



Abschlussprüfung Sommer 2025

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Datei Export auf einem cloudbasierten SharePoint

Export von RTC-Daten als Excel Datei auf einem cloudbasierten SharePoint

Abgabedatum: Köln, den dd.04. 2025

Prüfungsbewerber:

Prüflingsnummer: 5002364305

Robin Frank

Ahornweg 8

41540 Dormagen

The INEOS logo is displayed. It consists of the word "INEOS" in a bold, dark blue sans-serif font. The letter "E" is stylized with a large circle on its right side.

Ausbildungsbetrieb:

INEOS Manufacturing Deutschland GmbH

Alte Str. 201

50769 Köln

Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Tabellenverzeichnis.....	IV
Listings	IV
Abkürzungsverzeichnis	V
1. Einleitung.....	1
1.1 Projektumfeld	1
1.2 Projektziel	1
1.3 Projektbegründung.....	1
1.4 Projektschnittstellen.....	1
1.5 Projektabgrenzung	1
1.6 Use-Case	2
1.6.1 Prozessabläufe.....	2
1.6.2 Anwendungsfälle	2
2. Projektplanung	2
2.1 Projektphasen.....	2
2.2 Ressourcenplanung	3
2.3 Entwicklungsprozess	3
3. Analysephase.....	3
3.1 Ist-Analyse	3
3.2 Wirtschaftlichkeitsanalyse	3
3.2.1 Projektkosten.....	3
3.2.2 Amortisationsdauer	4
3.3 Anwendungsfälle	5
3.4 Lastenheft	5
4. Entwurfsphase.....	6
4.1 Zielplattform	6
4.2 Architekturdesign	6
4.3 Entwurf der Benutzeroberfläche	6
4.4 Datenmodell	7
4.5 Geschäftslogik.....	7
4.6 Maßnahmen zur Qualitätssicherung	7
4.7 Pflichtenheft	8
5. Implementierungsphase.....	8
5.1 Implementierung der Datenstrukturen.....	8
5.2 Implementierung der Benutzeroberfläche.....	8

5.3 Implementierung der Geschäftslogik9
5.3.1 RTC-Controller9
5.3.2 RTC-Service9
5.3.3 SharePoint-Service.....	.9
6. Abnahmephase.....	10
7. Einführungsphase.....	10
8. Dokumentation.....	11
9. Fazit	11
9.1 Soll-/Ist-Vergleich.....	11
9.2 Gesammelte Erfahrungen	12
9.3 Ausblick	12
Literaturverzeichnis	13
A Anhang	i
A.1 Detaillierte Zeitplanung.....	i
A.2 Verwendete Ressourcen	ii
A.3 Amortisation	iii
A.4 Use-Case-Diagramm.....	iv
A.5 Lastenheft (Auszug).....	v
A.6 Entwurf der Benutzeroberfläche	vi
A.7 Entity-Relationship-Modell	vii
A.8 Klassendiagramm	viii
A.9 Logeinträge.....	ix
A.10 Pflichtenheft.....	x
A.11 Excel-Struktur.....	xi
A.12 Datenmodell.....	xii
A.13 UI-JavaScript.....	xiii
A.14 Aktivitätsdiagramm	xv
A.15 RTC-Controller-Code	xvi
A.16 RTC-Service-Code	xvii
A.17 Excel-Formatierungsmethoden.....	xviii
A.18 Abhängigkeiten	xxi
A.19 SharePoint-Service-Code.....	xxii
A.20 Token Management	xxiv
A.21 SharePoint-API Konfiguration.....	xxv
A.22 Systemtest.....	xxvi
A.23 Usability Test	xxvii

A.24	Benutzeraktzeptanztest	xxviii
A.25	Benutzerdokumentation (Auszug)	xxix
A.26	Entwicklerdokumentation.....	xxxii

Abbildungsverzeichnis

Grafische Darstellung der Amortisation	iii
Use-Case-Diagramm	iv
Mockup des UIs	vi
Entity-Relationship-Modell.....	vii
Klassendiagramm	viii
Exemplarische Logeinträge.....	ix
Excel Struktur	xi
Datenmodell	xii
Datenflussmodell.....	xv
Auszug aus der Benutzerdokumentation	xxx
Entwicklerdokumentation	xxxi

Tabellenverzeichnis

Grobe Zeitplanung	2
Kostenaufstellung	4
Zeiteinsparung pro Jahr	5
Soll-/Ist-Vergleich.....	12
Detaillierte Zeitplanung	i
Lastenheft-Anforderungen	v

Listings

UI-JavaScript	xiv
RTC-Controller-Code.....	xvi
RTC-Service-Code	xvii
Excel Formatierungsmethoden	xx
Abhängigkeiten.....	xxi
SharePoint-Service-Code	xxiii
Token Management.....	xxiv
SharePoint-API Konfiguration	xxv

Abkürzungsverzeichnis

RTC	-	Rail Tank Car
RTCMS	-	Rail Tank Car Management System
SQL	-	Structured Query Language
HTML	-	Hypertext Markup Language
CSS	-	Cascading Style Sheets
API	-	Application Programming Interface
REST-API	-	Representational State Transfer - Application Programming Interface
IDE	-	Integrated Development Environment
CSV	-	Comma-Separated Values
JSON	-	JavaScript Object Notation

1. Einleitung

1.1 Projektumfeld

Die RTC-Anwendung wird zur Verwaltung von Kesselwagendaten verwendet und ist ein zentrales Instrument zur Erfassung und Bereitstellung von Daten. Das Projekt zielt darauf ab, eine automatisierte Funktion zu schaffen, die den Export von Daten aus der RTC-Anwendung in eine Excel-Datei ermöglicht und diese direkt auf einem cloudbasierten SharePoint im Tenant der INEOS Gruppe speichert. Dadurch werden manuelle Arbeitsabläufe überflüssig und die Effizienz der Datenbereitstellung erhöht.

1.2 Projektziel

Das Ziel des Projekts ist die Entwicklung einer automatisierten Exportfunktion für die RTC-Daten. Diese Funktion wird die Daten im Excel-Format exportieren, an die Anforderungen der internen und externen Nutzer angepasst formatieren und automatisiert auf einem SharePoint speichern. Dadurch sollen Zeitaufwand und Fehlerquellen minimiert sowie die Nutzung der Daten erleichtert werden.

1.3 Projektbegründung

Derzeit erfordert die Bereitstellung der RTC-Daten einen hohen manuellen Aufwand, was die Effizienz der Prozesse beeinträchtigt, und Fehlerpotenziale erhöht. Es besteht kein automatisierter Prozess, um diese Daten zu speichern und internen sowie externen Nutzern bereitzustellen. Externe Nutzer bekommen keinen Zugriff auf die Webanwendungen der INEOS, weshalb eine Anbindung auf einen cloudbasierten SharePoint programmiert werden soll. Das ermöglicht den externen Nutzern auf die Daten über den SharePoint zuzugreifen. Zudem wird nicht nur die Effizienz gesteigert, sondern auch die Genauigkeit und Verfügbarkeit der Daten verbessert. Dies hat positive Auswirkungen auf die Entscheidungsfindung und die Zusammenarbeit mit internen und externen Nutzern.

1.4 Projektschnittstellen

Das Projekt beinhaltet unterschiedliche technische und organisatorische Schnittstellen, die einen reibungslosen Datenfluss gewährleisten sollen. Als ersten Schritt ist es geplant, Daten aus der gegenwärtigen PostgreSQL-Datenbank der RTC-Anwendung zu exportieren. Die exportierten Daten werden anschließend auf einem cloudbasierten SharePoint gespeichert, unter Verwendung der Microsoft Graph REST-API. Zur Implementierung der Exportfunktion wird Apache POI verwendet, um die Excel-Datei zu erstellen. Um die Zugriffsberechtigung auf die Graph-API zu garantieren, ist es notwendig, dass die Systemadministration dem Systemkonto der Anwendung Vollzugriff auf das gewünschte Verzeichnis vergibt.

1.5 Projektbegrenzung

Das Projekt umfasst die Entwicklung und Implementierung der Exportfunktion sowie die Speicherung der Daten auf dem SharePoint. Es wird keine Entwicklung neuer Funktionen innerhalb der RTC-Anwendung außerhalb des Datenexports vorgenommen. Ebenso liegt die Nutzung der exportierten Daten durch externe Nutzer außerhalb des Projektumfangs.

1.6 Use-Case

1.6.1 Prozessabläufe

Der Datenexportprozess beginnt, wenn ein Nutzer in der RTC-Anwendung den Export initiiert. Basierend auf den benutzerdefinierten Anforderungen generiert die Anwendung daraufhin eine Excel-Datei mit den relevanten RTC-Daten. Die erstellte Datei wird anschließend automatisiert auf dem cloudbasierten SharePoint abgelegt.

1.6.2 Anwendungsfälle

Für die interne Nutzung können Mitarbeiter des Supply Chain Teams die exportierten Excel-Dateien zur umfassenden Analyse und gezielten Optimierung interner Prozesse heranziehen. Auf der externen Seite ermöglicht der Zugriff über SharePoint den Nutzern eine direkte Nutzung der Daten, um fundierte Transport- und Logistikentscheidungen zu treffen. Ein wesentlicher Vorteil des Systems liegt in der Automatisierung, die das Risiko von Fehlern, die typischerweise bei manuellen Prozessen auftreten können, signifikant reduziert. Durch diese integrierte Lösung wird nicht nur die Effizienz gesteigert, sondern auch die Datenqualität und Zuverlässigkeit des Informationsaustauschs verbessert.

2. Projektplanung

2.1 Projektphasen

Die IHK Köln schreibt vor, dass für die Umsetzung des Projektes 80 Stunden zur Verfügung stehen. Vor dem Projektstart wurde eine Aufteilung in verschiedene Phasen vorgenommen, die den gesamten Softwareentwicklungsprozess abdecken. Tabelle 1: Grobe Zeitplanung bietet eine Zeitplanung mit den Hauptphasen. Im Anhang A.1 Detaillierte Zeitplanung auf Seite [i](#). ist eine detailliertere Zeitplanung mit den einzelnen Schritten der verschiedenen Phasen zu finden.

Projektphase	Zeit
Analysephase	6 h
Entwurfsphase	9 h
Implementierungsphase	28 h
Testphase	12 h
Dokumentationsphase	10 h
Abnahme & Deployment	5 h
Pufferzeit	10 h
Gesamtaufwand	80 h

Tabelle 1: Grobe Zeitplanung

2.2 Ressourcenplanung

Im Bereich der personellen Ressourcen ist ein Auszubildender vorgesehen, der die Hauptverantwortung für die Umsetzung der Exportfunktion und API-Schnittstelle trägt. Mitarbeiter der Rail/Road Abteilung werden mit der Durchführung umfassender Tests betraut, während ein Senior Manager die Aufgabe des Code Reviews für die entwickelten Funktionen übernimmt. Auf der technischen Seite bildet eine Entwicklungsumgebung mit Apache POI die Grundlage für die Implementierung. Zusätzlich ist ein Zugriff auf die RTC-Anwendung und die zugehörige PostgreSQL-Datenbank sowie ein Zugang zum cloudbasierten SharePoint erforderlich. Diese strategische Ressourcenverteilung gewährleistet eine effiziente und strukturierte Projektdurchführung mit klaren Verantwortlichkeiten und notwendigen technischen Voraussetzungen.

2.3 Entwicklungsprozess

Der Entwicklungsprozess orientiert sich am traditionellen Wasserfallmodell mit deutlich abgrenzbaren Phasen. Bevor die nächste Phase beginnt, wird die vorherige abgeschlossen. Um den Fortschritt zu kontrollieren und die Qualität zu garantieren, werden Meilensteine festgelegt.

3. Analysephase

3.1 Ist-Analyse

Wie schon in 1.1 Projektumfeld dargelegt, wird im Rahmen der RTC-Anwendung eine Exportfunktion für den Bereich Rail/Road implementiert, die regelmäßig aktualisierte Kesselwageninformationen für interne und externe Nutzer bereitstellt. Die derzeitige Ist-Analyse offenbart signifikante Schwierigkeiten im bestehenden Prozess.

Aktuell erfolgt die Erstellung von Kesselwagenlisten manuell, wobei Informationen wie Wagennummern, Wagendetails und Produktkennzeichnungen direkt aus der RTC-Datenbank abgerufen und in Excel-Dateien eingetragen werden. Da es zu Eingabefehlern oder unvollständigen Datenübertragungen kommen kann, ist dieser Prozess zeitaufwendig und anfällig für Fehler. Es gibt keinerlei automatisierte Validierung der exportierten Daten.

Interne Mitarbeiter laden die Excel-Dateien nach ihrer Erstellung manuell in ein SharePoint-Verzeichnis hoch, auf das externe Nutzer Zugriff haben. Ein automatisiertes System zur Benachrichtigung über erfolgreiche Uploads existiert nicht. Externe Nutzer haben nur Zugriff auf veraltete oder unvollständige Daten, wenn Uploads fehlerhaft sind oder unterlassen werden.

Zu den weiteren Mängeln gehören das Fehlen einer Versionierung und revisionssicheren Protokollierung der Dateien sowie eine unzureichende Transparenz hinsichtlich des Zeitpunkts der Erstellung und der verantwortlichen Person. Bei hoher Auslastung kommt es aufgrund der starken Abhängigkeit vom verfügbaren Personal besonders häufig zu Verzögerungen oder sogar zum vollständigen Ausbleiben von Datenexporten.

3.2 Wirtschaftlichkeitsanalyse

3.2.1 Projektkosten

Bei der Entwicklung der Exportfunktion für die RTC-Anwendung werden die entstehenden Kosten detailliert berechnet. Weil die realen Personalkosten nicht gänzlich offengelegt werden können, wird die Berechnung auf Grundlage von Stundensätzen vorgenommen, die vom Projektbeauftragten bestimmt wurden. Die

Kosten für die unter 2.2 Ressourcenplanung aufgeführten verwendeten Ressourcen müssen ebenfalls eingeplant werden.

Der Auszubildende erhält einen Stundenlohn von 10,00 €. Für den Mitarbeiter der Rail/Road-Abteilung wurde ein Stundensatz von 85,00 € ermittelt. Der Stundensatz für den Senior Manager liegt bei 275,00 €. Pro Stunde Ressourcennutzung wird zusätzlich zu den individuellen Stundensätzen ein pauschaler Betrag von 15,00 € berechnet.

Das Projekt wird über eine Dauer von 80 Stunden durchgeführt. Tabelle 2: Kostenaufstellung zeigt eine Aufschlüsselung und Auflistung der Kosten für die verschiedenen Aspekte. Die im Rahmen des Projektes entstehenden Gesamtkosten belaufen sich auf 4590,00 €

Vorgang	Zeit	Kosten pro Stunde	Gesamtkosten
Entwicklungskosten (Auszubildender)	80 h	10,00 € + 15,00 € = 25,00 €	2.000,00 €
Testphase (Rail/Road Mitarbeiter)	12 h	85,00 € + 10,00 € + 15,00 € = 110,00 €	1.320,00 €
Code-Review (Senior Manager)	3 h	275,00 € + 10,00 € + 15,00 € = 300,00 €	1.160,00 €
Einführung (Rail/Road Mitarbeiter)	1 h	85,00 € + 10,00 € + 15,00 € = 110,00 €	110,00 €
Gesamtprojektkosten			4.590,00 €

Tabelle 2: Kostenaufstellung

3.2.2 Amortisationsdauer

Im Folgenden wird die Amortisation betrachtet, also die Dauer, bis sich die Entwicklung des Projektes amortisiert. Die Kosten wurden bereits in Tabelle 2: Kostenaufstellung festgelegt. Die automatische Erstellung der Excel-Datei spart Zeit bei der Datenaufbereitung. Auch durch den automatischen Upload und die Validierung der Daten, die der Anwender nicht manuell durchführen muss. In Tabelle 3: Zeiteinsparung pro Jahr sind die daraus resultierenden Zeitersparnisse detailliert aufgeführt. Durch die Beobachtung des Arbeitsprozesses eines Rail/Road-Mitarbeiters wurden die Vorgangszeiten ermittelt.

Für die Berechnung der jährlichen Zeiteinsparung wird angenommen, dass der Export etwa 65-mal im Jahr verwendet wird. Obwohl die Anwendung täglich einen Export durchführt, basiert die Berechnung der Zeiteinsparung auf 65-mal.

Vorgang	IST-Zustand	SOLL-Zustand	Zeitaufwand IST (pro Vorgang)	Zeitaufwand SOLL (pro Vorgang)	Jährliche Zeiteinsparung
Datenaufbereitung	Manuelle Übertragung in Excel	Automatische Generierung	10 Minuten	1 Minute	585 Minuten
Dateiupload	Manuelles Hochladen in SharePoint	Automatischer Upload	15 Minuten	2 Minuten	845 Minuten
Datenvieridierung	Keine automatische Überprüfung	Integrierte Validierungsmec hanismen	30 Minuten	1 Minute	1.885 Minuten
Benachrichtigungen	Keine Statusrückmel dungen	Automatische Benachrichtigu gen	-	1 Minute	-65 Minuten
Gesamteinsparung			55 Minuten pro Export	5 Minuten pro Export	3.250 Minuten

Tabelle 3: Zeiteinsparung pro Jahr

Dadurch ergibt sich eine jährliche Einsparung von $\frac{3.250 \text{ Minuten}}{60 \text{ Minuten/h}} \cdot (85,00 + 15,00) \text{ €/h} = 5416,67 \text{ €}$.

Die Amortisationszeit beträgt also $\frac{4590,00 \text{ €}}{5416,67 \text{ €/Jahr}} \approx 0,85 \text{ Jahre} \approx 10,2 \text{ Monate}$. Eine grafische Darstellung ist im Anhang A.3 Amortisation auf Seite [iii.](#) enthalten. Die ermittelte Zeit lässt also den Schluss zu, dass sich das Projekt aufgrund der erheblichen Einsparung an manueller Arbeit nach 10,2 Monaten amortisiert hat. Daher kann es als wirtschaftlich sinnvoll angesehen und umgesetzt werden.

3.3 Anwendungsfälle

Die Exportfunktion ist eine zentrale Komponente der RTCMS-Anwendung und ermöglicht die automatische sowie manuelle Erstellung und Bereitstellung von Kesselwagenlisten als Excel-Datei. Diese Funktion ist besonders wichtig für interne Nutzer, da sie eine effiziente und zuverlässige Bereitstellung der erforderlichen Daten sicherstellt.

Im folgenden Use-Case-Diagramm A.4 Use-Case-Diagramm auf Seite [iv](#) sind die verschiedenen Anwendungsfälle dargestellt, die von der Exportfunktion abgedeckt werden.

3.4 Lastenheft

Die funktionalen sowie nicht-funktionalen Anforderungen an die Exportfunktion werden im Lastenheft festgelegt. Die Priorisierung erfolgte mithilfe der (MoSCoW)-Methode, um zwischen unbedingt nötiges Muss, wichtigen sollte und optionalen könnte Anforderungen zu differenzieren.

A.5 Lastenheft auf Seite [v](#), fasst die grundlegenden Anforderungen an die Anwendung zusammen. Die Anforderungen sind so festgelegt, dass die wesentlichen Funktionen der Anwendung verlässlich realisiert werden, während Sollte- und Könnte-Funktionen optionale Ergänzungen sind.

Es muss gewährleistet werden, dass die Kesselwagenliste täglich automatisch erstellt und bereitgestellt wird. Fehlertoleranz, Benutzerfreundlichkeit und Sicherheit sind entscheidende Anforderungen für eine erfolgreiche Implementierung.

Die Verwendung von SharePoint als Ablageort garantiert, dass alle autorisierten Nutzer Zugang zu den aktuellen Daten haben.

Die Anwendung kann durch zukünftige Erweiterungen wie zusätzliche Exportformate oder E-Mail-Benachrichtigungen weiter optimiert werden.

4. Entwurfsphase

4.1 Zielplattform

In die bereits bestehende RTCMS-Anwendung wird die Exportfunktion integriert. Die Zielplattform wurde unter Berücksichtigung spezifischer Anforderungen ausgewählt. Die Exportfunktion wird in Groovy umgesetzt, um eine nahtlose Integration zu gewährleisten, da die RTCMS-Anwendung mit Groovy und dem Grails Framework entwickelt wurde. Die Applikation verwendet eine objektorientierte PostgreSQL-Datenbank, aus der die Exportfunktion die relevanten Datensätze für den Export abholt. Die Architektur orientiert sich an einem Client-Server-Modell und beruht auf einer REST-API, über die die Exportfunktion die erforderlichen Daten abruft. Die Ressourcen des Servers, auf dem die RTCMS-Anwendung läuft, sind für die Verarbeitung der Exportfunktion ausreichend. Daher sind zusätzliche Anpassungen der Hardware nicht notwendig.

4.2 Architekturentwurf

Bei der Implementierung der Exportfunktion wird das Model-View-Controller (MVC)-Architekturmuster verwendet, dabei handelt es sich um das Programmierparadigma von Grails, um eine deutliche Trennung der logischen Schichten der Anwendung sicherzustellen. Die Domain wickelt die Datenbankabfragen für den Export ab und fördert so eine effiziente Datenverarbeitung. Die Benutzeroberfläche beinhaltet eine Export-Schaltfläche und eine Anzeige für den Exportstatus, wodurch der Benutzer über den aktuellen Fortschritt informiert wird. Eine REST-Schnittstelle steuert den Exportvorgang.

Die Excel-Datei wird mit Hilfe der Apache POI-Bibliothek erstellt, da sie eine umfassende Unterstützung für verschiedene Excel-Formate bietet und bereits in anderen Anwendungen erfolgreich verwendet wurde.

4.3 Entwurf der Benutzeroberfläche

Die Exportfunktion wird mit einer Benutzeroberfläche in Form eines Webinterfaces entwickelt, das reibungslos in das bestehende RTCMS-Frontend integriert wird. Die Gestaltung folgt den Designrichtlinien der RTCMS-Anwendung, um eine einheitliche Benutzererfahrung zu gewährleisten. Der Benutzer hat mehrere Funktionen zur Bedienung. Eine Export-Taste bietet die Möglichkeit, den Exportprozess manuell zu starten. Ein Statussymbol gibt visuelles Feedback zum Fortschritt und Ergebnis des Exports. Darüber hinaus ermöglicht eine Historienliste den Überblick über die zuletzt erstellten Exportdateien. Erste Entwürfe des Designs wurden in Form eines Mockups erstellt, um sicherzustellen, dass es den Anforderungen der Nutzer entspricht. Dieses Mockup ist im Anhang A.6 Entwurf der Benutzeroberfläche auf Seite [vi](#) zu finden.

4.4 Datenmodell

Das Datenmodell der Exportfunktion gründet sich auf der bestehenden RTCMS-Datenbank und wurde um eine Entität zur Verwaltung des Exportprozesses ergänzt. Die Entität „graphapi“ speichert Daten wie name, last_successful_upload und last_error_message, um diese später im UI dem Anwender zurückzugeben. Um sicherzustellen, dass Exporte nachverfolgt werden können und um den Zugriff auf Dateien gemäß den Benutzerrechten zu steuern, verknüpft die Entität User diese Exporte mit den entsprechenden Benutzern. Ein detailliertes Entity-Relationship-Modell (ERM) nach der Chen Notation, ist im Anhang A.7 Entity-Relationship-Modell auf Seite [vii](#) zu finden und zeigt die genauen Beziehungen zwischen diesen Entitäten auf.

4.5 Geschäftslogik

Die Geschäftslogik der Exportfunktion legt die grundlegenden Abläufe fest, um die Exportprozesse zuverlässig und effizient durchzuführen. In Abschnitt 4.2 Architekturdesign wurden bereits der grundsätzliche Aufbau der Anwendung und die beteiligten Komponenten beschrieben. Die Implementierung basiert auf einer eindeutigen Trennung zwischen dem Zugriff auf Daten, deren Verarbeitung und der Bereitstellung der Exportdateien.

Der Exportprozess startet mit der Datenextraktion, bei der relevante Datensätze aus der PostgreSQL-Datenbank abgerufen und anhand festgelegter Kriterien gefiltert werden. Daraufhin werden die extrahierten Daten verarbeitet, indem sie in das notwendige Excel-Exportformat umgewandelt und für die weitere Bearbeitung vorbereitet werden.

In der Phase der Dateierstellung wird die Excel-Datei nach ihrer Verarbeitung erstellt und auf dem Server abgelegt. Durch die Automatisierung des Exportprozesses wird eine termingerechte Durchführung gewährleistet. Der Export wird jeden Tag um 07:00 Uhr automatisch durchgeführt. Ein berechtigter Nutzer kann zudem den Exportprozess manuell über die Benutzeroberfläche starten.

Ein erzeugtes Klassendiagramm dient der nachträglichen Dokumentation der Geschäftslogik und stellt die Struktur der beteiligten Komponenten sowie deren Beziehungen dar. Das Diagramm befindet sich im Anhang A.8 Klassendiagramm auf Seite [viii](#).

4.6 Maßnahmen zur Qualitätssicherung

Um die Qualität der entwickelten Exportfunktion zu garantieren, wurden verschiedene zielgerichtete Maßnahmen ergriffen. Nach Beendigung der Implementierungsphase überprüfte der Senior Manager den gesamten Quellcode gründlich. Der Schwerpunkt lag dabei auf der inhaltlichen Richtigkeit sowie auf der Beachtung bewährter Praktiken und der unternehmensinternen Coding-Standards. Durch diese Kontrolle wurde eine saubere Codestruktur, geeignete Dokumentation und effiziente Umsetzung sichergestellt. Gefundene Probleme wie suboptimale Variablennamen und fehlende Kommentare wurden sofort behoben, was einen wesentlichen Beitrag zur Gesamtqualität des Codes leistete.

Ein weiterer wesentlicher Punkt waren die gründlichen manuellen Tests, die von Angestellten der Rail/Road-Abteilung durchgeführt wurden. Die Tests beinhalteten verschiedene Szenarien, wie den Export mit variierenden Datenmengen, die Kontrolle der richtigen Excel-Formatierung, die Validierung des automatischen Uploads auf SharePoint sowie die Evaluierung der Benutzeroberfläche mit ihren Statusanzeigen. Die Testergebnisse wurden gewissenhaft festgehalten und ermittelte Schwierigkeiten behoben. Direkte Rückmeldungen der Fachabteilung erwiesen sich als besonders wertvoll, da sie sicherstellten, dass die Exportfunktion den tatsächlichen Anforderungen der Endbenutzer entspricht.

Zudem wurden weitreichende Logging- und Monitoring-Mechanismen implementiert, die eine effektive Erfassung von Fehlern und Systeminformationen ermöglichen. Um eine differenzierte Überwachung sicherzustellen, wurde das Logging-System mit verschiedenen Schweregraden (INFO, WARN, ERROR) konfiguriert. Mit detaillierten Kontextinformationen werden bedeutende Ereignisse wie erfolgreiche Exporte, Authentifizierungsprobleme und SharePoint-Zugriffsfehler protokolliert. Nicht nur hilft diese Maßnahme dabei, Probleme im laufenden Betrieb rasch zu erkennen und zu beheben, sie ist auch ein wesentlicher Faktor für die langfristige Wartbarkeit der Anwendung.

Exemplarische Logeinträge zur umgesetzten Qualitätssicherung sind im Anhang A.9 Logeinträge auf Seite [ix](#) festgehalten.

4.7 Pflichtenheft

Ein Pflichtenheft wurde erstellt, das auf den entwickelten Anforderungen basiert, nachdem die Entwurfsphase abgeschlossen war. Es basiert auf dem Lastenheft und umfasst die detaillierte Spezifikation der Exportfunktion. Es bildet die Basis für die Umsetzung und dafür, dass kontrolliert werden kann, ob alle festgelegten Anforderungen eingehalten wurden.

Der Anhang A.10 Pflichtenheft auf Seite [x](#) enthält das vollständige Pflichtenheft.

5. Implementierungsphase

5.1 Implementierung der Datenstrukturen

Zunächst wurde in der Implementierungsphase die erforderliche Datenstruktur für die GraphAPI-Funktionalität eingerichtet. Die GraphAPI-Entität wurde mit den folgenden Attributen implementiert, wie im Entity-Relationship-Modell in A.7 Entity-Relationship-Modell auf Seite [vii](#) dargestellt.

Mit dieser Entität kann der Exportprozess dokumentiert und können Erfolgs- und Fehlerzustände protokolliert werden. Dank der Verbindung zwischen GraphAPI und der bestehenden User-Entität kann nachvollzogen werden, welcher Benutzer einen Export ausgelöst hat.

Die Umsetzung der Datenstrukturen nahm Rücksicht auf die Anforderungen des Lastenhefts (vgl. Anhang A.5 Lastenheft auf Seite [v](#)) und wurde mit dem bestehenden PostgreSQL-Datenbankschema abgestimmt. Zum Zugriff auf die RTC-Daten waren keine neuen Datenstrukturen erforderlich, da die vorhandenen Entitäten der RTCMS-Anwendung verwendet werden konnten. Die Attribute der RTC-Entität, die für den Export von Bedeutung sind, entsprechen den Spalten in der Excel-Datei, wie in Anhang A.11 Excel-Struktur auf Seite [xi](#) dargestellt.

Daraus entstand das entwickelte Datenmodell mit der Integration der GraphAPI-Entität, welches im Anhang A.12 Datenmodell auf Seite [xii](#).

5.2 Implementierung der Benutzeroberfläche

Gemäß dem Mockup wurde die Benutzeroberfläche in Anhang A.6 Entwurf der Benutzeroberfläche auf Seite [vi](#) umgesetzt. Die Einbindung in das vorhandene Frontend der RTCMS-Anwendung wurde durch die Erweiterung der bestehenden Navigation und die Implementierung eines modalen Dialogs realisiert. Die Implementierung beinhaltet die nachfolgenden UI-Komponenten:

In der oberen Card der RTC-Liste ist ein neuer Button mit der Aufschrift „Export“ hinzugekommen. Nur autorisierte Benutzer können diesen Button sehen, mit dem das Export-Modul geöffnet werden kann. Die Zugriffsberechtigung wird durch die bestehenden Benutzerrechte der RTCMS-Anwendung gesteuert.

Mit dem Klick auf den Export-Button erscheint ein modales Fenster, das den Status des Exports in Echtzeit anzeigt und einen Button mit der Aufschrift „Export Now“ enthält. Dieses Modal wurde mit dem Bootstrap-Framework implementiert und in die bestehende GSP (Groovy Server Page) integriert.

Der derzeitige Status des Exports wird im Modal angezeigt. Hierfür kommen die Daten aus der vom Controller bereitgestellten GraphAPI-Entität zum Einsatz. Die Status-Visualisierung wurde mit verschiedenen CSS-Klassen für die unterschiedlichen Zustände umgesetzt. Im Status "In Arbeit" wird ein Ladeindikator mit der spinner-border-Klasse angezeigt. Beim Status "Erfolgreich" erscheint ein grüner Text mit Häkchen unter Verwendung der Klasse text-success. Für den Status "Fehler" wird ein Text in Rot mit Ausrufezeichen dargestellt, wofür die Klasse text-danger verwendet wird.

Der gesamte UI-Code wurde in einer separaten JavaScript-Datei export.js implementiert, die in Anhang A.13 UI-JavaScript auf Seite [xiii](#) zu finden ist.

5.3 Implementierung der Geschäftslogik

Die Umsetzung der Geschäftslogik orientierte sich am Aktivitätsdiagramm in Anhang A.14 Aktivitätsdiagramm auf Seite [xv](#). Die Geschäftslogik wurde in drei Hauptkomponenten unterteilt, die jeweils für verschiedene Aspekte des Exportprozesses zuständig sind.
Die Umsetzung der Geschäftslogik erfolgt nach dem MVC-Architekturmuster, wie im Abschnitt 4.2 Architekturdesign erläutert. Das Klassendiagramm in Anhang A.8 Klassendiagramm auf Seite [viii](#) illustriert die Beziehungen zwischen den implementierten Komponenten.

5.3.1 RTC-Controller

Der RTC-Controller umfasst zwei zentrale Methoden, deren Implementierung in Anhang A.15 RTC-Controller-Code auf Seite [xvi](#) zu finden ist.

Beim manuellen Export durch den Benutzer wird die exportToSharePoint()-Methode aufgerufen, um den Exportprozess in der GraphAPI-Entität zu dokumentieren. Die Methode scheduledExportToSharePoint() ist mit @Scheduled annotiert und wird täglich um 07:00 Uhr automatisch ausgeführt, gemäß der Vorgabe der Anforderung im Lastenheft A.5 Lastenheft auf Seite [v](#).

5.3.2 RTC-Service

Kernlogik für die Excel-Datei-Erstellung wird durch den RTC-Service implementiert. Die Implementierung nutzt die Apache POI-Bibliothek und orientiert sich am Code, der in Anhang A.16 RTC-Service-Code auf Seite [xvii](#) dokumentiert ist.

Der Service enthält verschiedene Hilfsmethoden zur Formatierung der Excel-Datei, die in Anhang A.17 Excel-Formatierungsmethoden auf Seite [xviii](#) detailliert beschrieben sind. Die Methoden erfüllen die im Pflichtenheft (Anhang A.10 Pflichtenheft auf Seite [x](#)) festgelegten Anforderungen an die Formatierung und Darstellung der Daten.

5.3.3 SharePoint-Service

Der SharePoint-Service wird unter Verwendung der in build.gradle (Anhang A.18 Abhängigkeiten auf Seite [xxi](#)) definierten Abhängigkeiten implementiert und basiert auf der Microsoft Graph API. Der SharePoint-Service, dessen vollständige Implementierung in Anhang A.19 SharePoint-Service-Code auf Seite [xxii](#) zu finden ist, bietet folgende Hauptfunktionen.

Der SharePoint-Service setzt ein Token-Caching-System um, das die Leistung verbessert und überflüssige API-Aufrufe verhindert. Die Methode getAccessToken() in Anhang A.20 Token Management auf Seite [xxiv](#) demonstriert die Umsetzung dieses Features.

Die SharePoint-Service-Konfiguration erfolgt anhand der in der application.yml festgelegten Einstellungen, wie auf Seite [xxv](#) in Anhang A.21 SharePoint-API Konfiguration mit Beispieldaten dargestellt. Um die Sicherheit zu steigern, werden alle sensiblen Informationen wie Client-IDs und Secrets über Umgebungsvariablen bereitgestellt.

6. Abnahmephase

Während der Abnahmephase wurde ein mehrstufiger Testprozess durchgeführt, der Systemtests, Usability-Tests und Akzeptanztests umfasste. Die Exportfunktion sollte unter echten Produktionsbedingungen gründlich überprüft werden.

Es wurde ein Systemtest durchgeführt, der auf Seite [xvi](#) im Anhang zu finden ist und sich auf die technische Integritätsprüfung von Datenexport, Formatierung und Schnittstellenperformanz konzentrierte. Zur Überprüfung der Verlässlichkeit und Stabilität des Systems wurden verschiedene Szenarien simuliert. Die Benutzeroberfläche und die Prozesseffizienz wurden im Rahmen der Usability-Tests bei ausgewählten Mitarbeitern überprüft. Im Anhang [A.23 Usability Test](#) auf Seite [xxvii](#) ist ein Auszug des Usability-Tests enthalten.

Die Benutzerfreundlichkeit, die intuitive Handhabung und die Workflow-Optimierung waren die zentralen Aspekte. In den Benutzerakzeptanztests bewerteten Stakeholder aus verschiedenen Abteilungen der Firma die Lösung hinsichtlich funktionaler Anforderungen, Systemkompatibilität und ökonomischem Vorteil. Im Anhang [A.24 Benutzerakzeptanztest](#) auf Seite [xxviii](#) ist ein Auszug des Usability-Tests zu finden.

Alle Testergebnisse waren positiv, es wurden 98% der funktionalen Anforderungen erfüllt und es traten keine kritischen Fehler auf. Die Rückmeldung bestätigte die Leistung und Datenintegrität vollständig. Kleinere Anpassungen an der Datenfilterung und am Systemupload wurden vorgenommen.

Der Projektverantwortliche erteilte nach gründlicher Prüfung die vollständige und uneingeschränkte Freigabe. Der Abnahmeprozess wird durch umfassende Dokumentationsunterlagen gewährleistet. Während der Abnahmephase wurde die technische Funktionalität der Exportlösung sowie ihr erhebliches Potenzial zur Prozessoptimierung bestätigt.

7. Einführungsphase

Um die Qualität des Codes zu garantieren, fängt der Deployment-Prozess mit einem manuellen Code-Review durch den Senior Manager an. Im Anschluss erfolgt die Integration in die bestehende, auf Linux basierende Tomcat-Serverumgebung. Zum Abschluss wird die Exportautomatisierung so eingestellt, dass der Export jeden Tag um 07:00 Uhr automatisch durchgeführt wird.

Das Konzept zur Schulung konzentriert sich auf die Weiterbildung der Mitarbeiter des Supply Chain-Teams und externer Stakeholder. Angestrebt wird, die neue Exportfunktion zu erklären und die Vorzüge der Automatisierung herauszustellen. Die Inhalte der Schulung umfassen die manuelle Auslösung des Exports, den Zugriff auf die exportierten Dateien im SharePoint sowie die Deutung der exportierten Kesselwagenlisten. Zusätzlich wird auf die Behebung von Fehlern eingegangen und es werden die verfügbaren Supportkanäle erklärt. Die Methoden zur Schulung umfassen eine Präsenzschulung für die Hauptnutzer, die Erstellung einer detaillierten Benutzerdokumentation zur Unterstützung des Lernprozesses.

8. Dokumentation

Die Anwendung wird durch die Benutzerdokumentation und Entwicklerdokumentation sowie durch die Projektdokumentation, in der die während der Umsetzung des Projekts durchlaufenen Phasen beschrieben werden, dokumentiert.

Die Benutzerdokumentation hat den Zweck, dem Endnutzer eine schnelle Orientierung im Programm zu bieten und ihm bei Problemen zu helfen. Es umfasst Einzelheiten zur Funktionsweise sowie eine Erläuterung der Navigation innerhalb der Applikation zur Ausführung des Exports auf den SharePoint. Die Verwendung der Export-Funktion wird gründlich dokumentiert und durch Screenshots erklärt. Im Anhang A.25 Benutzerdokumentation (Auszug) auf Seite [xxix](#) befindet sich ein Ausschnitt aus der Benutzerdokumentation.

Alle implementierten Klassen sind in der Entwicklerdokumentation detailliert beschrieben. Dies umfasst deren Eigenschaften und Vorgehensweisen sowie die Beziehungen zwischen ihnen. Diese Dokumentation bietet eine Übersicht und dient als Nachschlagewerk für die Weiterentwicklung oder Anpassung der Anwendung durch einen anderen Entwickler. Der Code wurde mit GroovyDoc dokumentiert. Die HTML kann automatisiert generiert werden, indem der Gradle-Befehl gradle groovydoc verwendet wird. Im Anhang A.23 auf Seite [xxv](#) ist ein Screenshot der erstellten Entwicklerdokumentation zu finden.

9. Fazit

9.1 Soll-/Ist-Vergleich

Zum Abschluss soll ein Rückblick der Projektdurchführung erfolgen. Die Zeitplanung wurde, wie in Tabelle 1: Grobe Zeitplanung dargestellt, größtenteils eingehalten. Das gesamte Vorhaben wurde innerhalb der von der IHK Köln festgelegten 80 Stunden realisiert. Die geringfügigen Abweichungen können wie folgt zusammengefasst werden.

Die Analysephase sowie die Entwurfsphase benötigten etwas mehr Zeit als eingeplant, durch umfangreiches Analysieren und Planen der RTC-Anwendung und Schnittstellen. Die Implementierungsphase brauchte, wie geplant, den größten Teil der Zeit. Die Testphase wurde genau nach Plan umgesetzt. In der Dokumentationsphase wurden mehr Stunden als die geplanten Arbeitsstunden benötigt, da die Projektdokumentation mehr Zeit in Anspruch nahm als geplant. Die vorgesehene Pufferzeit stellte sich als nützlich heraus, um kleinere Verzögerungen zu kompensieren.

Die detaillierte Vergleichung verdeutlicht, dass sämtliche Projektphasen dem ursprünglichen Zeitplan nahezu exakt entsprachen. Keine gravierenden Abweichungen, die den Erfolg des Projekts in Gefahr gebracht hätten, wurden festgestellt. Eine genaue Planung und die strikte Durchführung machten es möglich, dass der Fertigstellungstermin eingehalten wurde.

Phase	Geplant	Tatsächlich	Differenz
Analysephase	6 h	7 h	1 h
Entwurfsphase	9 h	11 h	2 h
Implementierungsphase	28 h	28 h	0 h
Testphase	12 h	12 h	0 h
Dokumentationsphase	10 h	13 h	3 h
Abnahme & Deployment	5 h	5 h	0 h
Pufferzeit	10 h	6 h	-4 h
Gesamtaufwand	80 h	76 h	-4 h

Tabelle 4: Soll-/Ist-Vergleich

9.2 Gesammelte Erfahrungen

Im Verlauf der Umsetzung des Projekts wurden wertvolle Erfahrungen gesammelt, die sowohl fachliche als auch methodische Aspekte betreffen. Um Effizienzpotenziale zu erkennen, wurde die Wichtigkeit einer gründlichen Untersuchung der aktuellen Situation deutlich. Die Komplexität der Schnittstellen zwischen RTC-Anwendung, SharePoint und Excel-Export wurde gründlich verstanden.

Die Wirtschaftlichkeitsanalyse belegte eindrucksvoll, dass das Projekt sich bereits nach 10,2 Monaten amortisiert.

Die technische Umsetzung stellte besondere Herausforderungen dar, eine verlässliche Exportfunktion zu entwickeln, eine Anbindung an den cloudbasierten SharePoint vorzunehmen und robuste Validierungsmechanismen zu implementieren.

9.3 Ausblick

Die ausgearbeitete Lösung stellt eine zuverlässige Basis für zusätzliche Verbesserungen dar. Zu den konkreten Optionen für die Erweiterung gehören das Hinzufügen weiterer Formate zur Exportfunktion, die Einführung von E-Mail-Benachrichtigungen und die Verbesserung der Validierungsprozesse.

Es gibt auch die Möglichkeit, dies auf andere Abteilungen oder Bereiche des Unternehmens auszuweiten. Die Lösung, erlaubt eine flexible Anpassung an verschiedene Anforderungen.

Literaturverzeichnis

Grails Dokumentation: The Grails Framework – Reference Documetation, Version 6.0.0,
<https://docs.grails.org/6.0.0/guide/single.html>

Microsoft: *Microsoft Graph REST API v1.0 reference*,
<https://learn.microsoft.com/en-us/graph/api/overview>

Apache POI Project: *Apache POI - the Java API for Microsoft Documents*,
<https://poi.apache.org/index.html>

Grails MVC: How Grails MVC works?,
<https://prosperasoft.com/blog/grails/how-grails-mvc-works/>

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase	6 h
1. Anforderungsanalyse	3 h
2. Analyse der RTC-Anwendung	3 h
Entwurfsphase	9 h
1. Erstellung des Datenflussmodells	3 h
2. Planung der Schnittstelle	3 h
3. Entwurf der Benutzerschnittstelle	3 h
Implementierungsphase	28 h
1. Entwicklung der Exportfunktion	10 h
2. Implementierung der API-Schnittstelle	10 h
3. Integration in die RTC-Anwendung	8 h
Testphase	12 h
1. System- und Usability Tests	8 h
2. Benutzerakzeptanztests	4 h
Dokumentationsphase	10 h
1. Erstellung der technischen Dokumentation	6 h
2. Erstellung des Anwenderhandbuchs	4 h
Abnahme und Deployment	5 h
1. Code-Review und Freigabe	3 h
2. Deployment	1 h
3. Einführung für den Abnehmer	1 h
Pufferzeit	10 h
Puffer für unerwartete Herausforderung oder Probleme	10 h
Gesamt	80 h

Tabelle 5: Detaillierte Zeitplanung

A.2 Verwendete Ressourcen

Hardware

- Büroarbeitsplatz mit Desktop Rechner

Software

- IntelliJ IDEA – Entwicklungsumgebung
- PostgreSQL – Datenbanksystem
- SharePoint
- Grails – Webanwendungs Framework
- Gradle – Buildtool
- Microsoft Windows 11 Enterprise – Betriebssystem
- Draw.io – Programm zur Erstellung von Diagrammen und Modellen

Personal

- Auszubildender Fachinformatiker für Anwendungsentwicklung – Umsetzung des Projektes
- Senior Manager – Review des Codes
- Mitarbeiter des Fachbereichs Supply Chain Rail/Road – Festlegung der Anforderungen, Mitgestaltung der Oberflächen
- Projektverantwortlicher - Abnahme der Anwendung

A.3 Amortisation

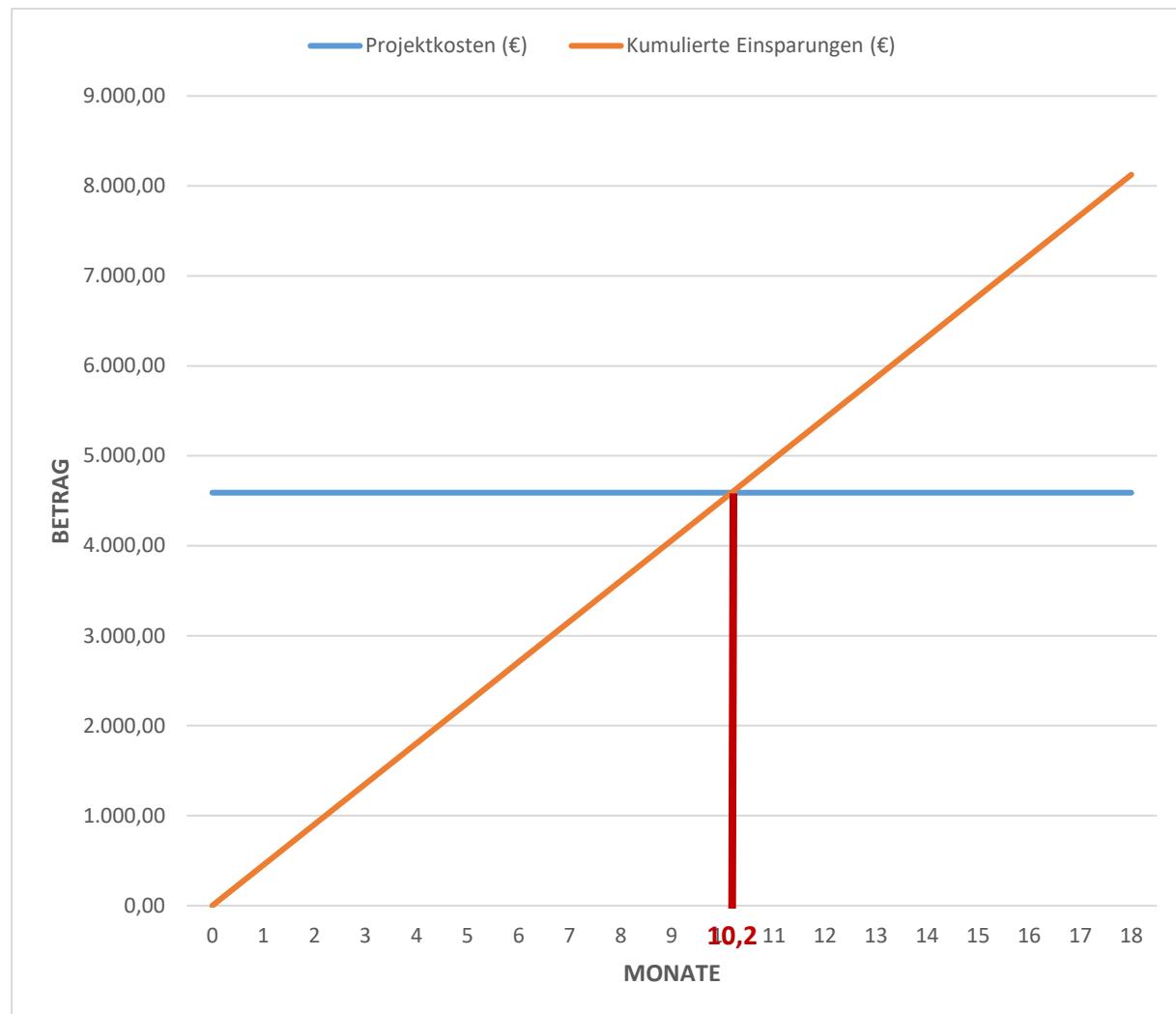


Abbildung 1: Grafische Darstellung der Amortisation

A.4 Use-Case-Diagramm

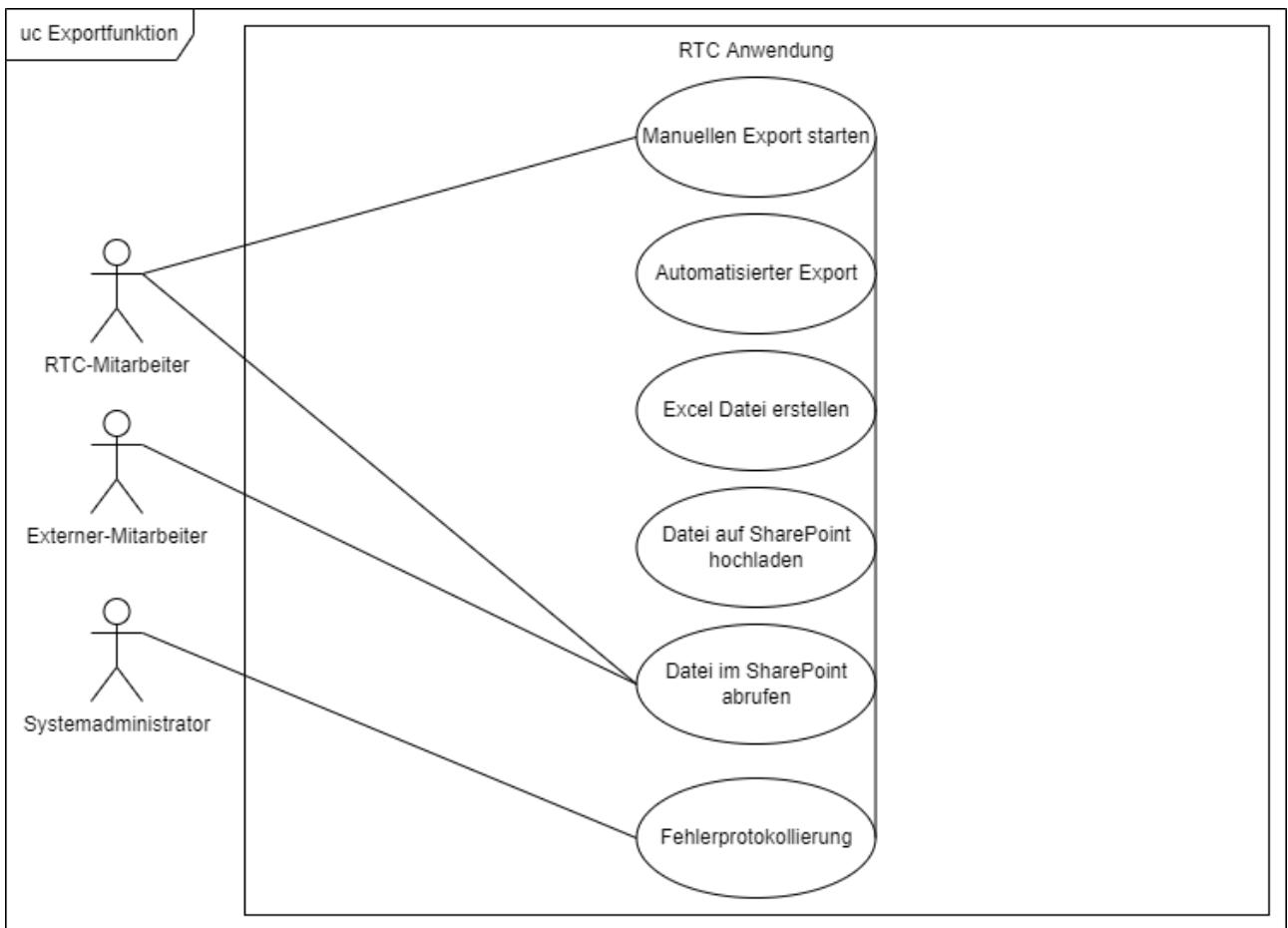


Abbildung 2: Use-Case-Diagramm

A.5 Lastenheft (Auszug)

Folgende Anforderungen müssen von der Anwendung erfüllt werden

Anforderung	Priorität
Die Kesselwagenliste muss täglich um 07:00 Uhr automatisiert als Excel-Datei exportiert werden.	Muss
Die generierte Excel-Datei muss automatisch in das SharePoint-Verzeichnis Rail/Road/Export/Dailys Bahn hochgeladen werden.	Muss
Alle Fehler im Export- oder Upload-Prozess müssen in ein Log geschrieben werden.	Muss
Es muss eine Funktion im RTC-UI geben, um den Export manuell anzustoßen.	Muss
Nach erfolgreichem Upload sollte eine Benachrichtigung im RTC-UI angezeigt werden.	Sollte
Nur berechtigte interne Nutzer sollten den manuellen Export anstoßen können.	Muss
Alte Dateien sollten bei einem neuen Export gelöscht werden.	Muss
Nach erfolgreichem Export/Upload könnte eine E-Mail-Benachrichtigung an bestimmte Nutzergruppen gesendet werden.	Könnte
Es könnte die Möglichkeit bestehen, zusätzlich zum Excel-Format weitere Formate (CSV, PDF) anzubieten.	Könnte

Tabelle 6: Lastenheft-Anforderungen

A.6 Entwurf der Benutzeroberfläche

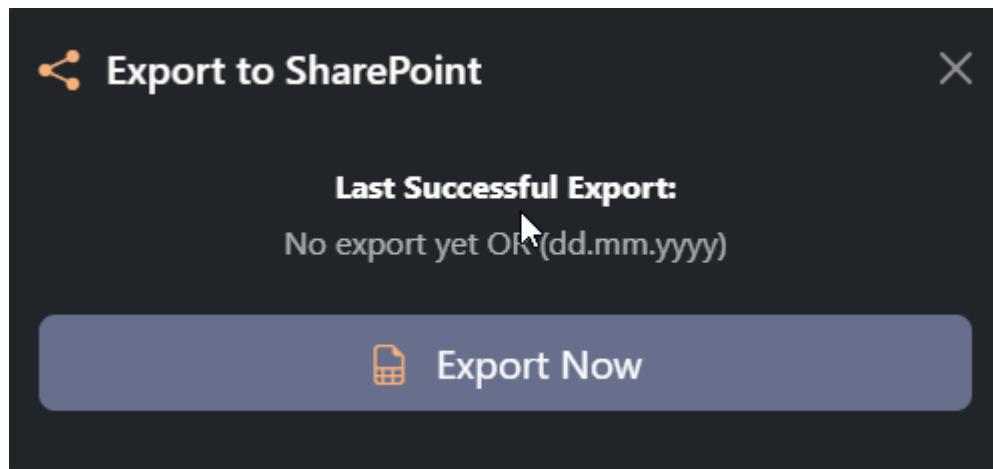


Abbildung 3: Mockup des UIs

A.7 Entity-Relationship-Modell

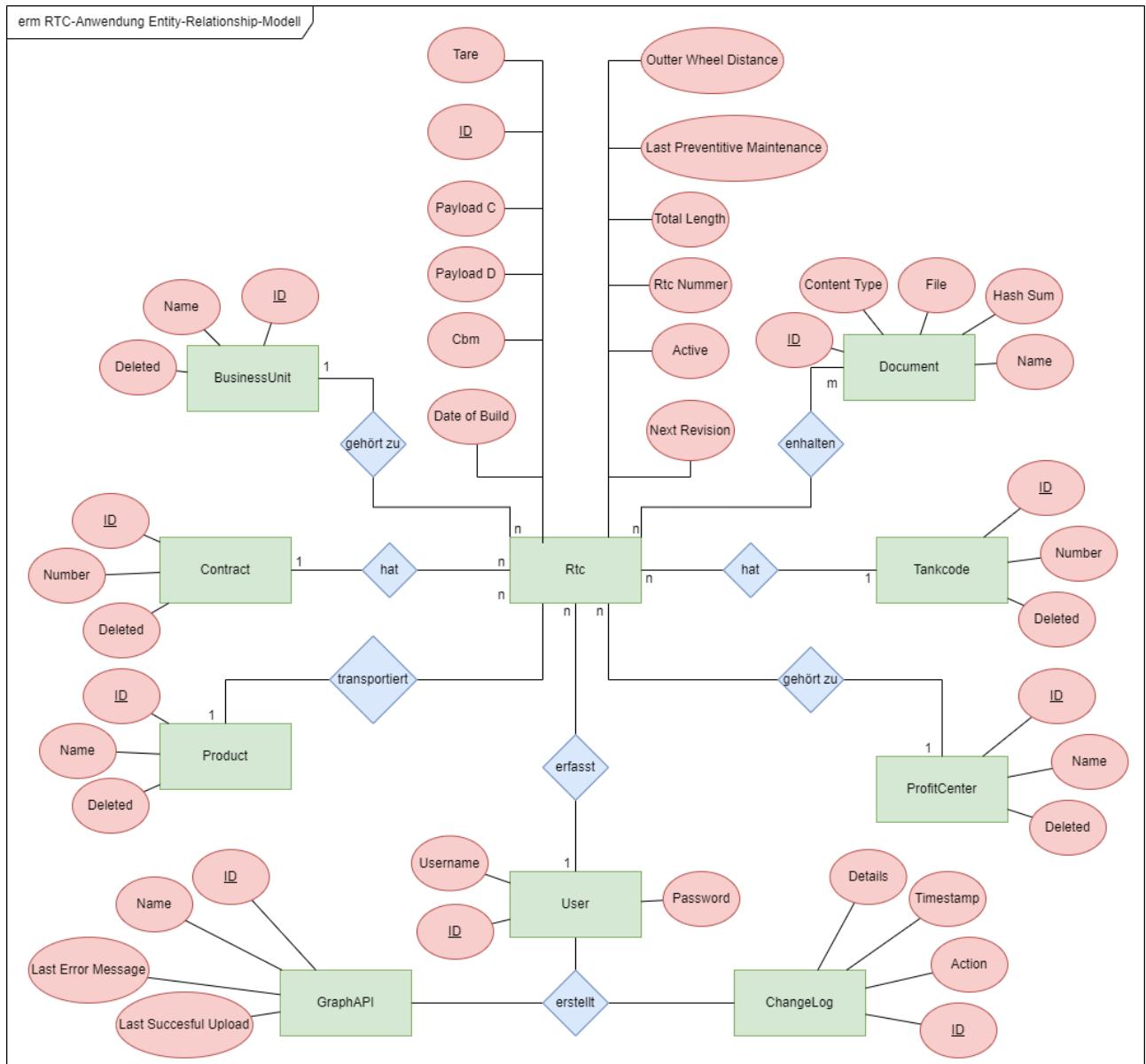


Abbildung 4: Entity-Relationship-Modell

A.8 Klassendiagramm

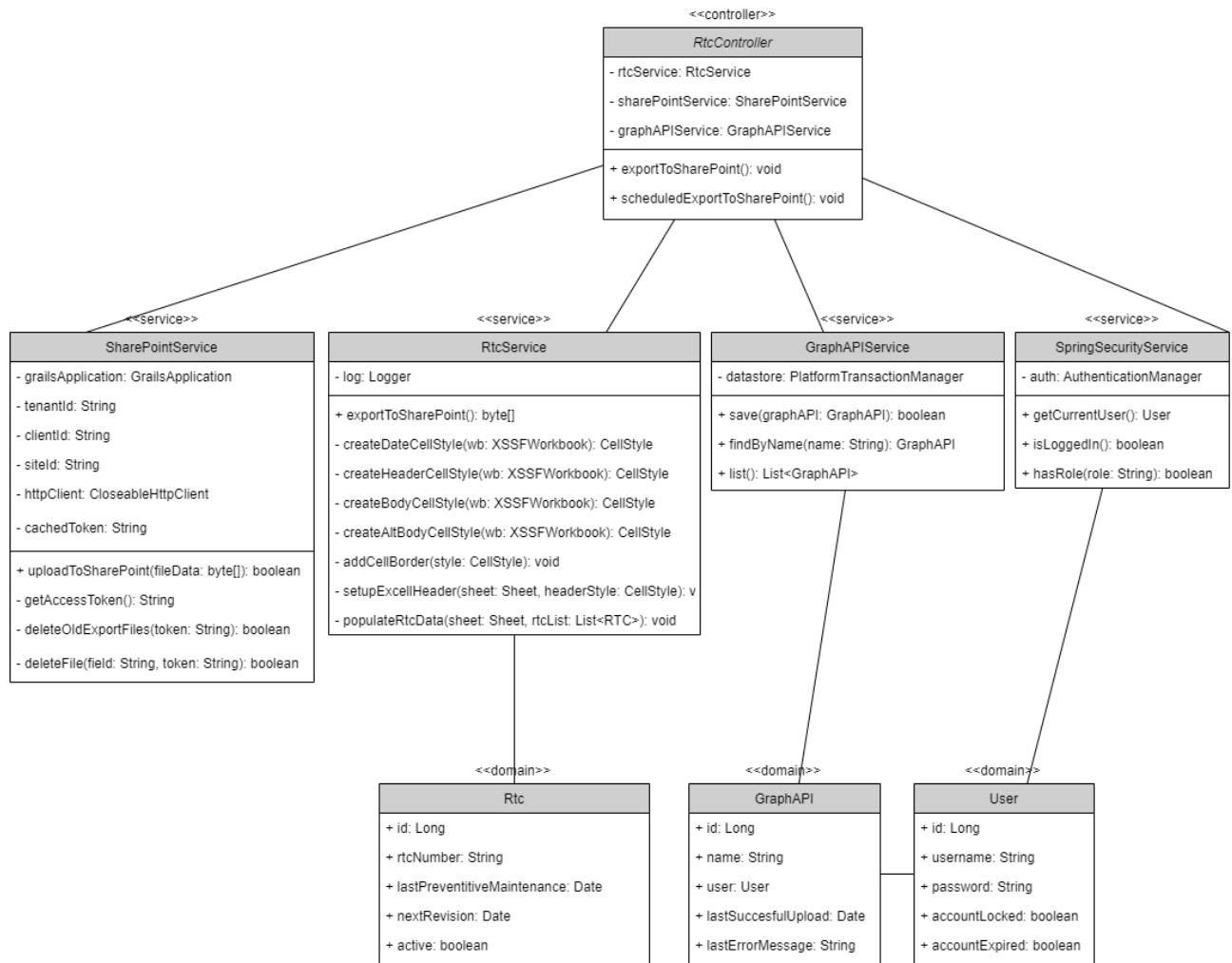


Abbildung 5: Klassendiagramm

A.9 Logeinträge

```
DEBUG [http-nio-8080-exec-3] d.i.rtcms.graphAPI.SharePointService      : Returning cached access token.  
DEBUG [http-nio-8080-exec-3] d.i.rtcms.graphAPI.SharePointService      : Returning cached access token.  
INFO [http-nio-8080-exec-3] d.i.rtcms.graphAPI.SharePointService       : Deleted file with ID: 0162B6MBI56KNRWZ6Y4JCYU2DSFNDE6A3I  
INFO [http-nio-8080-exec-3] d.i.rtcms.graphAPI.SharePointService       : Old export files deletion status: true  
INFO [http-nio-8080-exec-3] d.i.rtcms.graphAPI.SharePointService       : File 'Kesselwagenliste.xlsx' uploaded successfully.  
INFO [http-nio-8080-exec-3] de.ineos rtcms.RtcController                : Excel-Export erfolgreich und auf SharePoint hochgeladen!
```

Abbildung 6: Exemplarische Logeinträge

A.10 Pflichtenheft

Zielbestimmung

1. Plattform

- 1.1. Die Anwendung wird mit Groovy implementiert.
- 1.2. Die Entwicklung geschieht mit dem Grails Framework.
- 1.3. Die Anwendung wird in die bestehende RTCMS-Anwendung integriert.
- 1.4. Die Anwendung wird auf einem Tomcat-Server betrieben, der auf einem Linux-System läuft.
- 1.5. GitLab wird als Softwarelösung zur gehosteten Versionsverwaltung genutzt.
- 1.6. Gradle wird für den automatischen Build-Prozess genutzt.

2. Datenbank

- 2.2. Die Daten für den Export werden aus der bestehenden PostgreSQL-Datenbank der RTC-Anwendung abgerufen.
- 2.3. Die Daten zu den erfolgten Exports werden in der GraphAPI-Tabelle gespeichert.

3. Benutzeroberfläche

- 3.1. Die Benutzeroberfläche wird als Erweiterung der bestehenden RTCMS-Oberfläche realisiert.
- 3.2. Die Benutzeroberfläche wird mit dem Bootstrap Framework gestaltet und soll die bestehenden Designrichtlinien einhalten.
- 3.3. Grundlage zur Gestaltung ist Bootstrap 5.
- 3.4. Für die Anzeige des Export-Status wird ein Modal-Dialog implementiert.
- 3.5. Die Benutzeroberfläche enthält einen Button zum manuellen Auslösen des Exports.

4. Geschäftslogik

- 4.1. Die Excel-Dateien werden mit der Java-Bibliothek Apache POI erzeugt.
- 4.2. Die Kommunikation mit SharePoint erfolgt über die Microsoft Graph API.
- 4.3. Die Authentifizierung gegenüber der Graph API erfolgt mittels OAuth 2.0 Client Credentials Flow.
- 4.4. Der Export kann manuell durch berechtigte Benutzer ausgelöst werden.
- 4.5. Der Export wird täglich um 07:00 Uhr automatisch ausgeführt.
- 4.6. Alte Versionen der Datei werden vor dem Upload neuer Versionen gelöscht.

5. Fehlerbehandlung

- 5.1. Alle Fehler werden im Log dokumentiert.
- 5.2. Bei einem Fehler während des manuellen Exports wird eine entsprechende Meldung angezeigt.
- 5.3. Bei fehlgeschlagenen Exports wird die Fehlerursache in der GraphAPI-Tabelle gespeichert.

6. Sicherheit

- 6.1. Nur authentifizierte und autorisierte Benutzer können den manuellen Export auslösen.
- 6.2. Sensible Konfigurationsdaten wie Client-Secrets werden verschlüsselt gespeichert.
- 6.3. Die exportierten Daten dürfen nur im internen SharePoint gespeichert werden.

A.11 Excel-Struktur

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	RTC Number	Lessor	Contract	Last Preventive Maintenance	Next Revision	Date of Build	Product	Business Unit	Profit Center	Tankcode	Time (t)	Payload C (t)	Payload D (t)	cbm	Total Length
1	3212899665	ARETZ GMBH & CO.	1230191	24.05.2022	01.01.2070	14.03.2015	ACN	NITRILES	P26BH	23.05	84.39	71.92	36.22	15.73	
2	2902717851	ARETZ GMBH & CO.	1230181	16.05.2022	31.12.2025	10.11.2007	DIB/DD	OLIGOMERS	P26BH	18.13	89.23	80.8	38.06	13.28	
3	8643418609	ARETZ GMBH & CO.	1230181	15.11.2020	25.01.2042	02.11.1994	Methylpropadiene/OLIGOMERS	P15BH	18.17	93.73	88.88	36.25	10.91		

Abbildung 7: Excel Struktur

A.12 Datenmodell

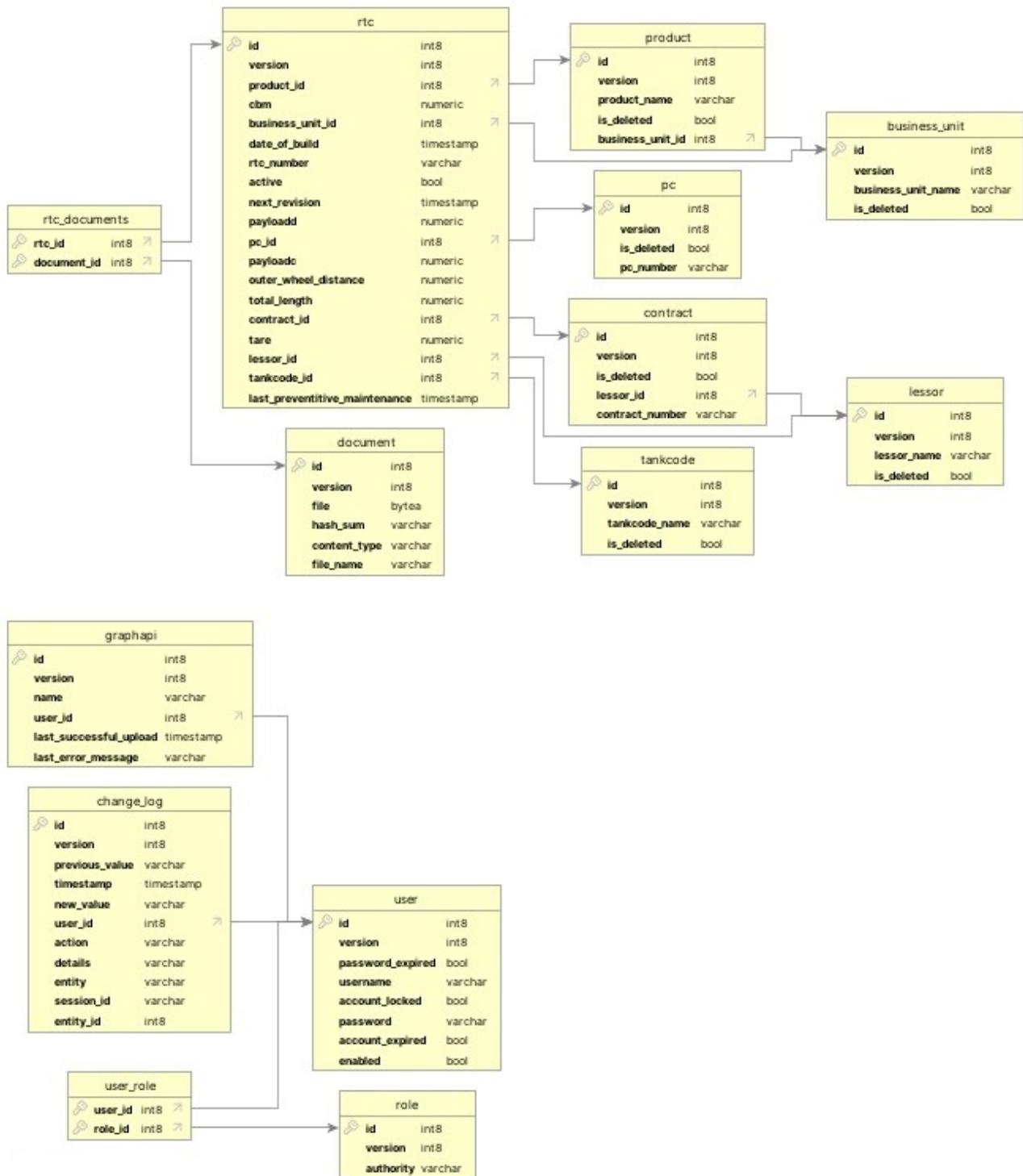


Abbildung 8: Datenmodell

A.13 UI-JavaScript

```
$(document).ready(function () {
    const exportBtn = $('#exportNowBtn');
    const spinner = $('#loadingSpinner');
    const statusMsg = $('#statusMessage');
    const lastExport = $('#lastExportDate');

    function toggleButtonState(isDisabled) {
        exportBtn.prop('disabled', isDisabled);
        exportBtn.find('span').text(isDisabled ? 'Exporting...' : 'Export Now');
    }

    function updateStatus(message, type = 'info') {
        // Entferne alle alert-* Klassen
        statusMsg
            .removeClass('d-none alert-info alert-success alert-danger alert-warning')
            .addClass('alert alert-' + type)
            .text(message);
    }

    function startExport() {
        toggleButtonState(true);
        spinner.removeClass('d-none');
        updateStatus('Export is in progress...', 'info');

        $.ajax({
            url: '/rtc/exportToSharePoint',
            type: 'POST',
            beforeSend: function () {
                console.log("Starting export...");
            },
            success: function () {
                spinner.addClass('d-none');
                updateStatus('Export successful! ✅', 'success');
                toggleButtonState(false);

                // Vorschlag: Nach erfolgreichem Export die Seite neu laden oder Modal neu öffnen
            },
            error: function (xhr, status, error) {
                console.error('Export error:', error);
                spinner.addClass('d-none');
                updateStatus('Export failed. ❌ Please try again.', 'danger');
                toggleButtonState(false);
            }
        });
    }
})
```

```
exportBtn.on('click', startExport);

$('#export').on('show.bs.modal', function () {
    lastExport.text(
        `${graphAPIInstance?.lastSuccessfulUpload ?: 'No export yet'}`);
});

// Reset Status-Anzeige & Spinner
spinner.addClass('d-none');
statusMsg.addClass('d-none').removeClass('alert-info alert-success alert-danger alert-warning');
toggleButtonState(false);
});
});
```

Listing 1: UI-JavaScript

A.14 Aktivitätsdiagramm

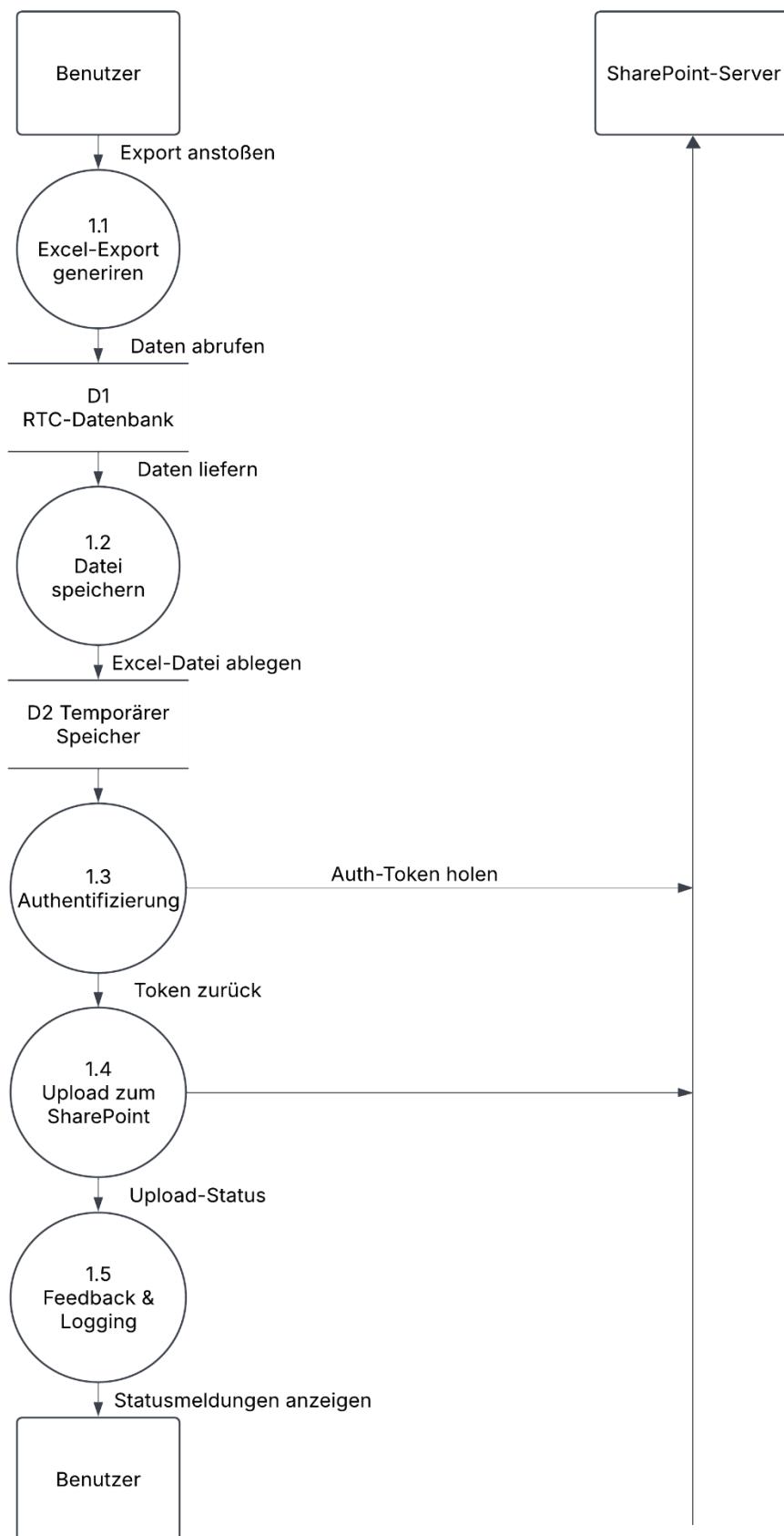


Abbildung 9: Datenflussmodell

A.15 RTC-Controller-Code

```
def exportToSharePoint() {
    try {
        if (sharePointService.uploadToSharePoint(rtcService.exportToSharePoint())) {
            log.info("Excel-Export erfolgreich und auf SharePoint hochgeladen!")

            def graphAPI = new GraphAPI(
                name: "RTC Export",
                user: springSecurityService.currentUser,
                lastSuccessfulUpload: new Date()
            )

            if (!graphAPIService.save(graphAPI)) {
                log.error("Fehler beim Speichern der GraphAPI-Instanz: ${graphAPI.errors}")
            }
        }
    } catch (Exception e) {
        log.error("Fehler beim Excel-Export", e)

        def graphAPI = new GraphAPI(
            name: "RTC Export",
            user: springSecurityService.currentUser,
            lastErrorMessage: e.message
        )
        graphAPIService.save(graphAPI)
    }
    redirect(action: "index")
}

@Scheduled(cron = "0 0 7 * * ?")
void scheduledExportToSharePoint() {
    log.info("Starte automatisierten Export für ${new Date()}")
    boolean success = sharePointService.uploadToSharePoint(rtcService.exportToSharePoint())

    if (success) {
        log.info("Automatisierter Export erfolgreich.")
    } else {
        log.error("Automatisierter Export fehlgeschlagen!")
    }
}
```

Listing 2: RTC-Controller-Code

A.16 RTC-Service-Code

```
byte[] exportToSharePoint() {  
    try {  
        def rtcList = getAllRtcs()  
  
        String templatePath = "grails-app/assets/templates/Template.xlsx"  
        XSSFWorkbook wb = new XSSFWorkbook(templatePath)  
  
        Sheet sheet = wb.getSheetAt(0)  
  
        CellStyle dateCellStyle = createDateCellStyle(wb)  
        CellStyle headerStyle = createHeaderCellStyle(wb)  
        CellStyle bodyStyle = createBodyCellStyle(wb)  
        CellStyle altBodyStyle = createAltBodyCellStyle(wb)  
  
        setupExcelHeader(sheet, headerStyle)  
  
        populateRtcData(sheet, rtcList, dateCellStyle, bodyStyle, altBodyStyle)  
  
        // Freeze the header row for better navigation  
        sheet.createFreezePane(0, 1)  
  
        // AutoSize all columns dynamically based on content  
        autoSizeAllColumns(sheet, 18)  
  
        // Optional: Add AutoFilter to the header row  
        sheet.setAutoFilter(new CellRangeAddress(0, 0, 0, 17))  
  
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream()  
        wb.write(outputStream)  
        wb.close()  
  
        byte[] fileData = outputStream.toByteArray()  
        return fileData  
  
    } catch (Exception e) {  
        log.error("Fehler bei der Excel Erstellung", e)  
    }  
    return fileData  
}
```

Listing 3: RTC-Service-Code

A.17 Excel-Formatierungsmethoden

```
private CellStyle createDateCellStyle(XSSFWorkbook wb) {  
    CreationHelper createHelper = wb.getCreationHelper()  
    CellStyle style = wb.createCellStyle()  
    style.setDataFormat(createHelper.createDataFormat().getFormat("dd.MM.yyyy"))  
    style.setAlignment(HorizontalAlignment.CENTER)  
    addCellBorder(style)  
    return style  
}  
  
private CellStyle createHeaderCellStyle(XSSFWorkbook wb) {  
    CellStyle style = wb.createCellStyle()  
    style.setAlignment(HorizontalAlignment.CENTER)  
    style.setVerticalAlignment(VerticalAlignment.CENTER)  
    style.setFillForegroundColor(IndexedColors.GREY_25_PERCENT.getIndex())  
    style.setFillPattern(FillPatternType.SOLID_FOREGROUND)  
    addCellBorder(style)  
  
    Font font = wb.createFont()  
    font.setBold(true)  
    font.setColor(IndexedColors.WHITE.getIndex())  
    style.setFont(font)  
  
    return style  
}  
  
private CellStyle createBodyCellStyle(XSSFWorkbook wb) {  
    CellStyle style = wb.createCellStyle()  
    style.setAlignment(HorizontalAlignment.LEFT)  
    addCellBorder(style)  
    return style  
}  
  
private CellStyle createAltBodyCellStyle(XSSFWorkbook wb) {  
    CellStyle style = createBodyCellStyle(wb)  
    style.setFillForegroundColor(IndexedColors.LIGHT_CORNFLOWER_BLUE.getIndex())  
    style.setFillPattern(FillPatternType.SOLID_FOREGROUND)  
    return style  
}  
  
private void addCellBorder(CellStyle style) {  
    style.setBorderBottom(BorderStyle.THIN)  
    style.setBorderTop(BorderStyle.THIN)  
    style.setBorderLeft(BorderStyle.THIN)  
    style.setBorderRight(BorderStyle.THIN)  
}  
  
private void setupExcelHeader(Sheet sheet, CellStyle headerStyle) {  
    Row headerRow = sheet.getRow(0) ?: sheet.createRow(0)
```

```
headerRow.setHeightInPoints(30)

def headers = [
    "RTC Number", "Lessor", "Contract", "Last Preventive Maintenance", "Next Revision", "Date of Build",
    "Product", "Business Unit", "Profit Center", "Tankcode", "Tare (t)", "Payload C (t)", "Payload D (t)",
    "cbm", "Total Length", "Outer Wheel Distance", "Active", "Documents"
]

headers.eachWithIndex { header, index ->
    Cell cell = headerRow.createCell(index)
    cell.setCellValue(header)
    cell.setCellStyle(headerStyle)
}
}

private void populateRtcData(Sheet sheet, def rtcList, CellStyle dateCellStyle, CellStyle bodyStyle, CellStyle altBodyStyle) {
    rtcList.eachWithIndex { rtc, rowIndex ->
        Row row = sheet.createRow(rowIndex + 1)
        row.setHeightInPoints(20)

        CellStyle rowStyle = (rowIndex % 2 == 0) ? bodyStyle : altBodyStyle

        createCell(row, 0, rtc.rtcNumber, rowStyle)
        createCell(row, 1, rtc.lesser?.toString(), rowStyle)
        createCell(row, 2, rtc.contract?.toString(), rowStyle)
        setDateCell(row, 3, rtc.lastPreventitiveMaintenance, dateCellStyle)
        setDateCell(row, 4, rtc.nextRevision, dateCellStyle)
        setDateCell(row, 5, rtc.dateOfBuild, dateCellStyle)
        createCell(row, 6, rtc.product?.toString(), rowStyle)
        createCell(row, 7, rtc.businessUnit?.toString(), rowStyle)
        createCell(row, 8, rtc.pc?.toString(), rowStyle)
        createCell(row, 9, rtc.tankcode?.toString(), rowStyle)
        createCell(row, 10, rtc.tare, rowStyle)
        createCell(row, 11, rtc.payloadC, rowStyle)
        createCell(row, 12, rtc.payloadD, rowStyle)
        createCell(row, 13, rtc.cbm, rowStyle)
        createCell(row, 14, rtc.totalLength, rowStyle)
        createCell(row, 15, rtc.outerWheelDistance, rowStyle)
        createCell(row, 16, rtc.active ? "Yes" : "No", rowStyle)
        createCell(row, 17, rtc.documents?.fileName, rowStyle)
    }
}

private void createCell(Row row, int columnIndex, def value, CellStyle style) {
    Cell cell = row.createCell(columnIndex)
    if (value instanceof Number) {
        cell.setCellValue(value.doubleValue())
    } else {

```

```
cell.setCellValue(value ?: "")  
}  
cell.setCellStyle(style)  
}  
  
private void setDateCell(Row row, int columnIndex, def dateValue, CellStyle style) {  
    Cell cell = row.createCell(columnIndex)  
    if (dateValue) {  
        cell.setCellValue(dateValue)  
        cell.setCellStyle(style)  
    } else {  
        cell.setCellValue("")  
        cell.setCellStyle(style)  
    }  
}  
  
private void autoSizeAllColumns(Sheet sheet, int numberOfColumns) {  
    (0..<numberOfColumns).each { columnIndex ->  
        sheet.autoSizeColumn(columnIndex)  
        int currentWidth = sheet.getColumnWidth(columnIndex)  
        int maxWidth = 10000 // Setze ein Limit, wenn nötig (in 1/256 Einheiten)  
        if (currentWidth > maxWidth) {  
            sheet.setColumnWidth(columnIndex, maxWidth)  
        }  
    }  
}
```

Listing 4: Excel Formatierungsmethoden

A.18 Abhängigkeiten

```
dependencies {  
    implementation('org.apache.httpcomponents.client5:httpclient5:5.4.2')  
    implementation('org.apache.httpcomponents.core5:httpcore5:5.3.3')  
    implementation('org.apache.poi:poi-ooxml:5.4.0')  
}
```

Listing 5: Abhängigkeiten

A.19 SharePoint-Service-Code

```
boolean uploadToSharePoint(byte[] fileData) {
    String token = getAccessToken()
    String url =
"https://graph.microsoft.com/v1.0/sites/${siteId}/drives/${driveId}/root:/${folderPath}/${fileName}:conte
nt"

    try {
        if (!deleteOldExportFiles()) {
            log.warn("Could not delete old export files before upload.")
        }

        HttpPut put = new HttpPut(url)
        put.setHeader("Authorization", "Bearer ${token}")
        put.setHeader("Content-Type", "application/octet-stream")
        put.entity = new ByteArrayEntity(fileData, ContentType.APPLICATION_OCTET_STREAM)

        httpClient.execute(put) { response ->
            int statusCode = response.code
            String responseBody = EntityUtils.toString(response.entity)
            if (statusCode in [HttpStatus.OK.value(), HttpStatus.CREATED.value()]) {
                log.info("File '${fileName}' uploaded successfully.")
                return true
            } else {
                log.error("Failed to upload file. Status: ${statusCode}, Response: ${responseBody}")
                return false
            }
        }
    } catch (Exception e) {
        log.error("Error uploading file to SharePoint", e)
        return false
    }
}

private boolean deleteOldExportFiles() {
    String token = getAccessToken()
    String url =
"https://graph.microsoft.com/v1.0/sites/${siteId}/drives/${driveId}/root:/${folderPath}:/children"

    try {
        HttpGet getRequest = new HttpGet(url)
        getRequest.setHeader("Authorization", "Bearer ${token}")

        httpClient.execute(getRequest) { response ->
            int statusCode = response.code
            String responseBody = EntityUtils.toString(response.entity)

            if (statusCode != HttpStatus.OK.value()) {
```

```
    log.error("Failed to list files. Status: ${statusCode}, Response: ${responseBody}")
    return false
}

def json = new JsonSlurper().parseText(responseBody)
def filesToDelete = json?.value?.findAll { it.name == fileName }

if (!filesToDelete) {
    log.info("No old files found for deletion.")
    return true
}

boolean allDeleted = filesToDelete.every { deleteFile(it.id, token) }
log.info("Old export files deletion status: ${allDeleted}")
return allDeleted
}

} catch (Exception e) {
    log.error("Error deleting old export files", e)
    return false
}
}

private boolean deleteFile(String fileId, String token) {
    String url = "https://graph.microsoft.com/v1.0/sites/${siteId}/drives/${driveId}/items/${fileId}"

    try {
        HttpDelete deleteRequest = new HttpDelete(url)
        deleteRequest.setHeader("Authorization", "Bearer ${token}")

        httpClient.execute(deleteRequest) { response ->
            int statusCode = response.code
            if (statusCode == HttpStatus.NO_CONTENT.value()) {
                log.info("Deleted file with ID: ${fileId}")
                return true
            } else {
                String responseBody = EntityUtils.toString(response.entity)
                log.error("Failed to delete file ${fileId}. Status: ${statusCode}, Response: ${responseBody}")
                return false
            }
        }
    } catch (Exception e) {
        log.error("Error deleting file with ID: ${fileId}", e)
        return false
    }
}
}
```

Listing 6: SharePoint-Service-Code

A.20 Token Management

```
private synchronized String getAccessToken() {  
    if (cachedToken && tokenExpiryTime && Instant.now().isBefore(tokenExpiryTime)) {  
        log.debug("Returning cached access token.")  
        return cachedToken  
    }  
  
    String url = "https://login.microsoftonline.com/${tenantId}/oauth2/v2.0/token"  
    List<BasicNameValuePair> params = [  
        new BasicNameValuePair("grant_type", "client_credentials"),  
        new BasicNameValuePair("client_id", clientId),  
        new BasicNameValuePair("client_secret", clientSecret),  
        new BasicNameValuePair("scope", "https://graph.microsoft.com/.default")  
    ]  
  
    try {  
        HttpPost post = new HttpPost(url)  
        post.entity = new UrlEncodedFormEntity(params)  
  
        httpClient.execute(post) { response ->  
            String responseBody = EntityUtils.toString(response.entity)  
            def json = new JsonSlurper().parseText(responseBody)  
  
            if (!json.access_token) {  
                throw new RuntimeException("Access token missing in response: ${responseBody}")  
            }  
  
            cachedToken = json.access_token  
            int expiresIn = json.expires_in ?: 3600 // Standardmäßig 1 Stunde  
            tokenExpiryTime = Instant.now().plusSeconds(expiresIn - 60) // 60 Sekunden Puffer  
  
            log.info("Fetched new access token, expires in ${expiresIn} seconds.")  
            return cachedToken  
        }  
  
    } catch (Exception e) {  
        log.error("Error getting access token", e)  
        throw new RuntimeException("Failed to fetch access token: ${e.message}", e)  
    }  
}
```

Listing 7: Token Management

A.21 SharePoint-API Konfiguration

```
microsoft:  
tenantId: xxxxx-xxxxxx-xxxx  
clientId: xxxxxx-xxxxxx-xxxx  
clientSecret: xxxxxxxxxxxxxxxxxxxx  
siteId: xxxxx-xxxxxx-xxxx  
driveId: x-xx-xxxxxx-xxxxxx  
folderPath: Dateien%20Dailys%20Bahn  
fileName: Kesselwagenliste.xlsx
```

Listing 8: SharePoint-API Konfiguration

A.22 Systemtest

Testziele

- Sicherstellen der technischen Funktionalität der Export- und Upload-Funktion.
- Überprüfung der Datenintegrität in den exportierten Excel-Dateien.
- Prüfung der erfolgreichen und fehlerfreien Übertragung zu SharePoint.

Testumgebung

- Entwicklungsumgebung inkl. Apache POI und SharePoint API
- PostgreSQL-Datenbank der RTC-Anwendung
- SharePoint-Testumgebung (cloudbasiert)

Verantwortlichkeiten

- Tester: Entwickler
- Testkoordination: Entwickler

Testfälle und Ergebnisse

Testfall-Nr.	Testfallbeschreibung	Eingabedaten	Erwartetes Ergebnis	Ergebnis
ST-01	Export manuell starten	Klick auf „Export Now“-Button	Excel-Datei wird erstellt	Erfolgreich
ST-02	Excel-Format prüfen	Excel-Datei öffnen	Daten korrekt formatiert (Spalten, Kopfzeilen, Zellformatierung)	Erfolgreich
ST-03	Automatischer Upload auf SharePoint	Nach Export automatisch hochladen	Datei erscheint im definierten SharePoint-Verzeichnis	Erfolgreich
ST-06	Automatischer Export (geplant)	Geplante Aufgabe um 7 Uhr	Excel-Datei wird automatisch erzeugt und hochgeladen	Erfolgreich

Abweichungen

Keine Abweichungen festgestellt.

A.23 Usability Test

Testziele

- Überprüfung der Benutzerfreundlichkeit der Exportfunktion
- Feststellung, ob der Prozess für Anwender verständlich und einfach ist

Testgruppe

- 3 interne Anwender aus dem Supply Chain Team

Testmethode

Beobachtung während der Nutzung + standardisierter Fragebogen zur Bewertung

Aufgaben für die Tester

1. Manuelle Durchführung des Exports
2. Kontrolle der erzeugten Excel-Datei
3. Abruf der Datei über SharePoint

Beobachtungen / Feedback

Tester	Aufgabe	Beobachtung / Anmerkung
Anwender 1	Export starten	Prozess intuitiv, keine Hilfe benötigt
Anwender 2	Excel prüfen	Struktur übersichtlich, keine Mängel
Anwender 3	Datei über SharePoint finden	Einfacher Zugriff, keine Schwierigkeiten

Fragebogen-Ergebnisse (Skala 1 = schlecht, 5 = sehr gut)

Kriterium	Durchschnitt
Verständlichkeit der Benutzeroberfläche	4,6
Simplizität des Exports	5
Auffindbarkeit auf SharePoint	4,6
Status-Feedback	4

Verbesserungsvorschläge

- Verbesserung der Fehlermeldung bei falscher SharePoint-Berechtigung (klarere Meldung)

A.24 Benutzeraktzeptanztest

Testziele

- Bestätigung, dass die Lösung den Anforderungen entspricht
- Abnahme durch die Stakeholder vor Go-Live

Teilnehmer

- Key-User aus Supply Chain
- Entwickler

Abnahmekriterien

- Excel-Datei enthält alle geforderten Datenfelder
- Automatischer Export läuft stabil und pünktlich
- Dateien sind für alle berechtigten Stakeholder zugänglich

Testfälle

Testfall-Nr.	Beschreibung	Akzeptanzkriterium	Ergebnis
UAT-01	Manuelle Exportfunktion nutzen	Excel-Datei korrekt erstellt und geladen	Erfolgreich
UAT-02	Geplante Exporte prüfen (automatisch)	Dateien um 7 Uhr verfügbar auf SharePoint	Erfolgreich

Freigabeerklärung

Die Export- und Upload Funktion der RTC-Anwendung wurde durch die definierten Benutzergruppen geprüft. Die Funktion erfüllt die zuvor festgelegten Anforderungen und wird zur Produktivsetzung freigegeben.

A.25 Benutzerdokumentation (Auszug)

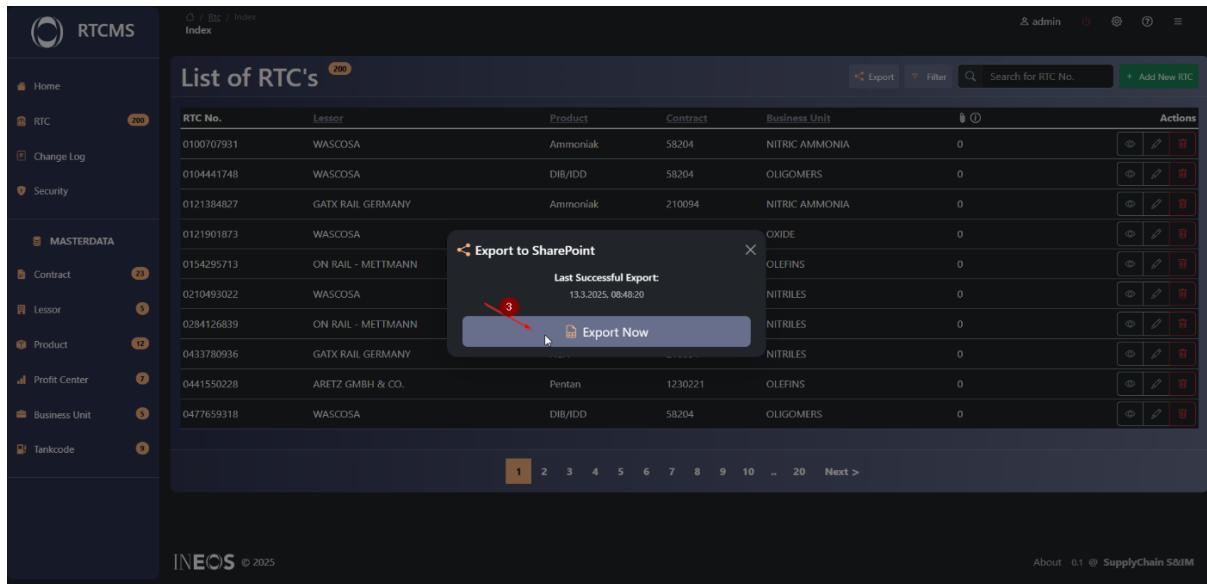
- Klicken Sie in der Navigationsleiste links auf die Schaltfläche **RTC**.

The screenshot shows the RTCMS Home page. On the left, there is a navigation sidebar with a red box around the 'RTC' button. The main area displays four cards: 'Currently Active RTCS' (113), 'Currently Inactive RTCS' (87), 'Upcoming Revisions in less than 1 year' (5), and 'Overdue Maintenance by more than 2 years' (161). Below these is a 'Product Distribution' section with a bar chart comparing various products. To the right, there is a 'Revisions by Criteria' search panel.

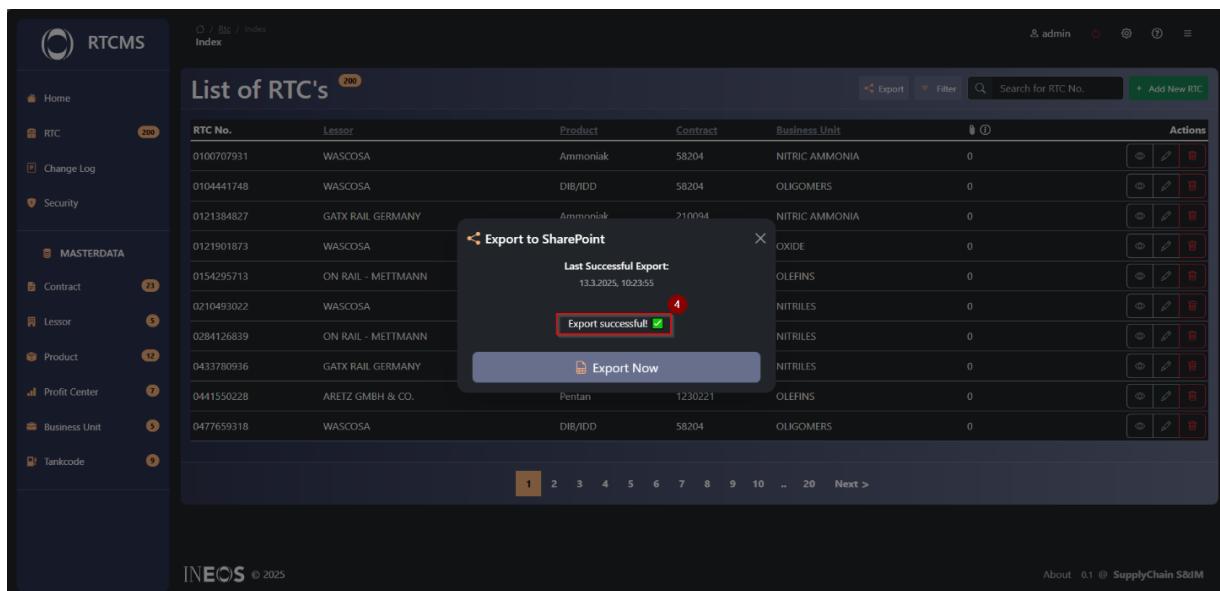
- Klicken Sie auf die Schaltfläche **Export**.

The screenshot shows the 'List of RTC's' page. The 'RTC' button in the sidebar has a red box around it. The main content area is a table with columns: RTC No., Lessor, Product, Contract, Business Unit, and Actions. An 'Export' button at the top right of the table has a red box and a number '2' above it, indicating it is the next step. The table contains 14 rows of RTC data.

3. Klicken Sie auf die Schaltfläche **Export Now**



4. Warten Sie, bis die Meldung „**Export successful!**“ erscheint.



5. Die Datei wurde automatisch auf SharePoint hochgeladen.

Abbildung 10: Auszug aus der Benutzerdokumentation

A.26 Entwicklerdokumentation (Auszug)

The screenshot displays two pages of developer documentation for Groovy classes:

- RtcService** (Groovy Class):
 - Constructor Summary**: Shows one constructor: `RtcService()`.
 - Methods Summary**:

Type Params	Return Type	Name and description
	Java.lang.Long	<code>count()</code> Zählt alle aktiven RTCs.
	Java.lang.Long	<code>countActiveRtc()</code> Zählt alle aktiven RTCs.
	Java.util.Map<Java.lang.String, Java.lang.Long>	<code>countByProject()</code> Gibt eine Map mit Projektnamen und der jeweiligen Anzahl zugehöriger RTCs zurück.
	Java.lang.Long	<code>countInactiveRtc()</code> Zählt alle inaktiven RTCs.
	Java.lang.Long	<code>countMostInactiveRtc()</code> Anzahl der RTCs, deren letzte vorliegende Wartung mehr als zwei Jahre zurückliegt.
	Java.lang.Long	<code>countMostRecentRtc()</code> Zählt RTCs innerhalb eines definierten Zeitraums für eine bestimmte Business Unit.
	Java.lang.Long	<code>countMostRecentRevision()</code> Anzahl der RTCs, deren nächste Revision innerhalb eines Jahres anstand.
void		<code>delete(java.io.Serializable id)</code>
byte[]		<code>exportToExcelSheet()</code> Exportiert RTC-Daten in eine Excel-Datei und lädt sie anschließend auf SharePoint hoch.
- SharePointService** (Groovy Class):
 - Properties Summary**: Shows one property: `grailsApplication`.
 - Constructor Summary**: Shows one constructor: `SharePointService(grails.core.GrailsApplication grailsApplication)`.
 - Methods Summary**:

Type Params	Return Type	Name and description
	boolean	<code>uploadToSharePoint(byte[] fileData)</code> Lädt eine Datei auf SharePoint hoch.

Abbildung 11: Entwicklerdokumentation