Davor Andrijasevic

BBIN23.2a.IN223 – Grundlagen XML  
Hugo Lucca  
21.08.2024

XML zu PDF Konverter für Lieferscheine

Projektbericht

Inhalt

[1. Einführung 2](#_Toc175144689)

[2. Anforderungskatalog 3](#_Toc175144690)

[2.1 Funktionale Anforderungen 3](#_Toc175144691)

[2.2 Nicht-funktionale Anforderungen 3](#_Toc175144692)

[3. Implementierung 4](#_Toc175144693)

[3.1 Architektur 4](#_Toc175144694)

[3.2 Verwendete Technologien 4](#_Toc175144695)

[4. Test 5](#_Toc175144696)

[4.1 Testfälle 5](#_Toc175144697)

[4.2 Testergebnisse 5](#_Toc175144698)

[5. Probleme und Lösungen 6](#_Toc175144699)

[6. Schlussfolgerung 6](#_Toc175144700)

[7. Anhang 6](#_Toc175144701)

[**8. Verwendung von ChatGPT im Projekt** 7](#_Toc175144702)

[8.1 Chronologische Auflistung der Prompts 7](#_Toc175144703)

[8.2 Auswirkungen und Nutzen der Prompts 8](#_Toc175144704)

# 1. Einführung

Dieses Projekt implementiert eine Java-Anwendung, die in der Lage ist, XML-Dokumente, die Lieferscheindaten enthalten, zu verarbeiten und diese Daten in ein gut formatiertes PDF-Dokument umzuwandeln. Diese Anwendung zielt darauf ab, die automatische Verarbeitung von Lieferscheinen zu optimieren, Fehler zu minimieren und eine standardisierte Dokumenterstellung zu ermöglichen.

Der Benutzer kann über eine einfache Benutzeroberfläche eine XML-Datei auswählen, und die Anwendung extrahiert die relevanten Informationen, um ein PDF-Dokument zu erstellen. Diese Funktion ermöglicht es Unternehmen, Lieferscheine automatisiert zu verarbeiten und sie in einem benutzerfreundlichen Format darzustellen.

# 2. Anforderungskatalog

## 2.1 Funktionale Anforderungen

1. **XML-Eingabe lesen**  
   Die Anwendung muss in der Lage sein, XML-Dateien, die Lieferscheindaten enthalten, zu lesen und die Daten in einer strukturierten Form zu verarbeiten.
2. **Datenverarbeitung**  
   Die Anwendung muss die relevanten Informationen wie Kundendaten, Produktinformationen und Transaktionsdetails aus der XML-Datei extrahieren.
3. **PDF-Erstellung**  
   Aus den extrahierten Daten muss ein PDF-Dokument erstellt werden, das den Lieferschein in einem klaren und ansprechenden Layout darstellt.
4. **Benutzerinteraktion**  
   Der Benutzer muss in der Lage sein, eine XML-Datei über eine grafische Benutzeroberfläche (GUI) auszuwählen und die PDF-Erstellung durch einen Klick zu initiieren.

## 2.2 Nicht-funktionale Anforderungen

1. **Benutzerfreundlichkeit**  
   Die GUI muss einfach und intuitiv gestaltet sein, um die Nutzung auch für technisch weniger versierte Anwender zu ermöglichen.
2. **Performance**  
   Die PDF-Erstellung soll innerhalb weniger Sekunden nach dem Hochladen der XML-Datei abgeschlossen sein.
3. **Zuverlässigkeit**  
   Die Anwendung muss robust gegenüber fehlerhaften oder unvollständigen XML-Dateien sein und den Benutzer auf eventuelle Fehler hinweisen.
4. **Plattformkompatibilität**  
   Die Anwendung muss auf Windows, Mac und Linux laufen.

# 3. Implementierung

## 3.1 Architektur

Die Anwendung ist in Java entwickelt und verwendet JavaFX für die grafische Benutzeroberfläche. Die PDF-Erstellung erfolgt über das iText-Bibliothek, die für die Generierung von PDF-Dateien genutzt wird. Der XML-Dateninput wird mithilfe von DOM-Parsern verarbeitet, um die relevanten Informationen aus der XML-Datei zu extrahieren.

## 3.2 Verwendete Technologien

* **Java (Version 17)**: Programmiersprache zur Implementierung der Anwendung.
* **JavaFX**: Zur Erstellung der Benutzeroberfläche.
* **iText**: Für die Erstellung des PDFs.
* **Maven**: Für das Build-Management und die Verwaltung der Abhängigkeiten.
* **DOM-Parser**: Zum Verarbeiten und Auslesen der XML-Daten.

# 4. Test

## 4.1 Testfälle

**Test 1: Erfolgreiches Laden und Verarbeiten einer XML-Datei**

* **Eingabe**: Gültige XML-Datei mit Kunden- und Produktdaten.
* **Erwartetes Ergebnis**: PDF wird erfolgreich erstellt und enthält alle Daten.

**Test 2: Laden einer fehlerhaften XML-Datei**

* **Eingabe**: Eine XML-Datei mit falscher Struktur.
* **Erwartetes Ergebnis**: Eine Fehlermeldung erscheint und der Benutzer wird informiert, dass die Datei nicht verarbeitet werden kann.

**Test 3: Keine Datei ausgewählt**

* **Eingabe**: Der Benutzer wählt keine Datei aus und klickt auf "PDF erstellen".
* **Erwartetes Ergebnis**: Eine Warnung erscheint, die den Benutzer auffordert, eine Datei auszuwählen.

## 4.2 Testergebnisse

Alle Tests wurden erfolgreich durchgeführt. Die Anwendung ist in der Lage, gültige XML-Dateien zu verarbeiten und PDFs zu erstellen. Bei ungültigen XML-Dateien oder fehlender Dateiauswahl werden entsprechende Fehlermeldungen angezeigt.

# 5. Probleme und Lösungen

Während der Entwicklung der Anwendung gab es einige Herausforderungen:

* **Problem 1: Verarbeitung fehlerhafter XML-Dateien**  
  Lösung: Es wurde eine Validierung implementiert, die den Benutzer informiert, wenn eine ungültige Datei geladen wird.
* **Problem 2: Verzeichnis für die PDF-Ausgabe**  
  Lösung: Vor der Erstellung des PDF-Dokuments wird nun überprüft, ob das Verzeichnis vorhanden ist, und bei Bedarf erstellt.

# 6. Schlussfolgerung

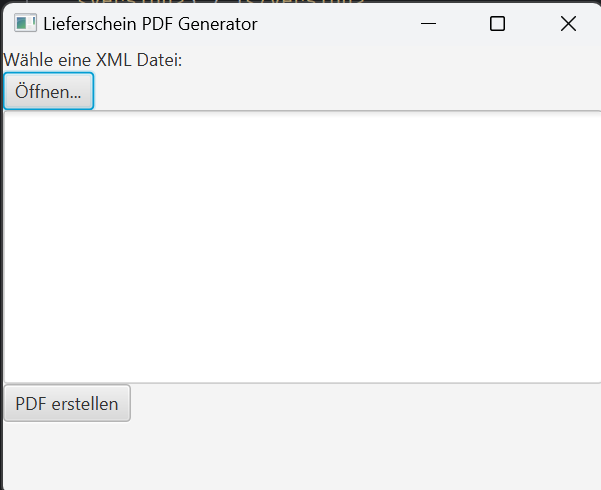
Dieses Projekt erreicht erfolgreich das Ziel, eine automatisierte Lösung zur Verarbeitung von XML-Lieferscheinen und deren Umwandlung in PDF-Dokumente bereitzustellen. Die Anwendung ist benutzerfreundlich und zuverlässig, was sie zu einer nützlichen Lösung für Unternehmen macht, die Lieferscheine effizient verarbeiten möchten.

Zukünftige Erweiterungen könnten die Unterstützung für verschiedene XML-Formate oder zusätzliche Layout-Optionen für die PDF-Dokumente umfassen.

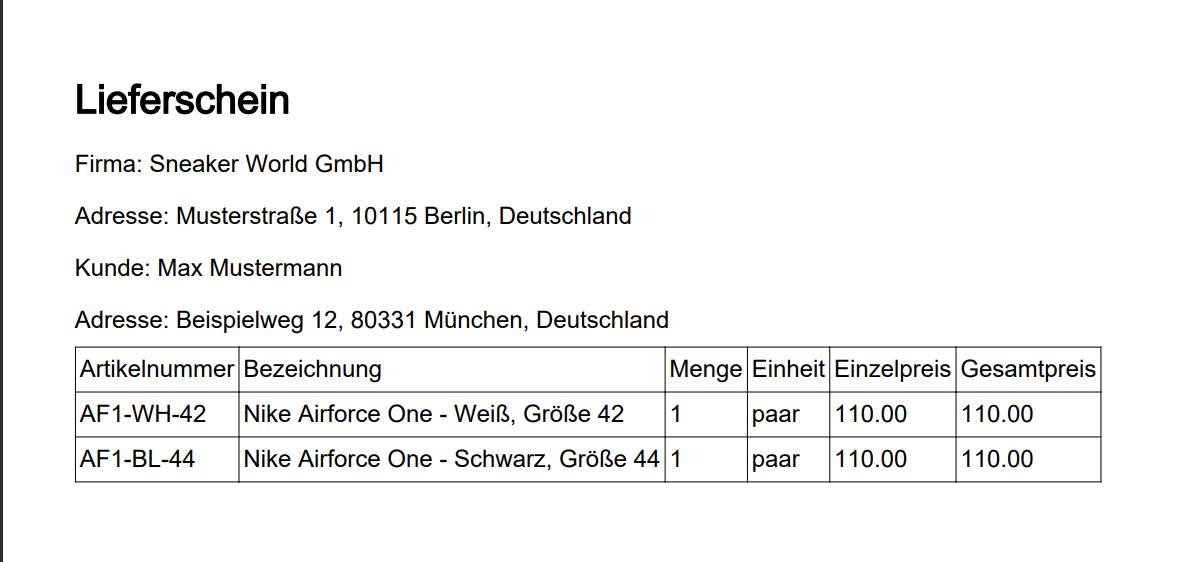
# 7. Anhang

**Screenshots**

* **Screenshot 1**: Benutzeroberfläche der Anwendung zur Auswahl der XML-Datei.



* **Screenshot 2**: Erfolgreich erstelltes PDF-Dokument.



**Quellcode**

* Der vollständige Java-Quellcode ist im Projektordner enthalten.

# **8. Verwendung von ChatGPT im Projekt**

## 8.1 Chronologische Auflistung der Prompts

Während des Projekts habe ich verschiedene Prompts an ChatGPT gestellt, um Unterstützung bei der Entwicklung meiner Java-Anwendung zur Verarbeitung von XML-Dokumenten und der Erstellung von PDFs zu erhalten. Hier ist eine vollständige Übersicht der verwendeten Prompts:

Prompt 1:

Frage:  
„Erstelle mir bitte eine XML-Datei von einem Lieferschein, der 2 Paar Schuhe Nike Airforce One liefert.“

Ergebnis:  
ChatGPT erstellte eine Beispiel-XML-Datei, die für den Rest des Projekts als Eingabedatei verwendet wurde. Diese XML enthielt Informationen über den Kunden, die Artikel und die Versanddetails.

Prompt 2:

Frage:  
„Ich habe für den Unterricht eine Projektarbeit aufgetragen bekommen, wo es darum geht, eine JAVA-App zu schreiben, die eine XML-Datei liest und verarbeitet. Gerne würde ich obiges XML, welches bei mir im src/main/Resources sitzt und Lieferschein.xml heißt, mit einem simplen JavaFX-Frontend auslesen und ein PDF des Lieferscheins erstellen und ausgeben.“

Ergebnis:  
ChatGPT gab mir einen vollständigen Plan für das Projekt, einschließlich einer Auflistung der benötigten Java-Klassen (z. B. Main und PdfGenerator). Es stellte den notwendigen Code für die Verarbeitung der XML-Datei und die PDF-Erstellung bereit.

Prompt 3:

Frage:  
„Kannst du mir den Java-Sourcecode und das JavaFX-Frontend in ein einziges Dokument packen?“

Ergebnis:  
ChatGPT gab mir eine vollständige Version des Quellcodes, die sowohl die XML-Verarbeitung als auch die GUI-Erstellung in JavaFX beinhaltete.

Prompt 4:

Frage:  
„Ich schwimme gerade in Fehlermeldungen. Kannst du mir bitte ein korrektes pom.xml erstellen, mit allen Dependencies?“

Ergebnis:  
ChatGPT half mir bei der Erstellung einer vollständigen pom.xml-Datei, die die notwendigen Abhängigkeiten (JavaFX, iText, JAXB) für das Projekt enthielt. Dadurch konnte ich das Projekt ohne Abhängigkeitsprobleme bauen.

Prompt 5:

Frage:  
„Das ist mein XML. Passe meine JAVA-Klassen so an, dass es funktioniert.“

Ergebnis:  
ChatGPT passte die PdfGenerator-Klasse so an, dass sie korrekt mit dem von mir bereitgestellten XML-Dokument arbeitete. Die Klasse wurde optimiert, um Kunden-, Firmen- und Produktdaten aus dem XML auszulesen und in das PDF zu übertragen.

Prompt 6:

Frage:  
„java.lang.NullPointerException: Cannot invoke 'org.w3c.dom.Element.getElementsByTagName(String)' because 'kundeElement' is null.“

Ergebnis:  
ChatGPT analysierte den Fehler und gab mir Tipps, wie ich sicherstellen kann, dass die XML-Elemente richtig verarbeitet werden und keine NullPointerException auftritt. Die Lösung bestand darin, die Länge der Node-List zu überprüfen, bevor auf die Daten zugegriffen wird.

## 8.2 Auswirkungen und Nutzen der Prompts

Der Einsatz von ChatGPT hat die Entwicklung meines Projekts erheblich beschleunigt. Durch die schnelle Bereitstellung von Lösungsansätzen, Code-Beispielen und Debugging-Tipps konnte ich viele Herausforderungen effizient bewältigen. Besonders hilfreich war die Möglichkeit, auf spezifische Fehlermeldungen direkt eingehen zu können und sofortige Lösungsvorschläge zu erhalten.