

Abschlussprüfung Sommer 2025

Fachinformatiker fr. Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Entwicklung eines Odoo-Moduls zur

**ZIP-komprimierten Dateiverwaltung** im

**Dokumentenmanagementsystem** (DMS)

Abgabetermin: 26. Mai 2025

**Diese Dokumentation wurde erstellt von:**

Mark Albers

Tienrade 1, 21031 Hamburg

E-Mail: [albers.mark02@gmail.com](mailto: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](mailto:Georg.Notter@Sirum.de)

Inhaltsverzeichnis

[Abkürzungsverzeichnis 3](#_Toc198732825)

[Begriffserläuterung 3](#_Toc198732826)

[1. Einleitung - 1 -](#_Toc198732827)

[1.1. Projektumfeld - 1 -](#_Toc198732828)

[1.2. Projektbeschreibung - 1 -](#_Toc198732829)

[1.3. Projektschnittstellen - 1 -](#_Toc198732830)

[1.4. Projektziel - 1 -](#_Toc198732831)

[2. Projektplanung - 2 -](#_Toc198732832)

[2.1. Zeitplanung - 2 -](#_Toc198732833)

[2.2. Ressourcenplanung - 2 -](#_Toc198732834)

[2.3. Entwicklungsprozess - 2 -](#_Toc198732835)

[2.4. SMART-Ziele - 3 -](#_Toc198732836)

[3. Analysephase - 3 -](#_Toc198732837)

[3.1. Ist-Analyse - 3 -](#_Toc198732838)

[3.2. Wirtschaftlichkeitsanalyse - 4 -](#_Toc198732839)

[3.2.1. „Make or Buy“-Entscheidung - 4 -](#_Toc198732840)

[3.2.2. Projektkosten - 4 -](#_Toc198732841)

[3.2.3. Amortisationsdauer - 5 -](#_Toc198732842)

[3.3. Nicht-monetäre Aspekte - 5 -](#_Toc198732843)

[3.4. Anwendungsablauf - 5 -](#_Toc198732844)

[3.5. Anforderungen - 5 -](#_Toc198732845)

[4. Entwurfsphase - 6 -](#_Toc198732846)

[4.1. Entwurf des Mockups - 6 -](#_Toc198732847)

[4.2. Entwurf der Benutzeroberfläche - 6 -](#_Toc198732848)

[4.3. Technischer Entwurf - 6 -](#_Toc198732849)

[4.4. Datenmodell - 6 -](#_Toc198732850)

[ir.attachment.export - 7 -](#_Toc198732851)

[ir.attachment - 7 -](#_Toc198732852)

[4.5. Maßnahmen zur Qualitätssicherung - 7 -](#_Toc198732853)

[4.6. Deployment - 8 -](#_Toc198732854)

[5. Implementierungsphase - 8 -](#_Toc198732855)

[5.1. Implementierung der Benutzeroberfläche - 8 -](#_Toc198732856)

[5.2. Implementierung der Datenstrukturen - 9 -](#_Toc198732857)

[5.3. Implementierung der Geschäftslogik - 9 -](#_Toc198732858)

[5.4. Testen der Anwendung - 9 -](#_Toc198732859)

[6. Abnahme- und Einführungsphase - 9 -](#_Toc198732860)

[7. Dokumentation - 10 -](#_Toc198732861)

[8. Fazit - 10 -](#_Toc198732862)

[8.1. Soll-/Ist-Vergleich - 10 -](#_Toc198732863)

[8.2. Lessons Learned - 10 -](#_Toc198732864)

[8.3. Ausblick - 10 -](#_Toc198732865)

[A. Anhang - 10 -](#_Toc198732866)

[A.1 Detaillierte Zeitplanung - 11 -](#_Toc198732867)

[A.2 Ressourcenplanung - 12 -](#_Toc198732868)

[A.3 Ablaufdiagramm - 13 -](#_Toc198732869)

# 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 |

# 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.

# Einleitung

## Projektumfeld

Die Sirum GmbH ist ein Systemhaus, der 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.

## Projektbeschreibung

Im Bereich der Dokumentenverwaltung wird bei der Firma ein zentrales Document Management System (DMS) eingesetzt, in den verschiedenen Dateitypen wie Rechnungen, Excel-Tabellen oder Verträge 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 wurde ein Modul entwickelt, das diesen Prozess deutlich vereinfacht. Über eine benutzerfreundliche Oberfläche können Benutzer einen Wizard öffnen, mit dem sie Dokumente anhand eines Datumsbereichs (von–bis) sowie nach Kategorien filtern können. Die ausgewählten Dateien werden anschließend in einem separaten Datensatz gespeichert. Dieser kann später wiederverwendet und als ZIP-Datei heruntergeladen werden.

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

## 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 (z. B. „Check Export“, „Pack as ZIP“) 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.

## 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.

# Projektplanung

## 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**](#_Anhang).

|  |  |
| --- | --- |
| 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

## Ressourcenplanung

Im Anschluss an die Zeitplanung wurden alle für die Umsetzung des Projekts benötigten Ressourcen im [Anhang **A.2: Ressourcenplanung**](#_Anhang) 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. Ziel war es, die Projektkosten so gering wie möglich zu halten.

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**](Make_or_Buy#_).

## 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 [**4.4: Maßnahmen zur Qualitätssicherung**](#_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.0: GitLab-Issue**](#_Anhang) definiert und bei Bedarf angepasst wurden. Diese iterative Vorgehensweise entspricht dem üblichen Kanban-Prozess und ermöglichte eine flexible Umsetzung.

## 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 |
| **S**pecific | 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. |
| **M**easurable | |  | | --- | | Die Umsetzung gilt als erfolgreich, wenn Benutzer mindestens 3 verschiedene Kategorien auswählen, einen Zeitraum eingrenzen, Dokumente in einem Datensatz speichern und diese als ZIP-Datei herunterladen können. | |
| **A**chievable | Das Ziel ist mit den vorhandenen Ressourcen (Odoo-Instanz, Entwicklungsumgebung, interner Reviewprozess) innerhalb der verfügbaren 80 Projektstunden umsetzbar. |
| **R**elevant | Das Projekt löst ein reales Problem im Arbeitsalltag: Es reduziert den manuellen Aufwand beim Dokumentendownload und erhöht die Benutzerfreundlichkeit des DMS. |
| **T**ime-bound | |  | | --- | | Das Modul wird innerhalb des vorgegebenen Projektzeitraums von 70–80 Stunden fertiggestellt und bis zum Projektende vollständig funktionsfähig in das bestehende System integriert. | |

Tabelle 2: SMART-Ziele

# Analysephase

## 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.

## Wirtschaftlichkeitsanalyse

Aufgrund des geschilderten Sachverhalts der in Abschnitt [**1.3 Projektbegründung**](#_Projektbegründung_1) und in Abschnitt [**3.1 Ist-Analyse**](#_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.

### „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. Darüber hinaus möchte die Geschäftsführung die Weiterentwicklung der internen Auszubildenden fördern und ihnen praxisnahe Erfahrungen in der eigenständigen Modulentwicklung ermöglichen.

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 einem flexiblen Datumsbereich zu filtern, sowie die Option, gefilterte Ergebnisse als eigenständigen Datensatz für spätere Downloads zu speichern.

### 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**](#_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.

Es ergeben sich aus den Projektvorgängen und dem Zeitrahmen von 80h für die Entwicklung die in Tabelle 2 dargestellten Projektkosten von 4.350 €.

|  |  |  |  |
| --- | --- | --- | --- |
| **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

### Amortisationsdauer

**Berechnung der Amortisationsdauer**

**SCHLIE FRAGEN WEGEN DEN 4500€ PRO MONAT (NICHT RICHTIG)**

Die Gesamtkosten für die Entwicklung und das Code-Review des Moduls belaufen sich auf 1.150 €, wie in Tabelle 2 dargestellt. Da der Kunde für die Nutzung des Systems monatlich 4.500 € zahlt, ergibt sich eine Amortisationsdauer von ca. 0,26 Monaten. Das bedeutet, dass sich die Investitionskosten für das Modul bereits innerhalb von etwa 8 Tagen vollständig amortisiert haben.

Aufgrund dieser sehr kurzen Amortisationszeit kann das Projekt aus wirtschaftlicher Sicht als äußerst effizient bewertet werden. Die schnelle Kostendeckung sorgt dafür, dass das Unternehmen bereits kurz nach Abschluss der Entwicklung einen nachhaltigen Mehrwert generiert.

## 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.

Nicht zuletzt steigert das Modul auch die Zufriedenheit der Anwender, da es deren Bedürfnisse gezielt adressiert und die tägliche Arbeit erleichtert. Dies kann langfristig zu einer höheren Akzeptanz und effizienteren Nutzung des DMS-Systems führen.

## 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**](#_Ablaufdiagramm).

## 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**](#_Anhang) 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.

# Entwurfsphase

## 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.1: Mockup / Entwurf**](#_Anhang) der Benutzeroberfläche.

## 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.0: Verzeichnisstruktur**](#_A._Anhang) 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.

## 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. „Entwurf“, „Offen“, „Abgeschlossen“)

**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.9: Klassendiagramm.**](#_Anhang)

## 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** sind alle Python-Klassen zur Abbildung der Datenlogik hinterlegt.
* Der Ordner **views** enthält die XML-Dateien zur Definition der Benutzeroberflächen.
* Im Verzeichnis **tests** befinden sich die automatisierten Unittests zur Sicherstellung der Codequalität.
* Weitere Konfigurationsdateien wie **\_\_init\_\_.py** und **\_\_manifest\_\_.py** dienen zur Initialisierung und Beschreibung des Moduls.

Diese saubere Trennung von Logik, Oberfläche und Tests erleichtert nicht nur die Wartung, sondern unterstützt auch eine modulare Erweiterbarkeit des Projekts.

Zur besseren Veranschaulichung befindet sich im [**Anhang A.10: Verzeichnisstruktur**](#_Anhang) ein Screenshot der vollständigen Ordner- und Dateistruktur des Moduls.

## 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 (Master) 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.

## 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.0: Verteilungsdiagramm**](#_A._Anhang)**.**

# Implementierungsphase

## Implementierung der Benutzeroberfläche

Zur Bedienung des Moduls wurde im DMS-Bereich von Odoo ein neuer Menüeintrag eingebunden, über den die Exportansicht (ir.attachment.export) 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.0: Screenshots der fertigen Benutzeroberfläche**](#_Anhang).

## Implementierung der Datenstrukturen

Zur Umsetzung der datenhaltenden Strukturen des Moduls wurden in Anlehnung an das entworfene [**A.0: Klassendiagramm**](#_Anhang) mehrere Modelle in Odoo erstellt. Die zentrale Rolle übernimmt 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 für Odoo gesetzt wurde. Die zugehörigen Felder wie start\_date, end\_date, attachment\_ids oder 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 zur Verknüpfung mit dem Standardmodell ir.attachment, 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 Stati „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. Damit lässt sich verhindern, dass dieselben Dateien mehrfach verarbeitet werden.

Zur technischen Umsetzung der Download-Funktion wurde keine separate REST-Schnittstelle benötigt, da die gesamte Verarbeitung innerhalb des Odoo-Frameworks erfolgt. Der ZIP-Export wird durch interne Methoden realisiert und direkt über die Oberfläche angestoßen.

Die Definition der Modelle und ihrer Beziehungen bildet somit die Grundlage für die automatisierte Verwaltung der Dokumentenexports und gewährleistet die vollständige Integration in das bestehende System.

## 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, Dateigröße und Erstellungsstatus – welche Anhänge (Attachments) im System vorhanden sind und zur Auswahl passen. Die Treffer werden im Feld attachment\_ids hinterlegt und gleichzeitig mit dem Flag is\_exported = 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.

Die Geschäftslogik wurde abschließend durch automatisierte [**A.0: Unittests**](#_Anhang) überprüft. Änderungen am Code konnten somit jederzeit reproduzierbar getestet werden.

## 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.

Screenshots des Unittest-Quellcodes befindet sich zur Veranschaulichung im Anhang [**A.0: Unittests**](#_Anhang).

# 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 auf dem QS-Branch. Die grundlegenden Funktionen, wie Filterung, ZIP-Erstellung und Export, 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.

# 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.0: Benutzerdokumentation**](#_Anhang) (Auszug).

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.0: Entwicklerdokumentation**](#_Anhang) (Auszug).

Zur Veranschaulichung der Datenstruktur wurde, wie bereits erwähnt, ein Klassendiagramm erstellt. Es dient als visuelle Unterstützung für die Systemübersicht. Das Klassendiagramm befindet sich im Anhang [**A.9: Klassendiagramm**](#_Anhang).

# Fazit

## Soll-/Ist-Vergleich

## Lessons Learned

## Ausblick

# Anhang

## A.1 Detaillierte Zeitplanung

|  |  |
| --- | --- |
| **Analyse** | **0 h** |
| Besprechung des Projekts und die einzelnen Phasen |  |
|  |  |
| **Entwurf** | **0 h** |
|  |  |
|  |  |
| **Implementierung inkl. Tests** | **0 h** |
|  |  |
|  |  |
| **Abnahme und Einführung** | **0 h** |
|  |  |
|  |  |
| **Dokumentation** | **0 h** |
|  |  |
|  |  |
| **Gesamt** | 1. **h** |

## A.2 Ressourcenplanung

**Hardware**

* Büroarbeitsplatz mit Firmenlaptop

**Software**

* Linux Mint – Betriebssystem
* Visual Studio Code – Entwicklerumgebung
* PostgresSQL – 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

