Dokumentation zum XML-Projekt im Master-Modul Internettechnologie

Lennart Mende
Richard Mende

HTWK Leipzig

Wintersemester 2025/26

Prof. Dr.-Ing. Andreas Pretschner

31. Dezember 2025

Inhaltsverzeichnis

1 Projekübersicht

1.1 Zielstellung

Als Grundlage dieses Projekts dient das Anlegen eines XML-Datensatzes, der aus mehreren Objekten besteht. Diese sollen jeweils mindestens ein Attribut sowie eine Datensequenz bestehen. Auf Basis dieses Datensatzes wird eine XSD-Schemadatei erstellt, sodass die XML-Datei validiert werden kann. Dabei wird überprüft, ob die XML-Datei der definierten Struktur entspricht und alle erforderlichen Angaben enthalten sind. Zusätzlich wird eine XSD-Schemadatei mithilfe der in *EditiX* integrierten automatischen Dokumenterstellung generiert.

Der zweite Teil des Projekts befasst sich mit der Transformation der XML-Datei in zwei unterschiedliche Ausganbeformate. Zunächst sollen die Daten aus der XML-Datei in Form einer HTML-Seite dargestellt werden, wofür eine XSLT-Transformationsdatei erstellt werden muss. Darüber hinaus erfolgt die FOP-Transformation in eine PDF erfolgen, wodurch eine PDF generiert wird. Eine schematische Übersicht dieser Transformationen zeigt Abbildung 1.

Der dritte Aufgabenteil beschäftigt sich mit der XML-Programmierung. Dabei soll eine Applikation erstellt werden, welche die Validierung, Abfrage und Transformation der XML-Datei ermöglicht.

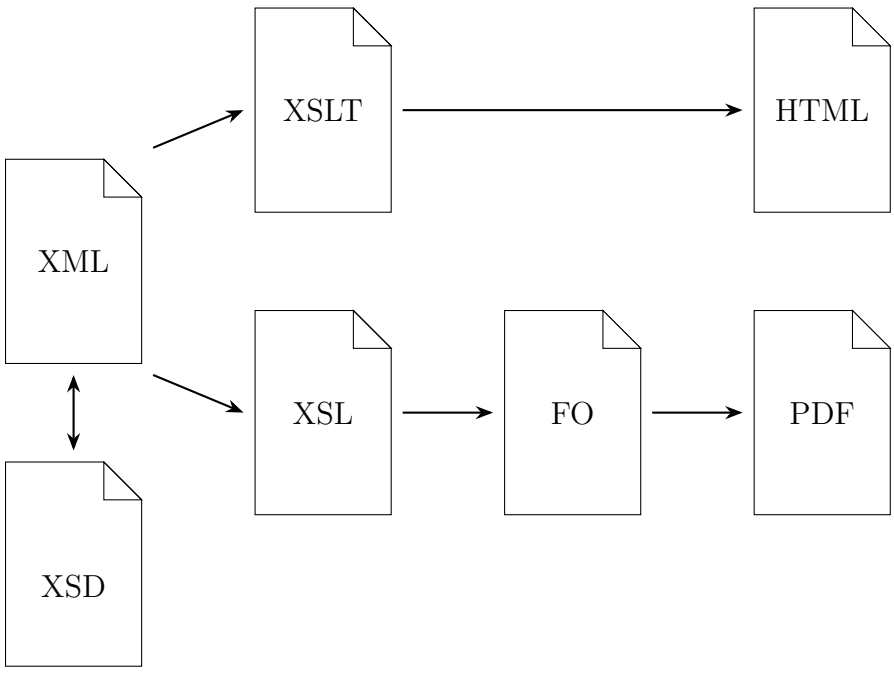

Abbildung 1: Übersicht der Dokumenttypen und Transformationspfade

1.2 Projektorganisation

Zur strukturierten Umsetzung des Projekts sowie zur Sicherstellung einer nachvollziehbaren Versionsverwaltung wurde ein öffentlich zugängliches *GitHub-Repository* unter `https://github.com/rimen13/Projekt_Internettechnologie` eingerichtet. Die Gliederung der Projektdateien ergibt sich folgendermaßen:

```
project/
|-- assets/
|   |-- src/
|   |   |-- products.xml
|   |   |-- products.xsd
|   |   |-- products-automatic-generated.xsd
|   |   |-- products-to-html.xslt
|   |   |-- products-to-pdf.xsl
|   |-- output/
|       |-- products.html
|       |-- products.fo
|       |-- products.pdf
|-- documentation/
|   |-- documentation.pdf
|-- programming/
    |-- src/
    |   |-- query_backend.py
    |   |-- query_from_project.py
    |   |-- query_interactive.py
    |   |-- validate_backend.py
    |   |-- validate_from_project.py
    |   |-- validate_interactive.py
    |   |-- transform_backend.py
    |   |-- transform_from_project.py
    |   |-- transform_interactive.py
    |-- output/
    |   |-- products.html
    |-- requirements.txt
```

Als Entwicklungswerkzeuge kamen der XML-Editor *EditiX* zur Erstellung und Dokumentation der XML- und XSD-Dateien sowie *Visual Studio Code* als integrierte Entwicklungsumgebung zur Umsetzung der XML-Programmierung zum Einsatz. Die inhaltliche Ausarbeitung orientiert sich an den in Moodle bereitgestellten Unterlagen zum Modul *Internettechnologie*. Die in der XML-Datei verwendeten Preise und Stückzahlen der elektronischen Bauteile basieren auf den Angaben der beiden Websiten `https://www.digikey.de` und `https://www.mouser.de`.

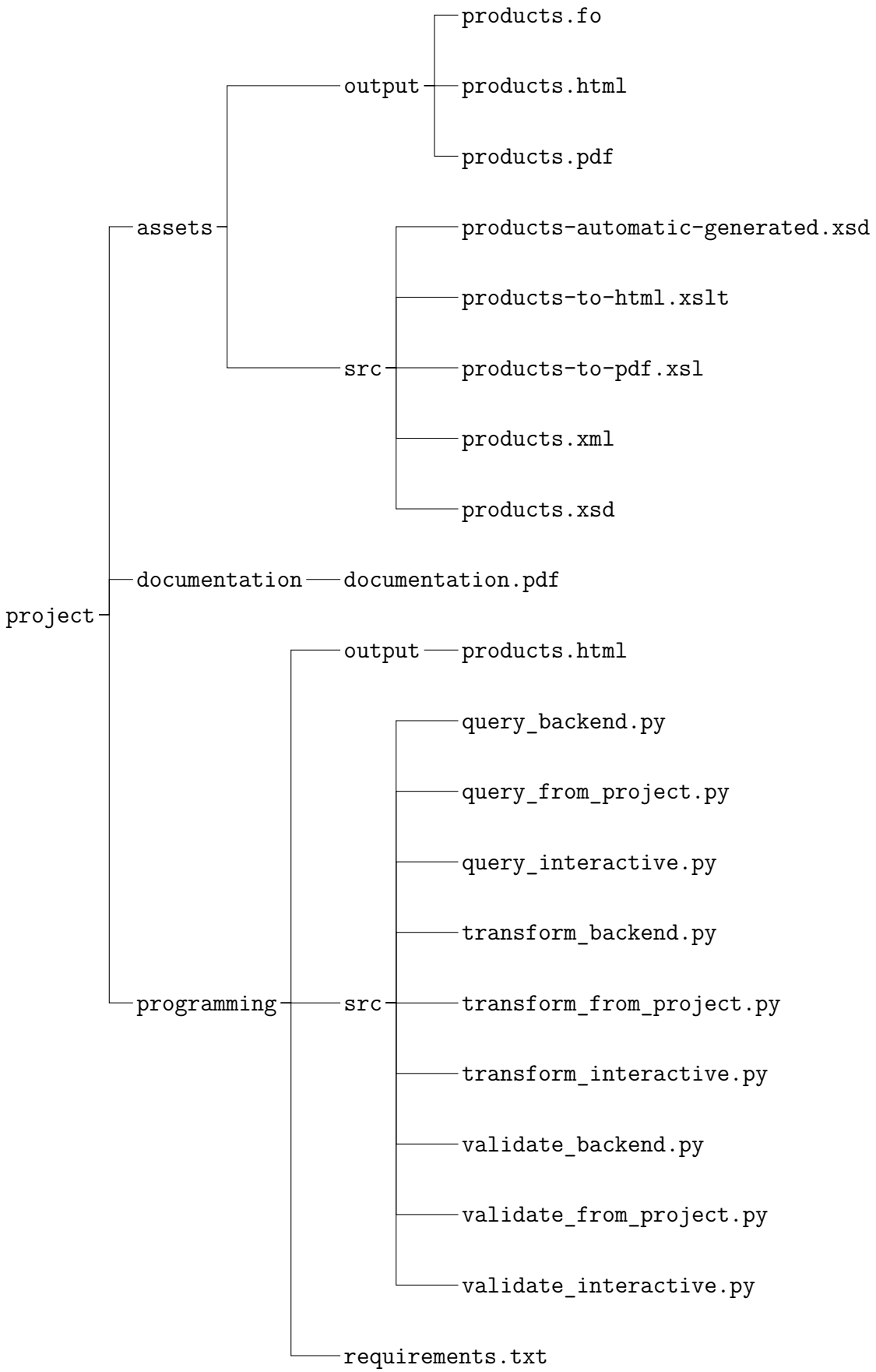

Abbildung 2: Projektordnerstruktur

2 Datensatz

2.1 XML-Datei

Als Beispieldatensatz für dieses Projekt wurde ein Katalog elektronischer Bauteile eines fiktiven Elektronikshops modelliert. Der vollständige Datensatz ist in der Datei *products.xml* abgelegt und umfasst insgesamt neun Produkte, darunter drei Widerstände, zwei Spulen, zwei Kondensatoren sowie zwei Dioden. Das Root-Element *products* kapselt alle enthaltenen *product*-Elemente. Jedes *product*-Element repräsentiert ein einzelnes Bauteil und ist über das Attribut *id* eindeutig identifizierbar. Zusätzlich wird über das Attribut *type* der grundlegende Bauteiltyp spezifiziert, bei Dioden optional erweitert durch das Attribut *subtype*. Unabhängig vom konkreten Bauteiltyp enthalten alle *product*-Elemente eine einheitliche Menge an Kindelementen zur Beschreibung allgemeiner Produkteigenschaften. Diese sind *amount*, *price* und *manufacturer*. Der Preis kann über ein *unit*-Attribut weiter typisiert werden, wodurch unterschiedliche Währungseinheiten realisierbar sind.

Ergänzend zu diesen allgemeinen Angaben besitzt jedes Produkt eine oder mehrere bauteilspezifische Datensequenzen, welche die relevanten physikalischen Kenngrößen modellieren. Die Bezeichnung dieser Sequenzen ist vom jeweiligen Bauteiltyp abhängig, *resistance* für Widerstände, *inductance* für Spulen oder *capacitance* für Kondensatoren. Für Dioden mehrere Kenngrößen angegeben werden (Durchlassspannung *forwardVoltage* sowie Sperrspannung *reverseVoltage*). Diese Elemente enthalten jeweils eine strukturierte Beschreibung des Zahlenwertes, bestehend aus Mantisse (*value*), Exponent (*exponent*), Einheit (*unit*) sowie optional einer Toleranzangabe. Der Exponent ist hierbei auf Werte beschränkt, die durch 3 teilbar sind, da nur diese später in physikalische Vorsilben wie µ, m, k etc. umgerechnet werden können. Die Toleranz kann über ein *unit*-Attribut in Prozent angegeben werden. Diese explizite Zuordnung der physikalischen Größen zu den jeweiligen Bauteiltypen erleichtert die formale Beschreibung des Datenmodells und bildet eine geeignete Grundlage für die spätere Validierung des XML-Dokuments mithilfe einer XSD-Schemadatei.

alter Text:

Als Thema für den Beispieldatensatz wurden in diesem Projekt elektronische Bauteile eines fiktiven Elektronikshops gewählt. Der vollständige Datensatz dieses Projekts ist in der Datei *products.xml* enthalten. Er umfasst 9 Objekte, davon 3 Widerstände, 2 Spulen, 2 Kondensatoren sowie 2 Dioden. Hintergrund ist, dass die vorhandenen Daten

auf einer Produkt-Website dargestellt werden können. Beim Inhalt des Datensatzes wurde sich deshalb auf die dafür wichtigsten Daten konzentriert. Das Root-Element ist *Products*, das alle *Product*-Elemente enthält. Diese können über den Eintrag im *id*-Attribut eindeutig identifiziert werden. Jedes dieser *Product*-Elemente umfasst die Elemente Anzahl, Preis und Hersteller sowie, abhängig vom mit dem *type*-Attribut spezifizierten Bauteil, eine oder zwei bauteilspezifische Datensequenzen. Diese beinhalten die physikalischen Größen des jeweiligen Bauteils, wobei der Wert, die Einheit und die Toleranz angegeben werden. Sie kann mithilfe des *unit*-Attributs relativ (bspw. mit %) oder absolut eingestellt werden. Die Einheit des Preises kann ebenfalls mit einem *unit*-Attribut definiert werden. Die Bezeichnung dieser Datensequenz ist für jedes Bauteil spezifisch. Während die Widerstände, Spulen und Kapazitäten jeweils nur eine spezifische physikalische Größe umfassen, können für Dioden mehrere angegeben werden. Diese konkrete Zuordnung der physikalischen Größen vereinfacht die anschließende Validierung der XML-Datei mit der XSD-Schemadatei.

2.2 XSD-Schemadatei

Zur formalen Beschreibung des XML-Datenmodells dient die XSD-Schemadatei *products.xsd*. Sie definiert die Struktur, Reihenfolge und erlaubte Häufigkeit der enthaltenen Elemente und stellt damit sicher, dass alle Angaben in der XML-Datei *products.xml* valide und konsistent sind. Insgesamt gewährleistet die XSD-Schemadatei eine eindeutige Definition des Datenmodells, unterstützt die Validierung von Datentypen, Elementreihenfolgen und Attributen. Das Root-Element *products* kapselt die *product*-Elemente, deren Anzahl mindestens 1 betragen muss. Jedes *product*-Element verweist auf ein oder mehrere bauteilspezifische Kindelemente, die über einen *choice*-Konstrukt modelliert sind. Dadurch werden die zulässigen Kombinationen physikalischer Größen wie *resistance*, *inductance*, *capacitance*, *forwardVoltage* und *reverseVoltage* zum jeweiligen Bauteil erlaubt.

Alle Produkte besitzen allgemeine Attribute wie *id*, *type* und optional *subtype*, die über eine *attributeGroup* zentral verwaltet werden. Die bauteilspezifischen Elemente nutzen jeweils einen eigenen *complexType*, der die Mantisse (*value*), den Exponenten (*exponent*), die Einheit (*unit*) und optional eine Toleranz (*tolerance*) kapselt. Die Typisierung der Elemente erfolgt über primitive Datentypen wie *float*, *short* oder *string*, während für Attribute Standardwerte und feste Werte genutzt werden, um Konsistenz zu gewährleisten und Abweichungen frühzeitig als Fehler zu erkennen. Für die Preisangaben wurde ein spezieller *PriceType* implementiert, der den Wert und die Einheit

kapselt.

Die automatisch durch *EditiX* erzeugte Datei *products-automatic-generated.xsd* kann ebenfalls zur Dokumentation herangezogen werden, enthält jedoch nicht die gleichen präzisen Typdefinitionen, erlaubt keine *choice*-Strukturen und ist daher für die Validierung der XML-Datei nur eingeschränkt geeignet. Es wird daher empfohlen, zur Validierung der XML-Datei die XSD-Datei *products.xsd* zu verwenden.

alter Text:

Das zugehörige Schema befindet sich in `products.xsd`. Es definiert Struktur, Reihenfolge und Häufigkeit der verwendeten Elemente. Das Schema dient dazu, die Angaben in der XML-Datei auf formale Richtigkeit zu überprüfen. Damit kann sichergestellt werden, dass die geforderten Angaben in der XML-Datei gemacht wurden. Diese Richtigkeit ist für die Transformation in andere Datenformate unerlässlich. Außerdem kann überprüft werden, ob die angegebenen Datentypen korrekt sind. Die erstellte XSD-Datei dieses Projekts ist in *products.xsd* zu finden. Beim Entwurf des XSD-Schemas wurde darauf geachtet, jedes Element über einen eigenen Typ zu kapseln, sodass spätere Änderungen gezielt und unabhängig vorgenommen werden können. Die verwendeten Datentypen wurden bewusst gewählt, um eine valide und konsistente Datenstruktur sicherzustellen. Standardwerte kommen dort zum Einsatz, wo Attribute fehlen, um sinnvolle und einheitliche Einstellungen zu gewährleisten. Feste Werte werden verwendet, um Abweichungen frühzeitig zu erkennen und als Fehler zu kennzeichnen. Darüber hinaus wurde die Datei *products-automatic-generated.xsd* automatisch durch die Dokumentenerstellung mit *EditiX* erzeugt. Diese Datei ist jedoch nicht so präzise wie *products.xsd*. Es können nicht an allen Stellen die gewünschten Datentypen angegeben werden. Außerdem ist es nicht möglich, ein *choice*-Element zu verwenden. Es wird daher empfohlen, zur Validierung der XML-Datei die XSD-Datei *products.xsd* zu verwenden.

3 Transformationen von XML-Daten

In diesem Kapitel wird die Umsetzung der Transformationen der XML-Daten in verschiedene Ausgabeformate beschrieben. Es werden die verwendeten XSLT-Templates zur Erzeugung von HTML sowie die XSL-FO-Transformation für PDF-Dokumente erläutert. Die hier beschriebenen Verfahren basieren auf der XML-Struktur und den in Kapitel 2 definierten Datentypen und Elementen.

3.1 XSLT-Transformation von XML zu HTML

Für die Darstellung der Produktdaten in tabellarischer Form wurde die Datei `products.xslt` erstellt. Die Tabellen sind nach Bauteiltypen getrennt: Widerstände, Spulen und Kondensatoren besitzen jeweils eine physikalische Größe, während Dioden zwei Kenngrößen (Forward Voltage, Reverse Voltage) sowie den Untertyp enthalten.

Die Templates greifen über XPath-Ausdrücke auf die Kindelemente der XML-Daten zu und extrahieren die entsprechenden Werte. Jede physikalische Kenngröße wird über ein gemeinsames Template ausgewertet, das Mantisse, Exponent, Einheit und optional vorhandene Toleranz verarbeitet. Die Exponenten werden dabei nach physikalischen Vorsilben interpretiert (p, n, µ, m, k, M). Fehlen optionale Werte wie die Toleranz, wird ein Platzhalter – in die Tabelle eingesetzt, um eine konsistente tabellarische Darstellung zu gewährleisten.

Die Tabellenüberschriften werden über `<th>`-Elemente generiert, wobei die Reihenfolge der Zellen der XML-Struktur entspricht. Die Templates sind modulartig implementiert, um spätere Anpassungen oder neue Bauteiltypen einfach integrieren zu können.

Beispielsweise wird die Einheit des Widerstandes (Ω) korrekt zusammen mit dem Exponenten angezeigt, sodass bei einer Umrechnung in physikalische Vorsilben die Lesbarkeit erhalten bleibt. Die Wiederverwendbarkeit der Templates erlaubt die einfache Anpassung für unterschiedliche Layouts oder neue Produktdaten.

3.2 XSL-FO-Transformation zu PDF

Für die FOP-Transformation wurde die XSLT-Logik der HTML-Ausgabe größtenteils wiederverwendet. Die Informationen werden über XPath auf die XML-Daten zugregriffen, die HTML-Syntax durch XSL-FO ersetzt und ein Master-Layout definiert, das Seitenränder, Schriftart und -größe festlegt.

Jedes Bauteil wird als Zeile in einer `fo:table` ausgegeben, wobei Tabellenüberschriften über `fo:table-header` und Zellen über `fo:table-cell` definiert werden. Optionale Werte, wie Toleranzen, werden ebenfalls mit Platzhalterwerten angezeigt, wenn diese in der XML nicht vorhanden sind.

Eine Besonderheit der FO-Transformation betrifft das Omega-Zeichen für Widerstän-

de. Da dieses Zeichen sonst eine halbe Zeile zu tief angezeigt wird, wird es mittels `baseline-shift` vertikal korrekt positioniert. Alle anderen physikalischen Einheiten werden standardmäßig inline ausgegeben.

Die Exponenten der physikalischen Größen werden in Vorsilben umgesetzt (p, n, µ, m, k, M). Es wird dabei sichergestellt, dass nur Exponenten verwendet werden, die durch 3 teilbar sind, um eine korrekte Zuordnung zu den SI-Vorsilben zu gewährleisten. Dies erleichtert die konsistente Darstellung der physikalischen Größen sowohl in HTML als auch in PDF.

3.3 Flexibilität und Wiederverwendbarkeit

Die Templates sind so generisch implementiert, dass Änderungen der XML-Struktur, neue Bauteiltypen oder zusätzliche Kenngrößen leicht integriert werden können. Die Verwendung von XPath-Ausdrücken in Verbindung mit `<xsl:apply-templates>` ermöglicht die modulare Verarbeitung aller relevanten Elemente.

Durch die Trennung von Layout (HTML vs. FO) und Datenlogik (XPath, Templates) wird eine hohe Wartbarkeit erreicht. Anpassungen am visuellen Erscheinungsbild, an den Tabellenlayouts oder an der Darstellung von physikalischen Größen können ohne Änderungen an der Datenstruktur umgesetzt werden.

Die Transformationen dienen dazu, die im Kapitel 2 definierten XML-Daten in unterschiedliche Ausgabeformate zu überführen. Die XSLT-Templates erlauben die Erzeugung von HTML-Tabellen, während die XSL-FO-Templates die Darstellung in PDF-Dateien ermöglichen. Optional vorhandene Werte werden durch Platzhalter ersetzt, Exponenten werden in physikalische Vorsilben umgesetzt, und Sonderzeichen wie Ω werden korrekt positioniert. Die modulare, XPath-basierte Implementierung sorgt für Flexibilität, Wiederverwendbarkeit und einfache Wartbarkeit der Transformationen.

alter Text:

4 Transformationen

4.1 XSLT von XML in HTML

Um die XML in HTML transformieren zu können, wurde `products.xslt` erstellt. Die tabellenförmige Ausgabe wurde getrennt für Widerstände, Spulen und Kondensatoren sowie für Dioden vorgenommen, da für erstere jeweils eine physikalische Größe angegeben ist, für letztere zwei physikalische Größen sowie der Untertyp.

Die physikalischen Größen selbst wurden mit einem gemeinsamen Template ausgewertet, da alle einen Wert und eine Einheit sowie einen optionalen Exponenten und eine optionale Toleranz aufweisen. Mit einer `xsl:when`-Abfrage wurde sichergestellt, dass bei Abwesenheit einer Toleranz dieses Feld mit einem Bindestrich gefüllt wird. Da das Feld für die physikalische Größe in der entsprechenden Tabelle jedoch auch ohne einen explizit vorgegebenen Exponenten korrekt ausgefüllt wird und alle weiteren Angaben in der Tabelle obligatorisch sind, ist dieses Vorgehen nur für die Toleranz notwendig.

Im Allgemeinen wurden die Templates so generisch wie möglich erstellt, um eine hohe Flexibilität für mögliche Veränderungen zu gewährleisten.

4.2 FOP-Transformation

Für die FOP-Transformation wurde hauptsächlich die XSLT für den HTML-Output wiederverwendet, da die gleichen Informationen mittels XPath ermittelt wurden. Es wurde lediglich die HTML-Syntax durch FO-Syntax ersetzt, zusätzlich wurde ein Master-Layout definiert.

Der einzige Unterschied besteht in der Behandlung des Ω. Dieses musste im Vergleich zu HTML und zu den anderen Zeichen des Dokuments stets um 0.145 em nach oben verschoben werden, da es sonst eine halbe Zeile zu tief steht.

Die beiden durchgeführten Transformationen sind sehr generisch – mit ihnen kann stets das angestrebte Output-Dokument erzeugt werden, sofern die übergebene XML den in Abschnitt 2 erläuterten Anforderungen der Schemadatei genügt.

5 Programmierung

5.1 Grundlegende Informationen

Durch die XML-Programmierung sollen folgende Funktionalitäten implementiert werden:

1. Validierung eines XML-Dokuments gegen ein XSD-Schema
2. Abfrage von Informationen aus einem XML-Dokument
3. Transformation eines XML-Dokuments in eine HTML-Website

Zur Umsetzung dieser Funktionalitäten wird im Rahmen dieses Projekts die Programmiersprache *Python* verwendet. Um das die in den folgenden Kapiteln beschriebenen Befehle in der Kommandozeile ausführen zu können, muss *Python* von der offiziellen Website https://www.python.org unter dem Reiter *Downloads* installiert werden. Für dieses Projekt wurde Version 3.12.8 genutzt. Außerdem müssen die beiden Module *lxml* und *tkinter* installiert werden.

Durch das Modul *lxml* ist es möglich, diese Funktionalitäten umzusetzen.

Damit die Befehle über die Kommandozeile aufgerufen werden können, muss zunächst ein lokaler Python Download erfolgen.

HIER MUSS ICH ES MAL MIT MEINEM EIGENEN LINUX SYSTEM PROBIEREN!

Für alle 3 Teilaufgaben existieren 3 getrennte Python-Dateien:

1. Backend-Datei (`*_backend.py`): enthält ausschließlich die Logik der jeweiligen Aufgabe und können nicht direkt ausgeführt werden
2. Projektmodus-Datei (`*_from_project.py`): führt die jeweilige Aufgabe mit den im Projekt erarbeiteten Dateien aus
3. Interaktiv-Datei (`*_interactive.py`): ermöglicht die Auswahl beliebiger Dateien

und führt die jeweilige Aufgabe für diese durch

Die jeweiligen Dateien enthalten den beschriebenen Code, der ausführlich kommentiert ist. Die Dateien können ausgeführt werden, indem man zunächst die virtuelle Umgebung aktiviert und in der Kommandizeile in das Verzeichnis `programming/src` wechselt und den Befehl `python desired_file.py` ausführt. Dabei ist `desired_file.py` ein Platzhalter für den eigentlichen Dateinamen, der an dieser Stelle natürlich eingesetzt werden muss.

5.2 Abfrage

Bei der Abfrage werden gezielt Informationen aus einem XML-Dokument extrahiert, indem XPath verwendet wird. Hierbei werden 3 verschiedene Abfragen durchgeführt:

1. Welche Produkt-IDs gibt es?

2. Wie heißen die Hersteller? (Hierbei sollen mehrfach aufgeführte Hersteller nur einmal aufgeführt werden.)

3. Was ist die Gesamtanzahl aller Widerstände im Lager?

Führt man die Datei `query_from_project.py` für den Projektmodus aus, dann wird in der Kommandozeile folgendes ausgegeben:

```
Query 1: Retrieve all product IDs.
Result: ['R001', 'R002', 'R003', 'L001', 'L002', 'C001', 'C002',
        'D001', 'D002']

Query 2: Retrieve unique manufacturer names.
Result: ['Diodes Incorporated', 'Micro Commercial Components',
        'Murata Power Solutions', 'Samsung Electro-Mechanics',
        'Stackpole Electronics', 'Taiyo Yuden']

Query 3: Calculate the total number of resistors in stock.
Result: 864907
```

Möchte man die interaktive Abfrage durchführen, kann man über die Datei `query_from_project.py` eine XML-Datei aus dem Datei-Explorer auswählen. Anschließend wird zunächst überprüft, ob die angegebene Datei das vorgegebene XSD-Schema erfüllt. Ist dies nicht der Fall, können die Abfragen nicht durchgeführt werden und es wird eine Fehlermeldung ausgegeben. Ist die XML-Datei gültig, werden ebenfalls die 3 Abfragen getätigt und die Ergebnisse ausgegeben.

5.3 Validierung

Um eine XML-Datei gegen ein XSD-Schema validieren zu können, kommen die Dateien `validate_from_project.py` und `validate_interactive.py` zur Anwendung. Dabei kann in ersterem Fall die XML-Datei gegen die XSD-Datei aus dem gegebenen Projekt validiert werden. Für die zweite Datei ist die Auswahl einer beliebigen XML- und XSD-Datei im Datei-Explorer möglich. Die Ausgabe in der Kommandozeile für eine gültige oder nicht gültige XML-Datei lautet folgendermaßen (kommt auch mit automatisch generierter XSD raus):

```
VALIDATION RESULT: products.xml is valid according to products.xsd
```

```
VALIDATION RESULT: products.xml is NOT valid according to products.xsd
```

5.4 Transformation

Die Transformation erfolgt über die Dateien `transfrom_from_project.py` und `transform_interact`
Dabei wird entweder die XML- und XSLT-Datei des gegebenen Projekts verwendet oder im interaktiven Modus werden diese beiden Dateitypen vom Anwender übergeben. Anschließend wird die Transformation in eine HTML-Datei durchgeführt und in den *output*-Ordner (in *programming*) gelegt, sodass sich der Anwender die entstehende HTML-Website anschauen kann. Im interaktiven Modus kann sich der Anwender den Zielordner selsbt aussuchen. In der Kommandozeile kommt der Hinweis:

```
HTML generated: products.html
```

Im Unterordner *programming/output* befindet sich bereits die HTML-Datei, die durch

die durchgeführte Transformation für die im Projekt gegebenen Dokumente entstanden ist.