HTL Leonding
next level

HÖHERE TECHNISCHE
BUNDESLEHRANSTALT
LEONDING

Smartbill Converter - Rechnungszuordnung mittels KI

DIPLOMARBEIT

verfasst im Rahmen der

Reife- und Diplomprüfung

an der

Höheren Abteilung für Informatik

Eingereicht von:
Sebastian Radić
Luis Schörgendorfer

Betreuer:
Gerald Unterrainer

Projektpartner:
PROGRAMMIERFABRIK GmbH

Leonding, 4. April 2025

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Leonding, 4. April 2025 Sebastian Radić & Luis Schörgendorfer

Abstract

Brief summary of our amazing work. In English. This is the only time we have to include a picture within the text. The picture should somehow represent your thesis. This is untypical for scientific work but required by the powers that are. Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Zusammenfassung

Zusammenfassung unserer genialen Arbeit. Auf Deutsch. Das ist das einzige Mal, dass eine Grafik in den Textfluss eingebunden wird. Die gewählte Grafik soll irgendwie eure Arbeit repräsentieren. Das ist ungewöhnlich für eine wissenschaftliche Arbeit aber eine Anforderung der Obrigkeit. *Bitte auf keinen Fall mit der Zusammenfassung verwechseln, die den Abschluss der Arbeit bildet!* Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Inhaltsverzeichnis

1 Einleitung

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

2 Umfeldanalyse

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula. Citing [**?**] properly.

Was ist eine GUID? Eine GUID kollidiert nicht gerne.

Kabellose Technologien sind in abgelegenen Gebieten wichtig [**?**].

3 Technologien

3.1 Foo

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat

sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum

placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

3.2 Bar

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis

viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.

Curabitur tellus magna, porttitor a, commodo a, commodo in, tortor. Donec interdum. Praesent scelerisque. Maecenas posuere sodales odio. Vivamus metus lacus, varius quis, imperdiet quis, rhoncus a, turpis. Etiam ligula arcu, elementum a, venenatis quis, sollicitudin sed, metus. Donec nunc pede, tincidunt in, venenatis vitae, faucibus vel, nibh. Pellentesque wisi. Nullam malesuada. Morbi ut tellus ut pede tincidunt porta. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam congue neque id dolor.

Donec et nisl at wisi luctus bibendum. Nam interdum tellus ac libero. Sed sem justo, laoreet vitae, fringilla at, adipiscing ut, nibh. Maecenas non sem quis tortor eleifend fermentum. Etiam id tortor ac mauris porta vulputate. Integer porta neque vitae massa. Maecenas tempus libero a libero posuere dictum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aenean quis mauris sed elit commodo placerat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus rhoncus tincidunt libero. Etiam elementum pretium justo. Vivamus est. Morbi a tellus eget pede tristique commodo. Nulla nisl. Vestibulum sed nisl eu sapien cursus rutrum.

3.2.1 Deeper

Nicht mehr im Inhaltsverzeichnis.

Deepest

Vermeide mich.

4 E-Rechnungsstandards und Compliance

Die Transformation von papierbasierten Geschäftsprozessen hin zu vollautomatisierten digitalen Workflows ist einer der wesentlichen Treiber der modernen IT-Ökonomie. Im Zentrum dieser Entwicklung steht die elektronische Rechnung (E-Rechnung). Anders als eine bloße PDF-Datei, die lediglich ein digitales Abbild eines Papierdokuments darstellt, definiert eine echte E-Rechnung strukturierte Daten, die von Maschinen ohne menschliches Zutun verarbeitet, geprüft und verbucht werden können.

In diesem Kapitel werden die theoretischen Fundamente und spezifischen Standards analysiert, die für die Entwicklung des *SmartBillConverter* maßgeblich sind. Der Fokus liegt dabei auf dem österreichischen Standard *ebInterface* sowie dem deutsch-europäischen Hybridformat *ZUGFeRD*, eingebettet in den Kontext der europäischen Norm EN 16931.

4.1 ebInterface 6.1 (Österreich)

4.1.1 Zweck und Historie

ebInterface ist der nationale österreichische Standard für die elektronische Rechnungslegung. Er wurde von der AUSTRIAPRO, einer Initiative der Wirtschaftskammer Österreich (WKO), entwickelt, um den Datenaustausch zwischen Unternehmen sowie zwischen Unternehmen und der öffentlichen Verwaltung zu standardisieren. Die Entwicklung von ebInterface reicht bis in die frühen 2000er Jahre zurück und hat sich über mehrere Versionen (4.0, 5.0, 6.0) bis zur aktuellen Version 6.1 weiterentwickelt.

Der primäre Zweck von ebInterface liegt in der Schaffung einer semantischen Interoperabilität. Während EDIFACT-Standards in großen Industriekonzernen dominierten, fehlte für KMUs (Kleine und mittlere Unternehmen) lange Zeit ein einfach zu implementieren-

des, XML-basiertes Format. ebInterface füllt diese Lücke. Seit 2014 ist die E-Rechnung an den Bund in Österreich verpflichtend, was die Verbreitung von ebInterface massiv gefördert hat. Rechnungen an den Bund müssen über das Unternehmensserviceportal (USP) eingebracht werden, welches ebInterface als primäres Format akzeptiert.[1]

4.1.2 Kernelemente und Struktur

Technisch gesehen ist ebInterface eine reine XML-Struktur. Im Gegensatz zu ZUGFeRD gibt es keine eingebettete PDF-Datei; die XML-Datei *ist* die Rechnung. Dies bedeutet, dass für die visuelle Darstellung (z.B. für die manuelle Prüfung durch einen Sachbearbeiter) ein Stylesheet (XSLT) oder ein Viewer notwendig ist.

Die Struktur eines ebInterface 6.1 Dokuments ist hierarchisch aufgebaut und umfasst folgende Hauptbereiche:

- **Root-Element**: `<Invoice>` definiert die Grundeigenschaften wie Währung, Sprache und Dokumententitel.
- **Header-Daten**: Hierzu zählen die eindeutige Rechnungsnummer (`InvoiceNumber`), das Rechnungsdatum (`InvoiceDate`) und der Leistungszeitraum (`Delivery`).
- **Biller (Rechnungssteller)**: Enthält detaillierte Informationen zum Leistenden, inklusive der gesetzlich vorgeschriebenen UID-Nummer (VAT Identification Number), Anschrift, Kontaktdaten und Bankverbindung.
- **InvoiceRecipient (Rechnungsempfänger)**: Analog zum Biller die Daten des Leistungsempfängers. Hier ist oft die Auftragsreferenz (Order Reference) entscheidend für die automatische Zuordnung.
- **Details (Positionen)**: Der Kern der Rechnung. Hier werden in einer Liste (`ItemList`) die einzelnen Positionen (`ListLineItem`) aufgeführt. Jede Position enthält Menge, Einheit, Einzelpreis, Beschreibung, Steuerreferenz und Zeilensumme.
- **Tax (Steuern)**: Eine Zusammenfassung der Steuerbeträge, gruppiert nach Steuersätzen. Dies ist essenziell für die Vorsteuerabzugs-Prüfung.
- **PaymentConditions**: Zahlungsziele, Skonto-Informationen und Fälligkeitsdaten.

[1] Vgl. AUSTRIAPRO: *ebInterface - Der österreichische Standard für die elektronische Rechnung*, https://www.ebinterface.at/, letzter Zugriff am 19.12.2025

4.1.3 Pflichtfelder und Validierung

Die Validität einer ebInterface-Rechnung wird durch ein XML Schema (XSD) definiert. Pflichtfelder sind jene, die laut Umsatzsteuergesetz (UStG) für eine ordnungsgemäße Rechnung erforderlich sind. Dazu gehören in Österreich unter anderem:

- Name und Anschrift des liefernden und empfangenden Unternehmers.
- Menge und handelsübliche Bezeichnung der Gegenstände.
- Tag der Lieferung oder sonstigen Leistung.
- Entgelt (Netto) und der darauf entfallende Steuerbetrag.
- Der anzuwendende Steuersatz.
- Ausstellungsdatum und fortlaufende Nummer.
- UID-Nummer des Leistenden (und ab 10.000 Euro Brutto auch des Empfängers).

Eine Besonderheit von ebInterface 6.1 ist die strikte Typisierung. Datumsfelder müssen dem ISO 8601 Format entsprechen, Beträge sind als Dezimalzahlen mit definierter Genauigkeit anzugeben. Dies verhindert Interpretationsspielräume, die bei OCR-basierten Verfahren oft zu Fehlern führen.

4.1.4 Beispielhafte XML-Struktur

Um die Struktur zu verdeutlichen, zeigt das folgende Listing einen gekürzten Ausschnitt einer validen ebInterface 6.1 Rechnung. Man erkennt deutlich die hierarchische Gliederung und die sprechenden Tag-Namen, die eine Implementierung erleichtern.

Listing 1: Ausschnitt einer ebInterface 6.1 Rechnung

```
<Invoice xmlns="http://www.ebinterface.at/schema/6p1/"
         GeneratingSystem="SmartBillConverter">
  <InvoiceNumber>2024-001</InvoiceNumber>
  <InvoiceDate>2024-01-15</InvoiceDate>
  <Delivery>
    <Date>2024-01-10</Date>
  </Delivery>
  <Biller>
    <VATIdentificationNumber>ATU12345678</VATIdentificationNumber>
    <Address>
      <Name>Musterfirma GmbH</Name>
      <Street>Hauptstrasse 1</Street>
      <Town>Wien</Town>
      <ZIP>1010</ZIP>
      <Country>Austria</Country>
    </Address>
  </Biller>
  <Details>
    <ItemList>
      <ListLineItem>
        <Description>Software Entwicklung</Description>
        <Quantity Unit="h">10.00</Quantity>
        <UnitPrice>100.00</UnitPrice>
```

```
24          <TaxItem>
25            <TaxPercent>20</TaxPercent>
26          </TaxItem>
27          <LineItemAmount>1000.00</LineItemAmount>
28        </ListLineItem>
29      </ItemList>
30    </Details>
31    <Tax>
32      <VAT>
33        <TaxedAmount>1000.00</TaxedAmount>
34        <TaxPercent>20</TaxPercent>
35        <Amount>200.00</Amount>
36      </VAT>
37    </Tax>
38    <TotalGrossAmount>1200.00</TotalGrossAmount>
39  </Invoice>
```

Dieses Beispiel illustriert, wie essenziell die korrekte Verschachtelung ist. Ein häufiger Fehler bei der Generierung ist die Diskrepanz zwischen den berechneten Zeilensummen (`LineItemAmount`) und den Steuer-Aggregaten im `Tax`-Block.

4.2 ZUGFeRD 2.3 und EN 16931 (Deutschland/EU)

4.2.1 Das hybride Konzept

ZUGFeRD (Zentraler User Guide des Forums elektronische Rechnung Deutschland) verfolgt einen anderen Ansatz als ebInterface. Es ist ein hybrides Format, das die Vorteile der menschlichen Lesbarkeit mit der maschinellen Verarbeitbarkeit kombiniert. Eine ZUGFeRD-Rechnung ist technisch gesehen eine PDF/A-3 Datei.

PDF/A-3 ist eine Erweiterung des PDF-Standards, die es erlaubt, beliebige Dateien als Anhang in das PDF einzubetten. Bei ZUGFeRD wird eine XML-Datei (genannt `factur-x.xml` oder `zugferd-invoice.xml`) in das PDF eingebettet.

- **Sichtbare Ebene**: Das PDF zeigt das Rechnungsbild, wie man es von Papier kennt.
- **Unsichtbare Ebene**: Das eingebettete XML enthält die gleichen Daten in strukturierter Form.

Dieses Konzept löst das Akzeptanzproblem bei kleineren Empfängern: Wer keine Software zur automatischen Verarbeitung hat, behandelt die Datei wie ein normales PDF. Wer automatisieren will, extrahiert das XML.

4.2.2 Profile und Konformität

ZUGFeRD 2.3 implementiert die europäische Norm EN 16931. Da die Anforderungen je nach Unternehmensgröße und Branche variieren, definiert ZUGFeRD verschiedene Profile:

1. **MINIMUM**: Enthält nur rudimentäre Daten, um eine Buchungshilfe zu bieten. Es erfüllt nicht die Anforderungen an eine steuerrechtliche Rechnung.
2. **BASIC WL (Without Lines)**: Enthält Kopfdaten und Summen, aber keine Positionsdaten. Nützlich für einfache Verbuchung, aber eingeschränkt in der Prüfung.
3. **BASIC**: Erfüllt die Anforderungen des deutschen UStG für Rechnungen unterhalb bestimmter Grenzen, ist aber eine Untermenge der EN 16931.
4. **EN 16931 (COMFORT)**: Das Standardprofil. Es bildet die europäische Norm vollständig ab und ist für den grenzüberschreitenden Verkehr sowie für B2G (Business to Government) geeignet. Dies ist das Zielformat für den *SmartBillConverter*.
5. **EXTENDED**: Enthält zusätzliche branchenspezifische Erweiterungen, die über die Norm hinausgehen.

4.2.3 CII-Struktur (Cross Industry Invoice)

Das XML-Format innerhalb von ZUGFeRD basiert auf der *Cross Industry Invoice* (CII) Syntax der UN/CEFACT. Dies ist ein globaler Standard für Lieferketten-Daten. Die Struktur ist extrem granular und tief verschachtelt. Ein einfacher Preis ist nicht nur eine Zahl, sondern ein komplexes Objekt mit Betrag, Währung, Basismenge und Einheitencode.

Beispielhafte Struktur-Tiefe in CII: `SupplyChainTradeTransaction` → `IncludedSupplyChainTra`
→ `SpecifiedLineTradeAgreement` → `NetPriceProductTradePrice` → `ChargeAmount`.
Diese Komplexität macht die Implementierung eines Mappers anspruchsvoll, da hunderte von Pfaden korrekt befüllt werden müssen, um Validierungsfehler zu vermeiden.[2] Im folgenden Listing ist ein Ausschnitt einer ZUGFeRD-XML dargestellt, der die Komplexität im Vergleich zu ebInterface verdeutlicht. Man beachte die tiefen Verschachtelungen für einfache Informationen wie den Steuerbetrag.

[2]Vgl. FeRD e.V.: *ZUGFeRD 2.3 - Das Datenformat für elektronische Rechnungen*, https://www.ferd-net.de/standards/zugferd-2.3/index.html, letzter Zugriff am 19.12.2025

Listing 2: Ausschnitt einer ZUGFeRD 2.3 (CII) Rechnung

```
<rsm:CrossIndustryInvoice
    xmlns:rsm="urn:un:unece:uncefact:data:standard:CrossIndustryInvoice:100" ...>
  <rsm:SupplyChainTradeTransaction>
    <ram:IncludedSupplyChainTradeLineItem>
      <ram:SpecifiedLineTradeAgreement>
        <ram:NetPriceProductTradePrice>
          <ram:ChargeAmount>100.00</ram:ChargeAmount>
        </ram:NetPriceProductTradePrice>
      </ram:SpecifiedLineTradeAgreement>
      <ram:SpecifiedLineTradeSettlement>
        <ram:ApplicableTradeTax>
          <ram:TypeCode>VAT</ram:TypeCode>
          <ram:CategoryCode>S</ram:CategoryCode>
          <ram:RateApplicablePercent>19.00</ram:RateApplicablePercent>
        </ram:ApplicableTradeTax>
      </ram:SpecifiedLineTradeSettlement>
    </ram:IncludedSupplyChainTradeLineItem>
    <ram:ApplicableHeaderTradeSettlement>
      <ram:SpecifiedTradeSettlementHeaderMonetarySummation>
        <ram:LineTotalAmount>100.00</ram:LineTotalAmount>
        <ram:TaxBasisTotalAmount>100.00</ram:TaxBasisTotalAmount>
        <ram:TaxTotalAmount currencyID="EUR">19.00</ram:TaxTotalAmount>
        <ram:GrandTotalAmount>119.00</ram:GrandTotalAmount>
      </ram:SpecifiedTradeSettlementHeaderMonetarySummation>
    </ram:ApplicableHeaderTradeSettlement>
  </rsm:SupplyChainTradeTransaction>
</rsm:CrossIndustryInvoice>
```

4.3 UBL vs. CII: Mapping-Überlegungen und Trade-offs

Die europäische Norm EN 16931 lässt zwei XML-Syntaxen zu: UBL (Universal Business Language) und UN/CEFACT CII. Während Netzwerke wie Peppol (Pan-European Public Procurement OnLine) historisch stark auf UBL (ISO/IEC 19845) setzen, hat sich das Forum elektronische Rechnung Deutschland (FeRD) bei ZUGFeRD für CII entschieden.

4.3.1 Strukturelle Unterschiede

UBL ist dokumentenzentriert. Es gibt eigene Schemas für `Order`, `Invoice`, `DespatchAdvice`. Die Struktur ist oft flacher und pragmatischer. **CII** ist prozesszentriert. Es versucht, alle Aspekte der Lieferkette in einem universellen Modell abzubilden. Dies führt zu einer höheren Abstraktion und Verschachtelung.

Ein konkretes Beispiel ist die Handhabung von Steuern:

- In **UBL** werden Steuern oft direkt auf Zeilenebene referenziert (`ClassifiedTaxCategory`).
- In **CII** gibt es komplexe `ApplicableTradeTax`-Strukturen, die sowohl auf Dokumentenebene (Summen) als auch auf Zeilenebene (Referenzen) konsistent gehalten werden müssen.

4.3.2 Herausforderungen für den Konverter

Für den *SmartBillConverter* bedeutet dies, dass das interne Datenmodell (das C# `Invoice`-Objekt) als "Intermediate Representation" fungieren muss. Es darf nicht zu stark an ebInterface gekoppelt sein, da sonst das Mapping auf CII extrem schwierig wird. Das Mapping von ebInterface (XML) direkt auf ZUGFeRD (CII) ist nicht trivial, da die semantischen Bäume unterschiedlich sind. ebInterface gruppiert oft logisch (z.B. "Biller"), während CII funktional gruppiert (z.B. "SupplyChainTradeTransaction"). Die Entscheidung, eine eigene interne Repräsentation zu nutzen, die von der KI befüllt wird und dann in beide Formate serialisiert werden kann, ist daher architektonisch notwendig. Tabelle 1 fasst die wesentlichen Unterschiede zusammen, die bei der Implementierung berücksichtigt wurden.

Merkmal	**ebInterface 6.1**
Basis-Standard	National (AUSTRIAPRO)
Dateiformat	Reines XML
Struktur-Tiefe	Flach bis Mittel
Steuer-Logik	Zentraler Tax-Block Header)
Pflichtfelder	Fokus auf UStG (AT)
Visualisierung	Benötigt Stylesheet

Tabelle 1: Vergleich zwischen ebInterface und ZUGFeRD

4.4 XSD-Validierung vs. Geschäftsregeln (BR-S-08 etc.)

Die Qualitätssicherung einer generierten E-Rechnung erfolgt in zwei Stufen. Ein Dokument kann technisch valide sein, aber inhaltlich falsch.

4.4.1 Syntaktische Validierung (XSD)

Die XML Schema Definition (XSD) prüft die Grammatik der Datei.

- Sind alle Pflicht-Tags vorhanden?
- Haben Datumsfelder das Format YYYY-MM-DD?
- Sind numerische Werte korrekt formatiert?
- Stimmt die Reihenfolge der Elemente? (Besonders in CII ist die Reihenfolge strikt vorgegeben).

Ein Verstoß gegen das XSD führt dazu, dass die Datei vom Empfängersystem meist gar nicht erst eingelesen werden kann ("Technical Rejection").

4.4.2 Semantische Validierung (Business Rules)

Selbst wenn das XML wohlgeformt ist, kann der Inhalt unlogisch sein. Die EN 16931 definiert daher eine Liste von Geschäftsregeln (Business Rules), die eingehalten werden müssen. Diese werden oft mittels *Schematron* geprüft.

Ein prominentes und im Projektverlauf problematisches Beispiel ist die Regel **BR-S-08** (Value Added Tax Breakdown). Diese Regel besagt: *Für jeden unterschiedlichen Steuercode und Steuersatz, der in den Rechnungspositionen verwendet wird, muss genau eine Zusammenfassung auf Dokumentenebene existieren, und die Summe der Steuerbeträge muss rechnerisch korrekt sein.*

Das Problem bei der Generierung durch KI (LLMs) ist, dass LLMs statistische Modelle sind, keine Taschenrechner. Ein LLM kann problemlos schreiben:

- Position 1: 100 Euro, 20% MwSt
- Position 2: 200 Euro, 20% MwSt
- Steuer-Summe: 55 Euro (statt korrekt 60 Euro)

Das LLM "schätzt" oder "halluziniert" die Summe oft, anstatt sie zu berechnen. Daher darf die Berechnung der Summen und Steuern **niemals** der KI überlassen werden. Die KI extrahiert die Einzelwerte (Menge, Preis, Steuersatz), aber die Aggregation und die Erstellung des `TaxBreakdown` für die Regel BR-S-08 muss durch deterministische Algorithmen im Backend (C#) erfolgen. Nur so kann die Validierung gegen Schematron-Regeln bestanden werden.[3]

[3]Vgl. CEN - European Committee for Standardization: *EN 16931-1:2017 Electronic invoicing - Semantic data model*, https://standards.cen.eu/dyn/www/f?p=204:110:0::::FSP_PROJECT:60602&cs=1B61B766636F9FB34B7DBD72CE9026C72, letzter Zugriff am 19.12.2025

5 Dokumentenverarbeitung und KI-Grundlagen

Die automatisierte Verarbeitung von Rechnungen ist ein klassisches Problem der Informatik, das durch moderne KI-Ansätze neu gelöst wird. Traditionelle Ansätze basierten auf Templates (Schablonen), bei denen für jeden Lieferanten definiert wurde, an welchen Koordinaten die Rechnungsnummer steht. Dieser Ansatz ist jedoch fragil und skaliert nicht. Der *SmartBillConverter* setzt daher auf einen generischen Ansatz mittels Large Language Models (LLMs). Dieses Kapitel beleuchtet die technologischen Grundlagen.

5.1 PDFs und Textextraktion

5.1.1 Die Natur des PDF-Formats

Das Portable Document Format (PDF) wurde von Adobe entwickelt, um Dokumente geräteunabhängig darzustellen. Es ist ein Seitenbeschreibungssprache. Das bedeutet, ein PDF speichert primär Anweisungen wie "Zeichne den Buchstaben "A" an Koordinate (100, 200) in Schriftart Helvetica". Es speichert *nicht* notwendigerweise die Information, dass dieses "A" Teil des Wortes "Rechnung" ist oder dass dieses Wort Teil einer Tabelle ist.

Für die Textextraktion ergeben sich daraus massive Probleme:

- **Verlust der Lesereihenfolge**: In einem PDF-Stream können die Zeichenbefehle in beliebiger Reihenfolge stehen. Ein zweispaltiger Text kann im Stream so gespeichert sein, dass erst die erste Zeile der linken Spalte, dann die erste Zeile der rechten Spalte kommt. Ein naiver Extraktor liest dann "Rechnungs Datum: 01.01.2024" als "Rechnungs 01.01.2024 Datum:".

- **Fehlende Wortgrenzen**: Oft werden Wörter nicht als String gespeichert, sondern jeder Buchstabe einzeln positioniert (Kerning). Leerzeichen sind oft gar keine Zeichen, sondern einfach Lücken in den Koordinaten.
- **Encoding-Probleme**: Manchmal nutzen PDFs benutzerdefinierte Encodings, sodass der Buchstabe "A" im Code als "X" gespeichert ist, aber visuell als "A" dargestellt wird. Ohne korrekte ToUnicode-Map ist nur "Datensalat" extrahierbar.

5.1.2 Lösungsansatz mit PdfPig

Im Projekt wird die Bibliothek *PdfPig* eingesetzt. Sie versucht, die logische Struktur aus den geometrischen Informationen zu rekonstruieren. PdfPig analysiert die "Bounding Boxes" aller Buchstaben. Durch Heuristiken (Abstandsanalyse) werden Buchstaben zu Wörtern und Wörter zu Zeilen zusammengefügt. Ein spezifisches Problem bei Rechnungen sind Tabellen. Da Tabellenlinien im PDF nur Vektorgrafiken sind und keine logische Verbindung zum Text haben, ist es für Algorithmen schwer zu erkennen, welche Zahl zu welcher Spalte gehört, besonders wenn Spalten rechtsbündig sind und der Textabstand variiert. Dennoch liefert die direkte Extraktion aus dem PDF-Stream (wenn möglich) immer präzisere Daten als der Umweg über OCR, da keine Zeichenverwechslungen (z.B. "8" vs "B") auftreten können.[4]

5.2 OCR mit Tesseract

Wenn Rechnungen nicht als digitales PDF, sondern als Scan (Rastergrafik) vorliegen, greift die Textextraktion ins Leere. Hier ist Optical Character Recognition (OCR) erforderlich.

5.2.1 Funktionsweise von Tesseract

Tesseract ist eine der führenden Open-Source-OCR-Engines, ursprünglich von HP entwickelt und heute von Google gepflegt. Moderne Versionen (ab 4.0) basieren auf LSTM (Long Short-Term Memory) neuronalen Netzen, einer Form von Recurrent Neural Networks (RNNs), die besonders gut für Sequenzdaten wie Text geeignet sind. Tesseract analysiert das Bild in mehreren Schritten: 1. **Layout Analysis**: Finden

[4]Vgl. UglyToad: *PdfPig - Read and extract text and other content from PDFs in C# (port of PdfBox)*, https://github.com/UglyToad/PdfPig, letzter Zugriff am 19.12.2025

von Textblöcke und Zeilen. 2. **Baseline Fitting**: Erkennen der Grundlinie einer Textzeile (wichtig bei schiefen Scans). 3. **Character Recognition**: Das neuronale Netz klassifiziert kleine Bildausschnitte als Zeichen. 4. **Word Recognition**: Wörterbuch-Abgleich, um die Wahrscheinlichkeit von Wortkombinationen zu bewerten.

5.2.2 Einflussfaktoren auf die Qualität

Die Qualität der OCR hängt massiv von der Vorverarbeitung ab:

- **Auflösung**: Unter 300 DPI sinkt die Erkennungsrate drastisch.
- **Binarisierung**: Die Umwandlung von Graustufen in reines Schwarz-Weiß (Thresholding) muss adaptiv erfolgen, um Schatten oder ungleichmäßige Beleuchtung auszugleichen.
- **Deskewing**: Schon eine Drehung um 1-2 Grad kann die Zeilenerkennung stören. Das Bild muss rechnerisch gerade gerückt werden.

Im Projektkontext werden für Tesseract spezifische Sprachpakete (`deu.traineddata`) geladen, um deutsche Umlaute und typische Vokabeln korrekt zu erkennen. Dennoch bleibt OCR fehleranfällig, weshalb die nachgelagerte KI-Korrektur essenziell ist.

5.3 LLMs für Informationsextraktion

Der Paradigmenwechsel in der Dokumentenverarbeitung kommt durch Large Language Models (LLMs). Anstatt Regeln zu programmieren ("Suche nach "Rechnungs-Nr." und nimm das Wort rechts daneben"), übergibt man dem LLM den gesamten unstrukturierten Text und bittet es, die Daten zu strukturieren.

5.3.1 Architektur und Funktionsweise

LLMs basieren auf der Transformer-Architektur. Sie wurden auf riesigen Textmengen trainiert und haben gelernt, statistische Zusammenhänge zwischen Wörtern (Tokens) vorherzusagen. Für die Extraktion ist die Fähigkeit des "In-Context Learning" entscheidend. Das Modell versteht den Kontext einer Rechnung. Es weiß, dass eine IBAN meist in der Nähe von "Bankverbindung" steht und wie eine IBAN aussieht, ohne dass man ihm einen Regex geben muss. Es kann Synonyme auflösen: Egal ob auf der Rechnung

"Total", "Gesamtbetrag", "Zahlbetrag" oder "Summe" steht, das LLM kann es auf das Feld `TotalAmount` mappen.

5.3.2 Modell-Vergleich und Evaluation

Im Rahmen der Entwicklung wurden verschiedene Modelle evaluiert (siehe Projektdokumentation):

- **Gemini 2.5 Flash**: Ein sehr schnelles und kosteneffizientes Modell von Google. Es zeigte im Projekt die beste Balance aus Geschwindigkeit und JSON-Konformität. Es hat ein großes Kontextfenster, was für lange Rechnungen wichtig ist.
- **Mistral (verschiedene Größen)**: Open-Source-Modelle, die lokal oder via API laufen können. Während große Modelle (Mistral Large) gut performen, neigen kleinere Modelle (7B) dazu, das JSON-Schema zu verletzen oder komplexe Tabellen zu halluzinieren.
- **Qwen**: Ein starkes Modell, das jedoch im Test oft Prompt-Logging und Training erforderte, was Datenschutzbedenken aufwirft.

5.3.3 Risiken: Halluzinationen

Das größte Risiko bei LLMs ist die Halluzination. Das Modell ist darauf trainiert, eine "plausible Fortsetzung" des Textes zu generieren. Wenn auf der Rechnung kein Lieferdatum steht, das JSON-Schema aber ein Lieferdatum verlangt, "erfindet" das Modell oft einfach eines (z.B. das Rechnungsdatum), um den User zufrieden zu stellen. Dies ist fatal für eine Finanzanwendung. Daher muss der Prompt so gestaltet sein, dass das Modell explizit "null" oder "nicht vorhanden" ausgibt, anstatt zu raten.

5.4 Prompt-Design für deterministische Ausgabe

Prompt Engineering ist die Kunst, die Eingabe so zu formulieren, dass das Modell das gewünschte Ergebnis liefert. Für die API-basierte Extraktion ist Determinismus das Ziel.

5.4.1 Techniken

- **System Prompting**: Dem Modell wird eine klare Rolle zugewiesen: "Du bist ein strikter Datenextraktions-Assistent. Du antwortest nur mit JSON. Du fügst keine Erklärungen hinzu."
- **JSON Mode / Function Calling**: Moderne APIs (wie OpenAI oder Gemini) bieten Modi an, die garantieren, dass der Output valides JSON ist. Das Modell wird gezwungen, Tokens zu generieren, die der Syntax entsprechen.
- **Schema Injection**: Das gewünschte JSON-Schema wird im Prompt mitgegeben. "Extrahiere die Daten in folgendes Format: { "invoiceNumber": "string", ... }". **Chain-of-Thought Unterdrückung**: Während bei Logikaufgaben "Denk nach Schritt für Schritt" hilft, ist es bei der Extraktion oft hinderlich, da es den Output "verunreinigt". Wir wollen nur die Daten.

Ein Beispiel für ein solches JSON-Schema, wie es im *SmartBillConverter* verwendet wird, zeigt Listing 3. Es definiert strikte Typen für die Extraktion.

Listing 3: JSON-Schema für die KI-Extraktion

```
{
  "type": "object",
  "properties": {
    "invoiceNumber": { "type": "string" },
    "invoiceDate": { "type": "string", "format": "date" },
    "totalAmount": { "type": "number" },
    "currency": { "type": "string", "enum": ["EUR", "USD"] },
    "items": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "description": { "type": "string" },
          "quantity": { "type": "number" },
          "unitPrice": { "type": "number" },
          "taxRate": { "type": "number" }
        }
      }
    }
  },
  "required": ["invoiceNumber", "invoiceDate", "totalAmount", "items"]
}
```

Ein spezifisches Problem im Projekt war die Konsistenz. Manchmal lieferte das gleiche Modell beim gleichen Prompt leicht unterschiedliche Ergebnisse (z.B. Datumsformat mal YYYY-MM-DD, mal DD.MM.YYYY). Dies muss durch strikte Anweisungen ("Formatiere alle Daten als ISO 8601") und niedrige Temperature-Einstellungen (Temperature = 0) minimiert werden.

5.5 Datenschutz und Sicherheit in KI-APIs

Die Nutzung von Cloud-basierten KI-APIs (wie Google Vertex AI oder OpenAI API) für Rechnungsdaten wirft Datenschutzfragen auf, da Rechnungen personenbezogene Daten (Namen, Adressen) und Geschäftsgeheimnisse (Preise, Lieferantenbeziehungen) enthalten.

5.5.1 Risikoanalyse

- **Training auf Kundendaten**: Das größte Risiko ist, dass der Anbieter (Google, OpenAI) die hochgeladenen Rechnungen nutzt, um seine Modelle zu trainieren. Dies könnte dazu führen, dass das Modell in Zukunft Informationen aus diesen Rechnungen "weiß". Enterprise-Verträge schließen dies meist explizit aus ("Zero Data Retention" für Training).
- **Datenübertragung**: Die Daten verlassen das geschützte Unternehmensnetzwerk. Eine Ende-zu-Ende-Verschlüsselung (TLS 1.3) ist Standard, aber der API-Endpunkt entschlüsselt die Daten zur Verarbeitung.
- **Serverstandort**: Nach DSGVO sollten personenbezogene Daten idealerweise den Europäischen Wirtschaftsraum (EWR) nicht verlassen. Bei US-Anbietern ist auf Angemessenheitsbeschlüsse (Data Privacy Framework) zu achten.

5.5.2 Lokale Alternativen

Um diese Risiken zu eliminieren, wäre der Einsatz lokaler LLMs (z.B. Llama 3, Mistral) via Ollama oder LM Studio eine Option. Diese laufen auf der eigenen Hardware (On-Premise). Der Nachteil ist der hohe Ressourcenbedarf (GPU-VRAM) und die oft schlechtere Leistung im Vergleich zu den riesigen Cloud-Modellen. Ein lokales 7B-Modell hat oft Schwierigkeiten mit komplexen, mehrseitigen Rechnungen, die ein großes Kontextfenster benötigen. Für den *SmartBillConverter* wurde aufgrund der Qualität und Entwicklungsgeschwindigkeit auf Cloud-APIs gesetzt, wobei im Produktionsbetrieb auf Enterprise-Lizenzen mit entsprechenden Datenschutzgarantien gewechselt werden müsste.

6 Anforderungsanalyse Frontend

Die Entwicklung einer benutzerfreundlichen Weboberfläche ist ein zentraler Erfolgsfaktor für die Akzeptanz des *SmartBillConverter*. Während das Backend die komplexen Aufgaben der Dokumentenverarbeitung und Format-Konvertierung übernimmt, muss das Frontend eine intuitive Schnittstelle bieten. Dieses Kapitel dokumentiert die systematische Analyse der Frontend-Anforderungen, die Modellierung der Datenstrukturen und die Benutzerinteraktionen.

6.1 Definition der Systemanforderungen (Frontend)

6.1.1 Funktionale Anforderungen

Die funktionalen Anforderungen an das Frontend definieren die Kernfunktionen des Systems:

- **Dokumenten-Upload**: Unterstützt PDF-Dateien und Bildformate (PNG, JPEG, BMP, TIFF) per Drag-and-Drop oder Dateiauswahl. Mehrere Dateien können gleichzeitig hochgeladen werden (bis zu 10 MB pro Datei). Der Upload-Fortschritt wird angezeigt. Falsche Dateiformate werden mit einer Fehlermeldung abgelehnt. Nutzt HTML5 File API und FormData.
- **Format-Auswahl**: Auswahl zwischen ebInterface 6.1 (Österreich) und ZUGFeRD 2.3 (Deutschland) über Radio-Buttons. Tooltips erklären die Unterschiede zwischen den Formaten.
- **Fortschrittsanzeige**: Zeigt den Verarbeitungsstatus mit Progress Bar (0-100%) und farbigen Status-Badges (wartend, verarbeitend, abgeschlossen, fehlgeschlagen). Grün bedeutet Erfolg, rot bedeutet Fehler, gelb bedeutet Warnung. Bei langen Verarbeitungen wird ein Spinner angezeigt.

- **Dokumenten-Vorschau**: PDFs werden mit ng2-pdf-viewer angezeigt (mit Zoom und Seitennavigation). Bilder werden automatisch skaliert. Das generierte XML wird mit Syntax-Highlighting angezeigt.
- **Download-Funktionalität**: XML-Dateien können einzeln oder als Batch heruntergeladen werden. Dateinamen folgen dem Schema `invoice_12345_ebInterface.xml`. Der Download nutzt Blob-URLs. Mehrere Dateien werden als ZIP-Datei gebündelt.
- **Fehlerbehandlung**: Dateien werden vor dem Upload validiert (Größe, Format). Bei Fehlern werden verständliche Fehlermeldungen angezeigt. Netzwerkfehler können mit einem Retry-Button wiederholt werden.

6.1.2 Nicht-funktionale Anforderungen

Diese Anforderungen definieren die Qualität und Benutzerfreundlichkeit der Anwendung:

- **Benutzerfreundlichkeit**: Die Bedienung soll selbsterklärend sein. Ein neuer Benutzer soll den Upload-Workflow in unter 60 Sekunden verstehen. Fehleingaben werden durch Validierung verhindert.[5]
- **Geschwindigkeit**: Die Seite soll in unter 2 Sekunden laden. Benutzerinteraktionen sollen unter 100ms reagieren.
- **Geräteunterstützung**: Die Anwendung ist primär für Desktop optimiert, funktioniert aber auch auf Tablets und Smartphones.
- **Browser-Unterstützung**: Die Anwendung funktioniert in Chrome, Firefox, Safari und Edge (aktuelle Versionen).[6]

6.2 Entity-Relationship-Diagramm (Datenmodell-Sicht)

Das Entity-Relationship-Diagramm modelliert die zentrale Datenstruktur des Systems. Die `Invoice`-Entität repräsentiert eine verarbeitete Rechnungsdatei mit allen extrahierten Kerninformationen:

[5] Vgl. Nielsen Norman Group: *Drag and Drop: How to Design for Ease of Use*, https://www.nngroup.com/articles/drag-drop/, letzter Zugriff am 19.12.2025

[6] Vgl. W3C: *Web Content Accessibility Guidelines (WCAG) 2.1*, https://www.w3.org/TR/WCAG21/, letzter Zugriff am 19.12.2025

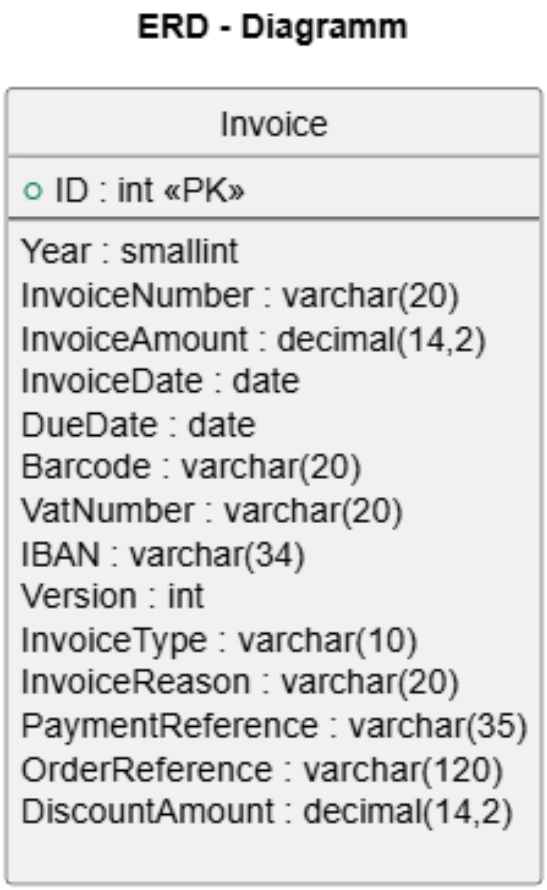

Abbildung 1: Entity-Relationship-Diagramm der Invoice-Entität

6.2.1 Detaillierte Attributbeschreibung

Das Datenmodell folgt den gesetzlichen Anforderungen des österreichischen UStG. Die einzelnen Attribute der `Invoice`-Entität haben folgende Bedeutung:

- **ID** (`int`, Primary Key): Eindeutige ID für jede Rechnung in der Datenbank. Wird automatisch hochgezählt.
- **Year** (`smallint`): Rechnungsjahr, extrahiert aus dem Rechnungsdatum. Wird für Jahresabfragen und Archivierung verwendet.
- **InvoiceNumber** (`varchar(20)`): Eindeutige Rechnungsnummer des Rechnungsstellers. Gemäß § 11 Abs. 1 Z 2 UStG muss jede Rechnung eine fortlaufende Nummer enthalten. Unterstützt Formate wie `RE-2025-00142` oder `INV/2025/0001`.
- **InvoiceAmount** (`decimal(14,2)`): Bruttorechnungsbetrag inklusive Umsatzsteuer. Mit 14 Stellen und 2 Nachkommastellen können Beträge bis zu 999.999.999.999,99 EUR gespeichert werden.
- **InvoiceDate** (`date`): Rechnungsdatum, an dem die Rechnung ausgestellt wurde. Wird für die Fälligkeitsberechnung und die Zuordnung zu Steuerperioden verwendet.
- **DueDate** (`date`): Fälligkeitsdatum, bis zu dem die Zahlung erwartet wird. Wird aus dem Zahlungsziel berechnet (z.B. „Zahlbar innerhalb 14 Tagen“). Wichtig für Mahnungen und Liquiditätsplanung.
- **VatNumber** (`varchar(20)`): Umsatzsteuer-Identifikationsnummer (UID) des Rechnungsstellers im Format `ATU12345678`. Muss bei EU-Lieferungen angegeben werden (§ 11 Abs. 1 Z 5 UStG).
- **IBAN** (`varchar(34)`): Bankkontonummer für Überweisungen. Maximal 34 Zeichen nach IBAN-Standard. Beispiel: `AT611904300234573201`.
- **Version** (`int`): Versionsnummer der Rechnung für Änderungsnachverfolgung. Beginnt bei 1 und wird bei jeder Korrektur erhöht.
- **InvoiceType** (`varchar(10)`): Rechnungstyp, z.B. `INVOICE` (Rechnung), `CREDIT` (Gutschrift) oder `ADVANCE` (Anzahlungsrechnung). Wichtig für die Buchhaltung.
- **PaymentReference** (`varchar(35)`): Zahlungsreferenz für die Zuordnung von Zahlungseingängen. Beispiel: `RF18539007547034`.

- **DiscountAmount** (`decimal(14,2)`): Skontobetrag bei vorzeitiger Zahlung. Oft 2-3% bei Zahlung innerhalb von 7-10 Tagen.

Diese Datenstruktur bildet die Grundlage für die persistente Speicherung aller verarbeiteten Rechnungen und erlaubt schnelle Abfragen, Reporting und Archivierung.

Im Frontend wird ein TypeScript-Interface verwendet, das die Backend-Entität widerspiegelt:

Listing 4: TypeScript Invoice-Interface

```typescript
export interface Invoice {
    id: number;
    invoiceNumber: string;
    invoiceAmount: number;
    invoiceDate: string;
    dueDate: string;
    vatNumber: string;
    iban: string;

    // UI-spezifische Felder
    ebInterfaceXml?: string;
    isValidXml?: boolean;
}
```

6.3 Use-Case-Diagramm (Benutzerinteraktionen)

Use-Case-Diagramme visualisieren die Interaktionen zwischen Benutzern und dem System. Das folgende Diagramm zeigt die Hauptanwendungsfälle:

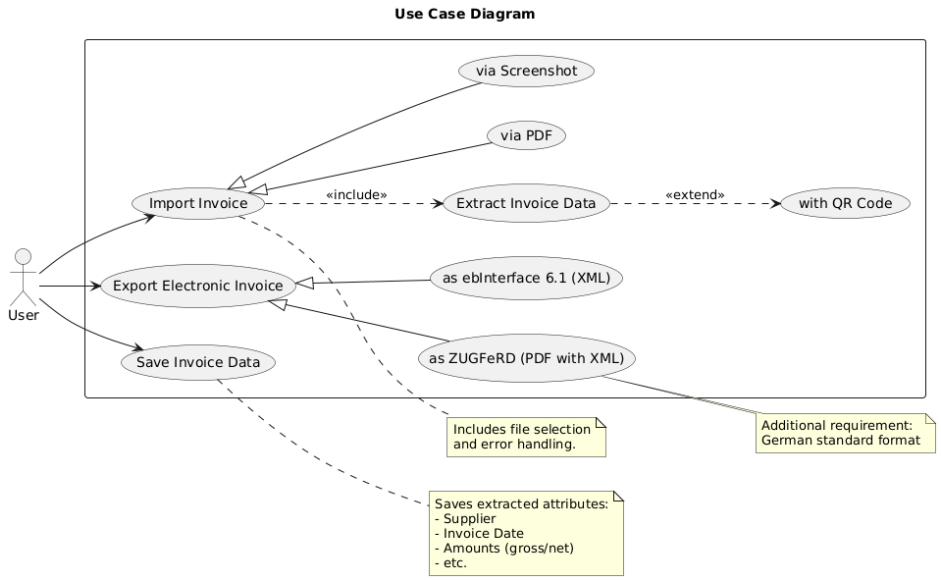

Abbildung 2: Use-Case-Diagramm des SmartBillConverter

6.3.1 Beschreibung der Use-Cases

Die Hauptanwendungsfälle werden im Folgenden detailliert beschrieben:

UC1: Import Invoice – Der primäre Use-Case ermöglicht das Hochladen von Rechnungsdokumenten. Der Benutzer wählt zwischen zwei Importvarianten:

- **UC1.1: via PDF** (`«extend»`): Import einer digitalisierten PDF-Rechnung. Dies ist der Standardfall für elektronisch erstellte Rechnungen. Das System extrahiert Text mittels PDF-Parser (PDF.js) ohne OCR-Verarbeitung.
- **UC1.2: via Screenshot** (`«extend»`): Import eines gescannten Bildes (PNG, JPEG, BMP, TIFF). Für diese Variante ist eine OCR-Verarbeitung (Tesseract) notwendig, um den Text aus dem Bild zu extrahieren.

UC2: Extract Invoice Data (`«include»`) – Dieser Use-Case ist in UC1 eingebettet und wird automatisch nach dem Import ausgeführt. Die Extraktion erfolgt in mehreren Schritten: (1) Text-Extraktion (PDF-Parser oder OCR), (2) Normalisierung durch LLM-basierte Analyse (Gemini/ChatGPT), (3) Extraktion strukturierter Daten (Rechnungsnummer, Betrag, Datum, etc.), (4) Validierung der extrahierten Daten gegen Geschäftsregeln.

UC3: Save Invoice Data – Nach erfolgreicher Extraktion werden die Rechnungsdaten in der PostgreSQL-Datenbank persistiert. Dieser Use-Case umfasst: (1) Validierung der Vollständigkeit aller Pflichtfelder, (2) Prüfung auf Duplikate anhand der Rechnungsnummer, (3) Speicherung der Invoice-Entität mit allen Attributen, (4) Rückgabe der generierten Datenbank-ID an das Frontend.

UC4: Export Electronic Invoice – Der finale Use-Case generiert die standardisierte elektronische Rechnung. Der Benutzer wählt das Zielformat:

- **UC4.1: as ebInterface 6.1 (XML)** (`«extend»`): Export als reine XML-Datei nach österreichischem ebInterface-Standard. Die XML-Struktur folgt dem offiziellen XSD-Schema des Bundesrechenzentrums.
- **UC4.2: as ZUGFeRD (PDF with XML)** (`«extend»`): Export als hybrides PDF/A-3-Dokument mit eingebetteter XML-Datei nach ZUGFeRD 2.3-Standard. Das sichtbare PDF entspricht der Original-Rechnung, während die maschinenlesbare XML-Datei im PDF eingebettet ist.

Der gesamte Workflow ist auf Einfachheit und Benutzerfreundlichkeit optimiert: Format wählen → Datei hochladen → Automatische Verarbeitung abwarten → XML herunterladen. Die durchschnittliche Bearbeitungszeit beträgt 5-15 Sekunden pro Rechnung, abhängig von der Dokumentenkomplexität.

6.4 Systemarchitektur (Frontend-Sicht)

Die Systemarchitektur folgt einer klassischen Drei-Schichten-Architektur mit klarer Trennung der Verantwortlichkeiten:

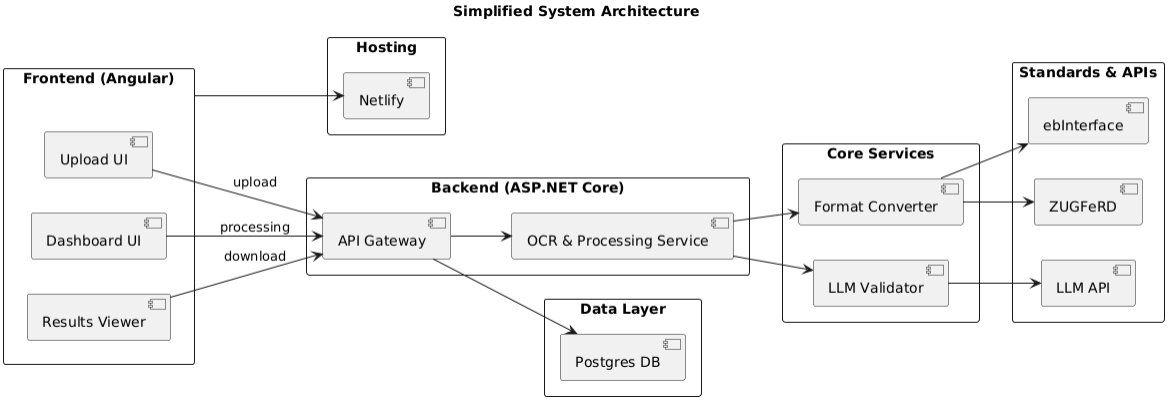

Abbildung 3: Systemarchitektur des SmartBillConverter

Die Architektur besteht aus drei Schichten: (1) **Präsentationsschicht**: Angular-Webseite im Browser, (2) **Anwendungsschicht**: ASP.NET Core Web API mit Verarbeitungslogik, (3) **Datenschicht**: PostgreSQL-Datenbank. Der `InvoiceService` im Frontend verwaltet alle HTTP-Anfragen und nutzt RxJS Observables für asynchrone Datenverarbeitung.[7] Der Ablauf: Benutzer lädt Datei hoch → Frontend prüft und sendet an Backend → Backend extrahiert Text (PDF-Parser oder OCR) und normalisiert mit KI → XML wird erstellt und zurückgesendet → Nutzer lädt XML herunter.

[7]Vgl. Angular Documentation: *Observables in Angular*, https://angular.io/guide/observables-in-angular, letzter Zugriff am 19.12.2025

7 Frontend-Architektur & Technologie-Stack

Die Wahl des richtigen Technologie-Stacks ist wichtig für den Projekterfolg. Dieses Kapitel dokumentiert die Architekturentscheidungen und die technologische Grundlage des *SmartBillConverter*-Frontends.

7.1 Projektinitialisierung mit Angular CLI

7.1.1 Framework-Evaluation

Die Wahl fiel auf Angular 19 aus folgenden Gründen:

- **Firmenanforderungen**: Das Partnerunternehmen gab keine konkrete Framework-Vorgabe, sondern ermöglichte eine freie Technologiewahl für das Frontend.
- **Schulische Vorkenntnisse**: Angular wurde im laufenden Schuljahr ausführlich behandelt, wodurch eine solide Wissensbasis vorhanden war.
- **Backend-Kompatibilität**: Das vorgegebene C#-Backend harmoniert gut mit Angular, da beide auf TypeScript bzw. stark typisierten Sprachen basieren.
- **Webapplikations-Eignung**: Für die geforderte Single-Page-Webapplikation bietet Angular eine ausgereifte Lösung mit integriertem Routing, HTTP-Client und Dependency Injection.

7.1.2 Projektsetup

Die Initialisierung erfolgte über die Angular CLI:

Listing 5: Angular-Projektgenerierung

```
1 ng new smart-bill-ui --routing --style=css --strict
```

```
cd smart-bill-ui
ng serve
```

Der Befehl `ng serve` startet einen Entwicklungsserver mit Hot Module Replacement. Für Produktion wird ein optimierter Build mit AOT-Kompilierung, Tree Shaking und Minification erstellt.

7.2 Auswahl der Bibliotheken

7.2.1 Bootstrap 5.3.7

Bootstrap wurde als CSS-Framework gewählt, weil es im schulischen Kontext bereits verwendet wurde und somit Vorkenntnisse im Umgang mit dem Grid-System, UI-Komponenten und Utility-Klassen vorhanden waren. Dies beschleunigte die Frontend-Entwicklung erheblich.[8]

```
npm install bootstrap@5.3.7
```

Die Integration erfolgte in `styles.css`:

```
@import 'bootstrap/dist/css/bootstrap.min.css';
```

7.2.2 ng2-pdf-viewer 10.4.0

Für die PDF-Vorschau wurde `ng2-pdf-viewer` gewählt. Bei der Suche nach einer einfachen Lösung zur Anzeige von PDFs im Browser erwies sich diese Bibliothek als geeignet, da sie direkt für Angular entwickelt wurde und eine unkomplizierte Integration ermöglicht.[9]

```
npm install ng2-pdf-viewer@10.4.0
```

Template-Integration:

```
<pdf-viewer [src]="pdfData" [render-text]="true"></pdf-viewer>
```

7.3 Projektstruktur

Angular folgt einer modularen Architektur mit Komponenten, Services und Routing:

[8] Vgl. Bootstrap Team: *Bootstrap 5 Documentation*, https://getbootstrap.com/docs/5.3/, abgerufen am 15.01.2025

[9] Vgl. Mozilla Foundation: *PDF.js Documentation*, https://mozilla.github.io/pdf.js/, abgerufen am 15.01.2025

Listing 6: Projektstruktur smart-bill-ui

```
src/app/
  components/
    upload/
      upload.component.ts        # Datei-Upload-Logik
      upload.component.html      # Upload-UI mit Drag & Drop
    invoice-list/
      invoice-list.component.ts # Rechnungsuebersicht
    processing/
      processing.component.ts    # Verarbeitungsstatus
  services/
    invoice.service.ts           # HTTP-Kommunikation
  models/
    invoice.model.ts             # TypeScript-Interfaces
  app.routes.ts                  # Routing-Konfiguration
  app.config.ts                  # Provider-Setup
```

7.3.1 Komponenten-Architektur

Die Anwendung besteht aus drei Hauptkomponenten:

- **UploadComponent**: Datei-Upload via Drag & Drop oder File-Input. Unterstützt Multi-File-Upload und zeigt Fortschrittsbalken. Validiert Dateitypen (PDF, PNG, JPEG) und Größenlimits.
- **InvoiceListComponent**: Zeigt Liste der hochgeladenen Rechnungen mit Status (pending, processing, completed, error). Erlaubt Download von ebInterface/ZUGFeRD-XML.
- **ProcessingComponent**: Zeigt Verarbeitungsstatus und Fehler. Stellt OCR-Ergebnisse und extrahierte Daten dar.

7.3.2 Services und Dependency Injection

Der `InvoiceService` verwaltet die HTTP-Kommunikation mit dem Backend:

Listing 7: Invoice Service

```
@Injectable({ providedIn: 'root' })
export class InvoiceService {
  private apiUrl = 'http://localhost:5000/api';

  constructor(private http: HttpClient) {}

  uploadInvoice(file: File): Observable<UploadResponse> {
    const formData = new FormData();
    formData.append('file', file);
    return this.http.post<UploadResponse>(
      `${this.apiUrl}/upload`, formData
    );
  }

  getInvoices(): Observable<Invoice[]> {
    return this.http.get<Invoice[]>(`${this.apiUrl}/invoices`);
  }
}
```

Der Service funktioniert folgendermaßen:

- **@Injectable**: Macht den Service für andere Komponenten verfügbar. `providedIn: 'root'` bedeutet, dass es nur eine Instanz im gesamten Projekt gibt.
- **apiUrl**: Speichert die Backend-Adresse (`http://localhost:5000/api`). Alle Anfragen gehen an diese URL.
- **constructor**: Bekommt den `HttpClient` automatisch von Angular bereitgestellt. Damit können HTTP-Anfragen (GET, POST) gesendet werden.
- **uploadInvoice**: Erstellt ein `FormData`-Objekt (benötigt für Datei-Uploads), hängt die Datei an und sendet sie per POST an `/api/upload`. Gibt ein `Observable` zurück, das später die Antwort vom Server liefert.
- **getInvoices**: Holt alle Rechnungen vom Backend per GET-Anfrage an `/api/invoices`. Gibt ein `Observable` zurück, das ein Array von Rechnungen enthält.

Das `Observable`-Pattern ermöglicht asynchrone Datenverarbeitung: Die Komponente abonniert das Observable mit `.subscribe()` und erhält die Daten, sobald die Server-Antwort eingetroffen ist.

7.4 Routing-Konfiguration

Das Routing definiert, welche Komponente bei welcher URL angezeigt wird:

Listing 8: Routing in app.routes.ts

```
1 export const routes: Routes = [
2   { path: '', redirectTo: '/upload', pathMatch: 'full' },
3   { path: 'upload', component: UploadComponent },
4   { path: 'invoices', component: InvoiceListComponent },
5   { path: 'processing/:id', component: ProcessingComponent },
6   { path: '**', redirectTo: '/upload' }
7 ];
```

Die Routen erlauben:

- `/upload`: Zeigt die Upload-Seite an (Standardroute). Wenn jemand auf die Startseite geht (`/`), wird automatisch auf `/upload` weitergeleitet.
- `/invoices`: Zeigt die Rechnungsübersicht mit allen hochgeladenen Rechnungen.
- `/processing/:id`: Zeigt den Verarbeitungsstatus einer bestimmten Rechnung. Die `:id` ist ein Platzhalter, z.B. `/processing/123` zeigt den Status von Rechnung 123.

- `**`: Wildcard-Route, die bei ungültigen URLs greift. Leitet zurück auf `/upload`, falls jemand eine nicht existierende Seite aufruft.

7.4.1 Navigation

Die Navigation kann auf zwei Arten erfolgen:

Programmatisch im TypeScript-Code:

```
constructor(private router: Router) {}

navigateToProcessing(invoiceId: number): void {
  this.router.navigate(['/processing', invoiceId]);
}
```

Hier wird der `Router`-Service verwendet, um per Code zu einer anderen Seite zu wechseln. Die Methode `navigate()` bekommt ein Array mit dem Pfad und den Parametern (z.B. `['/processing', 123]` wird zu `/processing/123`).

Im HTML-Template mit routerLink:

```
<a routerLink="/invoices" routerLinkActive="active">
  Rechnungen
</a>
```

Die `routerLink`-Direktive erstellt einen anklickbaren Link. `routerLinkActive="active"` fügt automatisch die CSS-Klasse `active` hinzu, wenn der Benutzer auf dieser Seite ist (nützlich für Navigation-Highlighting).

8 Implementierung der Upload-Komponente und Multi-File-System

Die Upload-Komponente ist das Hauptelement der SmartBillConverter-Anwendung. Sie steuert den gesamten Prozess vom Hochladen der Dateien bis zur Erstellung der XML-Dateien.

8.1 Entwicklung der Upload-Komponente mit Drag-and-Drop

Die Upload-Komponente wurde als eigenständige Angular-Komponente entwickelt. Sie nutzt die neue `@for`- und `@if`-Syntax von Angular 19 und `@ViewChild`, um auf HTML-Elemente im Template zuzugreifen.

8.1.1 Aufbau der Komponente

Die Komponente wird mit dem `@Component`-Decorator konfiguriert:

Listing 9: Upload-Component Decorator-Konfiguration

```
@Component({
  selector: 'app-upload',
  standalone: true,
  imports: [CommonModule, PdfViewerModule, HttpClientModule],
  providers: [InvoiceService],
  templateUrl: './upload.component.html',
  styleUrl: './upload.component.css'
})
export class UploadComponent {
  @ViewChild('fileInput') fileInput!: ElementRef<HTMLInputElement>;
  selectedFiles: FileUploadItem[] = [];
  selectedFormat: 'ebInterface' | 'ZUGFeRD' | null = null;
  isDragOver = false;
  isProcessing = false;

  constructor(private invoiceService: InvoiceService) {}
}
```

Der Parameter `standalone: true` bedeutet, dass die Komponente unabhängig funktioniert und nicht in ein zusätzliches Modul eingebunden werden muss. Das macht den Code übersichtlicher.

8.1.2 Drag-and-Drop-Funktion

Für das Hochladen per Drag-and-Drop werden drei Event-Handler verwendet:

Listing 10: Drag-and-Drop Template

```
<div class="upload-area"
  [class.drag-over]="isDragOver"
  (dragover)="onDragOver($event)"
  (drop)="onDrop($event)"
  (click)="fileInput.click()">
  <i class="bi bi-cloud-upload display-1"></i>
  <h4>Dateien hier ablegen oder klicken zum Auswaehlen</h4>
  <p>PDF, PNG, JPG, BMP, TIFF, GIF - Max. 10MB pro Datei</p>
</div>
```

Die Event-Handler verhindern das Standard-Browserverhalten (Datei öffnen) und verarbeiten stattdessen die Dateien in der Anwendung:

Listing 11: Event-Handler-Implementierung

```
onDragOver(event: DragEvent): void {
  event.preventDefault();
  this.isDragOver = true;
}

onDrop(event: DragEvent): void {
  event.preventDefault();
  this.isDragOver = false;
  if (event.dataTransfer?.files) {
    const fileArray = Array.from(event.dataTransfer.files);
    this.handleMultipleFiles(fileArray);
  }
}
```

8.1.3 Dateien prüfen und Vorschau erstellen

Die Methode `handleMultipleFiles` prüft, ob die Dateien unterstützte Formate haben, und erstellt Vorschauen:

Listing 12: Multi-File-Handling mit Validierung

```
private handleMultipleFiles(files: File[]): void {
  files.forEach(file => {
    const supportedTypes = ['application/pdf', 'image/png',
                            'image/jpeg', 'image/bmp'];
    if (!supportedTypes.includes(file.type)) {
      this.showFileTypeError(file.name);
      return;
    }

    const fileItem: FileUploadItem = {
      file: file,
      id: `file_${Date.now()}_${Math.random()}`,
      status: 'pending'
    };

    this.loadFilePreview(fileItem);
    this.selectedFiles.push(fileItem);
  });
}
```

Für die Vorschau wird zwischen Bildern und PDFs unterschieden. Bilder werden als Data-URL geladen, PDFs als Binärdaten:

Listing 13: FileReader API für Vorschauen

```
private loadFilePreview(fileItem: FileUploadItem): void {
  const reader = new FileReader();

  if (fileItem.file.type.startsWith('image/')) {
    reader.onload = (e) => fileItem.previewUrl = e.target?.result as string;
    reader.readAsDataURL(fileItem.file);
  } else if (fileItem.file.type === 'application/pdf') {
    reader.onload = (e) => {
      fileItem.pdfData = new Uint8Array(e.target?.result as ArrayBuffer);
    };
    reader.readAsArrayBuffer(fileItem.file);
  }
}
```

Die HTML5 FileReader API bietet zwei Methoden: `readAsDataURL()` für Bilder (als Base64-String) und `readAsArrayBuffer()` für PDFs (als Binärdaten für den PDF-Viewer).

8.2 PDF-Vorschau anzeigen

Für die PDF-Vorschau wird die Bibliothek `ng2-pdf-viewer` verwendet.

8.2.1 Einbindung des PDF-Viewers

Listing 14: ng2-pdf-viewer Installation

```
npm install ng2-pdf-viewer@10.4.0
```

Die PDF-Vorschau wird in einem Bootstrap-Modal (Popup-Fenster) angezeigt:

Listing 15: PDF-Viewer im Modal-Dialog

```
@if (showPreview && pdfData) {
  <div class="modal fade show d-block">
    <div class="modal-dialog modal-xl">
      <div class="modal-content">
        <div class="modal-header">
          <h5>PDF Vorschau: {{ getPreviewFileName() }}</h5>
          <button class="btn-close" (click)="togglePreview()"></button>
        </div>
        <div class="modal-body">
          <pdf-viewer
            [src]="pdfData"
            [render-text]="true"
            [show-all]="true"
            [fit-to-page]="true"
            style="width: 100%; height: 70vh;">
          </pdf-viewer>
        </div>
      </div>
    </div>
  </div>
}
```

Die Optionen bedeuten: `[render-text]` aktiviert Text-Auswahl und Suche, `[show-all]` zeigt alle Seiten gleichzeitig, `[fit-to-page]` passt die Größe an die Fensterbreite an. Das PDF wird als `Uint8Array` (Binärdaten) über `[src]` übergeben.

8.2.2 Speicher freigeben

Beim Schließen der Vorschau werden Blob-URLs freigegeben, um Speicherprobleme zu vermeiden:

Listing 16: Vorschau-Steuerung mit Cleanup

```
showFilePreview(fileId: string): void {
  const fileItem = this.selectedFiles.find(f => f.id === fileId);
  if (fileItem) {
    this.pdfData = fileItem.pdfData;
    this.previewFileId = fileId;
    this.showPreview = true;
  }
}

togglePreview(): void {
  this.showPreview = false;
  if (this.previewUrl) {
    URL.revokeObjectURL(this.previewUrl);
  }
}
```

Die Methode `URL.revokeObjectURL()` gibt Speicher frei, der von `URL.createObjectURL()` belegt wurde. Das ist wichtig, weil Browser diese URLs sonst bis zum Schließen der Seite speichern.

8.3 Datenstruktur für hochgeladene Dateien

Das `FileUploadItem`-Interface definiert, welche Informationen für jede hochgeladene Datei gespeichert werden:

Listing 17: FileUploadItem Interface-Definition

```
interface FileUploadItem {
  file: File;                // Original File-Objekt
  id: string;                // Eindeutiger Identifier
  status: 'pending' | 'processing' | 'completed' | 'error'; // Status
  progress?: number;         // Upload-Fortschritt (0-100)
  result?: any;              // Server-Response
  errorMessage?: string;     // Fehlermeldung bei Fehler
  previewUrl?: string;       // Data-URL f r Bildvorschau
  pdfData?: Uint8Array;      // Bin rdaten f r PDF-Vorschau
}
```

Der Status kann vier Werte haben: `pending` (wartet auf Verarbeitung), `processing` (wird gerade verarbeitet), `completed` (erfolgreich abgeschlossen) oder `error` (Fehler aufgetreten). TypeScript prüft automatisch, dass nur diese Werte verwendet werden.

Anstelle eines Enums wurde ein Union-Type verwendet. Das hat folgende Vorteile: Einfacherer Vergleich mit Strings, kleinere Dateigröße und einfachere Umwandlung in JSON.

8.3.1 Dateien verwalten

Die Komponente verwaltet alle hochgeladenen Dateien in einem Array:

Listing 18: File-Array-Management-Methoden

```
// Hinzuf gen neuer Dateien
this.selectedFiles.push(fileItem);

// Status-Update w hrend Verarbeitung
const currentFile = this.selectedFiles[this.currentProcessingIndex];
currentFile.status = 'processing';

// Entfernen einzelner Dateien
removeSingleFile(fileId: string): void {
  this.selectedFiles = this.selectedFiles.filter(f => f.id !== fileId);
}

// Alle Dateien l schen
clearAllFileData(): void {
  this.selectedFiles = [];
  this.allResults = [];
  this.processedCount = 0;
}
```

Die `filter()`-Methode erstellt ein neues Array ohne die zu entfernende Datei. Das ist wichtig für Angular's Change Detection (die automatische UI-Aktualisierung), die nur auf Änderungen der Array-Referenz reagiert. Eine direkte Änderung mit `splice()` würde die Referenz nicht ändern und die UI würde möglicherweise nicht aktualisiert werden.

Im HTML-Template werden die Dateien mit `@for` angezeigt:

Listing 19: Template mit FileUploadItem-Array

```
@for (fileItem of selectedFiles; track fileItem.id) {
  <div class="list-group-item">
    <h6>{{ fileItem.file.name }}</h6>
    <div class="progress">
      <div class="progress-bar"
           [class.bg-success]="fileItem.status === 'completed'"
           [class.bg-danger]="fileItem.status === 'error'"
           [style.width.%]="getProgressPercentage(fileItem)">
      </div>
    </div>
  </div>
}
```

Die `track`-Funktion verbessert die Performance: Angular aktualisiert nur geänderte Elemente, anstatt die gesamte Liste neu zu erstellen.

8.4 Dateien nacheinander verarbeiten

Die Anwendung verarbeitet mehrere Dateien nacheinander (sequenziell). Dafür werden Variablen wie `isProcessingAll`, `currentProcessingIndex` und `processedCount` verwendet.

8.4.1 Start und Verarbeitung

Die Methode `uploadAllFiles` startet die Verarbeitung aller ausgewählten Dateien:

Listing 20: Upload-Initialisierung

```
private uploadAllFiles(): void {
  this.isProcessingAll = true;
  this.currentProcessingIndex = 0;
  this.processedCount = 0;
  this.totalFiles = this.selectedFiles.length;
  this.allResults = [];

  this.selectedFiles.forEach(f => f.status = 'pending');
  this.processNextFile();
}
```

Diese Methode setzt alle Zähler und Status zurück und startet dann die Verarbeitung der ersten Datei mit `processNextFile()`.

Die Methode `processNextFile` verarbeitet die Dateien nacheinander:

Listing 21: Rekursive Dateiverarbeitung

```
private processNextFile(): void {
  if (this.currentProcessingIndex >= this.selectedFiles.length) {
    this.isProcessingAll = false;
    return;
  }

  const currentFile = this.selectedFiles[this.currentProcessingIndex];
  currentFile.status = 'processing';

  this.invoiceService.uploadInvoice(currentFile.file).subscribe({
    next: (response) => {
      currentFile.status = 'completed';
      currentFile.result = response;
      this.allResults.push(response);
      this.processedCount++;
      this.currentProcessingIndex++;
      this.processNextFile();  // Rekursiver Aufruf
    },
    error: (error) => {
      currentFile.status = 'error';
      currentFile.errorMessage = error.error?.error || 'Fehler';
      this.currentProcessingIndex++;
      this.processNextFile();  // Weiter trotz Fehler
    }
  });
}
```

Die Methode holt sich die aktuelle Datei aus dem Array, sendet sie ans Backend und wartet auf die Antwort. Bei Erfolg wird der Status auf `completed` gesetzt, bei Fehler auf `error`. In beiden Fällen wird der Index erhöht und die Methode ruft sich selbst auf, um die nächste Datei zu verarbeiten. Diese rekursive Logik (Methode ruft sich selbst auf) stellt sicher, dass die Dateien nacheinander und nicht gleichzeitig verarbeitet werden.

8.4.2 Fortschritt anzeigen und alle Downloads starten

Der Fortschritt wird basierend auf dem Status berechnet:

Listing 22: Progress-Berechnung

```
getProgressPercentage(fileItem: FileUploadItem): number {
  switch (fileItem.status) {
    case 'pending': return 0;
    case 'processing': return 50;
    case 'completed': return 100;
    case 'error': return 0;
  }
}

getProcessingProgress(): number {
  return this.totalFiles > 0
    ? (this.processedCount / this.totalFiles) * 100
    : 0;
}
```

Für den Download aller XML-Dateien wird `setTimeout` verwendet:

Listing 23: Batch-Download mit Staggering

```
downloadAllXml(): void {
  this.allResults.forEach((result, index) => {
    let xmlContent = result.zugferdXml || result.workflow?.ebInterfaceXml;
    let invoiceNumber = `invoice_${result.invoice?.id || Date.now()}`;

    setTimeout(() => {
      this.invoiceService.generateXmlDownload(
        xmlContent, invoiceNumber, this.selectedFormat!
      );
    }, index * 500);  // 500ms Verz gerung zwischen Downloads
  });
}
```

Die 500-Millisekunden-Verzögerung zwischen den Downloads verhindert, dass der Browser blockiert wird. Browser erlauben normalerweise nur 6 gleichzeitige Downloads pro Website, und die Verzögerung stellt sicher, dass dieses Limit nicht überschritten wird.

Die XML-Download-Funktion erstellt einen Download-Link:

Listing 24: XML-Download mit Blob-API

```
generateXmlDownload(xmlContent: string, invoiceNumber: string,
                    format: 'ebInterface' | 'ZUGFeRD'): void {
  const blob = new Blob([xmlContent], { type: 'application/xml' });
  const url = window.URL.createObjectURL(blob);
  const link = document.createElement('a');
  link.href = url;
  link.download = `${invoiceNumber}_${format}.xml`;

  document.body.appendChild(link);
  link.click();
  document.body.removeChild(link);
  window.URL.revokeObjectURL(url);
}
```

Diese Methode erstellt einen temporären Download-Link, klickt ihn automatisch an und entfernt ihn wieder. Die Datei wird direkt im Browser erstellt, ohne dass der Server nochmal kontaktiert werden muss.

Die Anwendung hat einen zweistufigen Ablauf: Zuerst wird das Format gewählt (ebInterface oder ZUGFeRD), dann erscheint der Upload-Bereich. Bei Formatwechsel werden vorherige Dateien gelöscht.

8.4.3 Gesamtablauf

Der Upload-Workflow hat folgende Schritte:

1. **Format-Auswahl:** Benutzer wählt ebInterface oder ZUGFeRD
2. **Datei-Selektion:** Drag-and-Drop oder Click-to-Upload
3. **Validierung:** MIME-Type-Prüfung und Größenlimit
4. **Vorschau-Generierung:** FileReader API lädt Preview-Daten
5. **Sequenzielle Verarbeitung:** Uploads nacheinander an Backend
6. **Status-Tracking:** Echtzeit-Updates in UI (pending → processing → completed/error)
7. **Ergebnis-Download:** XML-Dateien einzeln oder als Batch

Jeder Schritt hat eigene Fehlerbehandlung und zeigt dem Benutzer Feedback. Die Aufteilung in verschiedene Bereiche (Datei-Handling, Backend-Kommunikation, UI-Anzeige) macht den Code wartbar und erweiterbar.

8.4.4 Kommunikation mit dem Backend

Die Kommunikation mit dem Backend erfolgt über den `InvoiceService`:

Listing 25: InvoiceService Upload-Methoden

```
@Injectable({ providedIn: 'root' })
export class InvoiceService {
  private apiUrl = environment.apiUrl;

  constructor(private http: HttpClient) {}

  uploadInvoice(file: File): Observable<any> {
    const formData = new FormData();
    formData.append('file', file);
    return this.http.post(`${this.apiUrl}/Processing/upload`, formData);
  }

  uploadInvoiceForZugferd(file: File): Observable<any> {
    const formData = new FormData();
    formData.append('file', file);
    return this.http.post(`${this.apiUrl}/zugferd/upload-pdf`, formData, {
      responseType: 'text'
    }).pipe(
      map((xmlResponse: string) => ({ zugferdXml: xmlResponse }))
    );
  }
}
```

Die `FormData`-API ermöglicht das Hochladen von Dateien per HTTP POST. Der `Observable`-Ansatz von Angular's `HttpClient` ermöglicht asynchrone Verarbeitung mit RxJS-Operatoren wie `map`, `catchError` und `retry`.

8.4.5 Fehler behandeln

Die Fehlerbehandlung erfolgt auf mehreren Ebenen:

Listing 26: Mehrstufige Fehlerbehandlung

```
this.invoiceService.uploadInvoice(currentFile.file).subscribe({
  next: (response) => {
    currentFile.status = 'completed';
    currentFile.result = response;
    this.processedCount++;
    this.currentProcessingIndex++;
    this.processNextFile();
  },
  error: (error) => {
    console.error('Upload fehlgeschlagen:', error);
    currentFile.status = 'error';
    currentFile.errorMessage = error.error?.error ||
                               error.message ||
                               'Unbekannter Fehler';
    this.currentProcessingIndex++;
    this.processNextFile();  // Continue-on-error Pattern
  }
});
```

Das Continue-on-error Pattern bedeutet: Wenn eine Datei fehlschlägt, wird trotzdem mit der nächsten Datei weitergemacht. Die Fehlermeldung wird aus der Server-Antwort ausgelesen oder durch einen Standard-Text ersetzt.

Fehler werden in der Oberfläche mit Bootstrap Alerts angezeigt:

Listing 27: Error-Display im Template

```
@if (fileItem.status === 'error' && fileItem.errorMessage) {
  <div class="alert alert-danger alert-dismissible">
    <i class="bi bi-exclamation-triangle me-2"></i>
    {{ fileItem.errorMessage }}
  </div>
}
```

9 Umsetzung

Siehe tolle Daten in Tab. 2.

Siehe und staune in Abb. 4. Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

	Regular Customers	**Random Customers**
Age	20-40	>60
Education	university	high school

Tabelle 2: Ein paar tabellarische Daten

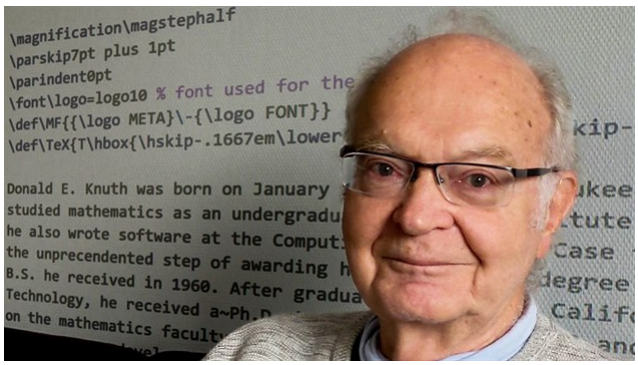

Abbildung 4: Don Knuth – CS Allfather

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus. Dann betrachte den Code in Listing 28.

Listing 28: Some code

```
# Program to find the sum of all numbers stored in a list (the not-Pythonic-way)

# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
    sum = sum+val

print("The sum is", sum)
```

10 Zusammenfassung

Aufzählungen:

- Itemize Level 1
  - Itemize Level 2
    - Itemize Level 3 (vermeiden)

1. Enumerate Level 1
   a. Enumerate Level 2
      i. Enumerate Level 3 (vermeiden)

Desc Level 1

Desc Level 2 (vermeiden)

Desc Level 3 (vermeiden)

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang