



SIRUM

Abschlussprüfung Sommer 2025

Fachinformatiker fr. Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

Entwicklung eines Odoo-Moduls zur
**ZIP-komprimierten Dateiverwaltung im
Dokumentenmanagementsystem (DMS)**

Abgabetermin: 24. Mai 2025

Diese Dokumentation wurde erstellt von:

Mark Albers
Tienrade 1, 21031 Hamburg
E-Mail: albers.mark02@gmail.com
Prüflingsnummer: (131)-54068

Ausbildungsbetrieb:

BFW-Berufsförderungswerk Hamburg GmbH
Marie-Bautz-Weg 16, 22159 Hamburg
Ansprechpartner: Dr. Olaf Kubillus

Betrieb für die Praktische Ausbildung:

Sirum GmbH
Am Sandtorkai 32, 20457 Hamburg
Ansprechpartner: Dipl. Ing. (FH) Georg Notter
E-Mail: Georg.Notter@Sirum.de

Inhaltsverzeichnis

Abkürzungsverzeichnis.....	4
Begriffserläuterung	4
1. Einleitung	1
1.1. Projektumfeld	1
1.2. Projektbeschreibung	1
1.3. Projektschnittstellen.....	1
1.4. Projektziel	1
2. Projektplanung	2
2.1. Zeitplanung	2
2.2. Ressourcenplanung.....	2
2.3. Entwicklungsprozess	2
2.4. SMART-Ziele	3
3. Analysephase	3
3.1. Ist-Analyse	3
3.2. Wirtschaftlichkeitsanalyse	4
3.2.1. „Make or Buy“-Entscheidung	4
3.2.2. Projektkosten	4
3.2.3. Amortisation	4
3.3. Nicht-monetäre Aspekte	5
3.4. Anwendungsablauf	5
3.5. Anforderungen	5
4. Entwurfsphase	6
4.1. Entwurf der Benutzeroberfläche	6
4.2. Technischer Entwurf	6
4.3. Datenmodell	7
4.4. Modulstruktur und Verzeichnisübersicht.....	7
4.5. Maßnahmen zur Qualitätssicherung	8
4.6. Zugriffsrechte und Sicherheit.....	8
4.7. Deployment	8
5. Implementierungsphase	9
5.1. Implementierung der Benutzeroberfläche	9
5.2. Implementierung der Datenstrukturen	9
5.3. Implementierung der Geschäftslogik	10
5.4. Testen der Anwendung	10
6. Abnahme- und Einführungsphase	11
7. Dokumentation.....	11
8. Fazit	12
8.1. Soll-/Ist-Vergleich.....	12
8.2. Lessons Learned	12
8.3. Ausblick.....	12

A.	Anhang	i
A.1	Detaillierte Zeitplanung	i
A.2	Ressourcenplanung	ii
A.3	Ablaufdiagramm	iii
A.4	Gitlab-Issue	iv
A.5	Mockup / Entwurf	v
A.6	Verzeichnisstruktur	vi
A.7	Manifest.....	vii
A.8	Klassendiagramm	viii
A.9	Verteilungsdiagramm	ix
A.10	Screenshots der fertigen Oberfläche	x
A.11	Quellcode	xi
A.11.1	Datei: <i>ir_attachment.py</i>	xi
A.11.2	Datei: <i>ir_attachment_export.py</i>	xi
A.11.3	Datei: <i>attachment_export_wizard_views.xml</i>	xii
A.11.4	Datei: <i>attachment_export_views.xml</i>	xii
A.11.5	Datei: <i>menu_views.xml</i>	xii
A.12	Unit-Tests	xiii
A.12.1	Datei: <i>test_ir_attachment.py</i>	xiii
A.12.2	Datei: <i>test_ir_attachment_export.py</i>	xiv
8.4.	Benutzerdokumentation.....	xvi
8.5.	Entwicklerdokumentation.....	xvii

Abkürzungsverzeichnis

TMS	Transport Management System
API	Application Programming Interface
DMS	Document Management System
ERP	Enterprise Resource Planning
MR	Merge Request
QS	Qualitätssicherung
XML	Extensible Markup Language
CI/CD	Continuous Integration / Continuous Delivery

Begriffserläuterung

Odoo ist eine integrierte ERP-Software mit einem dualen Lizenzmodell. Sie basiert auf dem Open-Core-Prinzip, bei dem der Kern der Software frei verfügbar ist, während zusätzliche Funktionen kommerziell angeboten werden. Odoo S.A. konzentriert sich dabei auf Module in den Bereichen ERP, CRM, Buchhaltung, CMS und E-Commerce.

Docker ist eine Containerplattform, mit der Software unabhängig vom zugrunde liegenden Betriebssystem betrieben werden kann. Anwendungen werden in sogenannten Containern gebündelt, die alle benötigten Komponenten enthalten. Dadurch lassen sich die Container flexibel kombinieren, einfach starten und auf verschiedenen Systemen wiederverwenden.

Unit-Tests sind automatische Tests zur Überprüfung einzelner Codebestandteile. Sie werden in einer isolierten Umgebung ausgeführt, in der gezielte Eingabewerte definiert und die Ausgaben mit den erwarteten Ergebnissen verglichen werden. Dadurch lassen sich Fehler frühzeitig erkennen und beheben.

Module sind in Odoo eigenständige Erweiterungen. Das beschriebene Modul ist ein Beispiel für ein solches Projekt, das aus mehreren strukturierten Dateien und Ordnern besteht. Module enthalten unter anderem Datenmodelle, Ansichten und Logik.

Modelle in Odoo definieren die Struktur von Datenobjekten, vergleichbar mit Tabellen in einer Datenbank. Sie werden in Python-Klassen angelegt und über das Attribut `_name = "..."` eindeutig benannt. Modelle sind zentral für die Datenverarbeitung und können in anderen Modulen referenziert werden.

Datensätze (Records) sind einzelne Einträge im DMS von Odoo, die jeweils eine Datei oder ein Dokument repräsentieren. Jeder Record enthält strukturierte Informationen wie Dateiname, Dateipfad, Erstellungsdatum und zugehöriger Benutzer. Diese Records ermöglichen die gezielte Verwaltung, Suche und Weiterverarbeitung von Dokumenten innerhalb des Systems.

Wizard ist in Odoo ein interaktives Dialogfenster, das den Benutzer schrittweise durch bestimmte Prozesse führt.

Views sind Darstellungen von Daten in der Benutzeroberfläche von Odoo.

1. Einleitung

1.1. Projektumfeld

Die Sirum GmbH ist ein Systemhaus und hat sich zum Ziel gesetzt, eine innovative Softwarelösung für die Transportlogistik zu entwickeln. Zurzeit beschäftigt Sirum etwa 30 Mitarbeiter. Während herkömmliche Lösungen oft nur einzelne Aspekte der Logistik abdecken, fokussiert sich Sirum auf eine ganzheitliche Optimierung, um Einschränkungen in der Gesamteffizienz zu überwinden. Mit dem Sirum Transport Management System ([TMS](#)), wurde eine umfassende Softwarelösung entwickelt, die auf dem Odoo-Framework basiert. Sirum setzt auf eine Software as a Service ([SaaS](#))-Architektur, die eine transparente und kalkulierbare Kostenstruktur ermöglicht.

1.2. Projektbeschreibung

Im Bereich der Dokumentenverwaltung wird bei der Firma ein zentrales Document Management System ([DMS](#)) eingesetzt, in dem verschiedene Dateitypen wie Rechnungen, Excel-Tabellen oder Verträge in Datensätze ([Records](#)) abgelegt und verwaltet werden. Da bisher keine Möglichkeit bestand, diese Dateien gezielt zu filtern und gesammelt herunterzuladen, mussten relevante Dokumente manuell gesucht, einzeln ausgewählt und heruntergeladen werden, ein zeitraubender und fehleranfälliger Prozess.

Ausgehend von einer konkreten Anforderung eines Kunden sollte ein Modul entwickelt werden, das diesen Prozess deutlich vereinfacht. Über eine benutzerfreundliche Oberfläche sollen Benutzer einen [Wizard](#) öffnen können, mit dem sie Dokumente anhand eines Datumsbereichs (von–bis) sowie nach [Modelle](#) filtern können. Die ausgewählten Dateien sollen anschließend in einem separaten Datensatz gespeichert werden, der später wiederverwendet und als ZIP-Datei heruntergeladen werden kann.

Das entwickelte Modul stellt somit eine umfassende Erweiterung des bisherigen DMS dar. Es soll zur Standardisierung der Dokumentenprozesse beitragen, den manuellen Aufwand reduzieren und die Übersichtlichkeit sowie die Nachvollziehbarkeit im Umgang mit großen Dokumentenmengen verbessern.

1.3. Projektschnittstellen

Das entwickelte Modul integriert sich vollständig in das bestehende Odoo-System. Es greift direkt auf das Modell `ir.attachment` zu, in dem alle Dokumente des DMS verwaltet werden. Dieses Modell dient als Datengrundlage für die Auswahl, Filterung und den Export von Anhängen.

Zusätzlich wurde eine neue Benutzeroberfläche ([View](#)) implementiert, die als Schnittstelle zwischen dem Anwender und den Daten dient. Alle relevanten Interaktionen, wie das Anwenden von Filtern, das Starten des Exports und das Anzeigen der Ergebnisse, erfolgen über diese View.

Eine weitere Schnittstelle besteht zu den [Wizards](#), die als Dialogfenster zur schrittweisen Eingabe von Filterkriterien und zur Durchführung der Aktionen dienen. Diese [Wizards](#) kommunizieren direkt mit dem erweiterten Datenmodell und steuern den Exportprozess.

Da alle Funktionen innerhalb des Odoo-Frameworks umgesetzt wurden, sind keine externen Systeme oder APIs notwendig.

1.4. Projektziel

Ziel des Projekts ist es, ein Modul zur effizienten Verwaltung und gezielten Bereitstellung von Dokumenten innerhalb des bestehenden DMS zu entwickeln. Es soll den Benutzern ermöglichen, Dokumente anhand eines definierten Datumsbereichs sowie bestimmter Kategorien zu filtern, die Ergebnisse übersichtlich darzustellen und bei Bedarf gebündelt als ZIP-Datei herunterzuladen.

2. Projektplanung

2.1. Zeitplanung

Für die Umsetzung des Projekts standen 80 Stunden zur Verfügung. Im Rahmen der Projektplanung wurden die Stunden auf verschiedene Bereiche aufgeteilt und somit der gesamte Projektablauf abgebildet. Eine grobe Zeitplanung kann aus Tabelle 1 entnommen werden. Eine detaillierte Zeitplanung mit den einzelnen Schritten der jeweiligen Bereiche befindet sich im Anhang [A.1: Detaillierte Zeitplanung](#).

Zeitplanung	Geplante Zeit
Analyse	8 h
Entwurf	12 h
Implementierung inkl. Tests	44 h
Abnahme und Einführung	5 h
Dokumentation	11 h
Gesamt	80 h

Tabelle 1: Zeitplanung

2.2. Ressourcenplanung

Im Anschluss an die Zeitplanung wurden alle für die Umsetzung des Projekts benötigten Ressourcen im Anhang [A.2: Ressourcenplanung](#) aufgelistet. Dabei wurden sowohl die eingesetzten Software- und Infrastrukturressourcen als auch das benötigte Personal berücksichtigt.

Für die Entwicklung kamen ausschließlich bereits lizenzierte oder intern bereitgestellte Werkzeuge und Umgebungen zum Einsatz, um Kompatibilität mit der bestehenden Systemlandschaft sicherzustellen und den Entwicklungsprozess effizient zu gestalten.

Im Vorfeld wurden bestehende Module mit ähnlicher Funktionalität geprüft. Da diese jedoch nicht alle Anforderungen erfüllten, wurde die Entwicklung eines eigenen Moduls beschlossen. Die vollständige Bewertung erfolgt in Abschnitt [3.2.1 Make-or-Buy-Entscheidung](#).

2.3. Entwicklungsprozess

Bei der Firma Sirum wird ein Kanban-basierter Entwicklungsprozess verwendet, der über das GitLab-Issue-Board organisiert ist. Das Board ist in die typischen Statusspalten **To Do**, **In Progress**, **Review** und **Done** unterteilt. Jedes neue Issue wird zunächst im Board erfasst und priorisiert.

Für jedes angelegte Issue wird ein eigener Git-Branch erstellt, auf dem die Umsetzung erfolgt. Nach der Entwicklung des jeweiligen Features werden Unit-Tests geschrieben, um die Funktionsweise des Moduls automatisiert zu überprüfen.

Anschließend erfolgt ein Code-Review durch einen Entwickler, der den Branch gemeinsam mit dem zugehörigen Issue prüft. Wenn keine Änderungen mehr notwendig sind, wird der Branch durch den unternehmensinternen Merge-Bot verarbeitet. Die dabei durchlaufenen Prüfschritte und Qualitätssicherungsmaßnahmen werden in Abschnitt [4.5: Maßnahmen zur Qualitätssicherung](#) näher beschrieben.

Durch dieses strukturierte Vorgehen werden Fehler frühzeitig erkannt und können bereits während der Implementierungsphase behoben werden. Gleichzeitig wird die Qualität des Quellcodes sichergestellt und der Entwicklungsprozess bleibt jederzeit transparent und nachvollziehbar.

Ein separates Pflichtenheft wurde im Rahmen des Projekts nicht erstellt, da die Anforderungen schrittweise im [A.4: GitLab-Issue](#) definiert und bei Bedarf angepasst wurden. Diese iterative Vorgehensweise entspricht dem üblichen Kanban-Prozess und ermöglichte eine flexible Umsetzung.

2.4. SMART-Ziele

Um zu überprüfen, ob die Entwicklung des Moduls den Anforderungen an ein Abschlussprojekt im Rahmen der Umschulung zum Fachinformatiker für Anwendungsentwicklung gerecht wird, wurden die Projektziele anhand der SMART-Kriterien definiert, diese werden in der Tabelle 2 erläutert.

Kriterium	Zielbeschreibung
Specific	Es soll ein Odoo-Modul entwickelt werden, mit dem Benutzer Dokumente im DMS gezielt nach einem Datumsbereich und einer Kategorie filtern, in einem Datensatz speichern und als ZIP-Datei herunterladen können.
Measurable	Die Umsetzung ist erfolgreich, wenn Benutzer mindestens drei Kategorien auswählen, einen Zeitraum eingrenzen, Dokumente in einem Datensatz speichern und diese als ZIP-Datei herunterladen können.
Achievable	Das Ziel ist mit den vorhandenen Ressourcen (Odoo-Instanz, Entwicklungsumgebung, interner Reviewprozess) innerhalb der verfügbaren 80 Projektstunden umsetzbar.
Relevant	Das Projekt löst ein reales Problem im Arbeitsalltag: Es reduziert den manuellen Aufwand beim Dokumentendownload und erhöht die Benutzerfreundlichkeit des DMS.
Time-bound	Das Modul wird innerhalb des vorgegebenen Projektzeitraums von 80 Stunden fertiggestellt und kann bis zum Projektende vollständig funktionsfähig in das bestehende System integriert.

Tabelle 2: SMART-Ziele

3. Analysephase

3.1. Ist-Analyse

Im aktuellen Zustand werden im DMS sämtliche Dateien zentral im Modell `ir.attachment` gespeichert. Dieses Modell dient als generische Ablage für alle Arten von Dokumenten, beispielsweise Rechnungen, Lieferscheine oder Excel-Dateien. Eine gezielte Filterung nach Kriterien wie Datum oder Kategorie ist im Standard nicht möglich. Die Suche und Auswahl von Dokumenten muss manuell über die Benutzeroberfläche erfolgen.

Ein weiterer Nachteil besteht darin, dass mehrere Dateien nur einzeln heruntergeladen werden können. Eine Funktion zum Sammel-Download mehrerer Dokumente, etwa durch eine ZIP-Komprimierung, ist nicht vorhanden. Gerade bei größeren Datenmengen stellt dies für den Benutzer einen hohen manuellen Aufwand dar. Zusätzlich limitieren moderne Webbrowser aus Sicherheitsgründen die Anzahl gleichzeitiger Downloads, was den Prozess weiter verlangsamt.

Die abgelegten Dokumente enthalten teilweise geschäftsrelevante und buchhaltungsbezogene Informationen. Eine strukturierte Verwaltung oder Trennung nach Dokumententypen ist aktuell nicht gegeben. Auch eine Möglichkeit, ausgewählte Dokumente temporär zu speichern, um sie später erneut herunterzuladen, existiert nicht.

Insgesamt ist die aktuelle Lösung funktional zwar ausreichend für die einfache Dokumentenablage, bietet jedoch keine effizienten Werkzeuge zur gezielten Auswahl, Verwaltung und strukturierten Bereitstellung von Dokumenten.

3.2. Wirtschaftlichkeitsanalyse

Aufgrund des geschilderten Sachverhalts der in Abschnitt [1.3 Projektbeschreibung](#) und in Abschnitt [3.1 Ist-Analyse](#) beschrieben wurde, ist die Umsetzung des Projekts erforderlich. Ob die Realisierung und die damit verbundenen wirtschaftlichen Aufwendungen gerechtfertigt sind, soll in den folgenden Abschnitten betrachtet werden.

3.2.1. „Make or Buy“-Entscheidung

Im Zuge der Projektplanung wurde geprüft, ob eine bestehende Lösung zur Erweiterung des DMS-Moduls verfügbar ist. Dabei wurde ein Modul im offiziellen Odoo-Appstore identifiziert, das den Download mehrerer Dokumente ermöglicht und grundsätzlich die gewünschte Funktionalität bietet.

Nach genauer Analyse wurde jedoch festgestellt, dass das Modul die spezifischen Anforderungen des Kunden nicht vollständig erfüllt.

Daher wurde entschieden, das benötigte Modul eigenständig zu entwickeln, um eine maßgeschneiderte Lösung bereitzustellen, die exakt auf die Anforderungen und bestehenden Abläufe zugeschnitten ist.

Beispielsweise fehlten im Drittanbieter-Modul die Möglichkeit, Dokumente nach Kriterien zu filtern, sowie die Option, gefilterte Ergebnisse als eigenständigen Datensatz für spätere Downloads zu speichern.

3.2.2. Projektkosten

Im Folgenden werden die während der Projektlaufzeit entstehenden Kosten kalkuliert. Grundlage der Kalkulation sind interne Stundensätze, die pauschal pro Rolle angesetzt wurden. Neben den Personalkosten für Entwicklung und Code-Review wurden keine zusätzlichen Kosten für Ressourcen wie Software oder Hardware angesetzt, da ausschließlich firmeneigene Infrastruktur und bereits lizenzierte Softwarekomponenten verwendet wurden (siehe Abschnitt [2.2 Ressourcenplanung](#)).

Das Modul wurde von zwei Mitarbeitern umgesetzt, wobei einer die Qualitätssicherung durch Code-Reviews übernahm. Für die Kalkulation wurde ein Stundensatz von 45 € für den Auszubildenden (Entwicklung) und 75 € für einen festangestellten Mitarbeiter (Code-Review) angesetzt.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	80 h	45 €	3.600 €
Code Review	10 h	75 €	750 €
Gesamt	90 h	//	4.350 €

Tabelle 3: Kostenaufstellung

3.2.3. Amortisation

Die Umsetzung des Projekts ist nicht nur aus funktionaler, sondern auch aus wirtschaftlicher Sicht sinnvoll. Der Kunde, der monatlich 4.500 € und somit jährlich 54.000 € für die Nutzung des Systems

zahlt, hat ein konkretes Interesse an der neuen Funktionalität geäußert. Die Möglichkeit, Dokumente effizient zu filtern und gesammelt herunterzuladen, spart dem Kunden wertvolle Arbeitszeit und verbessert die alltäglichen Abläufe erheblich.

Vor Einführung des Moduls mussten die benötigten Dokumente manuell im DMS gesucht, geöffnet, einzeln heruntergeladen und anschließend manuell archiviert werden. Für einen typischen Exportvorgang wurden im Schnitt etwa 20 Minuten pro Vorgang benötigt. Nach Einführung des Moduls beträgt der Zeitaufwand für denselben Vorgang nur noch ca. 2 Minuten.

Bei einer geschätzten Nutzung von 3 Exportvorgängen pro Tag ergibt sich folgende beispielhafte Zeitersparnis:

Vorher:

$$3 \text{ Vorgänge} \times 20 \text{ Minuten} \times 20 \text{ Arbeitstage} = 1.200 \text{ Minuten pro Monat (20 Stunden)}$$

Nachher:

$$3 \text{ Vorgänge} \times 2 \text{ Minuten} \times 20 \text{ Arbeitstage} = 120 \text{ Minuten pro Monat (2 Stunden)}$$

Angesichts dieser Ausgangslage wäre es langfristig nachteilig, auf die Umsetzung zu verzichten, da dies die Zufriedenheit des Kunden gefährden könnte. Ein möglicher Verlust dieses Kunden würde für das Unternehmen nicht nur einen erheblichen finanziellen Schaden bedeuten, sondern auch das Vertrauen in die Weiterentwicklung des Systems beeinträchtigen. Durch die geringe Projektlaufzeit und die kurze Amortisationsdauer lohnt sich die Investition in dieses Modul aus unternehmerischer Sicht auf jeden Fall.

3.3. Nicht-monetäre Aspekte

Durch die gezielte Filterung von Dokumenten nach Datum und Kategorie wird der bisher manuelle und zeitaufwändige Suchprozess deutlich vereinfacht. Benutzer können relevante Dokumente nun schneller auffinden und strukturierter verarbeiten.

Die Möglichkeit, gefilterte Dokumente in einem temporären Datensatz zu speichern und als ZIP-Datei herunterzuladen, erhöht zusätzlich die Effizienz und reduziert das Fehlerrisiko bei der Dateiauswahl. Dadurch wird die tägliche Arbeit nicht nur beschleunigt, sondern auch benutzerfreundlicher gestaltet.

Zudem trägt das Modul zu einer höheren Systemintegration bei, da es sich nahtlos in das bestehende Odoo-System einfügt und auf bereits vorhandene Strukturen wie das `ir.attachment`-Modell aufbaut. Eine zusätzliche Einarbeitung oder externe Tools sind nicht erforderlich.

3.4. Anwendungsablauf

Zur Erfassung der fachlichen Anwendungsfälle und zur Veranschaulichung des Nutzerablaufs wurde im Rahmen der Analyse ein Ablaufdiagramm erstellt. Es zeigt die einzelnen Schritte vom Start des Prozesses bis zum finalen Download der gefilterten Dokumente. Das vollständige Diagramm befindet sich im Anhang [A.3: Ablaufdiagramm](#).

3.5. Anforderungen

Das Lastenheft wurde nicht in direkter Abstimmung mit dem Kunden erstellt, sondern durch einen internen Wirtschaftsmitarbeiter, der als fachlicher Ansprechpartner agierte. Dieser sammelte die Anforderungen im Austausch mit dem Kunden und überführte sie in eine konkrete Aufgabenstellung.

Die dokumentierten Anforderungen wurden anschließend im firmeneigenen GitLab-System als Issue erfasst und bildeten die offizielle Grundlage für die Projektumsetzung. Das Issue enthält alle relevanten Informationen zur Zielsetzung, darunter die Möglichkeit, Dokumente im DMS nach einem Datumsbereich und einer Kategorie zu filtern, diese als Datensatz zu speichern und anschließend gebündelt als ZIP-Datei herunterzuladen.

Ein Screenshot des originalen GitLab-Issues befindet sich im Anhang [A.4: GitLab-Issue](#) zur Veranschaulichung der Anforderung.

Die technische Feinausarbeitung erfolgte iterativ im Rahmen der Merge-Requests ([MR](#)) in enger Abstimmung mit den Entwicklern. Auf diese Weise konnten funktionale Details, Validierungen und Abläufe schrittweise präzisiert und direkt im Entwicklungskontext umgesetzt werden.

4. Entwurfsphase

4.1. Entwurf der Benutzeroberfläche

Bevor mit der technischen Umsetzung begonnen wurde, erfolgte zunächst die Planung der Benutzeroberfläche auf konzeptioneller Ebene. Ziel war es, eine intuitive und übersichtliche Oberfläche zu gestalten, die sich visuell und funktional nahtlos in das bestehende Odoo-System einfügt.

Zu Beginn wurden bestehende Oberflächen aus anderen Odoo-Modulen innerhalb des Unternehmens analysiert. Dabei lag der Fokus auf wiederkehrenden Designmustern wie Layout-Strukturen, typischen Feldgruppen, Button-Anordnungen sowie dem allgemeinen Aufbau von [Wizards](#) und Listenansichten. Diese Analyse diente als Grundlage für ein einheitliches Erscheinungsbild und eine benutzerfreundliche Bedienung.

Anschließend wurde ein erster Entwurf der neuen Oberfläche, ein sogenanntes Mockup, erstellt. Dieses zeigt den grundlegenden Aufbau der Benutzeroberfläche, einschließlich der Filterfelder für Datum und Modell, der tabellarischen Übersicht mit Feldern wie Exportname, Dateigröße und Anzahl der Anhänge sowie der Schaltflächen zur Steuerung des Workflows („Check Export“, „Pack as ZIP“). Die Skizze wurde intern vorgestellt und im Abstimmungsprozess mit den zuständigen Entwicklern bestätigt.

Das Mockup diente während der weiteren Umsetzung als visuelle Orientierung und half dabei, die Anforderungen klar abzubilden. Es gewährleistete eine konsistente Umsetzung und erleichterte spätere Erweiterungen.

Zur Veranschaulichung befindet sich die entworfene Skizze im Anhang [A.5: Mockup / Entwurf](#) der Benutzeroberfläche.

4.2. Technischer Entwurf

Das entwickelte Modul erweitert das bestehende Modell `ir.attachment` um Funktionen zur Filterung, Zwischenspeicherung und komprimierten Bereitstellung von Dateien. Die Umsetzung erfolgt in Form eigener Python-Modelle und XML-Views, die sich vollständig in die bestehende Systemstruktur integrieren.

Das Modell `ir.attachment.export` übernimmt die Aufgabe, dem Benutzer eine benutzerfreundliche Oberfläche in Form eines [Wizards](#) bereitzustellen. Darüber kann der Nutzer Dokumente anhand eines Datumsbereichs und einer Modellauswahl filtern. Die gefundenen Anhänge werden anschließend zu einer ZIP-Datei gebündelt und zum Download bereitgestellt.

Die ZIP-Erstellung erfolgt serverseitig mit Standardbibliotheken (`zipfile`, `base64`, `BytesIO` und `datetime`) und wird zusätzlich als Datensatz gespeichert. Der Workflow wird über State-Felder und Aktionsbuttons wie „Check Export“ und „Pack as ZIP“ gesteuert.

Zur besseren Nachvollziehbarkeit der technischen Struktur befindet sich im Anhang [A.6: Verzeichnisstruktur](#) eine Übersicht über den Aufbau des Moduls. Die Darstellung zeigt die Aufteilung in Python-Modelle, Wizards und XML-View-Dateien, die gemeinsam die Funktionalität des Moduls abbilden.

4.3. Datenmodell

Das bereits vorhandene Modell `ir.attachment`, das in Odoo zur Speicherung aller Dateianhänge dient, wurde für dieses Projekt erweitert. Ergänzt wurde es um das zusätzliche Feld **`is_exported`** (Boolean), das kennzeichnet, ob ein Anhang bereits Bestandteil eines Exports ist.

Das Modell `ir.attachment.export` enthält Filterkriterien wie den Datumsbereich (**`start_date`**, **`end_date`**), eine Auswahl relevanter Modellkategorien (**`model_ids`**) sowie die daraus gefilterten Anhänge (**`attachment_ids`**). Letztere sind über eine Many2many-Beziehung mit `ir.attachment` verknüpft, was eine flexible und dynamische Zuordnung der Dateianhänge ermöglicht.

Die wichtigsten Modelle und Felder im Überblick:

`ir.attachment.export`

- **`start_date`**, **`end_date`**: Filter für den Erstellungszeitraum der Anhänge
- **`model_ids`**: Auswahl der betroffenen Modelle (z. B. Rechnungen)
- **`attachment_ids`**: Ergebnisliste der verknüpften Anhänge (Many2many zu `ir.attachment`)
- **`total_attachment_size`**: Automatische Berechnung der Gesamtgröße der Dateien
- **`state`**: Status des Vorgangs (z. B. „Draft“, „Open“, „Done“)

`ir.attachment`

- **`is_exported`**: Kennzeichnung, ob ein Dokument bereits Teil eines Exports ist

Das resultierende Datenmodell orientiert sich an der objektorientierten Struktur von Odoo und wurde zur besseren Übersicht als Klassendiagramm visualisiert. Die grafische Darstellung befindet sich im Anhang [A.8: Klassendiagramm](#).

4.4. Modulstruktur und Verzeichnisübersicht

Die technische Umsetzung des Moduls folgt dem standardisierten Aufbau von Odoo-Erweiterungen. Jede Funktionseinheit ist klar strukturiert und in eigenen Dateien abgelegt. So befinden sich die Python-Modelle beispielsweise in Dateien wie `ir_attachment_export.py`, während die zugehörigen Benutzeroberflächen in separaten XML-Dateien, wie `attachment_export_views.xml`, definiert sind.

Das Modul gliedert sich in mehrere thematisch getrennten Verzeichnisse:

- Im Ordner **`models`** befinden sich alle Python-Dateien, die die technische Logik des Moduls enthalten. Hier werden die Modelle definiert und deren Verhalten beschrieben.
- Der Ordner **`views`** beinhaltet die XML-Dateien zur Gestaltung der Benutzeroberflächen. Diese legen fest, wie Formulare, Listen und [Wizards](#) im Frontend dargestellt werden.
- Im Verzeichnis **`wizards`** beinhaltet auch XML-Dateien, die für die Gestaltung des Wizard zuständig sind.
- Der Ordner **`tests`** enthält die automatisierten Unittests, mit denen zentrale Funktionen des Moduls auf ihre Korrektheit geprüft werden.
- Ergänzend enthält das Modul die Konfigurationsdateien **`__init__.py`** zur Initialisierung und **`__manifest__.py`** zur Definition der Modulinformationen und Abhängigkeiten.

Diese saubere Trennung von Logik, Oberfläche und Tests erleichtert nicht nur die Wartung, sondern unterstützt auch eine modulare Erweiterbarkeit des Projekts. Wie erwähnt wird zur Veranschaulichung im [Anhang A.6: Verzeichnisstruktur](#) ein Screenshot der vollständigen Ordner- und Dateistruktur des

Moduls bereitgestellt. Ergänzend wird im Anhang [A.7: Manifest](#) die `__manifest__.py`-Datei erläutert, die als Grundlage für die Registrierung, Beschreibung und Integration des Moduls im Odoo-System dient.

4.5. Maßnahmen zur Qualitätssicherung

Zur Sicherstellung der Qualität des entwickelten Moduls wurden verschiedene Maßnahmen und Prüfprozesse durchgeführt.

Bereits während der Entwicklung wurden für zentrale Funktionen Unit-Tests erstellt. Diese Tests wurden lokal ausgeführt, um die Funktionsfähigkeit einzelner Komponenten frühzeitig zu validieren. Nach Abschluss der Entwicklung wurden die Tests erneut ausgeführt, um sicherzustellen, dass alle Funktionen wie erwartet arbeiten.

Bevor ein Feature in den Hauptzweig übernommen werden konnte, wurde für jedes GitLab-Issue ein Code-Review durch einen Entwickler durchgeführt. Dabei wurde der Quellcode geprüft und ggf. mit Kommentaren versehen, falls Verbesserungsbedarf bestand. Wenn dies erfolgreich geprüft wurde, dann musste es einen mehrstufigen Qualitätssicherungsprozess auf GitLab durchlaufen. Dieser bestand aus folgenden automatisierten Tests:

- **Deploy Test:** Prüfung, ob das Modul erfolgreich gebaut und gestartet werden kann
- **Translate Test:** Kontrolle auf korrekte Übersetzungsdateien und -strukturen
- **Lint Test:** Prüfung des Quellcodes auf Einhaltung von Syntax- und Stilvorgaben

Erst nach erfolgreicher Prüfung und Bestehen aller Tests konnte der Merge über den unternehmensinternen Merge-Bot erfolgen.

Dieser strukturierte Ablauf ermöglichte es, Fehler frühzeitig zu erkennen, die Codequalität konstant hochzuhalten und gleichzeitig einen kontrollierten Übergang der Features in das produktive System sicherzustellen.

4.6. Zugriffsrechte und Sicherheit

Um die Sicherheit der Exportfunktion zu gewährleisten, wurde eine rollenbasierte Zugriffskontrolle implementiert. Nur Benutzer, die einer berechtigten Odoo-Benutzergruppe (z. B. „Dokumentenmanager“) angehören, können Exporte erstellen und herunterladen. Die Sichtbarkeit des Menüeintrags sowie der Zugriff auf [Wizards](#) und Buttons ist über das Gruppen-Attribut in den XML-View-Dateien eingeschränkt. Zusätzlich wird die erzeugte ZIP-Datei nicht dauerhaft auf dem Server gespeichert und kann nur im Kontext eines aktiven, berechtigten Benutzers heruntergeladen werden. Dadurch wird verhindert, dass sensible Dokumente unbeabsichtigt oder unbefugt exportiert werden.

4.7. Deployment

Das entwickelte Modul wird innerhalb einer containerisierten Odoo-Umgebung betrieben, die durch die Firma bereitgestellt wurde. Die Entwicklung erfolgte mit Visual Studio Code, während die Quellcodeverwaltung und der Review-Prozess über GitLab abgewickelt wurden.

Die Anwendung wurde in einem Docker-Container ausgeführt, der das Odoo-System samt Modul beinhaltet. Die zugehörigen Daten werden über eine angebundene PostgreSQL-Datenbank verwaltet.

Benutzer greifen über einen Webbrowser auf die Odoo-Oberfläche zu, die das entwickelte Modul nahtlos integriert. Änderungen am Quellcode durchlaufen automatisierte Prüfprozesse, bevor sie in den Hauptzweig übernommen und auf der Zielumgebung bereitgestellt werden. Ein schematisches Verteilungsdiagramm zur Übersicht über die Systemkomponenten befindet sich im Anhang [A.9: Verteilungsdiagramm](#).

5. Implementierungsphase

5.1. Implementierung der Benutzeroberfläche

Zur Bedienung des Moduls wurde im DMS-Bereich von Odoo ein neuer Menüeintrag eingebunden, über den die Exportansicht aufgerufen werden kann. In dieser Ansicht wurde eine Tree-View implementiert, die eine tabellarische Übersicht aller angelegten Exporte darstellt.

Die Spalten der Übersicht enthalten unter anderem:

- **Export Name:** Bezeichnung des Exports
- **Modell(e):** Auswahl betroffener Modelle über ein many2many-Feld
- **Start Date / End Date:** Zeitraum zur Filterung der Anhänge
- **Total File Size:** Gesamtgröße aller enthaltenen Dateien
- **Attachments Total:** Anzahl der enthaltenen Anhänge

Über den Button „**Erstellen**“ öffnet sich ein Wizard. Dort kann ein neuer Export konfiguriert werden. Nachdem alle Felder ausgefüllt wurden, löst der Button „**Check Export**“ eine Prüfung aus, bei der die passenden Anhänge gefiltert und dem Export zugeordnet werden. Anschließend wechselt der Status des Datensatzes in „**Open**“.

In diesem „**Open**“-Status erhält der Benutzer eine detaillierte Übersicht über die gefundenen Dokumente, die gesetzten Filter und die Exportdetails. Sobald die Auswahl bestätigt ist, kann der Benutzer den Button „**Pack as ZIP**“ betätigen. Dies erzeugt die komprimierte ZIP-Datei und wechselt den Status auf „**Done**“.

Im „**Done**“-Status ist der Export abgeschlossen. Der Benutzer kann keine Änderungen mehr vornehmen, erhält jedoch eine lesbare Zusammenfassung der enthaltenen Dateien und Filtereinstellungen.

Die Umsetzung der Benutzeroberfläche erfolgte vollständig mit den Standardfunktionen des Odoo-Frameworks. Die Form- und Listenansichten wurden in der Datei `attachment_export_views.xml` definiert. Der mehrstufige Wizard zur Erstellung eines Exports wurde in einer separaten Datei (`attachment_export_wizard_views.xml`) gekapselt, um eine saubere und modulare Struktur zu gewährleisten.

Die verwendeten Buttons wie „**Check Export**“ und „**Pack as ZIP**“ wurden mit serverseitigen Python-Methoden verknüpft, welche die Statusübergänge steuern, Anhänge filtern und die ZIP-Datei erzeugen. Dadurch entsteht ein klar geführter und nachvollziehbarer Workflow für den Benutzer.

Ein Screenshot der fertigen Benutzeroberfläche befindet sich im [Anhang A.10: Screenshots der fertigen Benutzeroberfläche](#).

5.2. Implementierung der Datenstrukturen

Zur Umsetzung der datenhaltenden Strukturen des Moduls wurde, in Anlehnung an das entworfene [A.8: Klassendiagramm](#), die Datei `ir_attachment_export.py` in Odoo erstellt. Die zentrale Rolle übernimmt dabei das Modell `ir.attachment.export`, das für die Filterung und temporäre Zwischenspeicherung von Dokumenten verantwortlich ist.

Die Modelle wurden als Python-Klassen definiert, wobei durch das Attribut `_name = "ir.attachment.export"` ein eindeutiger Modellname innerhalb von Odoo vergeben wurde. Die zugehörigen Felder wie `start_date`, `end_date`, `attachment_ids` und `model_ids` bilden die notwendigen Attribute zur Eingabe und Verarbeitung der Filterkriterien sowie zur Verwaltung der exportierten Dateien ab.

Die Beziehung zwischen den Modellen erfolgt über Feldtypen wie Many2many, um eine flexible Verknüpfung mit dem Standardmodell `ir.attachment` herzustellen, in dem alle Anhänge gespeichert sind. Dadurch ist sichergestellt, dass mehrere Dokumente einem Export zugeordnet werden können. Zur Speicherung des Status des Exports wurde ein Selection-Feld mit den Zuständen „**draft**“, „**open**“ und „**done**“ integriert.

Ein zusätzliches Feld **is_exported** wurde im Standardmodell `ir.attachment` ergänzt, um zu kennzeichnen, ob ein Dokument bereits Teil eines Exports war. Dadurch lässt sich verhindern, dass dieselben Dateien mehrfach verarbeitet werden.

Zur technischen Umsetzung der Download-Funktion wurde keine separate REST-Schnittstelle benötigt, da das Odoo-Framework bereits serverseitige Aktionsmethoden sowie ein integriertes Dateihandling über das Modell `ir.attachment` bereitstellt. Dadurch konnte der gesamte Exportprozess, von der Filterung über die Zwischenspeicherung bis hin zur Erstellung der ZIP-Datei, vollständig innerhalb des Odoo-Systems realisiert werden. Diese enge Integration steigert die Wartbarkeit und reduziert potenzielle Sicherheitsrisiken, da keine externen Schnittstellen geöffnet oder gewartet werden müssen.

5.3. Implementierung der Geschäftslogik

Die Geschäftslogik des entwickelten Moduls wurde vollständig in der Modellklasse `ir.attachment.export` umgesetzt. Dabei handelt es sich um die zentrale Logikschicht, welche für das Filtern, Speichern, Verpacken und Validieren von Dokumenten verantwortlich ist. Ziel war es, die fachlichen Anforderungen so umzusetzen, dass der Benutzer jederzeit eine transparente und nachvollziehbare Prozessführung erlebt, von der Filterung bis zum finalen ZIP-Export.

Die Methode `action_check_attachments()` überprüft anhand definierter Kriterien, wie Datumsbereich, Modell und Dateigröße, welche Anhänge im System vorhanden sind und zur Auswahl passen. Die Treffer werden im Feld `attachment_ids` hinterlegt und gleichzeitig wird die Variable **is_exported** auf True versehen, um doppelte Exporte zu vermeiden. Wird kein Treffer gefunden, erfolgt eine kontrollierte Fehlermeldung an den Nutzer mittels der Odoo-internen `UserError`.

Über das `@api.depends`-Dekorator wurde mit `_compute_total_attachment_size()` eine Methode erstellt, die automatisch die Gesamtgröße der ausgewählten Dateien berechnet und benutzerfreundlich in Kilobyte, Megabyte oder Gigabyte anzeigt. Dies erlaubt dem Benutzer eine direkte Einschätzung der Exportmenge.

Ein besonderes Augenmerk lag auf der Implementierung der Methode `pack_zip()`. Diese erzeugt serverseitig eine komprimierte ZIP-Datei aller im Export befindlichen Anhänge. Die Erstellung erfolgt speicherschonend über einen `BytesIO`-Buffer. Dabei wird sichergestellt, dass gleichnamige Dateien korrekt durchnummeriert werden, um Konflikte zu vermeiden. Das Ergebnis wird anschließend als neuer `ir.attachment`-Datensatz dem System hinzugefügt und mit dem aktuellen Export verknüpft.

Um dem Benutzer ein konsistentes Nutzungserlebnis zu bieten, wurden Zustandsübergänge definiert. Die Auswahl und Zuordnung der Anhänge führt zu einem Wechsel von **Draft** zu **Open**, die finale ZIP-Erstellung auf **Done**. Diese Statuswerte werden intern im Feld `state` verwaltet und dienen auch der Steuerung der Oberflächenelemente.

Auch kleinere Hilfsfunktionen wie `_get_excluded_files_domains()` oder `_format_size()` sorgen im Hintergrund für eine klare Trennung von Geschäftsregeln und Darstellung, ganz im Sinne guter Softwarearchitektur.

Ein Auszug des relevanten Quellcodes ist im Anhang [A.11: Quellcode](#) dargestellt.

5.4. Testen der Anwendung

Zu Beginn wurden zentrale Funktionen des Moduls manuell über die Benutzeroberfläche getestet. Dabei wurden die einzelnen Schritte, von der Filterung der Dokumente über die Statusübergänge bis

hin zur Erstellung und dem Herunterladen der ZIP-Datei, direkt im System durchlaufen. Diese manuellen Tests dienten dazu, die Benutzerführung zu überprüfen und sicherzustellen, dass alle vorgesehenen Abläufe korrekt umgesetzt wurden.

Im Anschluss daran wurden automatisierte Unittests entwickelt, um die technischen Funktionsweisen einzelnen Methoden gezielt zu prüfen. Zum Erstellen der Unittest-Dateien wurden zunächst die notwendigen Bibliotheken aus dem Odoo-Testframework importiert. Danach wurde eine Testklasse eingerichtet, in der mithilfe der Methode `setUp` eine kontrollierte Testumgebung erzeugt wird. In dieser Umgebung werden künstliche Daten erzeugt, die für die Tests notwendig sind, darunter ein beispielhafter Datensatz sowie ein neuer Export-Datensatz mit einem vordefinierten Zeitrahmen.

Ziel der Tests war es, die Filter- und Exportfunktionalität des Moduls vollständig zu überprüfen. In einem der Tests wird z. B. geprüft, ob beim Ausführen der Funktion zur Überprüfung der Anhänge die Datei korrekt dem Export-Datensatz zugeordnet wird. Es wird kontrolliert, ob die Datei in der Ergebnisliste enthalten ist, ob die Gesamtanzahl und -größe der Anhänge korrekt berechnet wurde und ob das System den Status des Exports automatisch auf „**Open**“ setzt.

Ein weiterer Test behandelt die Erzeugung einer ZIP-Datei. Daraufhin wird geprüft, ob eine entsprechende ZIP-Datei im System gespeichert wurde. Zudem wird die Datei im Test entpackt und der Inhalt analysiert, um sicherzustellen, dass genau die erwarteten Dateien enthalten sind, sowohl vom Namen als auch vom Inhalt her.

Darüber hinaus wurde in einer separaten Testklasse das Verhalten von `is_exported` untersucht. Dabei wurde sichergestellt, dass Dateien nach einem Export korrekt als „exportiert“ markiert werden und diese Markierung automatisch zurückgesetzt wird, falls der zugehörige Export gelöscht wird. Diese Logik verhindert, dass Dateien fälschlicherweise doppelt verarbeitet oder ausgelassen werden.

Diese Testfälle decken zentrale Funktionen des Moduls ab, von der Auswahl und Verarbeitung der Anhänge über Statusänderungen bis zur abschließenden Archivierung. Dadurch konnte ein Großteil der potenziellen Fehlerquellen bereits vor dem manuellen Testen identifiziert und behoben werden.

Der Unittest-Quellcode befindet sich zur Veranschaulichung im Anhang [A.12: Unit-Tests](#).

6. Abnahme- und Einführungsphase

Zum aktuellen Stand wurde das entwickelte Modul noch nicht in die produktive Umgebung übernommen. Der Merge in den Master-Branch steht noch aus, da vor der finalen Abnahme durch den zuständigen Entwicklerteamleiter noch kleinere funktionale und visuelle Details überprüft werden müssen.

Das Modul befindet sich in der abschließenden Test- und Review-Phase. Die grundlegenden Funktionen wurden bereits erfolgreich getestet und bestehen die automatisierten CI-Prüfprozesse. Die abschließende Abnahme erfolgt nach Freigabe der letzten Anpassungen, woraufhin das Modul in die Zielumgebung gemerged und offiziell eingeführt wird.

7. Dokumentation

Im Verlauf wurde ein Benutzerhandbuch und eine Entwicklerdokumentation erstellt, um sowohl die Anwendung als auch die technische Umsetzung nachvollziehbar darzustellen.

Das Benutzerhandbuch richtet sich an die Anwender des Moduls und beschreibt die Nutzung der Oberfläche sowie den Ablauf eines Exports. Es erläutert die Schritte zur Filterung von Dokumenten, das Anlegen eines Exports sowie das Erstellen und Herunterladen einer ZIP-Datei. Ein Auszug des Benutzerhandbuchs befindet sich im Anhang [A.13: Benutzerdokumentation](#).

Die Entwicklerdokumentation dient der technischen Nachvollziehbarkeit und richtet sich an Entwickler, die das Modul warten oder erweitern sollen. Sie enthält eine Beschreibung der wichtigsten Klassen

sowie deren Felder und Methoden. Die Dokumentation wurde direkt im Quellcode verfasst und über die Entwicklungsumgebung strukturiert gepflegt. Ein Auszug der Entwicklerdokumentation befindet sich im Anhang [A.14: Entwicklerdokumentation](#).

8. Fazit

8.1. Soll-/Ist-Vergleich

Das Projekt kann insgesamt als erfolgreich bewertet werden, da alle im Vorfeld definierten Anforderungen vollständig umgesetzt wurden. Die geplanten Funktionen, darunter die Filterung von Dokumenten nach Datum und Kategorie, die Zwischenspeicherung der Ergebnisse sowie die Erstellung und der Download einer ZIP-Datei, wurden wie vorgesehen realisiert.

Während der Implementierungsphase zeigte sich, dass durch eine frühzeitigere Einhaltung der im Unternehmen etablierten Lint-Vorgaben (z. B. Code-Formatierung, Einrückungen und Dateistruktur) zusätzlicher Zeitaufwand hätte vermieden werden können. Diese kleineren Korrekturen führten stellenweise zu Verzögerungen im Review-Prozess. Ohne diese Nachbesserungen hätte das Projekt voraussichtlich sogar etwas früher abgeschlossen werden können.

Trotz dieses Optimierungspotenzials verlief die Entwicklung insgesamt reibungslos, und das Modul konnte wie geplant bereitgestellt werden.

8.2. Lessons Learned

Im Verlauf des Projekts konnte ich vielfältige praktische Erfahrungen sammeln und mein Wissen in mehreren Bereichen deutlich erweitern. Besonders die Arbeit mit dem Odoo-Framework und dem bestehenden DMS bot einen intensiven Einblick in die Modulstruktur, Modellierung und Erweiterung bestehender Systeme.

Ein weiterer Schwerpunkt lag in der Nutzung des GitLab-Issue-Boards. Ich habe gelernt, wie Aufgaben strukturiert erfasst, nachverfolgt und im Team effizient bearbeitet werden. Dabei konnte ich auch meine Kommunikationsfähigkeit stärken und regelmäßiges Feedback mit Entwicklern erfolgreich in den Entwicklungsprozess einbinden.

Darüber hinaus habe ich ein besseres Verständnis für sauberen, wartbaren Quellcode („Clean Code“) entwickelt. Die Bedeutung klarer Strukturen, nachvollziehbarer Methoden und der Einhaltung etablierter Standards wurde im Projektverlauf besonders deutlich, nicht zuletzt durch die automatisierten Lint-Tests.

Auch im Bereich der Qualitätssicherung konnte ich durch die Erstellung und Ausführung von Unittests wertvolle Einblicke gewinnen. Ich habe gelernt, wie Tests eine zuverlässige Absicherung der Funktionalität bieten und dazu beitragen, Fehler frühzeitig zu erkennen.

Nicht zuletzt wurde das Projekt in einer Docker-basierten Umgebung umgesetzt. Der Umgang mit Containern hat mir geholfen, die Vorteile einer standardisierten und isolierten Entwicklungsumgebung besser zu verstehen. Ergänzend konnte ich auch meine Kenntnisse im Bereich der Versionsverwaltung vertiefen, insbesondere im Hinblick auf Branch-Strategien, Commits und den Merge-Prozess.

8.3. Ausblick

Alle definierten Anforderungen wurden im Rahmen des Projekts erfolgreich umgesetzt. Für die Zukunft bestehen verschiedene Möglichkeiten, die Funktionalität des Moduls weiter auszubauen. Denkbar wären etwa zusätzliche Filteroptionen, wie etwa nach Dateigröße oder Dokumentenstatus, sowie automatisierte Exporte auf Basis festgelegter Zeitintervalle, um wiederkehrende Aufgaben zu vereinfachen.

Auch die Einführung einer Benutzerverwaltung könnte sinnvoll sein, um differenzierte Zugriffsrechte für Exporte zu definieren und damit die Sicherheit und Nachvollziehbarkeit der Nutzung zu erhöhen.

Ein weiterer Schritt könnte darin bestehen, das entwickelte Modul zentral über die Odoo-Kompetenzarchitektur (OKA) bereitzustellen. Dadurch könnten auch andere Nutzer oder Abteilungen innerhalb der Organisation von der Lösung profitieren und diese bedarfsgerecht in ihre Arbeitsabläufe integrieren.

A. Anhang

A.1 Detaillierte Zeitplanung

Analyse	8 h
Besprechung des Projekts und der Anforderungen	
Aufwandsschätzung & Nutzenbetrachtung	
Entwurf	12 h
Besprechung des Projekts und der Anforderungen	
Strukturierung & Mockup-Entwurf	
Diagramme erstellen	
Vorbereitung der Entwicklungsstruktur	
Implementierung inkl. Tests	44 h
Modelle und Beziehungen vorbereiten	
Umsetzung der Geschäftslogik	
Gestaltung der Benutzeroberfläche	
Modulstruktur & Integration	
Manuelles Testing	
Unit-Tests schreiben & ausführen	
Abnahme und Einführung	0 h
Dokumentation	16 h
Benutzerdokumentation	
Entwicklerdokumentation	
Gesamt	80 h

A.2 Ressourcenplanung

Hardware

- Büroarbeitsplatz mit Firmenlaptop

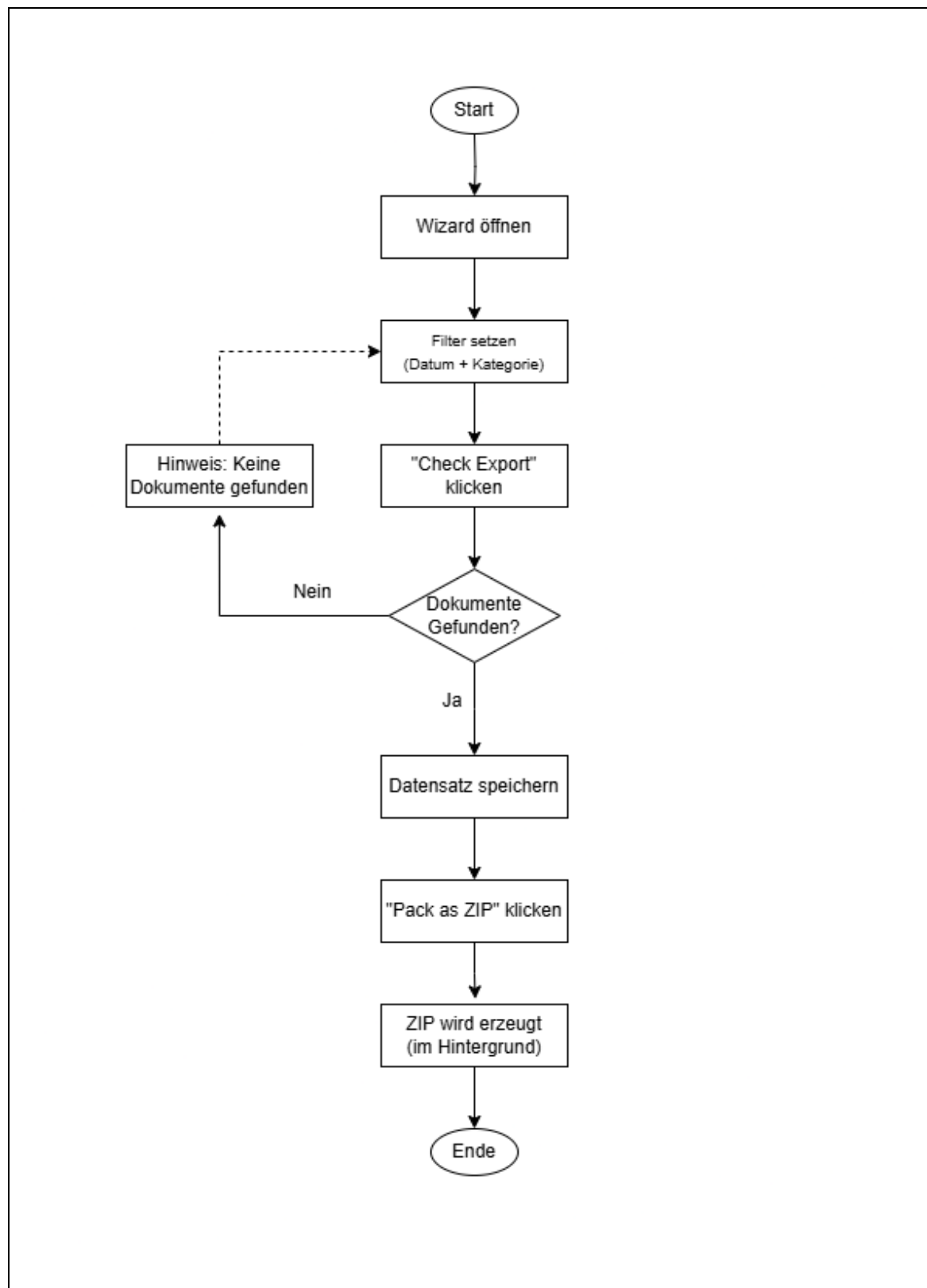
Software

- Linux Mint – Betriebssystem
- Visual Studio Code – Entwicklerumgebung
- PostgreSQL – Datenbanksystem
- Odoo 11.14 - ERP-System-Plattform, für die Entwicklung des Moduls
- Python 2.7 - Programmiersprache für die Modulentwicklung in Odoo 11
- Docker – Software für das Ausführen von Anwendungen in Containern
- Sublime Merge - Versionsverwaltung, CI/CD, Issue-Tracking und Code-Review
- Gitlab - Git-Client für visuelle Verwaltung von Branches und Merges
- Webbrowser (Chrome) - Für das Testen und Bedienen der Odoo-Oberfläche


Personal

- Auszubildender Anwendungsentwickler – Umsetzung des Projektes
- Anwendungsentwickler – Review der Pull-Requests
- Mitarbeiter der Fachabteilung – Festlegung der Anforderungen und Abnahme der Anwendung

A.3 Ablaufdiagramm



A.4 Gitlab-Issue



Erstelle einen Fileexport Module nach dem Muster des vorhandenen Finanzexport Module

Issue created 4 months ago by Maximilian Gloss

Anwendungsfall

Der Kunde wünscht, dass er alle im System gesicherten Dateien (Rechnungen, Belege etc.) sequenziell komprimiert (zip) herunterladen kann. Aktuell kann Dateianhänge nur einzeln herunterladen.

Diese Funktionalität soll in einem Fileexport Module umgesetzt werden, ähnlich zu dem vorhandenen Finanzexport Module.

Was ist das erwartete Verhalten?

Workflow

- Lege zwei neue Menüpunkte "Files" und "File Export" im DMS Module an. Siehe Screenshot 4).
- Unter dem Menüpunkt "Files" sollte über eine Mehrfachauswahl in der "tree view" eine action geben, die es ermöglicht mehr als eine Datei als komprimierte Zip-Datei (ab 2 Dateien als Zip) herunterzuladen.
- Unter dem Menüpunkt "File Export" soll das gleiche Schemata wie im Finanzexport angewendet werden. Dazu siehe die Screenshots 1), 2), 3). Hintergrund ist, das getracked werden soll, was schon heruntergeladen wurde und in einem revisionssicheren System hochgeladen wurde (Es soll vermieden werden Dateien doppelt hochzuladen)
- Achtet bitte unabhängig von den Screenshots auf eine saubere Anordnung der Felder in den Views.
- bitte beachten, dass die dateigrößen mitunter sehr groß sein können; das System sollte beim download nicht Gefahr laufen zu "crashen"

A.5 Mockup / Entwurf

Benutzeroberfläche
Attachment Export

DMS
Dateien
Verzeichnisse
Einstellungen

File Export
Attachment Export

Wizard

Name	Models IDs	Start Date	End Date	Attachments IDs
Example Export	Invoice, Country	01.01.1997	01.01.1997	11 Attachments

Wizard Draft

Wizard

Check Export
> Draft > Open > Done

Models
Start Date
End Date

Speichern
Verwerfen
0 Anhänge

Wizard Open

Wizard

Pack as ZIP
> Draft > Open > Done

Export Name

Name

Models

Models

Start Date

0.0.000

End Date

0.0.000

Export Name	Models	Start Date	End Date
X	X	X	X

Speichern
Verwerfen
0 Anhänge

Wizard Done

Wizard

> Draft > Open > Done

Export Name

Name

Models

Models

Start Date

0.0.000

End Date

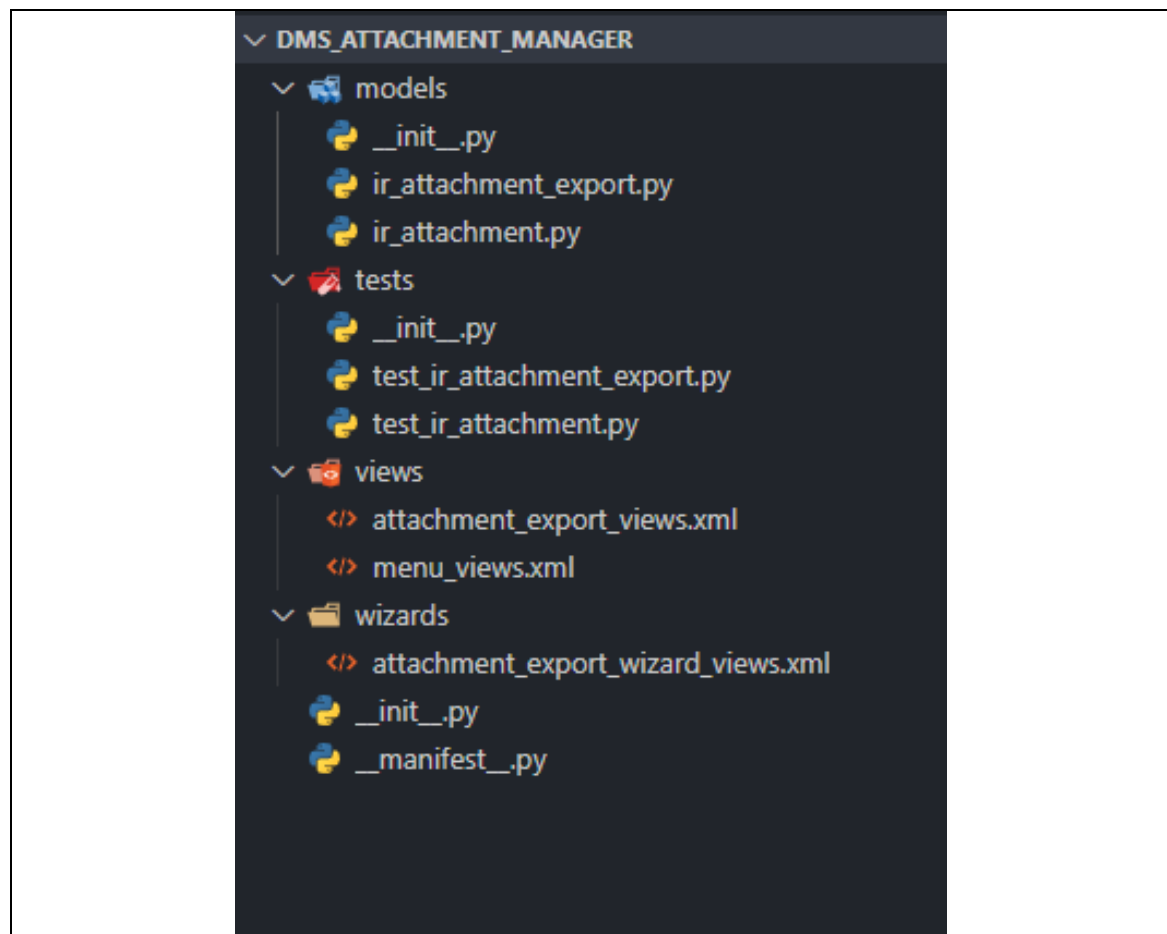
0.0.000

Export Name	Models	Start Date	End Date
X	X	X	X

Speichern
Verwerfen
1 Anhänge

Name.zip

A.6 Verzeichnisstruktur



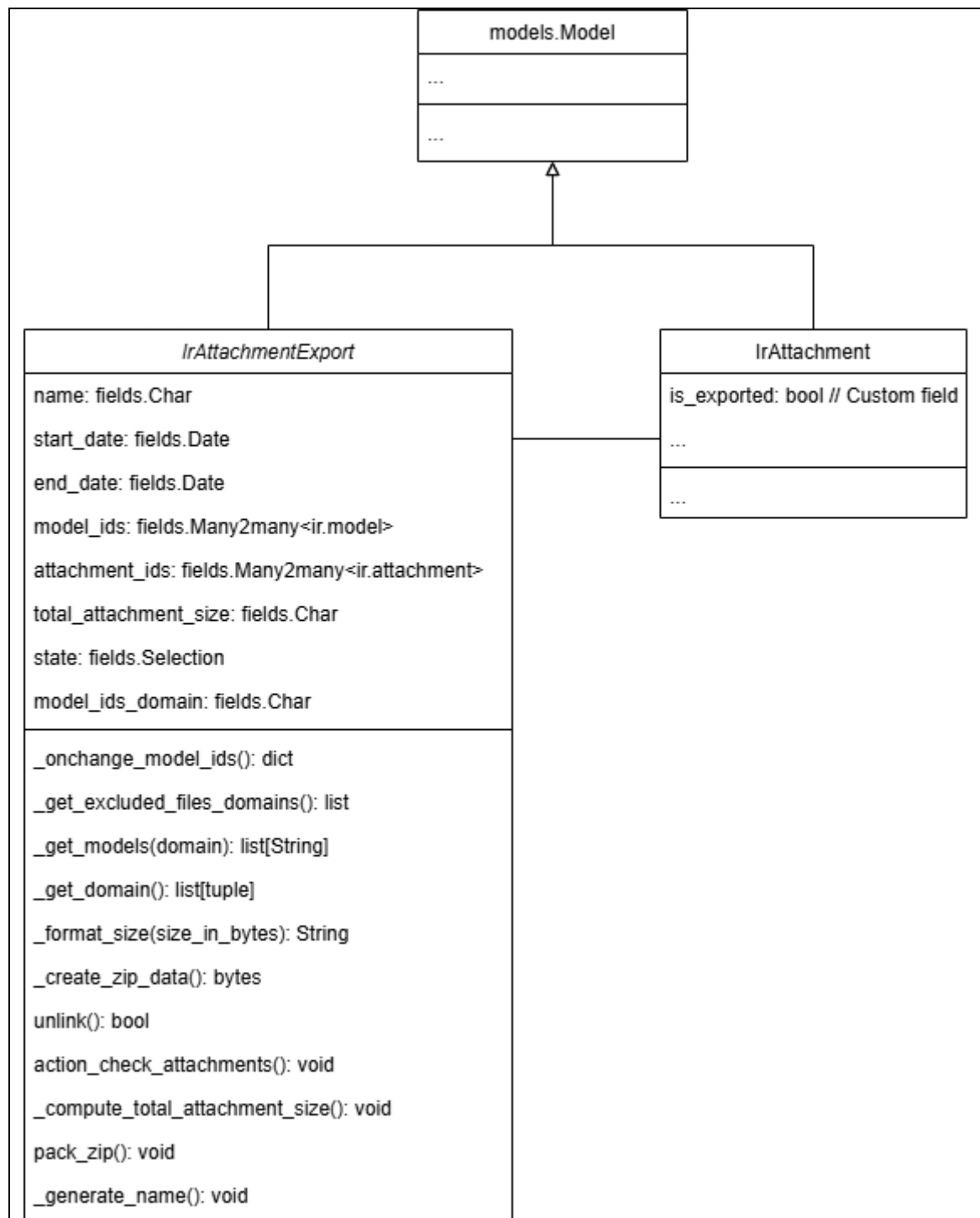
A.7 Manifest

Wichtige Aspekte im Manifest:	
Name des Moduls:	Hier wird der Name des Moduls festgelegt, unter dem es später im Odoo AppStore zu finden ist.
Autor:	Der Entwickler oder die Firma, die das Modul erstellt hat, wird angegeben. Es kann auch eine Verlinkung zur eigenen Website hinzugefügt werden.
Lizenz:	Die Lizenz, unter der das Modul vertrieben wird, wird hier definiert.
Kategorie:	Die Kategorie, in die das Modul im Odoo App-Store einsortiert werden soll, wird angegeben.
Abhängigkeiten (depends):	Hier werden die Module aufgelistet, von denen das Modul abhängt, um vollständig funktionsfähig zu sein. Diese Abhängigkeiten müssen erfüllt sein, damit das Modul korrekt funktioniert.
Daten (data):	Im Manifest wird auch festgelegt, welche Views (Ansichten) geladen werden sollen. Nur die hier angegebenen Views werden im System angezeigt.
Boolean-Werte:	Es können Einstellungen wie „installable“ und „auto_install“ vorgenommen werden. Mit „installable“ wird bestimmt, ob das Modul überhaupt installiert werden kann. Mit „auto_install“ wird festgelegt, dass das Modul automatisch installiert wird, sobald alle Abhängigkeiten erfüllt sind.

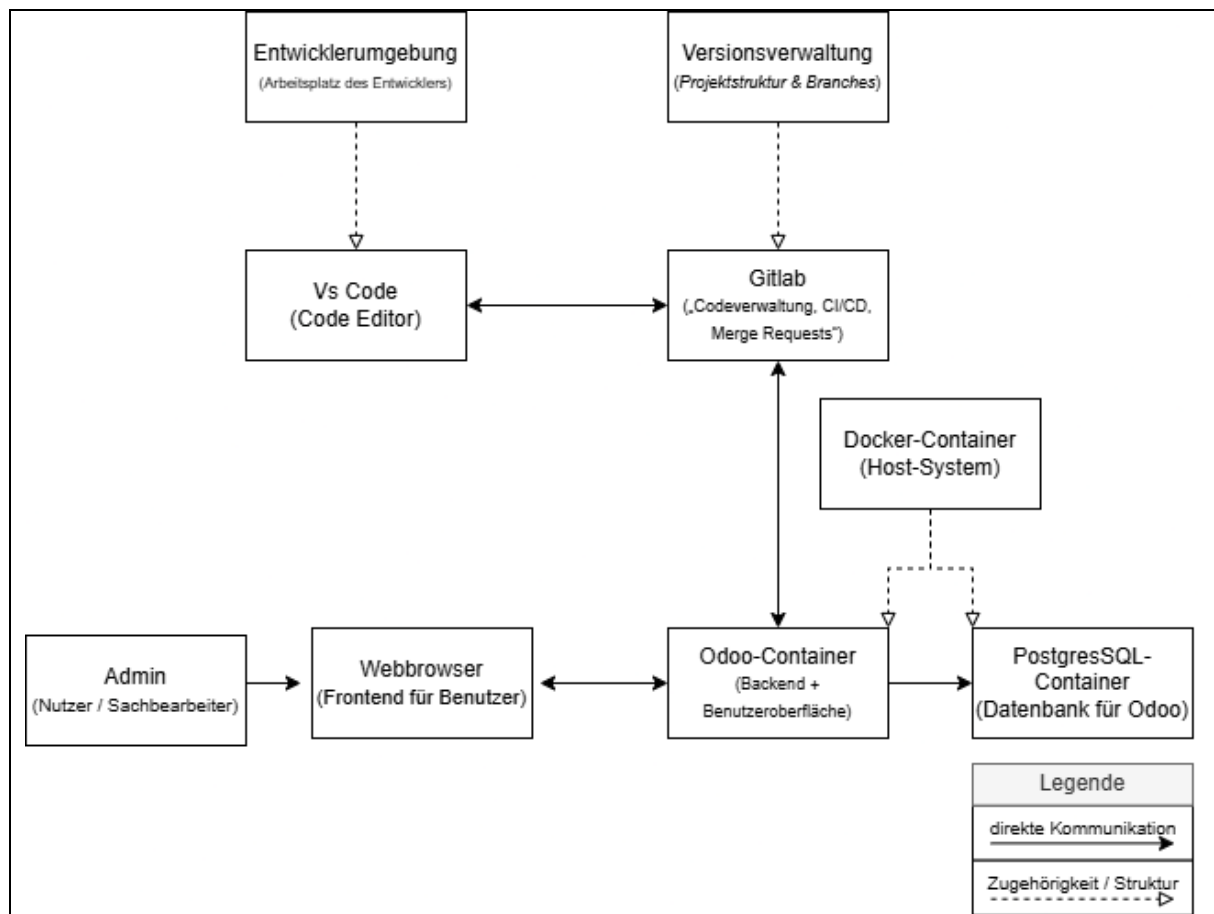
Aufbau der __manifest__.py

```
1  {
2      "name": "dms_attachments_manager",
3      "version": "11.0.1.0.0",
4      "author": "Sirum GmbH",
5      "website": "https://www.sirum.de",
6      "category": "TMS",
7      "license": "SEL-1",
8      "depends": ["base", "dms"],
9      "data": [
10         "views/attachment_export_views.xml",
11         "wizards/attachment_export_wizard_views.xml",
12         "views/menu_views.xml"
13     ],
14     "installable": True,
15     "auto_install": False,
16 }
```


A.8 Klassendiagramm

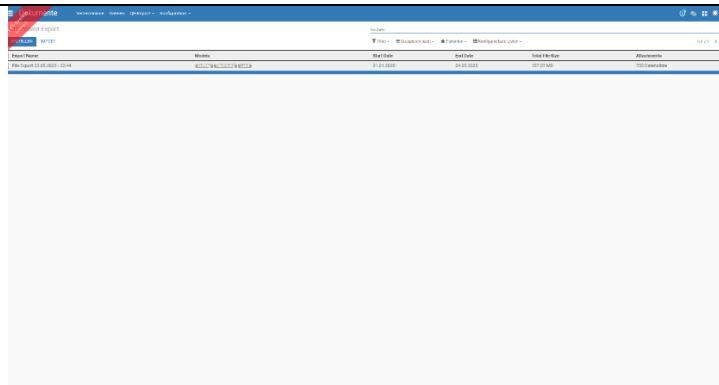


A.9 Verteilungsdiagramm

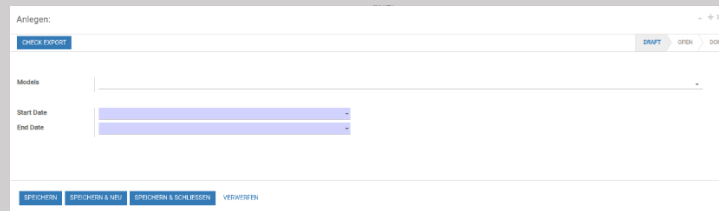


A.10 Screenshots der fertigen Oberfläche

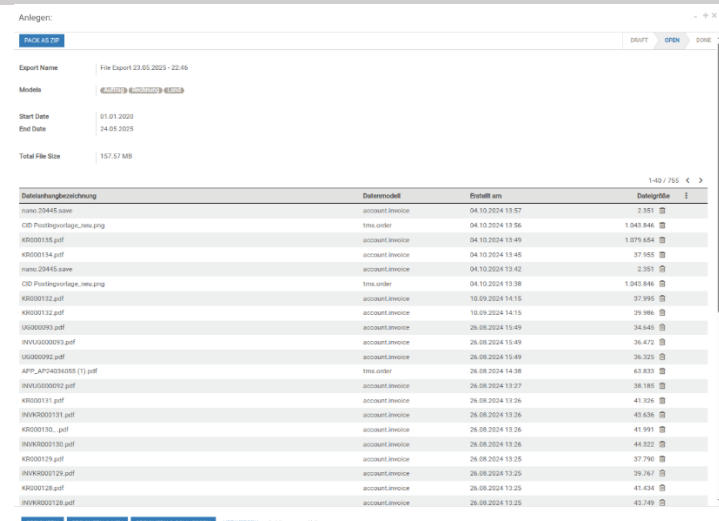
Benutzeroberfläche Attachment Export



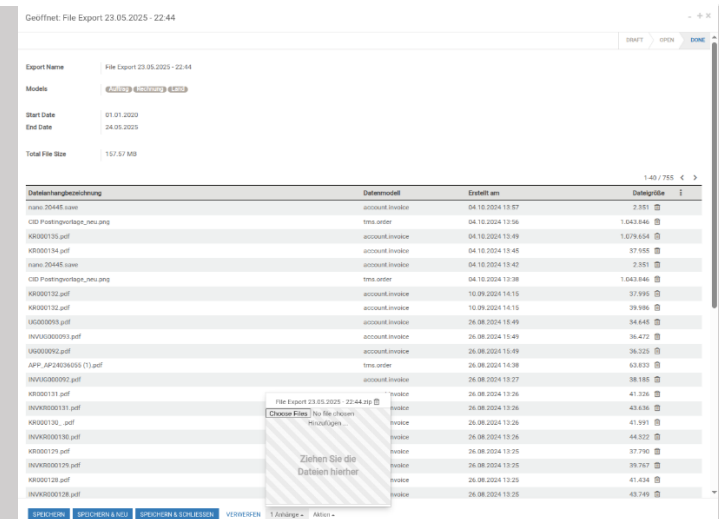
Wizard Draft



Wizard Open



Wizard Done



A.11 Quellcode

A.11.1 Datei: *ir_attachment.py*

```
1 from odoo import fields, models
2
3 class IrAttachment(models.Model):
4     _inherit = "ir.attachment"
5
6     is_exported = fields.Boolean(string="Exported", default=False)
```

A.11.2 Datei: *ir_attachment_export.py*

```
1 import base64
2 import zipfile
3 from datetime import datetime
4 from io import BytesIO
5 from odoo import _, api, fields, models
6 from odoo.exceptions import UserError
7
8 class IrAttachmentExport(models.Model):
9     _name = "ir.attachment.export"
10    _description = "Attachment Export"
11
12    start_date = fields.Date(required=True)
13    end_date = fields.Date(required=True)
14    model_ids = fields.Many2many(comodel_name="ir.model")
15    attachment_ids = fields.Many2many(comodel_name="ir.attachment")
16    total_attachment_size = fields.Char(compute = "_compute_total_attachment_size",
17                                       store=True)
18    state = fields.Selection([("draft", "Draft"), ("open", "Open"), ("done", "Done")],
19                           string="State", default="draft",)
20
21    def action_check_attachments(self):
22        self.attachment_ids = self.env["ir.attachment"].search(self._get_domain())
23
24        if not self.attachment_ids:
25            error_message = _("No attachments found with the given criteria.")
26            raise UserError(error_message)
27
28        for attachment in self.attachment_ids:
29            attachment.is_exported = True
30
31        self._generate_name()
32        self.state = "open"
33
34    def pack_zip(self):
35        zip_data = self._create_zip_data()
36        zip_name = "{}.zip".format(self.name)
37        encoded_zip = base64.b64encode(zip_data)
38        self.env["ir.attachment"].create(
39            {
40                "name": zip_name,
41                "datas": encoded_zip,
42                "datas_fname": zip_name,
43                "res_model": self._name,
44                "res_id": self.id,
45                "mimetype": "application/zip",
46            }
47        )
48        self.state = "done"
```

```
48 def unlink(self):
49     self.attachment_ids.write({"is_exported": False})
50     return super(IrAttachmentExport, self).unlink()
```

A.11.3 Datei: *attachment_export_wizard_views.xml*

```
1 <form string="Attachment Export Wizard">
2     <header>
3         <button name="action_check_attachments" string="Check Export" ... />
4         <button name="pack_zip" string="Pack as Zip" ... />
5         <field name="state" widget="statusbar" />
6     </header>
7     <sheet>
8         <group>
9             <field name="start_date" />
10            <field name="end_date" />
11            <field name="model_ids" widget="many2many_tags" />
12        </group>
13        <group>
14            <field name="attachment_ids" widget="one2many_list" />
15        </group>
16    </sheet>
17</form>
```

A.11.4 Datei: *attachment_export_views.xml*

```
1 <tree>
2     <field name="name" />
3     <field name="model_ids" widget="many2many_tags" />
4     <field name="start_date" />
5     <field name="end_date" />
6     <field name="total_attachment_size" />
7     <field name="attachment_ids" />
8 </tree>
```

A.11.5 Datei: *menu_views.xml*

```
1 <menuitem
2     id="cat_menu_dms_config_file_export"
3     name="File Export"
4     parent="dms.cat_menu_dms_config"
5 />
6 <menuitem
7     id="cat_menu_dms_config_attachment_wizard"
8     name="Attachments Export"
9     parent="cat_menu_dms_config_file_export"
10    action="dms_attachment_manager.ir_attachment_export_action"
11 />
```

A.12 Unit-Tests

A.12.1 Datei: *test_ir_attachment.py*

```
1 from odoo.tests.common import TransactionCase
2 from datetime import date, timedelta
3 import base64
4
5 class TestIrAttachmentExportsIsExported(TransactionCase):
6     def setUp(self):
7         super().setUp()
8         model = self.env["ir.model"].search([("model", "=", "res.country")], limit=1)
9
10        self.attachment = self.env["ir.attachment"].create({
11            "name": "flag_test.pdf",
12            "datas": base64.b64encode(b"Testflag").decode("utf-8"),
13            "datas_fname": "flag_test.pdf",
14            "res_model": model.model,
15            "res_id": 1,
16            "mimetype": "application/txt",
17            "file_size": 9,
18        })
19
20        self.export = self.env["ir.attachment.export"].create({
21            "start_date": date.today() - timedelta(days=1),
22            "end_date": date.today() + timedelta(days=1),
23            "model_ids": [(6, 0, [model.id])],
24        })
25
26    def test_is_exported_true_after_check(self):
27        """Testet, ob "is_exported" nach Exportprüfung auf True gesetzt wird."""
28        self.assertFalse(self.attachment.is_exported)
29
30        self.export.action_check_attachments()
31        self.assertTrue(self.attachment.is_exported)
32
33    def test_is_exported_false_after_unlink(self):
34        """Testet, ob "is_exported" zurückgesetzt wird, wenn Export gelöscht wird."""
35        self.export.action_check_attachments()
36        self.assertTrue(self.attachment.is_exported)
37
38        self.export.unlink()
39        self.attachment.invalidate_cache()
40        self.assertFalse(self.attachment.is_exported)
```

A.12.2 Datei: *test_ir_attachment_export.py*

```
1 from odoo.tests.common import TransactionCase
2 from datetime import date, timedelta
3 import base64
4 import zipfile
5 import io
6
7 class TestIrAttachmentExportZip(TransactionCase):
8     def setUp(self):
9         super().setUp()
10
11         model = self.env["ir.model"].search([("model", "=", "res.country")], limit=1)
12
13         self.attachment = self.env["ir.attachment"].create({
14             "name": "flag_test.pdf",
15             "datas": base64.b64encode(b"Testflag").decode("utf-8"),
16             "datas_fname": "flag_test.pdf",
17             "res_model": model.model,
18             "res_id": 1,
19             "mimetype": "application/txt",
20             "file_size": 9,
21         })
22
23         self.export = self.env["ir.attachment.export"].create({
24             "start_date": date.today() - timedelta(days=1),
25             "end_date": date.today() + timedelta(days=1),
26             "model_ids": [(6, 0, [model.id])],
27         })
28
29     def test_check_export_exists(self):
30         """Testet, ob Export erfolgreich erstellt wurde"""
31         self.export.action_check_attachments()
32         # File
33         self.assertIn(self.attachment, self.export.attachment_ids)
34         # Size
35         self.assertGreaterEqual(len(self.export.attachment_ids), 1)
36         # Bool
37         self.assertTrue(self.attachment.is_exported)
38
39     def test_state_draft(self):
40         """Testet, ob "state = draft" funktioniert"""
41         self.assertEqual(self.export.state, "draft")
42         self.assertFalse(self.export.attachment_ids)
43
44     def test_export_filters_and_assigns_attachments(self):
45         """Testet das Filtern und Zuweisen von Attachments"""
46         self.export.action_check_attachments()
47         self.assertEqual(self.export.state, "open")
48         self.assertIn(self.attachment, self.export.attachment_ids)
49         self.assertTrue(self.attachment.is_exported)
50         self.assertTrue(self.export.total_attachment_size)
51
52     def test_zip_state_done(self):
53         """Testet, ob der Status nach dem ZIP-Packen auf "done" gesetzt wird"""
54         self.export.action_check_attachments()
55         self.export.pack_zip()
56         self.assertEqual(self.export.state, "done")
57
58         # ZIP-Datei wurde erstellt
59         zip_attachment = self.env["ir.attachment"].search([
60             ("res_model", "=", "ir.attachment.export"),
```

```
61         ("res_id", "=", self.export.id),
62         ("mimetype", "=", "application/zip")
63     ], limit=1)
64     self.assertTrue(zip_attachment)
65
66     def test_zip_contains_attachment(self):
67         """Testet, ob das ZIP-Archiv die richtige Datei enthält"""
68         self.export.action_check_attachments()
69         self.export.pack_zip()
70
71         # ZIP-Datei laden
72         zip_attachment = self.env["ir.attachment"].search([
73             ("res_model", "=", "ir.attachment.export"),
74             ("res_id", "=", self.export.id),
75             ("mimetype", "=", "application/zip")
76         ], limit=1)
77
78         # Datei aus ZIP auslesen
79         zip_data = base64.b64decode(zip_attachment.datas)
80         with zipfile.ZipFile(io.BytesIO(zip_data), "r") as zip_file:
81             zip_filenames = zip_file.namelist()
82             self.assertIn(self.attachment.datas_fname, zip_filenames)
83             extracted_content = zip_file.read(self.attachment.datas_fname)
84             self.assertEqual(extracted_content, b"Testinhalt")
```


8.4. Benutzerdokumentation

Modul: *Attachment Export*

Version: *Odoo 11.14*

Funktion: *Export von Dokumenten aus dem DMS als ZIP-Datei*

Ziel der Funktion

Mit dem Modul *Attachment Export* können Benutzer Dokumente aus dem Odoo-DMS gezielt nach Datum und Kategorie filtern, als ZIP-Datei verpacken und anschließend herunterladen. Bereits exportierte Dateien werden automatisch als „verarbeitet“ markiert, um doppelte Exporte zu vermeiden.

Schritt-für-Schritt-Anleitung

1. Navigation zum Modul

- Öffne das Modul „Attachment Export“ über das Hauptmenü unter DMS > Einstellungen > Attachment Export.

2. Neuen Exportvorgang starten

- Klicke oben links auf die Schaltfläche „Erstellen“.

3. Filterkriterien festlegen

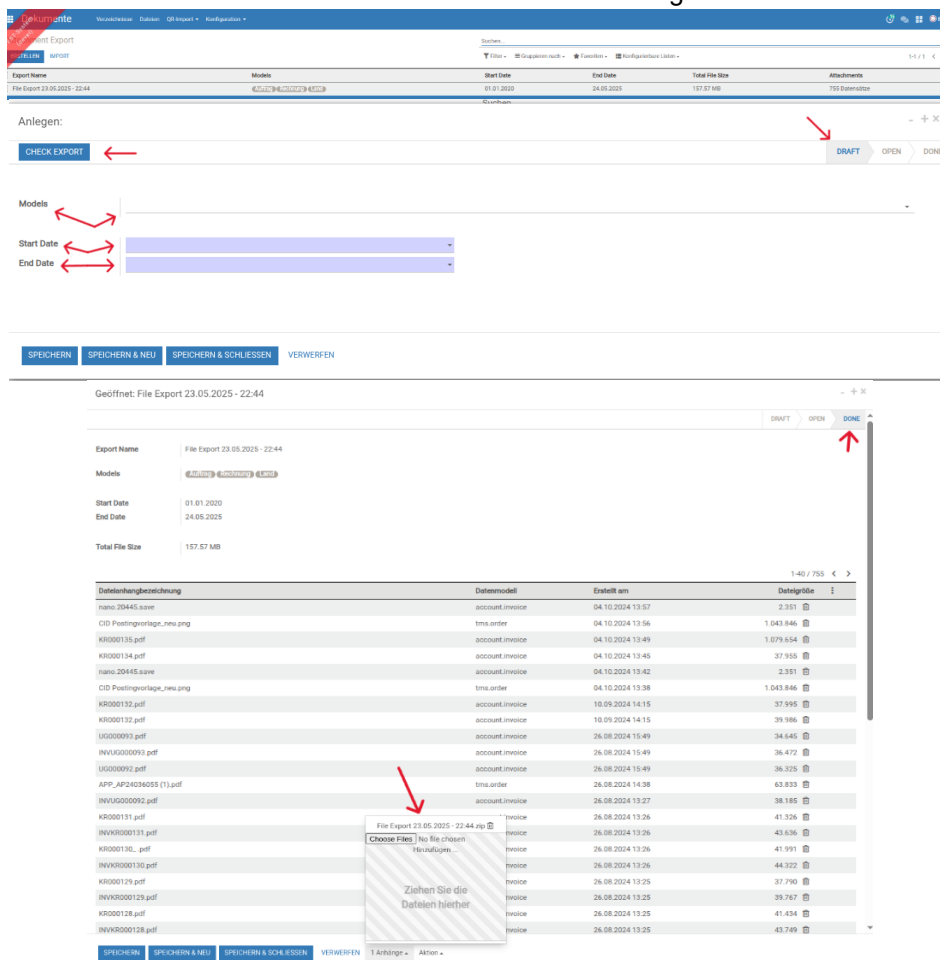
- Es erscheint der Wizard Draft.
- Setze die benötigten Filter (Datum von-bis, Modelle)

4. Dokumente suchen

- Klicke auf „Check Export“.
- Eine Übersicht aller gefundenen Dokumente wird angezeigt.

5. ZIP-Datei erstellen

- Klicke auf „Pack as ZIP“.
- Die ZIP-Datei erscheint nun im unteren Bereich als Anhang.



The screenshot displays the 'Attachment Export' wizard in Odoo. At the top, a table lists existing exports. Below, the 'Draft' form allows setting filters for 'Models', 'Start Date', and 'End Date'. A red arrow points to the 'CHECK EXPORT' button. Below the form, a table shows search results for documents. A red arrow points to the 'Pack as ZIP' button. A dialog box 'Choose Files' is shown with the text 'Ziehen Sie die Dateien hierher'.

8.5. Entwicklerdokumentation

Modulübersicht

Das entwickelte Modul erweitert Odoo 11.14 um eine Exportfunktion für Dokumente (*ir.attachment*). Ziel ist es, Dateianhänge anhand eines festgelegten Zeitraums und bestimmter Modelle zu filtern, diese in einer ZIP-Datei zu bündeln und als Anhang herunterladbar bereitzustellen. Das Modul basiert auf einem TransientModel und wird über einen Wizard gesteuert.

Erweiterte Klasse: *ir.attachment*

```
class IrAttachment(models.Model):  
    _inherit = „ir.attachment“
```

Variablename	Datentyp	Beschreibung
is_exported	Boolean	Kennzeichnet, ob ein Anhang bereits exportiert wurde. Verhindert Doppelexporte.

Hauptklasse: *ir.attachment.export*

```
class AttachmentExport(models.Model):  
    _name = „ir.attachment.export“
```

Felder:

Variablename	Datentyp	Beschreibung
name	Char	Name des Exports (automatisch generiert)
start_date	Date	Startdatum für den Filterzeitraum der Anhänge.
end_date	Date	Enddatum für den Filterzeitraum der Anhänge.
model_ids	Many2many	Auswahl der Odoo-Modelle, aus denen Anhänge exportiert werden sollen.
attachment_ids	Many2many	Gefundene Anhänge, die dem Filter entsprechen und exportiert werden.
total_attachment_size	Char	Gesamtgröße der ausgewählten Anhänge, dargestellt als lesbarer Text.
state	Selection	Status des Wizards: draft , open , done . Steuert den Ablauf.

Wichtige Methoden

```
def action_check_attachments(self):
```

In dieser Methode wird geprüft, ob es Attachments, mit den gesetzten Filtern, existieren, *is_exported* von den Attachments wird dann auf True gesetzt und *state* auf „open“. Falls kein Attachment gefunden wurde, wird ein „raise *UserError*“ returned.

```
def pack_zip(self):
```

In der Methode werden alle Dateianhänge der Attachments in eine Zip komprimiert. Dann wird in „*ir.attachment*“ ein neuer Eintrag erstellt mit den ganzen Daten. Status wird dann auf „done“ gesetzt.

```
def unlink(self):
```

Diese Methode wird ausgeführt, wenn der vorhandende Export gelöscht wird. Somit wird sichergestellt, dass das Feld **is_exported**: bool wieder auf **False** gesetzt wird. Dann kann das Attachment wieder Exportiert werden.