



Abschlussprüfung Winter 2024

Fachinformatiker für Anwendungsentwicklung  
Dokumentation zur betrieblichen Projektarbeit

# **Automatisierte Verfolgung und Benachrichtigung für verbesserte Liefertransparenz**

**Überwachungssystem für Warenlieferungen**

Abgabetermin: Lüneburg, den 27.11.2024

**Prüfungsbewerber:**

Jens Lange  
Hoßberg 17  
21376 Salzhausen

**VONMÄHLEN**

Northern Germany

**Ausbildungsbetrieb:**

VONMÄHLEN GMBH  
Vor dem Bardowicker Tore 49  
21339 Lüneburg

## Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>III</b>
<b>Tabellenverzeichnis</b>	<b>IV</b>
<b>Abkürzungsverzeichnis</b>	<b>V</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Projektumfeld . . . . .	1
1.2 Projektziel . . . . .	1
1.3 Projektbegründung . . . . .	1
1.4 Technische Projektschnittstellen . . . . .	2
1.4.1 Versanddienstleister: Zeitfracht (ZF) . . . . .	2
1.4.2 ERP-System: Microsoft Business Central . . . . .	2
1.4.3 Datenbank: PostgreSQL . . . . .	2
1.4.4 Workflow-Management-System: Apache Airflow . . . . .	2
1.4.5 Docker-Container . . . . .	3
<b>2 Projektplanung</b>	<b>3</b>
2.1 Projektphasen . . . . .	3
2.2 Abweichungen vom Projektantrag . . . . .	4
2.3 Ressourcenplanung . . . . .	4
2.4 Entwicklungsprozess . . . . .	4
<b>3 Analysephase</b>	<b>5</b>
3.1 Ist-Analyse . . . . .	5
3.2 Wirtschaftlichkeitsanalyse . . . . .	5
3.3 Kostenanalyse . . . . .	5
3.3.1 Einmalige Implementierungskosten . . . . .	5
3.3.2 Laufende Betriebskosten . . . . .	6
3.4 Vergleich: Manuelle vs. Automatisierte Prozesse . . . . .	7
<b>4 Entwurfsphase</b>	<b>7</b>
4.1 Zielplattform . . . . .	7
4.2 Architekturdesign . . . . .	7
<b>5 Implementierungsphase</b>	<b>8</b>
5.1 PostgreSQL-Datenbankmodellierung und -Erstellung . . . . .	8
5.2 Deployment auf dem Hetzner-Server . . . . .	9
5.3 Testdaten und Verifizierung des Datenbankzugriffs . . . . .	9
5.4 Implementierung der Geschäftslogik . . . . .	9

<b>6</b>	<b>Abnahmephase</b>	<b>12</b>
6.1	Testphase . . . . .	12
6.1.1	Tests mit dem Skript <i>trackings.py</i> . . . . .	12
6.1.2	Tests mit dem Skript <i>bc_db.py</i> . . . . .	13
<b>7</b>	<b>Einführungsphase</b>	<b>15</b>
7.1	Einrichtung neuer Docker-Container auf dem Hetzner-Server . . . . .	15
7.2	Installation und Konfiguration von Apache Airflow . . . . .	15
7.3	Starten und Verwalten des PostgreSQL-Datenbankservers . . . . .	15
7.4	Verbindung zwischen den Containern . . . . .	16
<b>8</b>	<b>Dokumentation</b>	<b>16</b>
8.1	Zielgruppe der Entwicklerdokumentation . . . . .	16
8.2	Inhalt der Entwicklerdokumentation . . . . .	16
8.3	Code Dokumentation . . . . .	17
<b>9</b>	<b>Fazit</b>	<b>17</b>
9.1	Soll-/Ist-Vergleich . . . . .	17
<b>Literaturverzeichnis</b>		<b>18</b>
<b>A</b>	<b>Anhang</b>	<b>i</b>
A.1	Lastenheft . . . . .	i
A.2	Komplette Zeitplanung . . . . .	ii
A.3	Zeitplanung nach Anpassungen . . . . .	iii

## **Abbildungsverzeichnis**

1	Skript zur Erstellung der Datenbank . . . . .	9
2	Validierung der Statuscodes . . . . .	10
3	Prüfung von Sendungen im Status 30 . . . . .	10
4	Verwendung von Integrationen . . . . .	11
5	Fehlerbehandlung bei API-Aufrufen . . . . .	11
6	Auszug Lastenheft . . . . .	i

**Tabellenverzeichnis**

1	Zeitplanung . . . . .	3
2	Vergleich von manuellen und automatisierten Prozessen . . . . .	7
3	<i>bc_tracking</i> . . . . .	8

## **Abkürzungsverzeichnis**

<b>ERP</b>	Enterprise Resource Planning
<b>SCM</b>	Supply Chain Management
<b>API</b>	Application Programming Interface

## 1 Einleitung

### 1.1 Projektumfeld

VONMÄHLEN ist ein deutsches Unternehmen, das sich auf die Entwicklung und Produktion von hochwertigen, designorientierten Lifestyle-Technologieprodukten spezialisiert hat. Die Firma bietet innovative Zubehörlösungen für den täglichen Gebrauch, insbesondere im Bereich Smartphone und Technik-Accessoires.

### 1.2 Projektziel

Die SCM-Abteilung wünscht sich eine automatisierte Überwachung der Warensendungen, um den manuellen Aufwand und das Risiko menschlicher Fehler zu minimieren. Das System soll den Status aller Warensendungen automatisch und regelmäßig überprüfen und bei Abweichungen oder Verzögerungen eine E-Mail-Benachrichtigung an die zuständigen Mitarbeiter der SCM-Abteilung senden. Zusätzlich soll das System im ERP-System (Microsoft Business Central) den Zeitpunkt der Auslieferung einer Sendung erfassen und aktualisieren, damit ersichtlich ist, wann eine Lieferung erfolgreich zugestellt wurde. Diese Information dient als Grundlage dafür, dass die SCM-Abteilung die Buchung der zugehörigen Rechnung anstoßen kann, sobald die Lieferung den Empfänger erreicht hat. Durch diese Automatisierung erwartet die SCM-Abteilung eine höhere Effizienz, da die Zustellvorgänge transparent und leicht nachvollziehbar sind und die Rechnungsstellung unmittelbar nach der Zustellung erfolgen kann. Eine detaillierte Auflistung der Anforderungen sind dem Lastenheft zu entnehmen. Siehe Anhang A.1: Lastenheft auf Seite i

### 1.3 Projektbegründung

Die größte Schwachstelle im aktuellen Prozess liegt im hohen manuellen Aufwand und der fehlenden Dokumentation. Täglich werden alle offenen Lieferaufträge nach solchen mit einer Trackingnummer durchsucht, was als Indikator dient, dass die Ware bereits auf dem Weg ist und überprüft werden muss. Dabei wird festgestellt, wie weit die Ware ist und ob Verzögerungen oder Probleme vorliegen. Die Prüfung erfolgt durch Eingabe der Trackingnummer im Webportal des Versanddienstleisters (z. B. DPD). Je nach Lieferstatus wird dann entsprechend gehandelt, und bei Auffälligkeiten wird die zuständige Abteilung informiert. Dieser manuelle Prozess birgt jedoch das Risiko, dass Überprüfungen ausgelassen oder ganze Aufträge übersehen werden. Auch kann es bei Personalmangel zum vollständigen Ausfall der Prüfungen kommen. Aufgrund dieser Fehleranfälligkeit und des hohen Zeitaufwands hat sich die Firma VONMÄHLEN entschieden, den Prozess zu automatisieren, um die Effizienz zu steigern und die Prozesssicherheit nachhaltig zu gewährleisten.

## 1.4 Technische Projektschnittstellen

Diese Schnittstellen gewährleisten eine reibungslose Interaktion zwischen den verschiedenen Systemen und Komponenten, die für das Projekt erforderlich sind.

### 1.4.1 Versanddienstleister: Zeitfracht (ZF)

- Art: SFTP-Schnittstelle
- Zweck: Bereitstellung von Trackingdaten
- Beschreibung: Der Versanddienstleister liefert täglich aktualisierte Tracking-Daten als CSV-Dateien über einen gesicherten SFTP-Server. Diese Daten werden vom System heruntergeladen, validiert und verarbeitet.

### 1.4.2 ERP-System: Microsoft Business Central

- Art: API-Schnittstelle
- Zweck: Austausch von Informationen zu Sendungsstatus und Aktualisierungen von Lieferdaten
- Beschreibung: Die API ermöglicht es, Statusinformationen und Lieferdetails aus der PostgreSQL-Datenbank in das ERP-System zu übertragen, um die automatisierte Weiterverarbeitung (z.B. Rechnungsstellung) zu gewährleisten.

### 1.4.3 Datenbank: PostgreSQL

- Art: SQL-Schnittstelle
- Zweck: Speicherung und Abfrage der Tracking-Daten
- Beschreibung: Diese Schnittstellen gewährleisten eine reibungslose Interaktion zwischen den verschiedenen Systemen und Komponenten, die für das Projekt erforderlich sind.

### 1.4.4 Workflow-Management-System: Apache Airflow

- Art: Workflow-Orchestrierungs-Tool
- Zweck: Automatisierung und Orchestrierung von Prozessen
- Beschreibung: Apache Airflow steuert die zeitgesteuerte Verarbeitung der Trackingdaten, die Kommunikation mit der Datenbank und die Synchronisierung mit dem ERP-System.



### 1.4.5 Docker-Container

- Art: Container-Übergreifende Kommunikation
- Zweck: Isolierte Ausführung von PostgreSQL und Apache Airflow
- Beschreibung: Docker-Container werden genutzt, um PostgreSQL und Apache Airflow in einer sicheren und skalierbaren Umgebung auszuführen. Die Container sind so konfiguriert, dass sie miteinander kommunizieren können, um Daten auszutauschen.

## 2 Projektplanung

### 2.1 Projektphasen

Für diese Umsetzung des Projektes standen dem Auszubildenden 80 Stunden zur Verfügung. Welche vor Projektbeginn in verschiedene Phasen aufgeteilt wurden.

Tabelle 1 zeigt eine grobe Zeitplanung.

Projektphase	Geplante Zeit
Analyse	12 h
Entwurf	12 h
Implementierung	38 h
Test	6 h
Einführung	3 h
Dokumentation	9 h
<b>Gesamt</b>	<b>80 h</b>

Tabelle 1: Zeitplanung

Eine detailliertere Zeitplanung ist im Anhang A.2: Komplette Zeitplanung auf Seite ii

## 2.2 Abweichungen vom Projektantrag

Es stellte sich während des Projektes heraus das einige Zeiten nicht eingehalten werden konnten. Darauf wird im Anhang A.3: Zeitplanung nach Anpassungen auf Seite iii weiter eingegangen.

## 2.3 Ressourcenplanung

Um die Planung der erforderlichen Ressourcen effizient zu gestalten, wurde gezielt auf bereits vorhandene Infrastruktur zurückgegriffen. Die Auswahl der Hardware und Software erfolgte so, dass keine zusätzlichen Anschaffungen erforderlich waren. Hardware war vollständig vorhanden, da alle Mitarbeiter über einen ausgestatteten Büroarbeitsplatz mit Standard-Peripherie verfügten. Die Softwareauswahl konzentrierte sich auf Open-Source-Tools wie PostgreSQL und Python sowie bereits lizenzierten Programme (Business Central API), die während der Projektlaufzeit genutzt wurden, um die Datenverarbeitung sicherzustellen. Der Auszubildende übernahm die Hauptverantwortung in der Entwicklung, was insgesamt 80 Stunden veranschlagte. Ein Entwickler wurde für 10 Stunden eingeplant, um gezielt bei technischen Fragen zu unterstützen. Für die infrastrukturelle Vorbereitung, wie das Aufsetzen der Docker-Container inklusive der PostgreSQL-Datenbank und Apache Airflow, wurde Zuarbeit durch die IT-Abteilung geleistet. Dies stellte sicher, dass eine stabile und funktionale Grundlage für die Entwicklung geschaffen wurde. Die SCM-Abteilung war schließlich dafür verantwortlich, die Anforderungen zu formulieren und das System während der Testphase auf seine Praxistauglichkeit zu überprüfen. Die Kombination aus vorhandenen Ressourcen und gezielter, bedarfsorientierter Planung sorgte dafür, dass die Projektkosten gering blieben und die notwendige Unterstützung jederzeit gewährleistet war.

## 2.4 Entwicklungsprozess

Für die Entwicklung dieses Projekts wurde das Wasserfallmodell als Vorgehensmodell gewählt. Das Wasserfallmodell zeichnet sich durch eine klare und lineare Abfolge von Phasen aus, was insbesondere bei Projekten mit fest definierten Anforderungen und einem strukturierten Ablauf von Vorteil ist, zudem ist es ein Vorgehensmodell der klassischen Softwareentwicklung, bei dem die Projektphasen sequentiell durchlaufen und bearbeitet werden (vgl. [FITTKAU UND RUF \[2008\]](#) S. 31). Da die Anforderungen an das System im Vorfeld detailliert analysiert und dokumentiert wurden, ermöglichte das Wasserfallmodell eine schrittweise und systematische Umsetzung der Projektphasen von der Anforderungsanalyse über die Implementierung bis hin zur abschließenden Testphase.

## 3 Analysephase

### 3.1 Ist-Analyse

Derzeit werden täglich alle Warensendungen manuell geprüft, indem Lieferungen mit Trackingnummern herausgefiltert und im Onlineportal des externen Versanddienstleisters überprüft werden. Mitarbeiter in der Abteilung Supply Chain Management (SCM) durchforsten Verkaufsaufträge nach Trackingnummern und prüfen im Versandportal den aktuellen Status der Sendungen. Der Fokus liegt hierbei darauf, Unregelmäßigkeiten wie Verzögerungen oder Fehlzustellungen zu identifizieren. Bei auffälligen Zuständen wird die zuständige Abteilung informiert, um die erforderlichen Maßnahmen einzuleiten.

### 3.2 Wirtschaftlichkeitsanalyse

Die Wirtschaftlichkeit des Projekts ließ sich unkompliziert anhand der eingesparten Arbeitszeit der SCM-Mitarbeiter berechnen, da diese durch die Automatisierung des Prüfungsprozesses deutlich reduziert wird.

### 3.3 Kostenanalyse

Die Kosten für die Implementierung des Systems umfassen zwei Hauptkategorien: Initiale Implementierungskosten und laufende Betriebskosten.

#### 3.3.1 Einmalige Implementierungskosten

Diese Kosten fallen nur einmal an und umfassen die Entwicklung des Systems, die Anschaffung notwendiger Software sowie die Einrichtung der Infrastruktur.

##### **Entwicklungskosten (inklusive Planung, Design und Implementierung):**

- Auszubildender: 80 Stunden a 7 €/Stunde = 560 €
- Entwickler-Support: 10 Stunden a 22 €/Stunde = 220 €
- Fachabteilung SCM: 2 Stunden a 30 €/Stunde

#### **Softwarelizenzen und Tools:**

- PostgreSQL: Kostenlos
- Visual Studio Code: Kostenlos
- Docker: Kostenlos
- Apache Airflow: Kostenlos
- Business Central Lizenz (während der Implementierung): 65,50 €/Monat, über 2 Monate = 131 €

#### **Hardware- und Infrastrukturkosten:**

- Serverkosten, Server vorhanden: 0€/Jahr

**Gesamtkosten für die Implementierung: 971 €**

#### **3.3.2 Laufende Betriebskosten**

Diese Kosten fallen während des Betriebs des Systems an.

#### **Wartung, Fehlerbehebung und Optimierungen:**

- Entwickler-Support (12 Stunden pro Jahr a 22€/Stunde) = 264 €/Jahr
- Erweiterung des bestehenden Hetzner Servers 120 €/Jahr

**Gesamtkosten für den laufenden Betrieb: 384 €/Jahr**

### 3.4 Vergleich: Manuelle vs. Automatisierte Prozesse

Tabelle 2

Kategorie	Manuelle Verarbeitung	Automatisierte Verarbeitung
Zeitaufwand Stunden/Jahr	189 h	12 h
Jährliche Kosten durch Zeitaufwand	5.670 €	264 €
Jährliche Kosten Server	0 €	120 €
Jährliche Fehlerkosten <sup>1</sup>	500 €	0 €
Fehleranfälligkeit <sup>2</sup>	Hoch	Gering
Skalierbarkeit	Gering	Hoch
Jährliche Gesamtbetriebskosten	6.170 €	384 €

Tabelle 2: Vergleich von manuellen und automatisierten Prozessen

## 4 Entwurfsphase

### 4.1 Zielplattform

Die Zielplattform für dieses Projekt ist die IT-Infrastruktur der VONMÄHLEN GmbH, die auf ein internes Netzwerk ausgerichtet ist. Kernkomponenten des Projekts, darunter die PostgreSQL-Datenbank und das Enterprise Resource Planning (ERP)-System BC, sind in die bestehende Serverarchitektur integriert. Zur Automatisierung von Aufgaben wie dem täglichen Datenabgleich und der Aktualisierung der Trackinginformationen werden Docker und Apache Airflow eingesetzt. Diese Werkzeuge ermöglichen eine nahtlose Integration und die zuverlässige Ausführung wiederkehrender Aufgaben. Der Zugriff auf Business Central erfolgt über eine abgesicherte Application Programming Interface (API)-Schnittstelle, während der Datentransfer über einen SFTP-Server abgewickelt wird, der die Trackingdaten vom externen Dienstleister sicher aufnimmt. Mit dieser Infrastruktur wird eine optimale Sicherheit und Performance erreicht, und gleichzeitig können die Anforderungen an eine hohe Verfügbarkeit und Skalierbarkeit erfüllt werden.

### 4.2 Architekturdesign

Python in Verbindung mit Apache Airflow wurde aufgrund der einfachen Integration und Technologiegleichheit im Unternehmen gewählt. Python ermöglicht schnelle Entwicklung, während Airflow die Orchestrierung und Überwachung der Datenpipelines übernimmt. Ideal für die automatisierte Verarbeitung der Tracking-Daten.

---

<sup>1</sup>Diese Fehlerkosten wurden aus Erfahrungswerten der SCM-Abteilung entnommen.

<sup>2</sup>Die Fehleranfälligkeit ist im Manuellen Prozess deutlich höher als im Automatisierten Prozess.

## 5 Implementierungsphase

---

Zusätzlich wurde Docker als Plattform gewählt, um eine isolierte, portable und skalierbare Umgebung für die Anwendungen bereitzustellen. Docker erleichtert die Verwaltung der eingesetzten Dienste wie PostgreSQL und Apache Airflow und ermöglicht eine schnelle Bereitstellung und Wiederherstellung der Anwendung. Die Containerisierung stellt sicher, dass die Anwendung unabhängig von der Umgebung läuft und problemlos erweitert werden kann.

PostgreSQL wurde als Datenbankmanagementsystem ausgewählt, da es eine leistungsstarke, zuverlässige und kostenlose Open-Source-Lösung darstellt. Es bietet umfangreiche Funktionen wie Transaktionssicherheit, Skalierbarkeit und eine hohe Leistung bei der Verarbeitung großer Datenmengen. Die Fähigkeit von PostgreSQL, komplexe Abfragen effizient zu verarbeiten, war entscheidend für die Auswahl, da die Tracking-Daten sowohl regelmäßig aktualisiert als auch konsolidiert werden müssen. Zusammen mit Docker ergibt sich eine robuste, skalierbare und leicht wartbare Architektur, die den Anforderungen des Projekts optimal entspricht.

## 5 Implementierungsphase

### 5.1 PostgreSQL-Datenbankmodellierung und -Erstellung

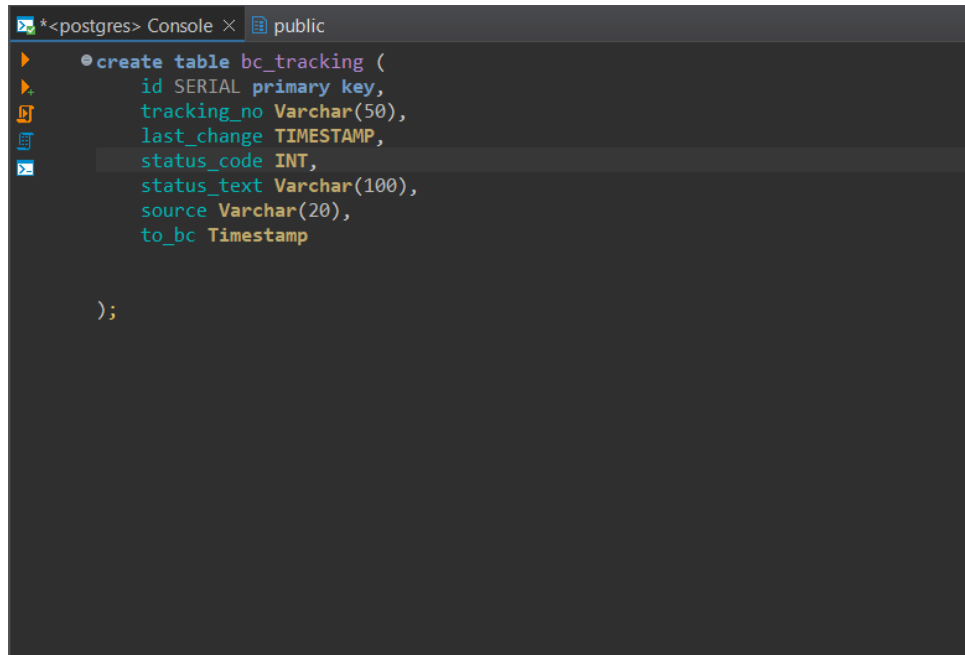
Die PostgreSQL-Datenbank umfasst eine einzige Tabelle namens *bc\_tracking*. Aufgrund der einfachen Projektanforderungen war ein komplexes ER-Modell nicht nötig. Die Datenbank speichert Tracking-Informationen von Zeitfracht und BC.

Tabelle 3 Die Struktur der Tabelle *bc\_tracking* ist wie folgt:

Spalte	Typ	Beschreibung
id	BigSerial	Primärschlüssel, eindeutiger Identifikator
tracking_no	VARCHAR	Eindeutige Sendungsnummer
last_change	TIMESTAMP	Letzte Statusänderung (Datum & Uhrzeit)
status_code	VARCHAR	Aktueller Statuscode
status_text	VARCHAR	Beschreibung des Statuscodes
source	VARCHAR	Datenquelle (z.B. ZF)
to_bc	TIMESTAMP	Zeitpunkt der Übertragung an BC

Tabelle 3: *bc\_tracking*

Die Tabelle wurde mit dem folgenden SQL-Skript erstellt:



```
*-<postgres> Console × public
● create table bc_tracking (
  id SERIAL primary key,
  tracking_no Varchar(50),
  last_change TIMESTAMP,
  status_code INT,
  status_text Varchar(100),
  source Varchar(20),
  to_bc Timestamp
);
```

Abbildung 1: Skript zur Erstellung der Datenbank

## 5.2 Deployment auf dem Hetzner-Server

Die PostgreSQL-Datenbank wurde auf einem Hetzner-Server installiert, um eine zuverlässige und performante Umgebung für die Anwendung bereitzustellen. Die Verwaltung der Datenbank erfolgt über DBeaver, das sowohl lokale als auch entfernte Datenbankverbindungen unterstützt.

## 5.3 Testdaten und Verifizierung des Datenbankzugriffs

Um die korrekte Funktion der Datenbank und der Anwendungen zu verifizieren, wurden Test-daten verwendet. Diese Testdaten wurden in die Tabelle *bc\_tracking* eingefügt und die Anwendungen darauf ausgeführt, um sicherzustellen, dass die Daten korrekt verarbeitet und die gewünschten Aktionen, wie das Senden von E-Mails, ausgeführt werden. Screenshots der Anwendung in der Entwicklungsphase mit Dummy-Daten befinden sich im Anhang A3 Abbildung 7

## 5.4 Implementierung der Geschäftslogik

Die Geschäftslogik wurde in Python implementiert und besteht aus zwei Hauptskripten: *trackings.py* und *bc\_db.py*. Das Skript *trackings.py* ist verantwortlich für das Einlesen von CSV-Dateien mit Tracking-Daten von Zeitfracht und das Speichern dieser Daten in der PostgreSQL-Datenbank. Das Skript führt folgende Schritte aus:

## 5 Implementierungsphase

---

- Einlesen aller CSV-Dateien aus einem Quellverzeichnis.
- Validierung der Dateiinhalte, insbesondere der notwendigen Spalten.
- Verarbeitung jeder Zeile und Einfügen der Daten in die Tabelle *bc\_tracking*.
- Bei erfolgreicher Verarbeitung wird die CSV-Datei aus dem Quellverzeichnis gelöscht.

Ein wichtiger Bestandteil ist die Validierung der Daten. Hier wird sichergestellt, dass nur gültige Statuscodes verarbeitet werden:

```
def validate_row_content(self, row):
    valid_status_codes = ["30", "40", "50"]
    if row["STATUS_CODE"] not in valid_status_codes:
        logging.warning(f"Invalid STATUS_CODE: {row['STATUS_CODE']}")
        return False
    return True
```

Abbildung 2: Validierung der Statuscodes

Das Skript *bc\_db.py* kümmert sich um die Verarbeitung der in der Datenbank gespeicherten Trackingdaten und interagiert mit der Business Central API. Die Hauptaufgaben sind:

- Überprüfen, ob bestimmte Tracking-Nummern länger als eine definierte Anzahl von Tagen in einem bestimmten Status verweilen.
- Senden von Benachrichtigungs-E-Mails, falls diese Bedingungen erfüllt sind
- Senden von Benachrichtigungs-E-Mails, falls Fehler auftreten
- Aktualisieren von Datensätzen in Business Central über die API.

Beispielsweise wird geprüft, ob Sendungen mit Statuscode 30 länger als 7 Tage nicht zugestellt wurden:

```
def check_status_delta_more_than_n_days(self, status_code, days_delta):
    query = text("""
        SELECT t.tracking_no,
               v.earliest_change,
               v.in_transit
        FROM in_transit_view v
        JOIN bc_tracking t
        ON v.tracking_no = t.tracking_no
        WHERE t.status_code = :status_code
        AND v.in_transit > INTERVAL :days_delta
    """)
```

Abbildung 3: Prüfung von Sendungen im Status 30



## 5 Implementierungsphase

---

Sowohl *trackings.py* als auch *bc\_db.py* nutzen verschiedene Integrationen, um die Funktionalität zu erweitern. Zum Beispiel wird das Decorator-Pattern verwendet, um Integrationen einzubinden:

```
@Use(DatabaseIntegration)
@Use(BusinessCentralIntegration)
@Use(MailerIntegration)
class bc_tracking(Action):
```

Abbildung 4: Verwendung von Integrationen

Dieses Muster ermöglicht eine modulare und erweiterbare Architektur der Anwendung.

Des Weiteren werden Ausnahmen und Fehler sorgfältig behandelt, um die Stabilität der Anwendung zu gewährleisten. Zum Beispiel wird beim Aktualisieren von Daten in BC auf mögliche API-Fehler reagiert:

```
try:
    update_data = {
        "vmDeliveredOn": current_timestamp
    }

    endpoint = f'salesShipmentHeaders({systemId})'

    response = self._bc.patch(
        endpoint=endpoint,
        payload=update_data,
        api="api/vonmaehlen/zeitfracht/v1.0"
    )

    if isinstance(response, dict):
        if "error" in response:
            logging.error(f"Failed to update vmDeliveredOn for tracking number {tracking_no}. Error: {response['error']}")
        else:
            logging.info(f"Successfully updated vmDeliveredOn for systemId {systemId}. Calling update_to_bc.")
            self.update_to_bc(tracking_no, current_timestamp)
    else:
        logging.error(f"Unexpected response format: {response}")

except BusinessCentralException as e:
    logging.error(f"Business Central API error when updating tracking number {tracking_no}: {e}")
```

Abbildung 5: Fehlerbehandlung bei API-Aufrufen

Die Klasse *Zf\_bc\_tracking* in *trackings.py* implementiert die Logik zum Einlesen und Verarbeiten der Trackingdaten von Zeitfracht. Zusammenfassend wurde die Geschäftslogik unter Verwendung von Python und verschiedenen Bibliotheken wie SQLAlchemy für den Datenbankzugriff, Logging für die Protokollierung und benutzerdefinierten Integrationen für die Anbindung externer Systeme realisiert.

## 6 Abnahmephase

### 6.1 Testphase

Alle Systeme wurden unter realistischen Bedingungen in einer speziell dafür eingerichteten Entwicklungsumgebung getestet. Diese Umgebung, die regelmäßig für Integration-Tests genutzt wird, stellt sicher, dass die Systeme stabil und zuverlässig funktionieren.

Zum Testsystem gehören unter anderem:

- Microsoft Dynamics 365 Business Central
- PostgreSQL-Datenbank (siehe Anhang)
- Apache Airflow für automatisierte Python-Skripte (siehe Anhang)
- SFTP-Server für den Datentransfer

#### 6.1.1 Tests mit dem Skript *trackings.py*

Die Tests bezogen sich hauptsächlich auf die folgenden Aspekte:

##### 1. Upload von Dummy-Daten auf den SFTP-Server

- Im Rahmen der Tests wurden regelmäßig Dummy-Daten auf den SFTP-Server geladen, um den vollständigen Prozess von der Datenerfassung bis zur Weiterverarbeitung zu simulieren.
- Diese Dummy-Daten bestanden aus korrekt formatierten Tracking-Datensätzen sowie absichtlich fehlerhaften Datensätzen, um die Robustheit der Validierungslogik zu prüfen.

##### 2. Abholung und Validierung der Daten durch das Skript

- Das Skript *trackings.py* wurde verwendet, um die hochgeladenen Dummy-Daten vom SFTP-Server abzuholen und in die PostgreSQL-Datenbank zu speichern.
- Eine umfassende Validierung der Daten wurde durchgeführt, um sicherzustellen, dass nur vollständige und korrekte Datensätze verarbeitet werden.
- Es wurden insbesondere folgende Validierungsszenarien getestet:
  - Vollständige und korrekte Datensätze: Prüfung, ob die Daten korrekt in die Datenbank eingefügt werden.
  - Fehlerhafte oder unvollständige Datensätze: Sicherstellung, dass fehlerhafte Datensätze (z.B. fehlende Pflichtfelder, falsche Datentypen) nicht in die Datenbank übernommen werden, sondern eine entsprechende Fehlermeldung generieren.

- Doppelte Datensätze: Dummy-Daten, die doppelt vorhanden waren, wurden hochgeladen, um sicherzustellen, dass das Skript diese erkennt und nicht erneut in die Datenbank einfügt.

### 3. Löschen der erfolgreich gelesenen Daten auf dem SFTP-Server

- Nach der erfolgreichen Verarbeitung der Dummy-Daten wurde das automatische Löschen der verarbeiteten Dateien auf dem SFTP-Server getestet.
- Es wurde sichergestellt, dass nur die Daten, die erfolgreich verarbeitet wurden, gelöscht wurden, während nicht verarbeitete oder fehlerhafte Dateien weiterhin auf dem SFTP-Server verbleiben.
- Das Skript wurde darauf getestet, korrekt zwischen verarbeiteten und nicht verarbeiteten Dateien zu unterscheiden und die Löschvorgänge nur bei Erfolg auszuführen.

### 4. Fehlerhafte Daten gezielt einfügen

- Es wurden gezielt falsche Daten auf den SFTP-Server hochgeladen, um zu prüfen, wie die Skripte auf inkonsistente Informationen reagieren.
- Ziel dieser Tests war es, sicherzustellen, dass das System solche fehlerhaften Daten erkennt und entsprechend darauf reagiert, z.B. durch Protokollierung im Log oder durch Benachrichtigung der zuständigen Personen.

### 5. Verifizierung der Datenintegrität

- Die Testdaten wurden genutzt, um die Datenintegrität in der PostgreSQL-Datenbank zu überprüfen.
- Es wurde sichergestellt, dass nur die validierten Daten korrekt in der Datenbank gespeichert wurden und dass keine fehlerhaften Daten Einfluss auf die vorhandenen Datensätze hatten.
- Dies beinhaltete auch die Konsistenzprüfung der Datenbank nach der Verarbeitung.

## 6.1.2 Tests mit dem Skript *bc\_db.py*

Die Testphase beinhaltet die folgenden Aspekte:

### 1. Abfrage und Verarbeitung der Tracking-Daten

- Das Skript *bc\_db.py* wurde getestet, um sicherzustellen, dass es in der Lage ist, die Tracking-Daten korrekt aus der PostgreSQL-Datenbank zu lesen und zu verarbeiten.
- Es wurden mehrere Testläufe durchgeführt, um sicherzustellen, dass das Skript die relevanten Daten korrekt abfragt und für die Weiterverarbeitung vorbereitet.

- Verarbeitungsszenarien: Verschiedene Szenarien, wie z.B. Datensätze mit unterschiedlichen Statuscodes, wurden geprüft, um sicherzustellen, dass das System die jeweilige Geschäftslogik korrekt anwendet.

## 2. Synchronisation mit Microsoft Business Central

- Ein wesentlicher Bestandteil des Skripts ist die Synchronisation der Tracking-Daten mit dem ERP-System Microsoft Business Central.
- Die Funktionalität wurde getestet, indem die API-Aufrufe mit verschiedenen Testdaten ausgeführt wurden.
- Es wurde insbesondere geprüft, dass die Statusinformationen korrekt in Business Central aktualisiert werden und dass die API fehlerfrei funktioniert.
- Hierzu wurden auch absichtlich falsche API-Daten verwendet, um sicherzustellen, dass das System korrekt auf Fehlermeldungen reagiert und diese verarbeitet.

## 3. Fehlerbehandlung und Benachrichtigung

- Das Skript wurde getestet, um sicherzustellen, dass es robust gegenüber Fehlern ist, die bei der Verarbeitung der Daten auftreten können.
- Es wurden Szenarien simuliert, in denen die API-Verbindung fehlschlägt, um zu prüfen, wie das Skript darauf reagiert.
- Zudem wurde die Benachrichtigungslogik getestet: Wenn eine Sendung länger als die definierte Anzahl an Tagen in einem bestimmten Status verweilt, wird automatisch eine Benachrichtigung an die zuständigen Mitarbeiter gesendet.
- Die Testläufe stellten sicher, dass diese E-Mails korrekt ausgelöst werden.

## 4. Korrekte Handhabung von Ausnahmefällen

- Es wurden Tests durchgeführt, um sicherzustellen, dass alle möglichen Ausnahmefälle korrekt behandelt werden, z.B. wenn die Datenbankverbindung vorübergehend nicht verfügbar ist oder unerwartete Datenformate auf der API-Ebene auftreten.
- Die Protokollierung der Fehler wurde ebenfalls geprüft, um sicherzustellen, dass alle Fehler im Log korrekt vermerkt werden, damit diese bei Bedarf nachvollzogen und behoben werden können.

## 5. End-to-End-Test der gesamten Datenpipeline

- Schließlich wurde ein End-to-End-Test der gesamten Datenpipeline durchgeführt, bei dem beide Skripte (*trackings.py* und *bc\_db.py*) zusammenarbeiteten.

- Ziel war es sicherzustellen, dass die Daten von der Erfassung über den SFTP-Server, die Validierung und Speicherung in der PostgreSQL-Datenbank bis hin zur Synchronisation mit Business Central nahtlos verarbeitet werden.
- Diese Tests beinhalteten auch die Simulation realer Arbeitsbedingungen, um die Performance und Zuverlässigkeit des Systems unter hoher Last zu prüfen.

## 7 Einführungsphase

In der Einführungsphase wurde das System in die produktive Umgebung überführt, um den automatisierten Ablauf der täglichen Prüfungen von Warensendungen sicherzustellen. Die folgenden Schritte wurden im Rahmen dieser Phase durchgeführt:

### 7.1 Einrichtung neuer Docker-Container auf dem Hetzner-Server

Auf dem Hetzner-Server wurden zwei neue Docker-Container erstellt, um eine isolierte und skalierbare Umgebung für die Anwendung bereitzustellen. Die Verwendung von Docker ermöglicht eine effiziente Verwaltung und einfache Wartung der benötigten Services sowie eine klare Trennung zwischen den verschiedenen Komponenten des Systems.

### 7.2 Installation und Konfiguration von Apache Airflow

In einem der Docker-Container wurde Apache Airflow installiert und konfiguriert. Apache Airflow dient hier als Workflow-Management-System, das die zeitgesteuerte und automatisierte Ausführung von Aufgaben sicherstellt. Im Airflow-Container wurden zwei zentrale Python-Skripte *trackings.py* und *bc\_db.py* integriert, die täglich als sogenannte "DAGs" (DirectedAcyclicGraphs) ausgeführt werden. Siehe Anhang A3 Abbildung 7. Durch die zeitgesteuerte Ausführung in Airflow werden diese Skripte täglich zu festgelegten Zeiten gestartet und gewährleisten somit eine kontinuierliche Aktualisierung der Trackingdaten.

### 7.3 Starten und Verwalten des PostgreSQL-Datenbankservers

Im zweiten Docker-Container wurde eine PostgreSQL-Datenbank eingerichtet, die als zentrales Speichermedium für die erfassten Trackingdaten fungiert. Diese Datenbank speichert die täglichen Updates der Trackingnummern und ermöglicht den Zugriff auf aktuelle sowie historische Daten für die weiteren Prüfungen und Analysen.

## 7.4 Verbindung zwischen den Containern

Die Docker-Container für Airflow und PostgreSQL sind so konfiguriert, dass sie nahtlos miteinander kommunizieren können. Apache Airflow kann so problemlos auf die PostgreSQL-Datenbank zugreifen, um die Datenbankabfragen und -speicherungen auszuführen, die für das tägliche Tracking der Warensendungen erforderlich sind. Die Netzwerkverbindung zwischen den Containern sorgt dafür, dass die Daten sicher und effizient verarbeitet werden können.

# 8 Dokumentation

## 8.1 Zielgruppe der Entwicklerdokumentation

Die Dokumentation richtet sich an administratives Personal der VONMÄHLEN GmbH. Der Umgang mit Apache Airflow und PostgreSQL ist diesem Personenkreis geläufig. Python wird als unternehmensübliche Programmiersprache genutzt. Die Dokumentation ist sowohl im Markdown-Format im GIT-Repository (*README.md*) als auch im internen Unternehmens-Wiki verfügbar.

## 8.2 Inhalt der Entwicklerdokumentation

Die Entwicklerdokumentation umfasst:

- Integration in Apache Airflow: Anleitung zur Einbindung der Software in die bestehende Airflow-Instanz.
- Funktionsweise der Software: Detaillierte Beschreibung der Anwendung und ihrer Hauptfunktionen.
- Fehlermeldungen und Maßnahmen: Übersicht der möglichen Fehlermeldungen und empfohlene Vorgehensweisen zur Fehlerbehebung.
- Erweiterungsmöglichkeiten: Anleitung zur Anpassung der Software bei Änderungen der API des Versanddienstleisters, wie z.B. neue Status-Codes oder geänderte CSV-Schemata.

### 8.3 Code Dokumentation

Der Programmcode enthält umfassende Python-Docstrings, die die Funktionen und Klassen erläutern. Diese richten sich an Personen, die den erstellten Programmcode anpassen oder besser verstehen möchten. Ergänzend dazu finden sich im Anhang:

- A9: Entwicklerdokumentation (Auszug Wiki)
- A10: Entwicklerdokumentation (Auszug Git)
- A6: Programmcode (Auszug *bc\_db.py*)
- A7: Programmcode (Auszug *trackings.py*)

## 9 Fazit

### 9.1 Soll-/Ist-Vergleich

Das Projektziel, ein automatisiertes System zur Überwachung und Benachrichtigung von Warensendungen zu entwickeln, wurde erfolgreich erreicht. Während des Projekts kam es jedoch zu zeitlichen Abweichungen von der ursprünglichen Planung. Besonders die Analyse- und Implementierungsphase nahmen mehr Zeit in Anspruch als zunächst kalkuliert. Die genauen Gründe für diese Verzögerungen sind der detaillierten Zeitplanung Anhang A.2: Komplette Zeitplanung auf Seite ii zu entnehmen. Solche Abweichungen sind im Rahmen komplexer Projekte nicht ungewöhnlich und wurden durch eine flexible Anpassung des Projektplans erfolgreich aufgefangen. Insgesamt konnte das Projekt trotz dieser Anpassungen im Rahmen der geplanten Ressourcen abgeschlossen werden. Die gewonnenen Erkenntnisse im Umgang mit unvorhergesehenem Mehraufwand und die Fähigkeit zur Anpassung der Planung sind wertvolle Erfahrungen für zukünftige Projekte.

## **Literaturverzeichnis**

### **Fittkau und Ruf 2008**

FITTKAU, Thomas ; RUF, Walter: *Ganzheitliches IT-Projektmanagement*. München : Oldenbourg Wissenschaftsverlag, 2008. – ISBN 978-3-486-58567-4



## A Anhang

### A.1 Lastenheft

#### **Lastenheft**

##### **Projekt: Automatisierte Überwachung der Warensendungen**

##### **1. Projektübersicht**

Projektname: Automatisierte Überwachung der Warensendungen

Auftraggeber: VONMÄHLEN GmbH, SCM-Abteilung

Projektverantwortlicher: Jens Lange

Ziel des Projekts: Entwicklung eines Systems zur automatisierten Überwachung und Benachrichtigung von Warensendungen zur Optimierung der Liefertransparenz und Effizienz der SCM-Abteilung.

##### **2. Ausgangssituation**

Derzeit erfolgt die Überwachung der Warensendungen manuell. Die Mitarbeiter der SCM-Abteilung prüfen täglich den Status offener Sendungen über das Webportal des jeweiligen Versanddienstleisters und kontrollieren, ob es Verzögerungen oder Probleme gibt. Dieser manuelle Prozess ist zeitaufwendig und fehleranfällig. Durch eine Automatisierung soll der Prozess effizienter und zuverlässiger gestaltet werden.

##### **3. Zielsetzung**

Die SCM-Abteilung wünscht sich eine automatisierte Überwachung aller Warensendungen, die sowohl den Status der Sendungen überwacht als auch bei Fehlern oder Verzögerungen benachrichtigt. Ziel ist es, durch eine automatische Prüfung und Benachrichtigung die Transparenz im Lieferprozess zu erhöhen, Zeitaufwand zu minimieren und Fehler zu reduzieren.

##### **4. Funktionale Anforderungen**

###### **4.1 Automatisiertes Tracking**

- Das System soll den Status aller Warensendungen täglich automatisch abrufen.
- Trackingdaten vom externen Versanddienstleistern ZF sollen verarbeitet werden können.

###### **4.2 Datenbankintegration**

- Die Trackingdaten sollen in einer PostgreSQL-Datenbank gespeichert werden.
- Das System soll den aktuellen Status jeder Sendung dokumentieren und den Verlauf historisch abbilden.
- Die Datenbank muss Abfragen zur Filterung nach Status, Versanddienstleister und Lieferdatum ermöglichen.

###### **4.3 Abgleich mit ERP-System**

- Das System soll die Trackingdaten mit den Lieferaufträgen im ERP-System (Microsoft Business Central) abgleichen.
- Es soll ersichtlich sein, wann eine Sendung zugestellt wurde, um die Rechnungsbuchung automatisch einleiten zu können.

###### **4.4 Benachrichtigungssystem**

- Bei Verzögerungen oder Fehlermeldungen soll eine automatische E-Mail-Benachrichtigung an die SCM-Abteilung gesendet werden.
- Verzögerungen sollen ab einer festgelegten Dauer (z. B. 3 Tage nach dem geplanten Liefertermin) gemeldet werden.

Abbildung 6: Auszug Lastenheft

## A.2 Komplette Zeitplanung

<b>Analysephase</b>	<b>10 h</b>
1. Erhebung der Anforderungen für die Speicherung und Verarbeitung der Trackingdaten	3 h
2. Analyse der bestehenden Systeme und Identifikation von Schwachstellen	4 h
3. Ermittlung der Projektkosten und Berechnung der Einsparungen	2 h
4. Erstellung des Wirtschaftlichkeitsberichts	1 h
<b>Entwurfsphase</b>	<b>13 h</b>
1. Entwurf der Architektur für die Datenbank und Integration mit dem ERP-System (Architekturdiagramm)	5 h
2. Design der Datenbankstruktur und der Schnittstellen für die Datenübertragung	5 h
3. Anbindung an das Monitoring- und Benachrichtigungssystem	3 h
<b>Implementierungsphase</b>	<b>38 h</b>
1. Implementierung der Schnittstellen zur Abholung und Speicherung der Trackingdaten in der Datenbank	10 h
2. Entwicklung der Integration des ERP-Systems zur Synchronisation und Datenabgleich	10 h
3. Programmierung der Prüfalgorithmen für die Trackingdaten und Einrichtung des Monitoring-Systems mit Python	8 h
4. Implementierung des Benachrichtigungssystems für die Fachabteilung	4 h
5. Integration aller Komponenten und Durchführung erster Funktionstests	3 h
6. Debugging und Behebung von initialen Fehlern	3 h
<b>Testphase</b>	<b>8 h</b>
1. Durchführung von Funktionstests zur Sicherstellung der korrekten Datenverarbeitung und -integration	4 h
2. Validierung der Datenintegrität und Konsistenzprüfungen	2 h
3. Testen des Monitoring-Systems und der Benachrichtigungsfunktionalität	2 h
<b>Einführungsphase</b>	<b>3 h</b>
1. Deployment des Systems auf die Produktionsumgebung und Durchführung der finalen Systemkonfiguration	2 h
2. Schulung der Fachabteilung zur Nutzung des neuen Systems	1 h
<b>Dokumentation</b>	<b>8 h</b>
1. Erstellung der technischen Dokumentation, einschließlich Systemarchitektur und Datenflussdiagramme	8 h
<b>Gesamt</b>	<b>80 h</b>

### A.3 Zeitplanung nach Anpassungen

Phase	Geplant	Tatsächlich	Differenz
Analysephase	10 h	12 h	2 h
Entwurfsphase	13 h	12 h	-1 h
Implementierungsphase	38 h	38 h	
Testphase	8 h	6 h	-2 h
Einführungsphase	3 h	3 h	
Dokumentation Erstellen	8 h	9 h	1 h
Gesamt	80 h	80 h	