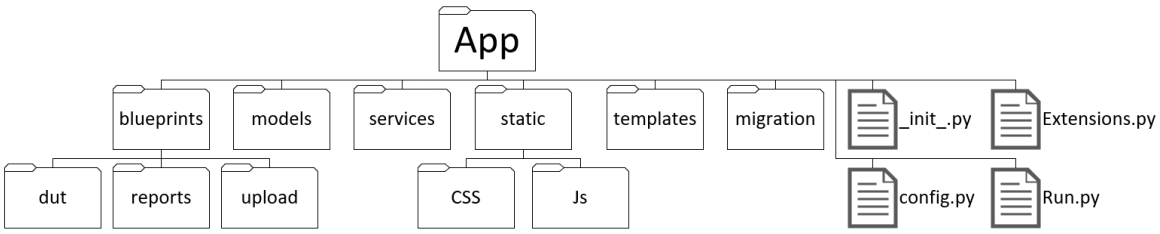

Abbildung 21: Grundlegende Ordnerstruktur der Applikation
Quelle: Eigene Darstellung mit Microsoft Visio

3.6.2. Präsentationsschicht

Die Präsentationsschicht besteht aus einer Kombination aus HTML-Templates und statischen Ressourcen (CSS, JavaScript), die im Verzeichnis templates/ bzw. static/ abgelegt werden.

- Die Templates für die einzelnen Seiten der Applikation für das Hochladen, die Darstellung der sich in der Datenbank befindenden Berichte und grafische Darstellungen bilden den Kern der Weboberfläche.
- Statische Dateien wie CSS-Stylesheets und das JavaScript für die Erstellung der Graphen dienen der Gestaltung und der interaktiven Darstellung von Messergebnissen.

  Die Kommunikation mit der Logikschicht erfolgt über die definierten Flask-Blueprints, die als modulare Controller agieren.

3.6.3. Applikationslogik

Die zentrale Geschäftslogik ist in modularen Blueprints und Servicekomponenten realisiert.

Die drei Blueprints `upload`, `reports` und `dut` kapseln die jeweiligen Funktionsbereiche:

- upload: Einlesen und Validieren von XML-Dateien.
- dut: Verarbeitung und Anzeige der Daten eines „Device Under Test“.
- reports: Zusammenstellung und Ausgabe von Auswertungen.

Unterstützt werden diese durch das Modul in dem Ordner services/, das die eigentliche Logik zur Datenverarbeitung bereitstellt. Dieser Ordner soll Dateien für folgende Aufgaben beinhalten:

- Eine Datei für das Übernehmen des Parsens und Einlesens der XML-Daten.
- Eine Datei für vordefinierte Datenbankabfragen.
- Eine Datei für Hilfsfunktionen, z. B. zur Zeitreihenanalyse und Datenaufbereitung.
- Eine Datei dient der Erstellung und Formatierung von Report-Daten für die grafische Darstellung.

Die Applikationslogik soll über die Datei `run.py` initialisiert werden, welche den Flask-Server startet und die Anwendungskonfiguration aus `config.py` einliest.

3.6.4. Datenhaltungsschicht

Die Datenhaltung soll über ein relationales Datenbanksystem erfolgen, das über das Flask-eigene SQLAlchemy ORM angebunden ist. Dabei sollen die Datenmodelle im Verzeichnis models/ definiert werden und die logischen Entitäten der Prüfanlage abbilden. Die Dateien werden nach den Tabellen in Abbildung 16 unterteilt. Außerdem sollen die Beziehungen zwischen den Modellen ermöglichen, dass eine strukturierte und relationale Abbildung der Prüfdaten dargestellt wird. Wodurch eine effiziente Abfrage und Analyse möglich ist.

Für die Migrationen werden mit der Bibiothek `Flask-Migration` verwaltet, wodurch das Verzeichnis `migrations` zustande kommen soll.

4. Implementierung

In diesem Kapitel wird die Umsetzung des im vorherigen beschriebenen Konzepts bis zum Punkt der Abgabe erläutert. Bei dieser Form der Applikation handelt es sich um einen lauffähigen Prototyp, der bisher nicht verlässlich einsetzbar wäre. Der Fokus der Implementierung lag bei diesem auf der Realisierung der Kernfunktionen zur Verarbeitung, Speicherung und Darstellung der XML-basierten Testberichte. Der Prototyp bildet damit die grundsätzlichen Funktionsabläufe der geplanten Web-Applikation ab, stellt jedoch noch kein vollständig ausgereiftes Produkt dar. Die Mängel, nötigen Änderungen und Erweiterungen bis zu einem nutzbaren Produkt werden in den Kapiteln 5 und 6 behandelt.

Die Implementierung erfolgte schrittweise in iterativen Entwicklungszyklen. Zunächst wurde das Grundgerüst der Flask-Anwendung erstellt, dann die Einlesefunktion erstellt. Anschließend die Datenbankanbindung und zuletzt die Benutzeroberfläche mit den Auswertungs- und Visualisierungsfunktionen integriert. Dabei wurde besonderer Wert auf eine modulare Struktur gelegt, um die Erweiterbarkeit des Systems zu gewährleisten.

Die Beschreibung der Implementierung wird hier an Architekturschichten angelehnt, um eine aufeinander aufbauende Erklärung zu schaffen.

4.1. Programmstruktur

Im folgenden Abschnitt wird die genaue Struktur des Programmcodes dargestellt, um mit dieser die in den folgenden Unterkapiteln zu beschreiben und der Gesamtstruktur besser zuordnen zu können. Die Programmstruktur folgt dem in Unterkapitel 3.6.1 beschriebenen Strukturmuster. Eine genaue Beschreibung der Struktur des Programmcodes. Struktur und der Inhalt der Ordner werden in der folgenden Abbildung 22 dargestellt. Die in der Abbildung vorkommenden Daten entsprechen genau den der Anwendung.

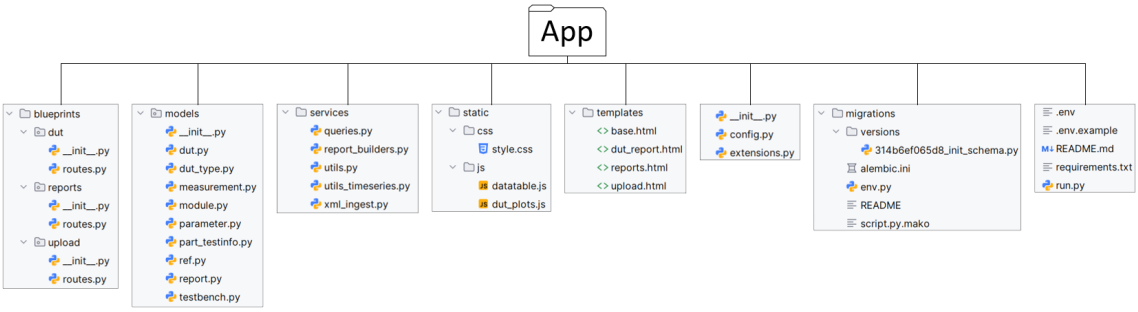

Abbildung 22: Genaue Struktur des Applikations-Ordners
Quelle: Eigene Darstellung mit Microsoft Visio

4.2. Entwicklung der Benutzeroberfläche

Die Benutzeroberfläche wurde in HTML, CSS und JavaScript unter Verwendung der Jinja2-Template-Engine realisiert.

Alle relevanten Dateien für die Darstellung der Seiten befinden sich in den Ordnern `static` und `templates`.

Der Ordner `templates` enthält alle relevanten Jinja2-Templates bzw. HTML-Dateien. Hierbei dient das HTML-Dokument `base.html` als das Grundgerüst für drei verschiedene Seiten. Es beinhaltet die Elemente den Navigator zum Wechsel zwischen den Seiten, die Überschrift und das Messagefeld für Fehler und Systemnachrichten, die auf allen drei Seiten vorkommen. Außerdem enthält es die Verknüpfung zu den CSS-Dateien `bootstrap.min.css` und `style.css` in `static`, die im Unterordner `css` liegen. Sie verändert das Aussehen der Benutzeroberfläche, um ein abgerundetes Design zu erhalten.

Die drei anderen HTML-Dateien `upload.html`, `reports.html` und `dut_analyse.html` werden durch Jinja2 in `base.html` eingebunden, um so folgende drei Seiten zu erstellen:

1. XML hochladen (`upload.html`): Hochladen und Einlesen von XML-Dateien. Die Seite ist in der folgenden Abbildung 23 dargestellt.

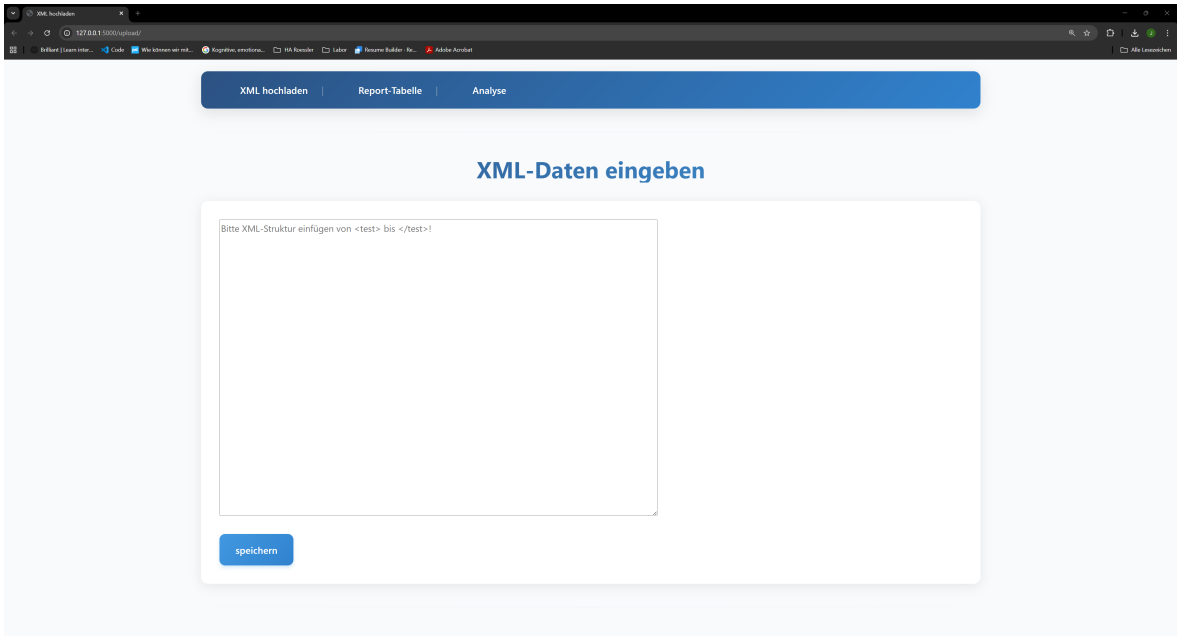

Abbildung 23: Erstellte Webseite für XML hochladen in Browserfenster
Quelle: Eigene Darstellung

2. Report-Tabelle (`reports.html`): Übersicht aller eingelesenen Testberichte in Tabelle. Die Seite ist in der folgenden Abbildung 24 dargestellt.

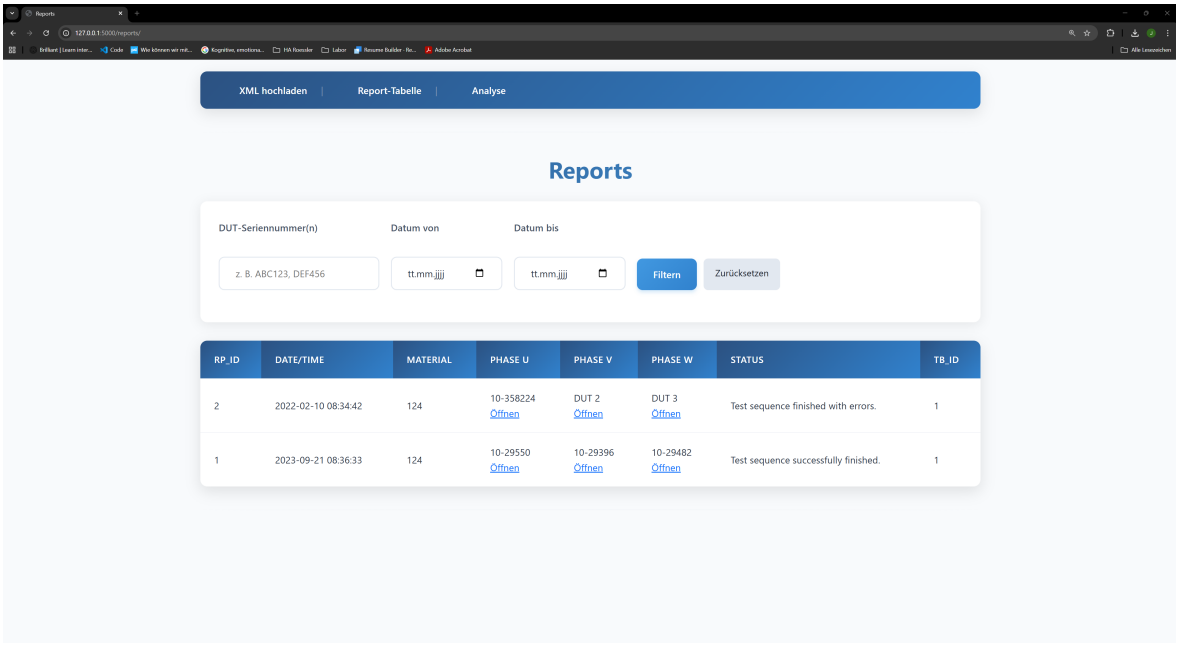

Abbildung 24: Erstellte Webseite für Report-Tabelle in Browserfenster
Quelle: Eigene Darstellung

3. Analyse (`dut_analyse.html`): Darstellung ausgewählter Messdaten eines DUT aus einem Testbericht in grafischer Form. Die Seite ist in der folgenden Abbildung 25 dargestellt.

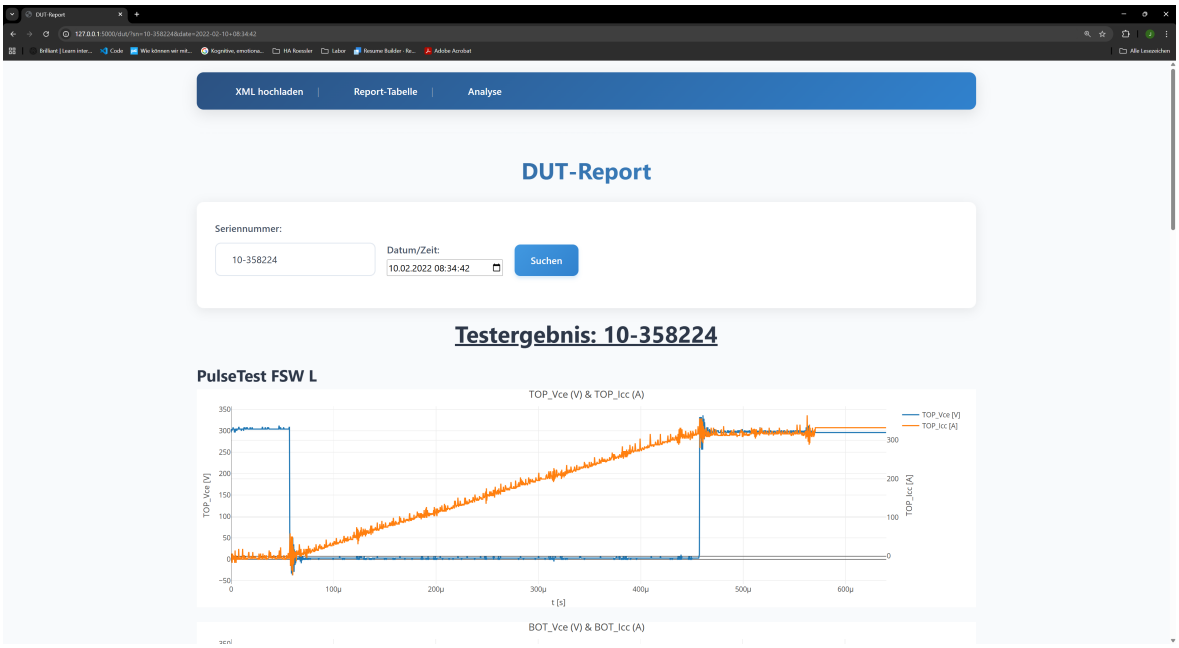

Abbildung 25: Erstellte Webseite für Analyse in Browserfenster
Quelle: Eigene Darstellung

Im Folgenden wird anhand der Beschreibung des Jinja2-Templates (`upload.html`) erläutert, wie die Übermittlung der Daten über einen Buttondruck an die Logikschicht abläuft. Dieses Prinzip

wird in allen Jinja2-Templates benutzt, um eingegebene Daten aus den Eingabefeldern der Webseite an die Logikschicht zu übermitteln.

Der Code aus `upload.html` ist in der Abbildung 26 für ein besseres Verständnis dargestellt.

```
{% extends "base.html" %}
{% block title %}XML hochladen{% endblock %}
{% block content %}
  <h1>XML-Daten eingeben</h1>
  <form method="post"> <!-- Übertragung der Form für späteren XML-string -->
    <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
    <textarea
            name="xml_data" rows="20" cols="100"
            placeholder="Bitte XML-Struktur einfügen von <test> bis </test>!">{{ xml_data }}
    </textarea><br><br>
    <button type="submit">speichern</button> <!-- Button für das Übergeben und ausführen -->
  </form>
{% endblock %}
```

Abbildung 26: Beispiel Programmcode aus upload.html
Quelle: eigene Darstellung

Wie bereits oben schon genannt, erweitert das Template `upload.html` die in `base.html` definierte Grundstruktur und fügt deren allgemeine Elemente wie Navigationsleiste, Kopfbereich und Mitteilungsfeld ein. Innerhalb des „blocks content“ wird der seitenindividuelle Inhalt definiert. Hierzu gehört ein Eingabefeld in Form eines HTML-Formulars, das zur Eingabe der XML-Struktur dient.

Das Formular verwendet die Methode „POST“, wodurch die eingegebenen Daten an die Serverlogik gesendet werden, sobald der Benutzer den Button *„speichern“* betätigt. Über das „csrf_token“-Feld wird ein Sicherheitsmechanismus implementiert, der Cross-Site-Request-Forgery-Angriffe verhindert. Der eigentliche Eingabebereich wird durch ein „<textarea>“-Element realisiert, das genügend Platz bietet, um den XML-Inhalt beginnend mit dem Stammelement „<test>“ bis zum schließenden „</test>“ vollständig einzufügen.

Beim Betätigen des Buttons wird der Inhalt des Textfeldes an die zugehörige Flask-Route, in diesem Fall `routes.py` im Unterordner `upload` in `blueprints` übergeben, wo die Daten serverseitig verarbeitet werden. Die Applikationslogik übernimmt in diesem Fall anschließend das Einlesen, Validieren und Speichern der XML-Struktur in der Datenbank.

Dieses Prinzip findet sich in gleicher Form auch in den anderen Templates für die Filterfunktionen der Applikation wieder, wodurch eine einheitliche und konsistente Interaktion zwischen Benutzeroberfläche und Logikschicht gewährleistet wird.

4.3. Einlesen und Verarbeiten von XML-Daten

Wie bereits im vorherigen Unterkapitel erläutert, beginnt das Einlesen und Verarbeiten mit der Übergabe des Inhaltes aus der Upload-Seite der Web-Applikation über die Post-Methode.

Die daraus erhaltenen Daten werden in die Funktion `upload()` in der Datei `route.py` für die Upload-Seite übergeben. weitergegeben und von der Funktion `ingest_xml()` aus `xml_ingest.py` verarbeitet wird. Bevor die Daten an `ingest_xml()` übergeben werden, werden diese durch die Funktion `parse_xml_string()` aus `utils` geparst. und somit die Struktur der XML-Daten grundlegend überprüft. Für das Parsen wird die Bibliothek `lxml` genutzt.

Die relevanten Daten werden in `ingest_xml()` durch Hilfsfunktionen extrahiert und in Variablen bzw. Listen vorsortiert und gespeichert.

Die Hilfsfunktionen nutzen hierbei die XPath-Adressierung über die Funktion `xpath()` aus der Bibliothek `lxml` bzw. aus dem Modul `lxml.etree`, um die relevanten Daten als Listen zurückzugeben. In Abbildung 27 ist zur Veranschaulichung ein Codeabschnitt dargestellt, in den über die Hilfsfunktionen die Variablen für das Vorsortieren und Speichern befüllt werden.

```
# ---- Start des Importvorgangs in einer Transaktion ----
with db.session.begin():
    modules = root.xpath("//testmodule")
    modulecount = len(modules)

    infoliste = extract_block("//info/*[@name]")
    testbenchliste = extract_block("//testbench/*[@name]")
    modulenamenliste = get_module_names()

    modulheaderliste = [[] for _ in range(modulecount)]
    extract_basic_data( x_path: "(//testmodule)[$k]/*[@name]", modulheaderliste)

    modulparameterliste = [[] for _ in range(modulecount)]
    extract_basic_data( x_path: "(//testmodule)[$k]/Phase_U/parameters/*[@name]", modulparameterliste)

    # Measurements aller Phasen sammeln
    modulmeasurementsliste_DUT_Phase_U = [extract_phase_rows(m,  phase_tag: "Phase_U") for m in modules]
    modulmeasurementsliste_DUT_Phase_V = [extract_phase_rows(m,  phase_tag: "Phase_V") for m in modules]
    modulmeasurementsliste_DUT_Phase_W = [extract_phase_rows(m,  phase_tag: "Phase_W") for m in modules]

    testsequencestatus = root.xpath("//string[@name='test sequence status']/text()")[0]
```

Abbildung 27: Codeausschnitt für die Listenspeicherung mit Hilfsfunktionen
Quelle: Eigene Darstellung

Nach dem Vorsortieren und Speichern werden ORM-Objekte für das Überführen der Daten in die Datenbank erstellt. Mit Hilfe der Suchfunktion `get_value_by_name()` werden die Daten anhand der Namen aus den Namen-Attributen gesucht, die zusammen mit dem Inhalt aus dem XML-Element in Tupeln gespeichert sind. den Namen-Attribute, welche zusammen mit dem Inhalt aus dem XML-Element in Tupeln gespeichert sind, aus den Listen gesucht. Der Inhalt wird zurückgegeben, außer wenn der Name nicht gefunden wird. Dann wird ein Standardwert zurückgegeben. Dieser Standardwert löst beim Eintragen in die Datenbank bewusst Fehler aus, um Fehler in der Übertragung und der Verarbeitung der XML-Daten zu erkennen.

Für den Code zu `get_value_by_name()` siehe Abbildung 28.

```python
def get_value_by_name(listname, name, default=None, strip=True, cast=None):  Jonnifen
    keys = {str(name).casefold()}
    for tup in listname:
        if not tup:
            continue
        key = str(tup[0]).casefold()
        if key in keys:
            if len(tup) < 2:
                return default
            val = tup[1]
            if strip and isinstance(val, str):
                val = val.strip()
            if cast is not None:
                try:
                    return cast(val)
                except Exception:
                    return default
            return val
    return default
```

Abbildung 28: Codeausschnitt für die Suchfunktion in den Listen
Quelle: Eigene Darstellung

Die ORM-Objekte werden durch die Funktionen `.session.add()` und `.session.flush()` schon in die Datenbank geschrieben, um die automatisch generierten Primärschlüssel der Objekte erhalten zu können. Die richtige Transaktion der ORM-Objekte wird erst am Ende der Funktion `ingest_xml()` ausgeführt, damit nur vollständig einpflegbare XML-Daten in die Datenbank überführt werden können. Bei einem Fehler werden alle Transaktionen rückgängig gemacht und die ausstehenden Änderungen gelöscht und es wird ein Fehler zurückgegeben.

Neben dieser Sicherung werden alle vermeintlichen Eintragungen bzw. ORM-Objekte noch mal in der Datenbank über Funktionen `execute()` und `select()` gesucht. Wenn das Element bereits in der Datenbank existiert, wird es nicht eingefügt, um Dopplungen zu vermeiden. Ein Codebeispiel für die Sicherung ist in Abbildung 29 dargestellt. Die Prüfungen sind in diesem Prototyp nicht identisch und können geringfügige Variationen beinhalten.

```
# --- Testbench ---
testbench = Testbench(
    Testbench_name=get_value_by_name(testbenchliste, name: "Testbench_name"),
    Testbench_HW_revision=get_value_by_name(testbenchliste, name: "Testbench_HW_revision"),
    Version_GUI=get_value_by_name(testbenchliste, name: "Version_GUI"),
    Version_controller=get_value_by_name(testbenchliste, name: "Version_controller"),
)
# Duplikate vermeiden: einfacher Fingerprint
tb_row = db.session.execute(
    db.select(Testbench.tb_id).where(
        Testbench.Testbench_name == testbench.Testbench_name,
        Testbench.Testbench_HW_revision == testbench.Testbench_HW_revision,
        Testbench.Version_GUI == testbench.Version_GUI,
        Testbench.Version_controller == testbench.Version_controller,
    ).limit(1)
).first()
if tb_row:
    tb_id = tb_row[0]
else:
    db.session.add(testbench)
    db.session.flush()
    tb_id = testbench.tb_id
```

Abbildung 29: Beispiel ORM-Objekt
Quelle: Eigene Darstellung

Bei einem erfolgreichen Einfügen wird die Report-ID aus dem Eintrag aus der Datenbank an `upload()` zurückgegeben. Diese erstellt dann eine Erfolgsnachricht, welche dann auf der Benutzeroberfläche der Seite ausgegeben wird. In Abbildung 30 ist der Code aus der `routes.py` von der Upload-Seite dargestellt.

```python
from flask import render_template, request, redirect, url_for, flash
from . import bp
from ...services.utils import parse_xml_string
from ...services.xml_ingest import ingest_xml

@bp.route("/", methods=["GET", "POST"])  Jonnifen
def upload():
    if request.method == "POST":
        xml_string = (request.form.get("xml_data") or "").strip()
        if not xml_string:
            flash( message: "Bitte XML einfügen.",  category: "warning")
            return redirect(url_for("upload.upload"))
        try:
            root = parse_xml_string(xml_string)
            rp_id = ingest_xml(root)
            flash( message: f"✅ XML gespeichert (Report {rp_id}).",  category: "success")
        except Exception as e:
            flash( message: f"❌ Fehler beim Import: {e}",  category: "danger")
        return redirect(url_for("upload.upload"))
    return render_template("upload.html")

```

Abbildung 30: Code routes.py aus Upload-Seite
Quelle: Eigene Darstellung

4.4. Implementierung der Datenbank

In diesem Unterkapitel werden die Einbindung der Datenbank in die Applikation sowie die Implementierung grob erläutert. Der Hauptfokus liegt hierbei auf der Einbindung und der Methode zum Erstellen über Flask-SQLAlchemy. Die Datenbank wurde mithilfe der Bibliothek Flask-SQLAlchemy erstellt. Hierbei wurde die Bibliothek Flask-Migration für die Verwaltung von Datenbankschemata genutzt.

SQLAlchemy stellt ORM bereit, mit welchem die relationalen Datenbanktabellen als objektorientierte Python-Klassen dargestellt werden können. Dadurch konnte die in 3.4 dargestellte Datenbankstruktur, ohne die aktive Nutzung von SQL-Befehlen, definiert und umgesetzt werden. Hierbei entspricht jede Klasse einer der Tabellen. Die Attribute der Klassen bilden die Spalten der Tabellen ab. Die Fremdschlüssel zwischen den Tabellen werden über Relationship-Objekte beschrieben. Die Klassen werden im Ordner `models` definiert. Zur Veranschaulichung wird der Aufbau an dem Beispiel der Tabelle bzw. der Klasse `Report_Table` aus der Datei `report.py` kurz erläutert, siehe Abbildung 31.

```python
from ..extensions import db

class Report(db.Model):  Jonnifen
    __tablename__ = "Report_Table"
    Rp_id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    Material_number = db.Column(db.Integer, db.ForeignKey("DUT_Typs_Table.Material_number"), nullable=False)
    Date_Time_Report = db.Column(db.DateTime, nullable=False)
    Configuration = db.Column(db.String(255), nullable=False)
    ID_Carrier_left = db.Column(db.String(255), nullable=False)
    DUT_id_phase_u = db.Column(db.Integer, db.ForeignKey("DUT_Table.DUT_id"), nullable=False)
    DUT_id_phase_v = db.Column(db.Integer, db.ForeignKey("DUT_Table.DUT_id"), nullable=False)
    DUT_id_phase_w = db.Column(db.Integer, db.ForeignKey("DUT_Table.DUT_id"), nullable=False)
    test_sequence_status = db.Column(db.String(100), nullable=False)
    tb_id = db.Column(db.Integer, db.ForeignKey("testbench_Table.tb_id"), nullable=False)

    __table_args__ = (
            db.UniqueConstraint(
                'Material_number','Date_Time_Report','Configuration','tb_id',
                'DUT_id_phase_u','DUT_id_phase_v','DUT_id_phase_w',
                name='uq_report_fingerprint'
            ),
    )
```

Abbildung 31: Beispiel der Tabelle-Klasse Report-Table
Quelle: Eigene Darstellung

Alle Klassen aus `models` sind identisch aufgebaut und folgen demselben Schema. Alle Klassen besitzen den Import der globalen Instanz `db` aus dem Modul `extensions`. Diese Instanz bindet SQLAlchemy in die Flask-Applikation ein und wird beim Starten der Hauptapplikation initialisiert. Über diese Instanz `db` können alle Modelle registriert und Datenbankoperationen wie Abfragen, Einfügungen oder Aktualisierungen ausgeführt werden. Die Funktion zum Erstellen der Hauptapplikation ist in Abbildung 32 abgebildet. Sie liegt in der Datei `__init__.py` und wird über `run.py` ausgeführt.

```python
from flask import Flask, redirect, url_for
from .config import Config
from .extensions import db, migrate, csrf
from flask_wtf.csrf import generate_csrf

def create_app(config_class: type[Config] = Config) -> Flask:  2 usages  Jonnifen
    app = Flask(__name__)
    app.config.from_object(config_class)

    # Extensions
    db.init_app(app)
    migrate.init_app(app, db)
    csrf.init_app(app)
    app.jinja_env.globals["csrf_token"] = generate_csrf

    # Blueprints
    from .blueprints.upload import bp as upload_bp
    from .blueprints.dut import bp as dut_bp
    from .blueprints.reports import bp as reports_bp

    app.register_blueprint(upload_bp, url_prefix="/upload")
    app.register_blueprint(dut_bp, url_prefix="/dut")
    app.register_blueprint(reports_bp, url_prefix="/reports")

    @app.route("/")  Jonnifen
    def index():
        return redirect(url_for("upload.upload"))

    return app
```

Abbildung 32: Funktion für das Erstellen der Hauptapplikation
Quelle: Eigene Darstellung

Die Variable `__tablename__` in Abbildung 31 legt fest, wie die Tabelle in der Datenbank heißen soll. In Zeile 5 bis 14 werden die Attribute bzw. die Tabellenspalte definiert. Am Ende der Klasse wird ein Table-Constraint über `__table_args__` definiert, welches als datenbankseitiger Duplizierungsschutz dient. Dieser sorgt dafür, dass derselbe XML-Bericht versehentlich mehrfach importiert wird. Es können nicht zwei Datensätze mit denselben Werten existieren.

Dies dient als zusätzliche Sicherheitsinstanz neben dem selbst erstellten Duplizierungsschutz in `ingest_xml()`. Die Verbindungen der Datenbank werden über die Klasse Config in der Datei `config.py` definiert. Hierbei wird im Fall, dass die Datenbank nicht vorhanden ist und getestet werden soll, eine SQLite-Datenbank erstellt und genutzt. Die genaue Bezeichnung der `DATENBANK_URL` und dem `FLASK_SECRET_KEY` ist in `.env` definiert. In der folgenden Abbildung 33 ist die Klasse `Config` zum Verbinden mit der Datenbank abgebildet.

```python
import os
from dotenv import load_dotenv
load_dotenv()

class Config:  Jonnifen
    SQLALCHEMY_DATABASE_URI = os.getenv("DATABASE_URL", "sqlite:///dwt_dev.db")
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    SECRET_KEY = os.getenv("FLASK_SECRET_KEY", "dev-secret")
```

Abbildung 33: Config-Klasse für Datenbankverbindung
Quelle: Eigene Darstellung

Zur Verwaltung und Versionierung des Datenbankschemas wird Flask-Migrate eingesetzt, welches auf dem Migrationstool Alembic basiert. Diese Erweiterung erkennt automatisch Änderungen an den ORM-Modellen und erzeugt daraus sogenannte Migrationsskripte. Die Skripte sind im Ordner `versions` unter `migrations` zu finden. Diese Skripte enthalten die notwendigen SQL-Befehle, um das Schema der Datenbank an Modellversionen anzupassen. Durch die Befehle werden die Migrationen generiert und auf die bestehende Datenbank angewendet. Dadurch ist es möglich, die Struktur der Datenbank schrittweise zu erweitern oder zu verändern, ohne bestehende Daten zu verlieren. Diese Methode wurde hauptsächlich für die einfachere Entwicklung implementiert und wird nicht zwangsläufig in zukünftigen Versionen weiterverwendet.

4.5. Technischer Ablauf der Visualisierung

Im folgenden Unterkapitel wird der Ablauf des Erstellens der Tabelle für die in der Datenbank enthaltenen Berichte, so wie die Erstellung der Graphen grob beschrieben und ein Beispiel der Ergebnisse gegeben.

4.5.1. Erstellen der Berichtstabelle

Das Erstellen der Berichtstabelle für die Seite „Report-Tabelle“ läuft wie folgt ab:

1. Beim Laden der Seite oder beim Bestätigen bzw. Zurücksetzen der Filterbedingungen wird die Funktion `report()` der Seite ausgeführt. Diese übermittelte die Filterinhalte über die GET-Methode an die Funktion `query`$_{,}$$eports()ausderDatei$`queries.py`$.DieseFunktionfhrtberdb einenSE Befehlberdie Reportdatenaus, welchedie Filterbedingungenerfllen.$

1. `Die ausgewählten Datensätze werden nach Rp_id geordnet und an die Funktion report() aus der Route-Datei der Seite zurückgegeben. Die Datei route.py ist für die Veranschaulichung in Abbildung 34 dargestellt.`

2. `Diese Funktion lädt das Template der Seite neu und übergibt die ausgewählten Datensätze.`

3. Durch das erneute Laden wird mittels der HTML-Datei reports.html und der JavaScript-datatable.js eine Tabelle mit den Ausgewählten Datensätzen erstellt.

```python
from flask import render_template, request
from . import bp
from ...services.queries import query_reports

@bp.route("/", methods=["GET"])  Jonnifen
def reports():
    serial_q = (request.args.get("serial") or "").strip()
    date_from = (request.args.get("from") or "").strip()
    date_to   = (request.args.get("to") or "").strip()

    rows = query_reports(serial_q, date_from, date_to)

    return render_template(
        template_name_or_list: "reports.html",
        rows=rows,
        serial=serial_q,
        date_from=date_from,
        date_to=date_to,
    )
```

Abbildung 34: Code routes.py aus Report-Tabellen-Seite
Quelle: Eigene Darstellung

Dieser Ablauf wird jedes Mal, wenn die Seite geladen oder z. B. durch das Ändern der Filterbedingungen refreshed wird, durchgeführt. Nach dem Laden der Seite für die Report-Tabelle werden erst alle in der Datenbank enthaltenen Berichte ausgegeben, bis Filterargumente festgelegt und bestätigt werden. Ein Ausschnitt dieser Tabelle ist in Abbildung 35 dargestellt.

```python
from flask import render_template, request, redirect, url_for, flash
from . import bp
from ...services.utils import parse_xml_string
from ...services.xml_ingest import ingest_xml

@bp.route("/", methods=["GET", "POST"])  Jonnifen
def upload():
    if request.method == "POST":
        xml_string = (request.form.get("xml_data") or "").strip()
        if not xml_string:
            flash( message: "Bitte XML einfügen.",  category: "warning")
            return redirect(url_for("upload.upload"))
        try:
            root = parse_xml_string(xml_string)
            rp_id = ingest_xml(root)
            flash( message: f"✅ XML gespeichert (Report {rp_id}).",  category: "success")
        except Exception as e:
            flash( message: f"❌ Fehler beim Import: {e}",  category: "danger")
        return redirect(url_for("upload.upload"))
    return render_template("upload.html")

```

Abbildung 35: Ausschnitt der Report-Tabelle
Quelle: Eigene Darstellung

4.5.2. Erstellen der Messwert-Graphen

Das Erstellen der Graphen ähnelt vom grundlegenden Ablauf her der Seite „Report-Tabelle“. Der Ablauf für das Erstellen der Graphen der Analyse-Seite läuft wie folgt ab:

1. Die Variablen zum Finden des gesuchten DUTs können über das Eingeben der in die Suchfelder auf der Analyse-Seite oder über Links in der Tabelle auf der Seite der Report-Tabelle eingetragen werden. Das Laden dieser Variablen geschieht über die GET-Methode. und wird über die Funktion `dut_root()` aus der Route-Datei für die Analyse-Seite durchgeführt.

2. Die Funktion `dut_root()` führt über `db` SELECT-Befehle aus, um die notwendigen Datensätze für die darzustellenden Testmodule (PulsTest und XPowerTest) zu erhalten. Hierfür werden Funktionen, unter anderem die Funktion `get_series_vals_and_unit()` aus `queries.py`, verwendet, um Datensätze zu suchen. Für eine Darstellung der Funktion siehe Abbildung 36.

3. Die ausgewählten Datensätze werden noch geordnet und am Ende lädt die Funktion die Seite neu.

4. Beim Neuladen werden die Daten an die HTML-Datei `dut_analyse.html` weitergegeben. In dieser werden die geordneten Datensätze mit Hilfe der JavaScript-Dateien `dut_plot.js`

und `plotly-latest.min.js` verarbeitet und die Graphen für die Analyse-Seite erstellt und ausgegeben.

```
42    def get_series_vals_and_unit(rp_id: int, dut_id: int, mod_id: int, name: str):
43        row = db.session.execute(
44            db.select(Measurement.meas_value, Measurement.unit)
45              .where(Measurement.Rp_id==rp_id,
46                     Measurement.DUT_id==dut_id,
47                     Measurement.mod_id==mod_id,
48                     Measurement.meas_name==name)
49              .limit(1)
50        ).first()
51        if not row:
52            return [], None
53        raw, unit = row
54        vals = []
55        if raw:
56            try:
57                vals = floatstring_split(raw)
58            except Exception:
59                vals = []
60        return vals, (unit or None)
```

Abbildung 36: Beispielcode einer Such-Funktion
Quelle: Eigene Darstellung

Dieser Ablauf wird jedes Mal, wenn die Seite geladen oder z. B. durch das Ändern der Suchbedingungen refreshed wird, durchgeführt. In der Folge sind zwei auf diese Weise erstellte Graphen dargestellt. In Abbildung 37 ist ein Graph für die Darstellung der PulsTest-Daten dargestellt. Die Abbildung 38 ist ein Graph für die Darstellung der XPowerTest-Daten.

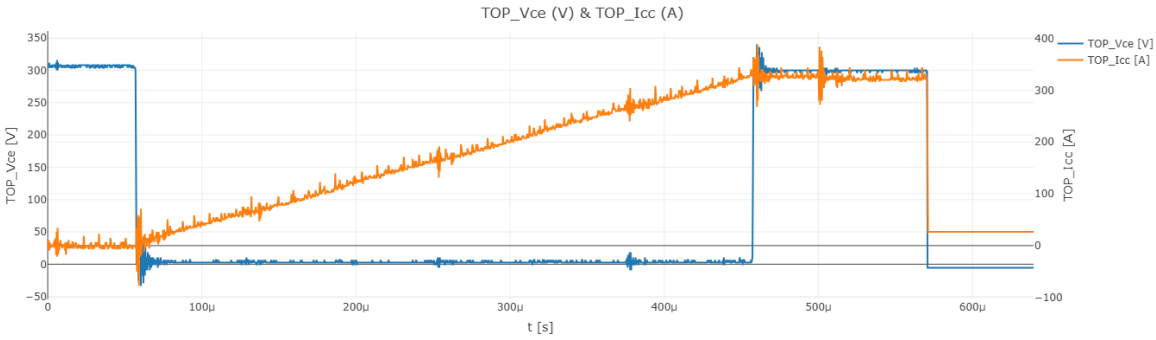

Abbildung 37: Beispiel-Graph für PulsTest-Daten
Quelle: Eigene Darstellung

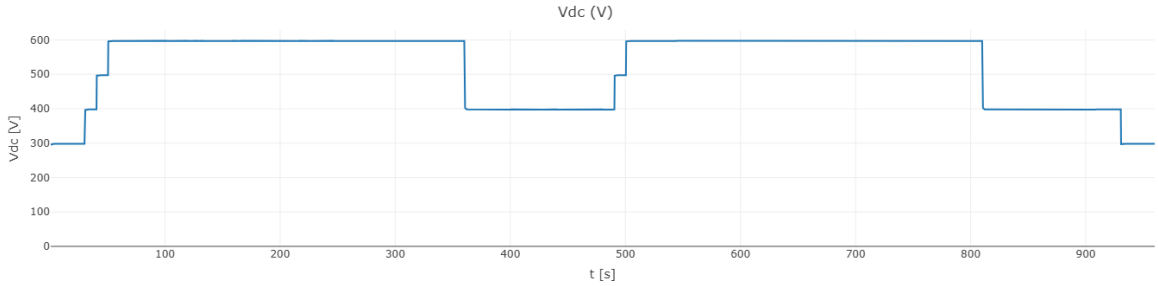

Abbildung 38: Beispiel-Graph für XPowerTest-Daten
Quelle: Eigene Darstellung

Die auf diese Weise erstellten Graphen wurden mit den späteren Nutzern überarbeitet, um die fachgerechte Darstellung und Benennung der Graphen zu gewährleisten.

5. Test der Applikation

In diesem Kapitel werden die Testmethode, die Ergebnisse und die gefundenen Mängel behandelt. Der Prototyp der Web-Applikation wurde hierbei manuell getestet. Die Zeitanforderungen sowie die zahlreichen Strukturvariationen der XML-Berichte haben das Erstellen von automatisierten Testmethoden erschwert. Daher wurde, um einen funktionellen Prototyp fertigzustellen, die Software vorerst manuell getestet. Hierbei wurden vorerst Tests durchgeführt, um primäre Schwachstellen und Mängel festzustellen.

5.1. Testmethode und Durchführung

Der Prototyp wurde, um seine grundlegenden Funktionen zu testen, mit mehreren möglichst unterschiedlichen XML-Berichten getestet. Hierfür stand ein Adminreport des Teststandes, der mehrere Testberichte enthält, zur Verfügung. Es wurden 20 Berichte für Tests ausgewählt. Es wurde darauf geachtet, zwei Berichte mit der gleichen Materialnummer und somit DUTs des gleichen Typs zu verwenden. Hierbei wurde ein Test mit einem erfolgreichen Testergebnis und ein Test, bei dem ein Fehler aufgetreten ist, zum Testen ausgewählt. Somit werden 10 DUT-Typen getestet.

Hierbei wurde nachfolgendem Testablauf gefolgt:

1. Einlesen des XML-Berichtes, bei Fehler genauere Analyse der Ursache, sonst fortfahren.
2. Überprüfen der Werte in der Datenbank. Hierbei wurde eine grafische Verwaltungsoberfläche für Datenbankmanagementsysteme (HeidiSQL) zu Hilfe genommen, um sich die Dateneinträge genauer erfassen zu können. Die Datenbankeinträge des Berichtes werden auf Existenz und Richtigkeit überprüft.
3. Überprüfen der Ausgabe des Berichtes in der Tabelle auf der Webseite „Report-Tabelle“. Der Tabellenbeitrag „Richtigkeit“ überprüft.
4. Überprüfen der ausgegebenen Graphen und Werte auf der Webseite für das visuelle Ausgeben der Daten. Die Graphen und Werte werden auf Existenz und Richtigkeit überprüft.
5. Eintragen der Ergebnisse in eine Tabelle für einen allgemeinen Überblick.

Die verwendeten XML-Berichte zum Testen wurden im Ordner „Testreports“ im Programmordner gespeichert. Vor dem Darstellen der Ergebnisse sollte noch angemerkt werden, dass es sich bei allen XML-Berichten um Berichte handelt, die dargestellt werden sollten. Ausnahme hierbei bildet der DUT-Type mit der ID 115, welcher einer Sondertestform entspricht und nur aus Driver-Consumption-Tests besteht, die hintereinander durchgeführt werden. Zudem wurde bei DUT-Type mit der ID 110 ein zweiter Test mit positivem Ergebnis verwendet, da keine negativen Tests in dem Adminreport vorhanden waren. Die Ergebnisse werden so wie die Tabelle im nachfolgenden Unterkapitel dargestellt und erläutert.

5.2. Ergebnisse der Testmethode

In der folgenden Tabelle 4 werden die Ergebnisse grob dargestellt. Die Erläuterung der aufgetretenen Mängel wird im Folgenden genauer beschrieben und erste Lösungsansätze werden genannt.

DUT-Type	Ergebnisse des Berichtes	Einlesen	Datenbankeinträge vorhanden	Tabelleneintrag vorhanden	Graphen vorhanden	Aufgetretene Fehler
124	Erfolgreich	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Vorhanden und korrekt	Keine
124	Fehler	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Erwartete Werte vorhanden	Keine
119	Erfolgreich	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Flowrate-Graph fehlt	Anderer Name für Flowrate
119	Fehler	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Keine vorhanden (erwartet)	Keine
118	Erfolgreich	Fehler	Keine vorhanden	Keine vorhanden	Keine vorhanden	Fehler durch fehlende Seriennummern
118	Fehler	Fehler	Keine vorhanden	Keine vorhanden	Keine vorhanden	Fehler durch fehlende Seriennummern
114	Erfolgreich	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Flowrate-Graph fehlt	Anderer Name für Flowrate
114	Fehler	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Flowrate-Graph fehlt	Anderer Name für Flowrate
112	Erfolgreich	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Flowrate-Graph fehlt	Anderer Name für Flowrate
112	Fehler	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Flowrate Graph-fehlt	Anderer Name für Flowrate
113	Erfolgreich	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Flowrate-Graph fehlt	Anderer Name für Flowrate
113	Fehler	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Flowrate Graph-fehlt	Anderer Name für Flowrate
116	Erfolgreich	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Flowrate-Graph fehlt	Anderer Name für Flowrate
116	Fehler	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Vorhanden und korrekt	Keine
120	Erfolgreich	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Flowrate-Graph fehlt	Anderer Name für Flowrate
120	Fehler	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Flowrate-Graph fehlt	Anderer Name für Flowrate
110	Erfolgreich	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Flowrate-Graph fehlt	Anderer Name für Flowrate
110	Erfolgreich	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Flowrate-Graph fehlt	Anderer Name für Flowrate
115	Erfolgreich	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Keine vorhanden (erwartet)	Keine
115	Fehler	Erfolgreich	Vorhanden und korrekt	Vorhanden und korrekt	Keine vorhanden (erwartet)	Keine

Tabelle 4: Tabelle der Testergebnisse

Quelle: eigene Darstellung

Bei der Auswertung der Tests sind zwei maßgebliche Mängel erkennbar geworden. Im Folgenden werden diese kurz erklärt, ihr Ursprung erläutert und ein Lösungsansatz bestimmt.

1. **Fehler beim Einlesen von DUTs mit nur einer oder zwei Seriennummern**
   Beim Einlesen der XML-Berichte mit der ID-Nummer 118 ist aufgefallen, dass der Prototyp keine Berichte, die weniger als drei Seriennummern enthalten, einlesen kann. Diese Bauform kommt zwar selten vor und wurde daher bei der Erstellung nicht berücksichtigt, was zu diesem Fehler beim Testen geführt hat. Dieser Mangel ist nicht gravierend und kann durch eine Anpassung der Einlesefunktion behoben werden. Hierbei muss zuvor Rücksprache mit den späteren Nutzern gehalten werden, da diese Änderung eine Anpassung der Dopplizierungssicherung benötigt. Alternativ können auch mehrere Einträge für die Seriennummer mit einer Sonderänderung erstellt werden, was diese notwendige Änderung umgeht. Der Unterschied der XML-Strukturen wird in Abbildung 39 für eine bessere Veranschaulichung dargestellt.

Info Bericht mit ID 124:

```
<info>
    <string name="Time">2023-09-21_08:36:33</string>
    <string name="Material_number">124</string>
    <string name="Configuration">L_3DUT_3P1Q</string>
    <string name="ID_Carrier_left">Carrier 1</string>
    <string name="ID_DUT_R">10-29550</string>
    <string name="ID_DUT_S">10-29396</string>
    <string name="ID_DUT_T">10-29482</string>
</info>
```

3 Seriennummern

Info Bericht mit ID 118:

```
<info>
    <string name="Time">2022-01-06_13:38:42</string>
    <string name="Material_number">118</string>
    <string name="Configuration">L_1DUT_3P1Q</string>
    <string name="ID_Carrier_left">CARRIER_LEFT</string>
    <string name="ID_DUT_R">DWT-6300066-020</string>
</info>
```

1 Seriennummern

Abbildung 39: Beispiel Fehler Seriennummeranzahl
Quelle: Eigene Darstellung

2. **Fehler beim Ausgeben der Flowrate-Graphen**
   Bei allen gelb markierten Zeilen in der Tabelle 4 fehlte der Graph für die Flowrate der Kühlung. Dieser Fehler kann auf einen Unterschied in der Benennung des Namens der Messwertreihe zurückgeführt werden. Die DUTs werden je nach Typ mit Luft oder Wasser gekühlt, wodurch sich der Name der Messwertreihe ändert. Sie wird zwar richtig in die Datenbank eingelesen, nur die Funktionen für das Suchen und Darstellen kennen nur den Namen für die luftgekühlten DUTs. Daher muss in diesen Funktionen angepasst werden, um auch die Messwertreihen mit Wasserkühlung zu erfassen und weiterzugeben. Diese Änderung benötigt keine größeren Anpassungen. Der Unterschied der XML-Strukturen wird in Abbildung 40 für eine bessere Veranschaulichung dargestellt.

Flowrate Bericht mit ID 116:

```
<floatblock name="Flowrate(air)" size="698" duration="959.952" unit="m/s">80.00,80.00,80.00
```

Name bei Luftkühlung — Einheit bei Luftkühlung

Flowrate Bericht mit ID 116:

```
<floatblock name="Flowrate" size="1054" duration="1460.07" unit="l/min">100.0,100.0,100.
```

Name bei Wasserkühlung — Einheit bei Wasserkühlung

Abbildung 40: Beispiel Fehler Wassserkühlung vs Luftkühlung
Quelle: Eigene Darstellung

6. Fazit und Ausblick

In diesem abschließenden Kapitel der Arbeit werden die Ergebnisse prägnant zusammengefasst. Zudem erfolgt eine kritische Bewertung der Ergebnisse sowie ein Ausblick auf zukünftige Arbeiten und Weiterentwicklungen der Applikation.

6.1. Zusammenfassung der Ergebnisse

Ziel dieser Arbeit war die Entwicklung einer modular aufgebauten, serverbasierten Web-Applikation zur Verarbeitung, Speicherung und Visualisierung von XML-Daten, die aus den Prüfverfahren eines Umrichter-Teststands stammen. Die Anwendung sollte die automatisch erzeugten XML-Berichte einlesen, die enthaltenen Messdaten in einer Datenbank speichern und anschließend grafisch darstellen.

Die Arbeit umfasste die Grundlagen zur Erstellung, Beschreibung und Bewertung der Applikation, die fundamentale Konzeption, die Implementierung sowie die Testung eines funktionalen Prototyps.

Bei der Konzeptentwicklung und -umsetzung wurde insbesondere auf die vorher festgelegten Anforderungen sowie die während des Erstellungsprozesses geforderten Änderungsvorschläge durch das Unternehmen geachtet. Zudem wurde besonderer Wert auf eine modulare, erweiterbare und wartbare Architektur gelegt, um schnelle Änderungen durchzuführen. Durch den Einsatz von Flask als Webframework und Flask-Blueprint konnte eine klare Trennung zwischen Präsentations-, Logik- und Datenhaltungsschicht umgesetzt werden. Das ORM-Framework Flask-SQLAlchemy wurde hierbei für die Datenbankerstellung und -anbindung genutzt, wodurch die Verarbeitung und Verwaltung der XML-Strukturen effizient und strukturiert realisiert werden konnte.

Die entwickelte Applikation bzw. der Applikations-Prototyp ist in der Lage, einen Großteil der vom Teststand erzeugten XML-Berichte automatisiert zu verarbeiten, redundante Einträge zu vermeiden und die Daten konsistent in der Datenbank abzulegen. Über die Benutzeroberfläche können vorhandene Berichte durchsucht, gefiltert und ausgewählte Datensätze visualisiert werden. Damit wurde das Hauptziel dieser Arbeit, die automatisierte Datenaufbereitung und visuelle Darstellung von Prüfdaten, weitestgehend erreicht. Hierfür wird eine Unregelmäßigkeit in der Berichtstruktur zugrunde gelegt, die dem Entwickler entgangen ist und durch bestimmte Bauformen der zu testenden Einrichtung ausgelöst wird.

6.2. Kritische Bewertung

Obwohl die entwickelte Applikation bzw. der Prototyp die Kernfunktionen weitestgehend erfüllt und es positives Feedback der späteren Nutzer gegeben hat, zeigt die Konzeption und die derzeitige Umsetzung noch große Schwächen auf. Insbesondere eine aussagekräftige Visualisierung der Berichtsdaten in Bezug auf die Aussage der Testergebnisse ist mangelhaft. Die Darstellung

sagt wenig über die Einordnung der Berichte aus. Es wird sich nur auf einzelne Berichtsteile in der Analyse beschränkt, womit die Einordnung in einen Gesamtkontext nicht ausreichend dargestellt ist.

Nach dem derzeitigen Konzept kann die Applikation nur zur Analyse für das Fachpersonal eingesetzt werden und benötigt noch viele Anpassungen und Erweiterungen, um in der geplanten Umgebung eingesetzt zu werden. Hierbei müssen zudem intensive Tests durchgeführt werden, welche alle bisher bekannten Typen und Fehlervarianten des Teststandes bestmöglich abdecken. Die bisherigen Tests waren nur zum Erkennen grundlegender Mängel zu nutzen und können nur einen grundlegenden Erfolg des Ansatzes belegen. Es müssen automatisierte und gründliche Tests für das Prüfen der Applikation erstellt werden, um eine genauere Einschätzung zu gewinnen.

Ein weiterer Schwachpunkt ist die Struktur und Lesbarkeit des Applikationscodes. Diese ist kaum vorhanden. Die Applikation besitzt zu viele ansatzweise identische Funktionen, welche ähnliche Aufgabenbereiche haben und zu einer generelleren und somit flexibleren Funktion zusammengefasst werden sollten.

Dennoch sind die Grundstruktur und Ansätze des Konzeptes vielversprechend und können mit einigen geringeren Anpassungen und Erweiterungen zu einer geeigneten Applikation führen. Insbesondere die Wahl der genutzten Bibliotheken und Methoden sowie die gewählte Systemarchitektur haben sich als geeignet für die Erstellung der gewünschten Applikation herausgestellt.

Der derzeitige Prototyp bzw. Applikationsstand stellt hierbei nur einen hastigen Ansatz dar, der aufgrund der allgemein knapp bemessenen Zeit und des Umfangs des Projektes nicht alle gewünschten Ziele zur Gänze erfüllen konnten. Dennoch bestätigt der aktuelle Stand die prinzipielle Umsetzbarkeit und Leistungsfähigkeit des gewählten Konzeptes.

6.3. Ausblick und möglichkeiten für zukünftige Erweiterungen

Diese Arbeit bietet eine vielversprechende Grundlage für zukünftige Arbeiten und Erweiterungen. Der aktuelle Prototyp erfüllt bereits erfolgreich die essenziellen Funktionen zur Verarbeitung, Speicherung und Anzeige von XML-Daten, hat jedoch noch Schwächen in den Bereichen Stabilität, Benutzerführung und Visualisierung.

Der nächste wesentliche Schritt ist die Behebung der durch das Testen bekannten Mängel und die Entwicklung von automatisierten Tests, welche alle bekannten Varianten der XML-Berichte abdecken und die Datenverarbeitung auf logische Konsistenz prüfen, um somit das Einlesen und Verarbeiten für alle Testberichte zu gewährleisten. Zudem sollte die Stabilität und Wartbarkeit des Applikationscodes optimiert werden.

Nach dem Beheben der grundlegenden Mängel sollte die grafische Analyse optimiert werden, beispielsweise durch Vergleichsansichten und statistische Auswertungen. So wie die Integration einer Benutzerverwaltung mit unterschiedlichen Rollen (z. B. Administrator, Prüfer, Techniker), um den Zugriff auf Daten zu steuern.

Mittelfristig soll die Applikation weiterentwickelt werden, um vollständige Kundenberichte zu erstellen, die sowohl intern als auch extern verwendet werden, und um die alten Speichermethoden vollständig zu ersetzen.

Sobald die Datenbank als Bericht umfasst, soll langfristig auf eine Entwicklung der Applikation zu einer Analyseplattform für die Erkennung der Alterungsprozesse der DUTs hingeführt werden. Hierbei könnte auch Machine Learning zur Erkennung von Fehlerbildern einen möglichen Entwicklungsansatz darstellen.

Insgesamt zeigt der aktuelle Prototyp, dass das zugrunde liegende Konzept tragfähig ist und ein erhebliches Potenzial für die zukünftige Nutzung im produktiven Umfeld bietet. Hierbei sind die mittelfristigen Ziele mit diesem Ansatz mit großer Sicherheit umsetzbar. Für die langfristigen Ziele der Applikation müsste zu einem fortgeschrittenen Zeitpunkt eine Evaluierung durchgeführt werden.

i. Literaturverzeichnis

Literatur

[1] A. GmbH, *USTB DWT (XCT0006-1) Main Manual V1.0*, Aachen: AixControl GmbH, 2018.

[2] G. Neumann, „Auszeichnungssprache," in *Enzyklopädie der Wirtschaftsinformatik – Online-Lexikon*, Online; abgerufen am 18.08.2025, Berlin: GITO, 2019. besucht am 18. Aug. 2025. Adresse: https://wi-lex.de/index.php/lexikon/technologische-und-methodische-grundlagen/sprache/auszeichnungssprache/

[3] P. Brezany, *XML und Datenbanken*, Vorlesungsskript, Online; abgerufen am 22.09.2025, 2003. besucht am 22. Sep. 2025. Adresse: https://homepage.univie.ac.at/peter.brezany/teach/kfk/02ws-vo/skriptum/14-01-03.pdf

[4] M. Becher, „XML-Grundlagen," in *XML: DTD, XML-Schema, XPath, XQuery, XSL-FO, SAX, DOM*. Wiesbaden: Springer Fachmedien Wiesbaden, 2022, ISBN: 978-3-658-35435-0. DOI: 10.1007/978-3-658-35435-0_1 Adresse: https://doi.org/10.1007/978-3-658-35435-0_1

[5] B. Jung. „Homepage Webhilfe: Grundlagen XPath." Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: https://www.homepage-webhilfe.de/XML/XPath/

[6] Python Software Foundation. „xml.etree.ElementTree — The ElementTree XML API." Online; abgerufen am 21.08.2025, besucht am 21. Aug. 2025. Adresse: https://docs.python.org/3/library/xml.etree.elementtree.html#module-xml.etree.ElementTree

[7] Python Software Foundation. „xml.dom.minidom — Minimal DOM implementation." Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: https://docs.python.org/3/library/xml.dom.minidom.html

[8] Python Software Foundation. „xml.sax — Support for SAX2 parsers." Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: https://docs.python.org/3/library/xml.sax.html

[9] S. Richter. „lxml — XML and HTML with Python." Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: https://lxml.de/

[10] Python Software Foundation. „beautifulsoup4 4.13.5." Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: https://pypi.org/project/beautifulsoup4/

[11] Python Software Foundation. „xmlschema 4.1.0." Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: https://pypi.org/project/xmlschema/

[12] Python Software Foundation. „defusedxml 0.7.1." Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: https://pypi.org/project/defusedxml/

[13] C. Stefanescu. „untangle: Convert XML to Python objects." Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: https://untangle.readthedocs.io/en/latest/

[14] Pallets Projects, *Flask Documentation*, https://flask.palletsprojects.com/en/stable/, Zugriff am 10. Oktober 2025, 2024.

[15] Pallets Projects, *Flask Application Lifecycle*, https://flask.palletsprojects.com/en/stable/lifecycle/, Zugriff am 22. Oktober 2025, 2024.

[16] P. Kennedy, *How Are Requests Processed in Flask?* https://testdriven.io/blog/how-are-requests-processed-in-flask/, Zugriff am 22. Oktober 2025, 2024.

[17] P. Projects. „Flask Documentation – Blueprints.“ Zugriff am 12. Oktober 2025. Adresse: https://flask.palletsprojects.com/en/stable/blueprints/

[18] E. Fuchs, *SQL: Grundlagen und Datenbankdesign*, 1. Ausgabe. Bodenheim: HERDT-Verlag, Juli 2021, ISBN: 978-3-98569-009-1.

[19] F. Herrmann, *Datenorganisation und Datenbanken: Praxisorientierte Übungen mit MS Access 2016*. Wiesbaden: Springer Vieweg, 2018, ISBN: 978-3-658-21330-5. DOI: 10.1007/978-3-658-21331-2 Adresse: https://doi.org/10.1007/978-3-658-21331-2

[20] A. Gadatsch, *Datenmodellierung: Einführung in die Entity-Relationship-Modellierung und das Relationenmodell*, 2., aktualisierte Auflage. Wiesbaden: Springer Vieweg, 2019, ISBN: 978-3-658-25729-3. DOI: 10.1007/978-3-658-25730-9 Adresse: https://doi.org/10.1007/978-3-658-25730-9

[21] Oracle Corporation. „MySQL Connector/Python.“ Version 9.1.0, zuletzt abgerufen am 09. Oktober 2025, Oracle Corporation. Adresse: https://pypi.org/project/mysql-connector-python/

[22] Federico Di Gregorio und Daniele Varrazzo. „psycopg2 – PostgreSQL Database Adapter for Python.“ Version 2.9.10, zuletzt abgerufen am 09. Oktober 2025, Psycopg Project. Adresse: https://pypi.org/project/psycopg2/

[23] MariaDB Foundation. „MariaDB Connector/Python Documentation.“ Zuletzt abgerufen am 09. Oktober 2025, MariaDB Foundation. Adresse: https://mariadb.com/docs/connectors/mariadb-connector-python

[24] M. Grinberg. „The Flask Mega-Tutorial, Part IV: Database.“ Zugriff am 8. Oktober 2025. Adresse: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-iv-database

[25] Pallets Projects, *Using SQLAlchemy with Flask*, Zugriff am 8. Oktober 2025, 2025. Adresse: https://flask.palletsprojects.com/en/stable/patterns/sqlalchemy/

[26] Pallets Projects, *Flask-SQLAlchemy Documentation (Version 3.x)*, Zugriff am 8. Oktober 2025, 2025. Adresse: https://flask-sqlalchemy.readthedocs.io/en/stable/quickstart/

[27] S. Ramírez, *SQLModel Documentation*, Zugriff am 8. Oktober 2025, 2025. Adresse: https://sqlmodel.tiangolo.com/

[28] C. O. Wilke, *Datenvisualisierung – Grundlagen und Praxis: Wie Sie aussagekräftige Diagramme und Grafiken gestalten*, übers. von B. Gür. Heidelberg: Dpunkt.Verlag / O'Reilly, 2020, ISBN: 3960091214.

[29] *Chart.js Documentation — Getting Started*, https://www.chartjs.org/docs/latest/, Zugriff am 21. Oktober 2025, 2025.

[30] *What is D3? — D3.js*, https://d3js.org/what-is-d3, Zugriff am 21. Oktober 2025, 2025.

[31] *Plotly JavaScript Open Source Graphing Library*, https://plotly.com/javascript/, Zugriff am 21. Oktober 2025, 2025.

[32] *Features — Apache ECharts*, https://echarts.apache.org/en/feature.html, Zugriff am 21. Oktober 2025, 2025.

[33] *vis.js — Community Edition*, https://visjs.org/, Zugriff am 21. Oktober 2025, 2025.

[34] *Recharts — A composable charting library built with React components*, https://recharts.org/, Zugriff am 21. Oktober 2025, 2025.

[35] *ISO/IEC 9126: Information technology — Software product evaluation — Quality characteristics and guidelines for their use*, Consulted via summarized PDF provided, Geneva: International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC), 1991.

[36] MariaDB Foundation, *MariaDB.org — The MariaDB Foundation*, https://mariadb.org/, Zugriff am 12. Oktober 2025, 2025.

ii. Anhangsverzeichnis

iii. Erklärung

Hiermit erkläre ich, dass ich die von mir eingereichte Bachelor- / Masterarbeit „Datenverarbeitung und Visualisierung von Umrichter-Testbench-Daten“ selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.
Ort und Datum

persönliche Unterschrift

(Name des Verfassers)

Zusatz zur Nutzung von KI-Assistenzsystemen
Im Rahmen der Erstellung dieser Arbeit habe ich KI-gestützte Assistenzwerkzeuge (*QuillBot* und *ChatGPT*) *ausschließlich* zu folgenden Zwecken eingesetzt: (i) Rechtschreib- und Grammatikprüfung, (ii) sprachliche Glättung und stilistische Hinweise zu von mir verfassten Textpassagen, (iii) grobe thematische Einordnung sowie Vorschläge zur Strukturierung/Gliederung.

Es wurden *keine* inhaltlich tragenden Texte, Abbildungen oder Tabellen von KI-Systemen generiert und unverändert übernommen. Sämtliche inhaltlichen Aussagen, Formulierungen und Ergebnisse stammen von mir. Alle verwendeten Quellen (einschließlich solcher, auf die KI-Werkzeuge ggf. hingewiesen haben) wurden von mir eigenständig geprüft, bewertet und ordnungsgemäß zitiert.