Institut für Strukturmechanik und Leichtbau

RWTH AACHEN UNIVERSITY

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

KONZEPTION EINER DATENMANAGEMENTSOFTWARE ZUR REALISIERUNG VON BUCHHALTUNGSWORKFLOWS

Diese Seminararbeit wurde vorgelegt am

Fachbereich 9
Medizintechnik und Technomathematik
Fachhochschule Aachen, Campus Jülich

von

Delvenne Tobias
Matrikelnummer: 3654037

und wurde betreut von:

1. Prüfer

Prof. Dr. rer. nat. Alexander Voß

Fachbereich 9

FH Aachen

2. Prüferin

Helena Heuser, M.Sc.

Institut für Strukturmechanik und Leichtbau

RWTH Aachen University

Aachen, Dezember 2025

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema

Konzeption einer Datenmanagementsoftware zur Realisierung von Buchhaltungsworkflows

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Im Rahmen der Erstellung dieser Arbeit wurde das KI-System "KI.connect.nrw" unterstützend zur sprachlichen Überarbeitung sowie zur fachlichen Reflexion und Präzisierung eigenständig entwickelter Argumente genutzt. Eine Übernahme von KI-generierten Texten oder inhaltlichen Lösungsvorschlägen erfolgte nicht. Sämtliche fachlichen Aussagen, Bewertungen und Schlussfolgerungen wurden eigenständig erarbeitet und verantwortet. Die Nutzung erfolgte im Einklang mit der Zweckbestimmung des Systems sowie unter Beachtung datenschutz- und urheberrechtlicher Vorgaben.

Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Tobias Delvenne

Aachen, den 15. Dezember 2025

Unterschrift des Studierenden

Kurzfassung

Buchhaltung ist ein essenzieller Bestandteil jeder Institution. Ohne eine verlässliche und aussagekräftige Übersicht über den Finanzhaushalt sind sowohl die Planung von Investitionen als auch die laufende Verwaltung von Finanzmitteln kaum möglich. Für diese Aufgaben werden häufig etablierte Systeme eingesetzt, die zwar über einen umfangreichen Funktionsumfang und breiten Support verfügen, jedoch oftmals Defizite in der Bedienbarkeit aufweisen. An dieser Stelle setzt die vorliegende Arbeit an.

Durch die Automatisierung verschiedener Arbeitsabläufe und die Bereitstellung konfigurierbarer Übersichten soll der Verwaltungsaufwand innerhalb der Buchhaltungsabteilung reduziert werden. Zu diesem Zweck werden zunächst die aktuell verwendeten Technologien analysiert und bestehende Schwachstellen herausgearbeitet. Darauf aufbauend wird ein eigenes Softwaresystem entwickelt, beginnend mit einer strukturierten Anforderungsanalyse. Da die Zielgruppe nicht aus IT-Fachkräften besteht, sondern aus Mitarbeitenden der Buchhaltung, liegt ein besonderer Fokus auf intuitiver Bedienbarkeit. Zu diesem Zweck werden verschiedene Designentwürfe (Mockups) entworfen und im Rahmen eines Minimum Viable Product (MVP) umgesetzt.

Im weiteren Verlauf werden die dabei eingesetzten Technologien sowie die technischen Herausforderungen der Entwicklung erläutert. Abschließend werden mögliche Erweiterungen des Systems diskutiert und ein Fazit gezogen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACID	Atomicity, Consisteny, Isolation, Durability.
API	Application Programming Interface.
CRUD	Create, Read, Update, Delete.
CSV	Comma-Separated Values.
FI	Finanzbuchhaltung.
GUI	Graphical User Interface.
HCM	Human Capital Management.
HTTP	Hypertext Transfer Protocol.
MaCoCo	Management Cockpit für Controlling.
MVP	Minimum Viable Product.
OAS	OpenAPI Specification.
OData	Open Data Protocol.
ORM	Object-Relational Mapping.
REST	Representational State Transfer.
RWTH	Rheinisch-Westfälische Technische Hochschule Aachen.
URI	Uniform Resource Identifier.
US	User Stories.

1 Einleitung

Im Folgenden wird die Problemstellung, welche dieser Arbeit zugrunde liegt erklärt. Außerdem wird das Ziel der Arbeit erläutert und bereits ein kleiner Überblick über die zu erwartenden Funktionen gegeben. Des Weiteren wird der Aufbau und die Vorgehensweise in dieser Arbeit beschrieben.

1.1 Motivation

Die von der Rheinisch-Westfälische Technische Hochschule Aachen (RWTH) eingesetzte Buchhaltungssoftware SAP stellt über das Webinterface FIORI verschiedene Subsysteme bereit, unter anderem zur Erfassung von Personalbelastungen sowie zur Verwaltung eingehender und ausgehender Zahlungen. Durch die Aufteilung in mehrere voneinander weitgehend isolierte Subsysteme und der fehlenden Kommunikation zwischen diesen ist es der Buchhaltungsabteilung jedoch nicht ohne erheblichen manuellen Aufwand möglich, eine konsolidierte und strukturierte Übersicht über die Finanzsituation einzelner Projekte zu erhalten. Dass die verwendete Software zusätzlich durch nicht funktionierende Filteroptionen und unübersichtliche Menüs zu weiterem Mehraufwand führt, ist ein wesentlicher Grund für die Entwicklung eines maßgeschneiderten Systems, das diese Datenverarbeitungsfunktionen einfacher und zugänglicher bereitstellen soll.

Hinzu kommt, dass die in der aktuell eingesetzten Software verfügbaren Datensätze häufig nur zeitversetzt aktualisiert werden. Dadurch kann es vorkommen, dass die Daten mehrere Monate hinter den tatsächlich bereits erfolgten Ausgaben oder Einnahmen liegen, was eine aktuelle Budgetübersicht unmöglich macht. Eine belastbare Finanzplanung ist unter diesen Bedingungen kaum möglich. Viele projektbezogene Ausgaben insbesondere Personalkosten lassen sich jedoch über Monate oder sogar Jahre hinweg zuverlässig prognostizieren und müssen mit den bewilligten Finanzierungsmitteln abgeglichen werden. Die vorhandene Software unterstützt diesen Bedarf nicht, sodass eine ergänzende Eigenentwicklung notwendig erscheint.

Ziel dieser Arbeit ist die Konzeption und prototypische Umsetzung einer Software, die projektbezogene Finanzübersichten und Verwendungsnachweise aus den vorhandenen SAP-Exporten erzeugt und ergänzt. Neben der Verarbeitung der Comma-Separated Values (CSV)-Daten sollen auch manuell hinzugefügte Informationen, insbesondere künftige oder prognostizierte Ausgaben, berücksichtigt werden. Die entwickelte Lösung soll modular, erweiterbar und langfristig wartbar sein, um zukünftige organisatorische oder technische Änderungen ohne grundlegende Anpassungen zu ermöglichen. Des Weiteren soll diese Arbeit eine Entscheidungsgrundlage schaffen, anhand derer bewertet werden kann, ob eine Weiterentwicklung organisatorisch sinnvoll ist.

Die entwickelte Software soll SAP nicht ersetzen, sondern gezielt ergänzen, indem sie bestehende Schwächen adressiert und Arbeitsprozesse erleichtert. Durch die automatisierte Aufbereitung und Zusammenführung der Daten wird der zeitliche Aufwand für manuelle Nachbearbeitung reduziert und die Fehleranfälligkeit verringert, was zu einer zuverlässigeren und effizienteren Finanzübersicht führt.

1.2 Vorgehensweise und Aufbau der Arbeit

Die vorliegende Arbeit ist so strukturiert, dass sie zunächst die notwendigen fachlichen und technischen Grundlagen vermittelt und darauf aufbauend die konzeptionelle Entwicklung des MVP nachvollziehbar darstellt.

Zu Beginn werden in Kapitel 2 die theoretischen Grundlagen erläutert, die für das Verständnis der späteren Architekturentscheidungen erforderlich sind. Dazu zählen insbesondere grundlegende Konzepte der Softwarearchitektur, der Representational State Transfer (REST)-Kommunikation sowie der relationalen Datenmodellierung.

Darauf aufbauend erfolgt in Kapitel 3 eine systematische Anforderungsanalyse, in der sowohl funktionale als auch nicht-funktionale Anforderungen erhoben und durch User Stories ergänzt werden.

Kapitel 4 beschreibt anschließend die Konzeption des MVP, einschließlich des Architekturentwurfs, der Datenmodellierung, der Benutzeroberfläche sowie der eingesetzten Technologien.

Das darauf folgende Kapitel 5 widmet sich der prototypischen Umsetzung ausgewählter Kernfunktionen und zeigt exemplarisch, wie die konzeptionellen Elemente in Software überführt wurden.

Kapitel 6 diskutiert die erzielten Ergebnisse im Hinblick auf die definierten Anforderungen und bewertet die Eignung des MVP für den Einsatz im buchhalterischen Arbeitskontext.

Abschließend fasst Kapitel 7 die wesentlichen Erkenntnisse kurz zusammen.

2 Stand der Technik

In nachfolgendem Kapitel wird der aktuelle Stand der Technik an der RWTH erklärt. Dazu werden das eingesetzte Buchhaltungssystem von SAP und dessen Module sowie die eigens entwickelte Software Management Cockpit für Controlling (MaCoCo) erläutert. Außerdem werden die theoretischen Grundlagen beschrieben.

2.1 Aktuelle Buchhaltungssysteme an der RWTH

Die derzeit eingesetzte Technik verfügt zwar theoretisch über die meisten benötigten Funktionalitäten, ist in der praktischen Nutzung jedoch häufig nur schwer bedienbar oder arbeitet teilweise nicht zuverlässig. So ist es beispielsweise in der SAP-Finanzübersicht nicht möglich, Buchungen nach Jahren zu gruppieren, da entsprechende Abfragen über das Webinterface nach mehreren Minuten ohne Ergebnis abbrechen. Ebenso lassen sich die Daten aus verschiedenen Subsystemen nicht über gemeinsame Schlüsselattribute verknüpfen, sodass etwa eine Zuordnung von Mitarbeitenden zu ihren jeweiligen Entgeltstufen nicht ohne zusätzliche manuelle Schritte möglich ist.

Die Eigenentwicklung MaCoCo der RWTH stellt zwar erweiterte Möglichkeiten zur Datenanalyse bereit, ist jedoch kostenpflichtig und deckt nicht alle für dieses Projekt relevanten Funktionen ab. Durch die Bestrebung der RWTH, große Teile der finanzbuchhalterischen Prozesse über MaCoCo abzubilden, hat sich zudem im Laufe der Zeit für unsere Buchhaltungsabteilung eine zunehmende Unübersichtlichkeit ergeben.

Insbesondere für den hier betrachteten Anwendungsfall der Erzeugung von Verwendungsnachweisen bietet MaCoCo weder eine ausreichend klare Aufbereitung der projektbezogenen Finanzdaten noch eine einfache Möglichkeit zur manuellen Eingabe und Verwaltung von Plandaten. Aufgrund dieser funktionalen Einschränkungen stellt MaCoCo keine geeignete Alternative zur entwickelten Lösung dar.

Im Rahmen des Projekts SARA wurde im Jahr 2014 RWTH weit das Softwaresystem SAP für betriebswirtschaftliche Abläufe eingeführt. Dabei wurden unter anderem die Bereiche Finanzbuchhaltung, Anlagenbuchhaltung, Drittmittelverwaltung, Haushalt und Budgetierung auf SAP umgestellt. Im Jahr 2016 folgte die Integration des Personalwesens, einschließlich Reisekostenabrechnung, sowie der Logistik mit den Themengebieten Beschaffung und Facility Management.[1]

SAP ist modular aufgebaut. An der RWTH sind insbesondere folgende Module im Einsatz:

- FI: Finanzbuchhaltung
- FI-AA: Anlagenbuchhaltung
- CO: Controlling
- PS: Projektsystem

- PSM: Public Sector Management
- BW: Business Warehouse
- HCM: Personalwesen
- OM: Organisationsmanagement
- MM: Materialwirtschaft
- FI-TV: Reisekostenmanagement

[1]

Das Projekt MaCoCo wurde ebenfalls im Jahr 2016 initiiert, nachdem sich verschiedene Einschränkungen der neu eingeführten SAP-Landschaft gezeigt hatten. In einer Kooperation zwischen dem Lehrstuhl für Controlling und dem Lehrstuhl für Software Engineering wurde die Software mit dem Ziel entwickelt, das Controlling an Lehr- und Forschungseinrichtungen der RWTH zu professionalisieren.[2]

2.2 Aktuelle Workflows

Derzeit werden die benötigten Daten manuell aus verschiedenen SAP-Subsystemen zusammengesucht, einzeln exportiert und anschließend in mehrere Excel-Tabellen übertragen. Die Ausgaben werden manuell mit den bewilligten Drittmitteln verrechnet um einen Überblick über die noch verfügbaren Gelder zu erhalten. Dieser Prozess ist zeitaufwendig und mit einem erheblichen verwaltungstechnischen Aufwand verbunden. Zudem ist die manuelle Zusammenführung der Daten anfällig für Übertragungs- und Eingabefehler.

Durch eine digitale Abbildung und Automatisierung dieses Workflows können menschliche Fehler systematisch vermieden und der Aufwand für wiederkehrende Tätigkeiten deutlich reduziert werden. Gleichzeitig wird eine konsistente, reproduzierbare und nachvollziehbare Datenbasis geschaffen, die als Grundlage für weitere Analysen und Auswertungen dient.

2.3 Theoretische Grundlagen

In diesem Abschnitt werden die grundlegenden technischen Konzepte erläutert, die für das Verständnis der später beschriebenen Architekturentscheidungen und Implementierungsdetails erforderlich sind.

2.3.1 Client-Server-Architektur

Die Client-Server-Architektur beschreibt ein verteiltes System, in dem mindestens ein zentraler Server spezifische Dienste bereitstellt, während die Clients diese Dienste anfragen. Ein Client ist somit ein Prozess, der eine bestimmte Funktionalität konsumiert, während der Server diese bereitstellt. Eine strikte Trennung ist in der Praxis jedoch nicht immer gegeben, da ein Server wiederum als Client gegenüber anderen Servern auftreten kann, etwa wenn ein Anwendung Anfragen an einen separaten Datenbankserver stellt.[3]

Für die Kommunikation zwischen Client und Server wird in webbasierten Architekturen üblicherweise das Hypertext Transfer Protocol (HTTP) verwendet. HTTP ist ein zustandsloses, textbasiertes Übertragungsprotokoll, bei dem der Client eine *Request* sendet und der Server darauf mit einer *Response* antwortet.[4] Zu den zentralen HTTP-Methoden, die auch in dieser Arbeit eingesetzt werden, gehören:

- **GET**: Abrufen von Ressourcen oder Daten,
- **POST**: Erstellen neuer Ressourcen bzw. übermitteln von Daten,
- **PUT**: Ersetzen einer bestehenden Ressource,
- **PATCH**: Teilweises Aktualisieren einer Ressource,
- **DELETE**: Löschen einer Ressource.

2.3.2 REST-Architekturstil

Der REST-Architekturstil wurde von Roy Thomas Fielding im Rahmen seiner Dissertation definiert [5]. REST beschreibt keine spezifische Technologie, sondern architektonische Prinzipien für verteilte Systeme, insbesondere für webbasierten Datenaustausch. Die grundlegende Idee besteht darin, Ressourcen über eindeutige Uniform Resource Identifier (URI)s zu adressieren und sämtliche Interaktionen über ein einheitliches Interface, typischerweise HTTP, abzuwickeln.

REST basiert auf mehreren konzeptionellen Einschränkungen, die die Skalierbarkeit und Austauschbarkeit von Systemkomponenten fördern. Dazu gehören insbesondere:

- **Client-Server-Entkopplung**: Die Benutzeroberfläche ist strikt von der Datenverarbeitung getrennt.
- **Zustandslosigkeit**: Jeder Request enthält alle Informationen, die der Server benötigt; Sessions auf Serverseite sind nicht notwendig.
- **Einheitliche Schnittstellen (Uniform Interface)**: Ressourcen werden über standardisierte HTTP-Methoden manipuliert.
- **Cachefähigkeit**: Ressourcen sollen Client- oder serverseitig gecacht werden können.
- **Mehrschichtige Systemarchitektur**: Aufrufe und Antworten können beliebig viele Schichten durchlaufen. Client und Serveranwendung müssen nicht direkt miteinander verbunden sein.

[6]

Ein praktischer Bestandteil dieses Uniform-Interface-Prinzips ist die semantische Zuordnung der grundlegenden Create, Read, Update, Delete (CRUD)-Operationen zu den entsprechenden HTTP-Methoden. Diese Zuordnung stellt sicher, dass REST-basierte Web-Application Programming Interface (API)s konsistent und vorhersehbar sind.[7] Die typischen Abbildungen lauten:

- **Create** → **POST**: Erstellen neuer Ressourcen unter einer vom Server verwalteten URI.
- **Read** → **GET**: Abrufen bestehender Ressourcen ohne Änderung ihres Zustands.
- **Update** → **PUT/PATCH**: PUT ersetzt eine Ressource vollständig, während PATCH Teiländerungen vornimmt.

- **Delete → DELETE**: Entfernen einer Ressource.

Diese semantische Kopplung zwischen CRUD-Operationen und HTTP-Methoden bildet die Grundlage für die in dieser Arbeit implementierte REST-API, da sämtliche Operationen auf Projekten, Importvorgängen und Finanzübersichten anhand dieser Prinzipien umgesetzt werden.

Durch diese Eigenschaften ermöglicht REST eine lose Kopplung zwischen Client und Server sowie eine hohe Flexibilität bei der Weiterentwicklung einzelner Systemteile.[5] Dies ist insbesondere für das in dieser Arbeit vorgestellte System relevant, da die Desktop-Graphical User Interface (GUI) unabhängig vom Backend betrieben werden kann und sich alternative Frontends leicht integrieren lassen.

2.3.3 API-Dokumentation mit OpenAPI und SwaggerUI

Die OpenAPI Specification (OAS) ist ein standardisiertes Format zur Beschreibung von REST-basierten Web-APIs. Es ermöglicht die formale Definition aller Endpunkte, Datenmodelle, Parameter und Antwortstrukturen in einer maschinen- und menschenlesbaren Form. Eine korrekt definierte OAS bildet die Grundlage für automatisierte Dokumentation und Testwerkzeuge.[8]

SwaggerUI ist ein Werkzeugset, das auf der OAS aufbaut und eine interaktive Weboberfläche zur Verfügung stellt, in der API-Endpunkte ausführlich dokumentiert und direkt getestet werden können. Frameworks wie FastAPI erzeugen diese Dokumentation automatisch, indem sie Typhinweise und Datenmodelle aus dem Quellcode ableiten.

Die Vorteile einer formalen API-Spezifikation umfassen unter anderem:

- **Automatisierte und immer aktuelle Dokumentation**,
- **Interaktive Testmöglichkeiten ohne separate Tools**,
- **Verbesserte Wartbarkeit und Konsistenz der API**.

Für dieses Projekt ist die automatische SwaggerUI-Dokumentation besonders relevant, da sie die interne Entwicklung unterstützt und die Erweiterbarkeit des Systems erleichtert.

2.3.4 CSV als Datenformat

Das CSV-Format ist ein textbasiertes Format, das tabellarische Daten in Zeilen- und Spaltenstruktur speichert. Es gehört zu den am weitesten verbreiteten Formaten für Datenexporte aus Informationssystemen, da es einfach, universell lesbar und ohne proprietäre Software nutzbar ist.[9]

CSV-Dateien besitzen allerdings keine fest definierte Spezifikation[9]. Typische Herausforderungen, die auch in dieser Arbeit auftreten, umfassen:

- **Uneinheitliche Trennzeichen** (z. B. “;” vs. “,”),
- **fehlende oder mehrzeilige Header**,
- **uneinheitliche Datums- und Zahlenformate**,
- **Darstellung numerischer Werte als Text**,

- **keine eingebaute Unterstützung für Datentypen**.

Das MVP implementiert daher Mechanismen zur automatischen Erkennung von Datentypen, zur Normalisierung von Datums- und Betragsfeldern sowie zur Bereinigung problematischer Headerstrukturen.

2.3.5 SQLite: Grundlagen eingebetteter Datenbanken

SQLite ist ein relationales Datenbankmanagementsystem, das sich vollständig als Bibliothek in eine Anwendung einbetten lässt. Es benötigt keinen separaten Serverprozess und speichert alle Daten in einer einzigen lokalen Datei [10]. Durch die Unterstützung des vollständigen SQL-Standards (inkl. Atomicity, Consisteny, Isolation, Durability (ACID)-Konformität) eignet es sich besonders für Desktopanwendungen, lokale Datenanalysen und prototypische Entwicklungen.

Zu den wesentlichen Eigenschaften von SQLite zählen:

- **Serverlosigkeit**: keine Installation oder Wartung eines separaten Datenbankservers,
- **ACID-garantierte Transaktionen**: sichere Schreib- und Leseoperationen,
- **hohe Performance bei lokalen Datenzugriffen**,
- **breite Unterstützung in Programmiersprachen und Frameworks**.

Für das MVP dient SQLite als persistente Grundlage für Projekte, Importtabellen, Plandaten und Zwischenauswertungen. Die Wahl fällt insbesondere aufgrund der einfachen Integration und der guten Eignung für Einzelplatzsysteme.

3 Anforderungsanalyse

Die Entwicklung eines Systems zur automatisierten Aufbereitung und Analyse projektbezogener Finanzdaten setzt eine präzise Definition der fachlichen, organisatorischen und technischen Anforderungen voraus. Dieses Kapitel beschreibt die Rahmenbedingungen, die Zielgruppe sowie die funktionalen und nicht-funktionalen Anforderungen und verweist dabei auf die in Abschnitt A.1 formulierten User Stories (US).

3.1 Rahmenbedingungen und Einschränkungen

Die Konzeption und Entwicklung des Systems unterliegt mehreren organisatorischen, technischen und regulatorischen Rahmenbedingungen. Wie bereits erwähnt, ist der Zugriff auf die Open Data Protocol (OData)-Services von SAP aufgrund des notwendigen Genehmigungsprozesses durch mehrere Abteilungen der RWTH nicht im zeitlichen Rahmen dieser Seminararbeit realisierbar. Ebenso ist es aus rechtlichen Gründen nicht gestattet, Daten aus dem Webinterface mittels Reverse Engineering auszulesen.[11] Daher müssen sämtliche Daten aus den bereitgestellten CSV-Exporten sowie aus manuellen Eingaben gewonnen werden (vgl. US-09).

3.1.1 Verfügbare Datenquellen

Für diese Seminararbeit stehen zwei wesentliche Datenquellen zur Verfügung: die CSV-Exporte aus der Personalbelastungsanzeige, im Folgenden als *Human Capital Management (HCM)* bezeichnet, und die CSV-Exporte der Finanzstellenbuchungen, im Folgenden als *Finanzbuchhaltung (FI)* bezeichnet.

Über die HCM-Daten lassen sich Mitarbeitende ihren jeweiligen Entgeltstufen sowie einzelnen Projekten zuordnen. Darüber hinaus enthalten die CSV-Exporte Informationen zu gezahlten Gehältern und Zuwendungen. Letztere umfassen beispielsweise jährlich gewährte Sonderzahlungen wie das Weihnachtsgeld in Form eines 13. Monatsgehalts.

Die FI-Daten umfassen unter anderem eingehende Drittmittelzahlungen, bezahlte Rechnungen und vergebene Aufträge. Somit bilden sie die zentrale Grundlage für die Darstellung projektbezogener Ein- und Ausgaben.

Im Hinblick auf die Benutzerfreundlichkeit ist anzumerken, dass SAP OData-Schnittstellen anbietet, die eine regelmäßige und automatisierte Aktualisierung dieser Daten ermöglichen würden.[12] Da die Freischaltung dieser Schnittstellen jedoch durch mehrere Abteilungen der RWTH genehmigt werden muss, ist die Umsetzung dieser Funktionalität im Rahmen dieser Seminararbeit nicht realisierbar.

Charakteristika der verfügbaren Daten

Die aus den SAP-Systemen exportierten Daten weisen mehrere Besonderheiten auf, die bei der Verarbeitung und Analyse berücksichtigt werden müssen. Zum einen werden die Exporte ausschließlich als CSV-Dateien bereitgestellt, ohne dass eine garantierte einheitliche Struktur oder ein standardisiertes Schema vorliegt. Zum anderen enthalten die Daten sowohl historisch gewachsene als auch domänenspezifische Bezeichnungen, die eine direkte maschinelle Verarbeitung ohne zusätzliche Übersetzungs- und Bereinigungsprozesse erschweren.

Ein weiterer Aspekt ist die teilweise zeitverzögerte Aktualisierung der Datensätze, insbesondere in den Bereichen Personalbelastung und Drittmittelfinanzierung. Diese Verzögerungen können mehrere Wochen bis Monate betragen, wodurch die Exporte nicht immer den aktuellen Stand der Finanzsituation abbilden. Zusätzlich werden die Werte nicht relational verknüpft exportiert, sodass keine konsistenten Primär- oder Fremdschlüsselbeziehungen zwischen den verschiedenen Systemen (FI, HCM) vorhanden sind. Dies führt dazu, dass Zuordnungen zwischen Mitarbeitenden, Projekten und Kostenarten häufig nur über eine manuelle Betrachtung möglich sind.

CSV-Struktur

Die Struktur der exportierten CSV-Dateien variiert stark zwischen den einzelnen SAP-Subsystemen und sogar zwischen verschiedenen Exportvorgängen desselben Systems. Diese Inhomogenität erschwert eine automatisierte Weiterverarbeitung erheblich. Die wichtigsten strukturellen Herausforderungen umfassen:

- **Uneinheitliche Headerzeilen:** Manche Exporte enthalten keine Spaltenbezeichnungen, andere hingegen verwenden mehrzeilige Header oder mehrfach vorkommende Bezeichner. Dadurch ist eine eindeutige Zuordnung der Spalten nicht ohne manuelle Nachbearbeitung möglich.
- **Uneinheitliche Datumsformate:** Datumsangaben liegen je nach Export entweder im Format `MM/YYYY` oder `TT.MM.YYYY` vor. Ein konsistentes Parsing erfordert daher eine Normalisierung der Formate.
- **Numerische Werte als Text:** Beträge werden standardmäßig als Text exportiert und enthalten Tausendertrennzeichen (z. B. `1.234,56`). Dies verhindert eine numerische Sortierung und erfordert eine explizite Umwandlung in numerische Datentypen.
- **Domänenspezifische Kodierungen:** Viele Felder enthalten SAP-interne Schlüssel, Kürzel oder Domänenbeschreibungen, die ohne Übersetzungstabellen nicht interpretierbar sind. Ein Beispiel dafür ist der Belegart-Schlüssel **D3** für Drittmittel.

Das folgende Beispiel zeigt exemplarisch eine CSV-Struktur, die ähnliche Probleme wie die SAP-Exporte aufweist. Es illustriert mehrere der beschriebenen Probleme gleichzeitig, insbesondere mehrzeilige bzw. unvollständige Header bzw. Header in mehreren Zeilen, leere Spalten, redundante Bezeichnungen sowie uneinheitliche Formatierungen numerischer Werte und Daten:

```
;;;;;;;;;
;;;;;;;;;
;;Geburt;Wohnort;Euro;Euro;;;;
Name;Alter;Stadt;Stadt;Gehalt;Sozialabgaben;Netto;Geburtsdatum;Ausbildungsbeginn;
"Tom";"25";"Berlin";"Berlin";"3000";"500";"2.500,00EUR";"2000.1.21";"2023/09";
"Anna";"28";"";"";"3200";"600";"2.600,00EUR";"1997.09.21";"2020/09";
"Max";"20";"Hamburg";"Hamburg";#;#;#;"2005.07.13";"2025/09";
```

Listing 3.1: CSV-Beispiel

Diese Herausforderungen machen eine systematische Datenbereinigung und Normalisierung zwingend erforderlich, bevor eine Analyse oder Zusammenführung der unterschiedlichen Datenquellen möglich ist.

3.1.2 Technische Rahmenvorgaben

Zusätzlich zu den organisatorischen und datenbezogenen Einschränkungen bestehen mehrere technische Vorgaben, die die Architektur und Umsetzung des Systems wesentlich beeinflussen:

- **Verwendung von Python:** Die Implementierung des Systems hat zwingend in der Programmiersprache Python zu erfolgen. Das ist eine Vorgabe der Auftraggebenden und ist mit der Wartbarkeit zu begründen.
- **Lokale Desktop-Anwendung unter Windows:** Die Anwendung soll als lokale Desktop-Applikation auf Windows-Arbeitsplätzen der Buchhaltung betrieben werden. Daraus ergibt sich, dass sämtliche Datenverarbeitung und Speicherung lokal erfolgen muss. Diese Einschränkung gilt nur für das MVP.
- **Serverbasierte Multi-User-Funktionalität** Um die Software zukunftssicher und skalierbar zu machen, ist es wichtig, die Software so zu gestalten, dass eine spätere Umstellung auf eine Multiuser-Architektur mit einem Client-Server-Modell möglichst einfach zu gewährleisten ist.(vgl. US-22)
- **CSV-basierte Datenimporte:** Als Datenbasis stehen ausschließlich die manuell erzeugten CSV-Exporte aus SAP Fiori zur Verfügung. Diese Dateiformate sind fest vorgegeben und bilden die einzige maschinell verarbeitbare Eingabequelle. Die Architektur des Systems muss daher auf den Import, die Bereinigung und Normalisierung solcher CSV-Dateien ausgerichtet sein.
- **OData-Importe** Eine spätere Anbindung an die von SAP bereitgestellten OData-Services soll im Rahmen der Planung berücksichtigt jedoch nicht implementiert werden.

Tabelle 3.1: Übersicht der Rahmenbedingungen und Einschränkungen

ID	Kurzbeschreibung
RB-01	Zugriff auf SAP-OData-Services ist aufgrund des Genehmigungsprozesses im Rahmen der Seminararbeit nicht realisierbar. Kein direkter Online-Zugriff.
RB-02	Auslesen von Daten aus dem SAP-Webinterface mittels Reverse Engineering ist rechtlich unzulässig.
RB-03	Sämtliche Daten müssen aus bereitgestellten CSV-Exporten und manuellen Eingaben gewonnen werden.
RB-04	Verwendete Datenquellen sind ausschließlich die HCM- und FI-CSV-Exporte mit projektbezogenen Personal- und Finanzinformationen.
RB-05	Die CSV-Exporte besitzen keine einheitliche Struktur und sind teilweise zeitverzögert; eine Bereinigung und Normalisierung ist zwingend erforderlich.
RB-06	Die Implementierung hat zwingend in Python zu erfolgen.
RB-07	Das MVP ist als lokale Desktop-Anwendung unter Windows mit vollständig lokaler Verarbeitung und Speicherung auszuführen.
RB-08	Die Architektur soll eine spätere Umstellung auf eine Multi-User-Client-Server- Architektur ermöglichen.
RB-09	Eine spätere Anbindung an SAP-OData-Services ist konzeptionell zu berücksichtigen, wird im MVP aber nicht implementiert.

3.2 Zielgruppe und Stakeholder

Die primäre Zielgruppe des Systems sind die Mitarbeitenden der Buchhaltung, die regelmäßig Finanz- und Personaldaten aus SAP exportieren und weiterverarbeiten müssen. Da diese Nutzenden üblicherweise nicht über tiefgehende technische Kenntnisse verfügen, ist eine klar strukturierte und intuitive Benutzeroberfläche entscheidend für die erfolgreiche Nutzung des Systems.(vgl. US-17)
Auch visuelle Werkzeuge zur Definition von Filter- oder Zuordnungsregeln müssen ohne Programmierkenntnisse nutzbar sein(vgl. US-18). Neben den operativen Nutzenden sind die Auftraggebenden wichtige Stakeholder, da sie besonderen Wert auf Erweiterbarkeit und langfristige Wartbarkeit legen (vgl. US-22). Die IT-Abteilung schließlich stellt Anforderungen hinsichtlich Nachvollziehbarkeit und Fehlerdiagnose, welche in den User Stories zur Klarheit von Fehlermeldungen wiederzufinden sind (vgl. US-23).

3.3 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben das Verhalten des Systems sowie die Funktionen, die das System bereitstellen muss. Hier werden alle Anforderungen an die zu entwickelnde Software erhoben. Später werden die im Rahmen des MVP zu berücksichtigen Funktionen abgegrenzt.

Tabelle 3.2: Zielgruppen und zentrale Anforderungen

Stakeholder	Zentrale Anforderungen	User Stories
Mitarbeitende der Buchhaltung	Intuitive, klar strukturierte Benutzeroberfläche; möglichst wenige Interaktionen pro Arbeitsschritt.	US-17, US-18, US-19
Mitarbeitende der Buchhaltung	Visuelle Konfiguration von Filtern, Regeln und Zuordnungen ohne SQL- oder Programmierkenntnisse.	US-14, US-18
Auftraggebende	Modularer, erweiterbarer Aufbau des Systems, um zukünftige Schnittstellen, Exporte und Architekturänderungen ohne grundlegende Umbauten zu ermöglichen.	US-22
IT-Abteilung	Nachvollziehbarkeit von Fehlern und Systemzuständen durch klare, verständliche Fehlermeldungen und Logging.	US-23

3.3.1 Projektverwaltung

Um projektbezogene Finanzübersichten erstellen zu können, muss das System die Anlage und Verwaltung beliebiger Projekte ermöglichen (vgl. US-01). Für jedes Projekt sollen die bewilligten Mittel pro Jahr hinterlegt und den entsprechenden Kategorien wie z.B. Personalkosten der Finanzübersicht zugeordnet werden können (vgl. US-02). Darüber hinaus ist die Möglichkeit erforderlich, Projekte vollständig zu entfernen, wobei sicherheitsrelevante Bestätigungsdialoge den Löschvorgang absichern müssen (vgl. US-03 und US-19).

Für jedes Projekt müssen die aus SAP stammenden CSV-Exporte aus den Modulen HCM und FI eingelesen und verarbeitet werden können (vgl. US-04). Dazu gehört insbesondere, das System uneinheitliche oder fehlerhafte Headerstrukturen erkennt, bereinigt und automatisiert geeignete Korrekturvorschläge generiert (vgl. US-05). Die Nutzenden sollen zusätzlich die Möglichkeit haben, Spaltenbezeichnungen manuell anzupassen (vgl. US-06) und jedem Feld einen passenden Datentyp zuzuweisen (vgl. US-07). Die so definierten Zuordnungen und Datentypen müssen als wiederverwendbare Masken gespeichert werden können, um zukünftige Importe von gleich strukturierten Daten effizient durchführen zu können (vgl. US-08).

Neben den über CSV importierten Daten sollen langfristig auch die SAP-OData-Schnittstellen als alternative Importquelle genutzt werden können, um den manuellen Aufwand zu reduzieren. Ergänzend müssen Nutzende weitere Daten manuell erfassen können, insbesondere geplante oder zukünftige Finanzvorgänge wie prognostizierte Personalkosten oder noch nicht gebuchte Rechnungen. Diese sogenannten `Plandaten` sollen später automatisch durch die entsprechenden SAP-Einträge ersetzt werden, sobald diese vorliegen (vgl. US-09 und US-13).

Tabelle 3.3: Funktionale Anforderungen Projektverwaltung

User Story	Kurzbeschreibung
US-01	Anlegen neuer Projekte zur getrennten Verwaltung projektbezogener Finanzdaten.
US-02	Speichern projektspezifischer Einstellungen und Spaltenkonfigurationen, um wiederkehrende Importe zu vereinfachen.
US-03	Löschen bestehender Projekte mit Sicherheitsabfrage zur Vermeidung unbeabsichtigter Löschvorgänge.
US-04	Import von CSV-Dateien aus den Modulen HCM und FI.
US-05	Automatische Erkennung und Korrekturvorschläge für fehlerhafte oder uneinheitliche Spaltenbezeichnungen.
US-06	Manuelle Anpassung von Spaltennamen zur Verbesserung der Verständlichkeit.
US-07	Festlegung passender Datentypen pro Spalte (Text, Datum, Zahl) für korrekte Verarbeitung.
US-08	Speichern von Spalten- und Typzuordnungen als wiederverwendbare Masken für konsistente Importe.
US-09	Manuelles Ergänzen zusätzlicher Datensätze, insbesondere zukünftiger Ausgaben und Plandaten, die in SAP noch nicht erfasst sind.
US-13	Automatische Berücksichtigung von Plandaten in Auswertungen und späteres Ersetzen durch tatsächliche SAP-Buchungen.

3.3.2 Datenverarbeitung, Analyse und Export

Nach dem Import müssen die Daten weiter analysiert werden können. Dazu gehört die Möglichkeit, Datensätze nach frei definierbaren Kriterien zu filtern und zu gruppieren, um nur die jeweils relevanten Informationen anzuzeigen (vgl. US-10 und US-11). Auf dieser Grundlage sollen monatliche projektbezogene Übersichten generiert werden, die Einnahmen, Ausgaben, Personalkosten und die daraus resultierenden Budgetentwicklungen darstellen (vgl. US-12). Hierbei müssen sowohl die realen SAP-Daten als auch die manuell erfassten Plandaten berücksichtigt werden, um einen vollständigen Überblick über die finanzielle Situation zu vermitteln (vgl. US-13).

Für die Zuordnung der Daten zu den Zielspalten der Finanzübersicht soll ein regelbasiertes System bereitstehen, das ohne technische oder SQL-spezifische Kenntnisse nutzbar ist (vgl. US-14). Alle berechneten Übersichten müssen exportierbar sein. Der Export ist in erster Linie im CSV-Format vorzusehen, darüber hinaus sollen jedoch auch druckbare Ausgabeformen wie PDF oder XLS unterstützt werden (vgl. US-15 und US-16). Dies ermöglicht die Erstellung formaler Verwendungsnachweise oder die Weitergabe von Finanzübersichten an externe Stellen.

Tabelle 3.4: Funktionale Anforderungen Datenverarbeitung, Analyse und Export

User Story	Kurzbeschreibung
US-10	Filtern von Daten nach frei definierbaren Kriterien.
US-11	Gruppieren von Datensätzen nach beliebigen Attributen zur Erstellung aggregierter Übersichten.
US-12	Monatliche projektbezogene Finanzübersicht mit Einnahmen, Ausgaben und Personalkosten.
US-13	Einbeziehung von Plandaten in die Auswertung zur Beurteilung zukünftiger Budgetentwicklungen.
US-14	Regelbasierte Zuordnung von Daten zu Kategorien (z. B. Einnahmen, Ausgaben, Personalkosten) ohne SQL-Kenntnisse.
US-15	Export der erzeugten Übersichten als CSV-Datei.
US-16	Export der Übersichten als PDF oder XLS zur Erstellung formaler Nachweise und Berichte.

3.3.3 Benutzeroberfläche

Das System benötigt eine grafische Benutzeroberfläche, die ohne technische Vorkenntnisse bedienbar ist und alle wichtigen Arbeitsschritte übersichtlich abbildet(vgl. US-17). Es muss Dialoge zur Erstellung von Projekten und zum Import von CSV-Dateien geben, idealerweise auch per Drag-and-Drop. Um die Headerinformationen der CSV-Daten zu bearbeiten, soll eine grafische Oberfläche vorhanden sein, in der die Zuordnung von Originalspaltennamen zu einem neuen Spaltennamen möglich ist. Die Felder für den neuen Spaltennamen sollen bereits mit den automatisch angepassten Headerfeldern befüllt sein. Durch Klicken in dieses Feld sollen die Namen aber noch bearbeitbar sein (vgl. US-06). Außerdem soll zu jeder Spalte ein Dropdown-Menü existieren, welches es den Nutzenden ermöglicht ein Datentyp für das Feld festzulegen. Datentypen können Text, Datum und Zahl sein. Die Datentypen sollen ebenfalls bereits mit den geschätzten richtigen Werten populiert sein (vgl. US-07).

Für jedes Projekt soll eine Finanzübersicht verfügbar sein, bestehend aus aktuellem Kontostand, Einnahmen, Ausgaben, gezahlten Gehältern, geplanten Ausgaben sowie der Verrechnung mit den bewilligten Finanzierungsmitteln.

Für die Erstellung der Übersichten muss eine Zuordnung von Quell- zu Zielspalten möglich sein. Dazu soll das User Interface entsprechende Konfigurationseinstellungen bereitstellen (vgl. US-18). Im Exportbereich sollen die Nutzenden das Ausgabeformat auswählen und die zu exportierenden Spalten festlegen können (vgl. US-15). Sämtliche Arbeitsabläufe sollen mit möglichst wenigen Interaktionen auskommen, um die Bedienbarkeit zu maximieren. Alle erstellten Daten müssen auch wieder gelöscht werden können. Um versehentliches Löschen zu vermeiden muss bei jedem Löschvorgang eine Bestätigungsanfrage gestellt werden (vgl. US-19).

Tabelle 3.5: Funktionale Anforderungen Benutzeroberfläche

User Story	Kurzbeschreibung
US-17	Klar strukturierte, intuitive Benutzeroberfläche ohne technischen Vorwissensbedarf.
US-18	Visuelle Konfiguration von Regeln, Filtern und Zuordnungen über die GUI, ohne Programmierkenntnisse.
US-19	Sicherheitsabfragen bei kritischen Aktionen (z. B. Löschen von Projekten), um unbeabsichtigte Datenverluste zu vermeiden.

3.4 Nicht-funktionale Anforderungen

Neben den funktionalen Anforderungen muss das System mehrere qualitative Eigenschaften erfüllen, die die Bedienbarkeit, Leistungsfähigkeit, Sicherheit sowie die langfristige Erweiterbarkeit und Wartbarkeit betreffen. Die im Anhang aufgeführten User Stories konkretisieren diese Anforderungen.

Ein wesentlicher Aspekt betrifft die **Usability**. Da die primären Nutzenden keine technischen Vorkenntnisse besitzen, muss die Benutzeroberfläche intuitiv verständlich, klar strukturiert und logisch aufgebaut sein (vgl. US-17). Fehlermeldungen sollen präzise formuliert sein und konkrete Hinweise zur Problemlösung geben (vgl. US-23). Darüber hinaus sollen typische Arbeitsschritte mit möglichst wenigen Interaktionen durchgeführt werden können. Visuelle Konfigurationsoptionen sollen komplexe Tätigkeiten erleichtern(vgl. US-18).

Auch die **Performance** stellt eine zentrale Anforderung dar. Das System muss in der Lage sein, große CSV-Dateien ohne spürbare Verzögerungen zu importieren, zu filtern und zu gruppieren (vgl. US-20). Zudem sollen sowohl der Programmstart als auch das Laden bestehender Projekte schnell erfolgen, um den Einsatz im regelmäßigen Arbeitsalltag der Buchhaltung nicht zu verlangsamen.

Im Hinblick auf die **Sicherheit** ist im Rahmen des MVP sicherzustellen, dass alle sensiblen Finanz- und Personaldaten ausschließlich lokal verarbeitet und gespeichert werden (vgl. US-21). Da keine Übertragung an externe Systeme erfolgt, reduziert sich die Angriffsfläche auf den jeweiligen Arbeitsplatz. Zukünftig, im Kontext einer möglichen Multi-User- oder Serverarchitektur, müssen erweiterte Sicherheitsmechanismen wie Zugriffskontrollen, Verschlüsselung oder Netzwerkabsicherung ergänzt werden. Für das MVP gilt jedoch ausdrücklich das Prinzip der reinen lokalen Verarbeitung.

Auch spielt die **Erweiterbarkeit** der Software eine zentrale Rolle. Die Architektur muss so gestaltet sein, dass zukünftige Erweiterungen ohne grundlegende Eingriffe in die bestehende Struktur möglich sind (vgl. US-22). Dies umfasst einen modularen Aufbau, der Import, Analyse, Export, Benutzeroberfläche und Datenhaltung klar voneinander trennt. Neue Datenquellen oder zusätzliche SAP-Exporte sollen sich durch die Erweiterung einzelner Module integrieren lassen, ohne dass die Gesamtarchitektur angepasst werden muss. Ebenso muss die interne Datenrepräsentation so flexibel sein, dass unterschiedliche CSV-Strukturen oder neue Mapping-Mechanismen unterstützt werden können. Die spätere Integration automatisierter Schnittstellen wie SAP OData sowie der mögliche Wechsel von lokalen zu externen Datenbanken sollen konzeptionell bereits vorgesehen sein.

Schließlich ist für eine langfristige Nutzbarkeit eine hohe **Wartbarkeit** erforderlich. Diese setzt eine klare Trennung zwischen Benutzeroberfläche, Geschäftslogik und Datenzugriff voraus, um gezielte Weiterentwicklungen zu ermöglichen. Darüber hinaus soll das System so gestaltet sein, dass zentrale Funktionen wie CSV-Parsing oder Filtermechanismen über die Benutzeroberfläche konfiguriert werden können, ohne den Quellcode anpassen zu müssen. Ein strukturiertes Fehler- und Ereignis-Logging unterstützt die Diagnose von Problemen und trägt ebenfalls zur Wartbarkeit bei (vgl. US-23).

Tabelle 3.6: Nicht-funktionale Anforderungen

User Story	Kurzbeschreibung
US-17	Hohe Usability durch eine intuitive, klar strukturierte und logisch aufgebaute Benutzeroberfläche.
US-18	Reduktion der Komplexität durch visuelle Konfigurationsmöglichkeiten für Filter- und Zuordnungsregeln.
US-20	Performante Verarbeitung großer CSV-Dateien ohne spürbare Verzögerungen beim Import, Filtern und Gruppieren.
US-21	Ausschließlich lokale Verarbeitung und Speicherung aller sensiblen Finanz- und Personaldaten im MVP.
US-22	Modularer, erweiterbarer Systemaufbau zur Unterstützung zukünftiger Schnittstellen, Exporttypen und Architekturänderungen.
US-23	Klare, verständliche und hilfreiche Fehlermeldungen sowie Logging zur Unterstützung von Diagnose und Wartung.

4 Konzeption des MVP

Auf Grundlage der zuvor erhobenen funktionalen und nicht-funktionalen Anforderungen wird in diesem Kapitel das Konzept für das zu entwickelnde MVP vorgestellt. Ziel ist es, eine konsistente, modular aufgebaute und erweiterbare Systemarchitektur zu definieren, die die Kernanforderungen erfüllt und gleichzeitig die spätere Erweiterbarkeit insbesondere im Hinblick auf Multi-User-Funktionalität sowie verschiedener Benutzeroberflächen gewährleistet.

Das Kapitel umfasst die Zielsetzung des MVP, den grundlegenden Architekturentwurf, das Datenmodellierungskonzept sowie den Entwurf der Benutzeroberfläche. Darüber hinaus wird begründet, warum bestimmte Technologien ausgewählt wurden und wie sie zur Erfüllung der Anforderungen beitragen.

4.1 Zielsetzung des MVP

Das MVP soll eine erste funktionsfähige Version des Systems darstellen, welche die grundlegenden Prozesse des Datenimports, der Normalisierung und der finanziellen Auswertung implementiert. Es fokussiert sich auf die Kernfunktionalitäten, die für die tägliche Arbeit der Buchhaltung relevant sind, ohne bereits sämtliche Komfortfunktionen oder langfristig geplanten Erweiterungen wie Multi-User oder OData-Datenimport zu implementieren.

Im Mittelpunkt steht die Funktionalität, CSV-Exporte aus SAP strukturiert zu importieren, zu bereinigen und auszuwerten. Auf Grundlage dieser Daten soll das MVP eine übersichtliche Darstellung der Einnahmen, Ausgaben und Personalkosten auf Monatsbasis ermöglichen und grundlegende Exportfunktionen bereitstellen. Ergänzend können zukünftige Finanzvorgänge als Plandaten erfasst werden, um eine Abbildung der Budgetentwicklung zu ermöglichen.

Das MVP verfolgt darüber hinaus das Ziel, eine Architektur zu etablieren, die spätere Erweiterungen wie alternative Benutzeroberflächen oder den Einsatz externer Datenbanken vereinfacht. Diese Aspekte werden jedoch nicht funktional umgesetzt, sondern lediglich strukturell vorbereitet.

4.2 Architekturentwurf

Das System wird als modular aufgebautes Client-Server-System konzipiert. Obwohl die Anwendung im MVP ausschließlich lokal ausgeführt wird, bietet die Trennung zwischen Frontend und Backend erhebliche Vorteile. Sie ermöglicht die spätere Auslagerung des Backends auf einen externen Server, ohne dass Anpassungen an der Benutzeroberfläche erforderlich sind. Ebenso eröffnet sie die Möglichkeit, alternative Frontends wie ein Web- oder Mobile-Interface einzusetzen, ohne die zugrunde liegende Logik verändern zu müssen.[13]

Die Kommunikation zwischen Benutzeroberfläche und Backend erfolgt vollständig über eine interne REST-API. Die Desktopanwendung, welche mit PySide6 implementiert wird, fungiert als Client und interagiert ausschließlich über klar definierte HTTP-Endpunkte mit dem Backend. Auf diese Weise entsteht eine lose Kopplung zwischen Benutzeroberfläche und Geschäftslogik, was sowohl die Austauschbarkeit des Frontends als auch die langfristige Erweiterbarkeit erleichtert.[5]

Das Backend selbst ist in mehrere klar abgegrenzte Schichten gegliedert. Die API-Schicht (FastAPI) dient als Zugangspunkt zur Geschäftslogik und stellt sämtliche Funktionen nach außen bereit. Die Geschäftslogik selbst ist in Services organisiert, die unabhängig von der konkreten Datenhaltung implementiert sind. Der Zugriff auf die Datenbank erfolgt über eine Repository-Schicht, deren Interfaces einheitliche Zugriffsmuster definieren. Diese Struktur ermöglicht es, die zugrunde liegende SQLite Datenbank im MVP künftig ohne Veränderungen an der Geschäftslogik durch alternative Systeme wie MariaDB oder PostgreSQL zu ersetzen.[14]

Die Modularität ergibt sich somit aus der Entkopplung von Benutzeroberfläche, Geschäftslogik und Datenzugriff. Jede dieser Schichten kann unabhängig weiterentwickelt, ausgetauscht oder erweitert werden. Die REST-basierte Kommunikation stellt sicher, dass selbst tiefgreifende Änderungen an der Datenhaltung oder der Architektur des Backends keine Auswirkungen auf die Bedienoberfläche haben. Gleichzeitig schafft diese Struktur die Grundlage dafür, das System in späteren Entwicklungsstufen als Multi-User-System mit externem Server zu betreiben, ohne das grundlegende Architekturkonzept verändern zu müssen. Die folgende Abbildung zeigt den Aufbau der Architektur mit ihren Schichten. Die Pfeile zeigen die Datenflussrichtung sowie die Interaktionsrichtung.

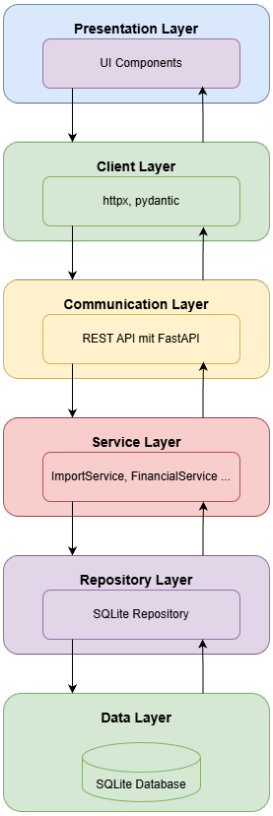

Abb. 4.1

4.3 Datenmodellierung

Die Datenmodellierung des MVP verfolgt das Ziel, die CSV-Exporte aus SAP so aufzubereiten, dass sie unabhängig von ihrer ursprünglichen Struktur einheitlich verarbeitet werden können. Die relevanten Kernobjekte wie Projekte, Importvorgänge oder Zuordnungskonfigurationen werden als feste Datenmodelle abgebildet und bilden die Grundlage für sämtliche weitere Auswertungen. (vgl. Abbildung A.6) Die eigentlichen Buchungs- und Personaldaten aus den CSV-Dateien werden hingegen in dynamisch erzeugten Tabellen pro Projekt gespeichert, deren Schemas direkt aus den bereinigten Spaltenüberschriften gebildet werden.(vgl. Abbildung A.7) Dadurch entfällt die Notwendigkeit eines starren Tabellenschemas. Das ist insbesondere angesichts der inhaltlich und strukturell wechselnden SAP-Exporte ein Vorteil. Während der Importphase werden Datums- und Betragsangaben in einheitliche, maschinenlesbare Formate transformiert, um zuverlässige Aggregationen und Berechnungen zu ermöglichen. Die Trennung zwischen stabilen fachlichen Entitäten und den dynamischen Importtabellen schafft damit sowohl Flexibilität als auch Konsistenz und bildet eine robuste Grundlage für spätere Erweiterungen des Systems und Sicherheit bezüglich sich verändernder CSV-Strukturen.

Ein Nachteil dieses Ansatzes besteht darin, dass die dynamisch generierten Importtabellen nicht über klassische Foreign-Key-Beziehungen in das restliche Datenmodell eingebunden werden können. Da Tabellenname und Spaltenstruktur erst zur Laufzeit entstehen, lassen sich keine stabilen referenziellen Verknüpfungen definieren. Wird eine Tabelle umbenannt oder ein Importvorgang entfernt, können bestehende Verweise nicht durch das Datenbankmanagementsystem abgesichert oder automatisch aktualisiert werden. Dies erhöht das Risiko inkonsistenter Konfigurationen, da alle Relationen ausschließlich über frei benennbare Zeichenketten (z. B. Tabellennamen oder Spaltenbezeichner) realisiert werden.

Mit zunehmender Zahl an Projekten führt dieser Ansatz zudem zu einer starken Fragmentierung der Datenbasis. Für jedes Projekt entstehen mehrere eigenständige Tabellen mit teils ähnlicher Struktur, was langfristig zu einem Wildwuchs an Tabellen führt. Dies erschwert die Wartung und Fehlersuche.

Diese Nachteile sind für ein MVP tolerierbar, da der Ansatz maximale Flexibilität bei unstrukturierten CSV-Dateien bietet und die Implementierung deutlich vereinfacht. Für ein zukünftiges produktives System wäre jedoch eine Anpassung des Datenbankschemas zu betrachten.

4.4 Entwurf der Benutzeroberfläche

Die Benutzeroberfläche des MVP ist so konzipiert, dass sie die wesentlichen Arbeitsschritte der Buchhaltung in klar nachvollziehbarer Reihenfolge abbildet. Sie besteht aus einer Projektverwaltung, einem Bereich für den CSV-Import sowie einer Ansicht zur Darstellung und Analyse der eingelesenen Daten. Des Weiteren werden verschiedene Ansichten zur Konfiguration bereitgestellt. Der Entwurf folgt dem Prinzip einer möglichst reduzierten und leicht verständlichen Bedienführung, da die Zielgruppe keine technischen Vorkenntnisse besitzt. Ein zentraler Bestandteil ist ein visueller Filter- und Zuordnungsmechanismus, der die Erstellung von Regeln ohne direkte SQL-Kenntnisse ermöglicht. Die gesamte Oberfläche ist funktional gehalten und beschränkt sich auf die für den MVP notwendigen Funktionen.[15] Zur prototypischen Entwicklung des Designs wurden Mockups mit der Software Figma erstellt. Die

beschriebenen Gestaltungskonzepte adressieren insbesondere die in den User Stories formulierten Anforderungen an eine einfache, nicht-technische Bedienbarkeit und eine visuelle Konfiguration der Filter- und Zuordnungsregeln.(vgl. US-14, US-17)

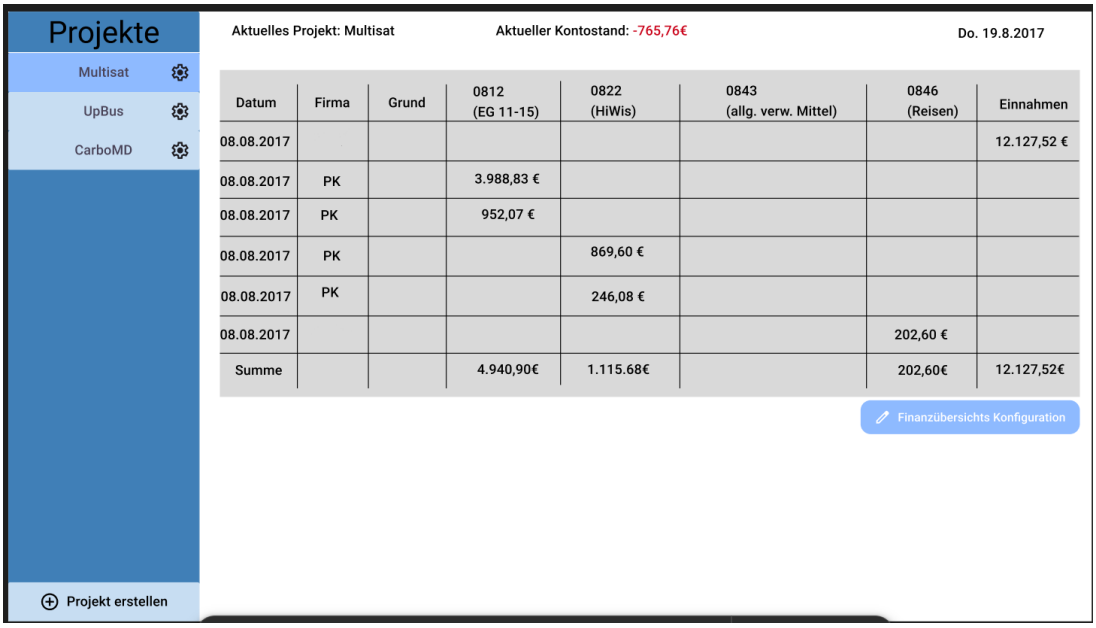

Abb. 4.2: Mockup der Startoberfläche mit Projektliste und Finanzübersicht

Abbildung 4.1 zeigt das Mockup der Startoberfläche für bereits konfigurierte Projekte. Links befindet sich eine Übersicht aller Projekte. Durch Klicken auf einen Eintrag wird die entsprechende Projektübersicht geöffnet.

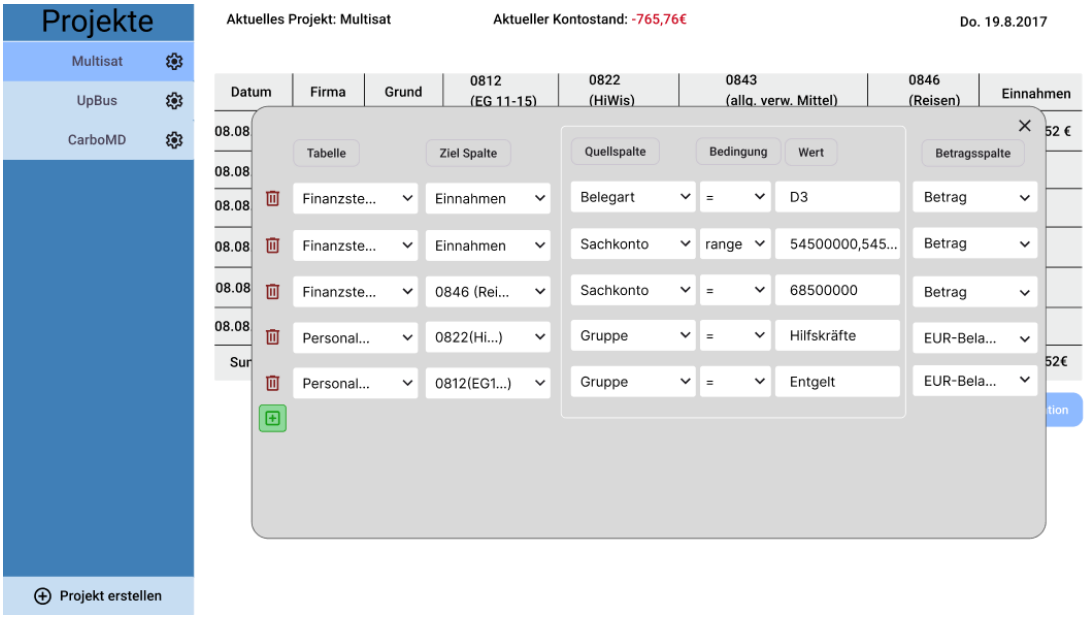

Abb. 4.3: Mockup der Zuordnung von Quellspalten und Filtern zu Zielspalten der Finanzübersicht

In Abbildung 4.2 wird das Design für die Zuordnung der Spalten zu den in der Übersicht gewünschten Zielspalten zusammen mit den entsprechenden Filtern dargestellt. Links wird der Datenursprung in Form der Quelltabelle ausgewählt. Anschließend erfolgt die Auswahl der Zielspalte in der Finanzübersicht. Um die Werte aus den Originaldaten eindeutig zuordnen zu können, wird ein Filter definiert, der aus einer Quellspalte, einer Bedingung und einem Wert besteht, der diese Bedingung erfüllt. Rechts wird zusätzlich festgelegt, in welcher Spalte der Originaldaten der zu übernehmende Wert zu finden ist. Diese visuelle Konfiguration unterstützt die in den User Stories formulierte Anforderung, Klassifizierungsregeln ohne direkte SQL-Kenntnisse definieren und anpassen zu können.

4.5 Technologieauswahl

Die für das MVP ausgewählten Technologien orientieren sich an der Zielsetzung, eine lokal lauffähige Anwendung mit klarer Trennung von Benutzeroberfläche und Geschäftslogik zu realisieren. Python dient als zentrale Programmiersprache, da es eine schnelle Entwicklung und eine umfangreiche Unterstützung für Datenverarbeitung bietet. Außerdem wird die Wartbarkeit erhöht, da die am Institut eingesetzte Programmiersprache Python ist.

Für das Backend wird FastAPI eingesetzt, um die Geschäftslogik über eine interne REST-Schnittstelle bereitzustellen. FastAPI ist im Vergleich zu anderen Frameworks wie Django eher leichtgewichtig und bringt keinen eigenen Object-Relational Mapping (ORM), kein Session-Management und keine Templating-Engine mit sich. Django ist für das Projekt überdimensioniert und würde den Implementierungsaufwand nur erhöhen, ohne Vorteile zu bieten. Flask ist zwar ähnlich leichtgewichtig wie FastAPI, bietet aber keine automatische Dokumentation.[16, 17]

Die Daten werden lokal in einer SQLite-Datenbank gespeichert, da diese keinen zusätzlichen Installationsaufwand erfordert und gleichzeitig für spätere Erweiterungen durch andere Datenbanken vorbereitet ist.[18]

Die Benutzeroberfläche wird mit PySide6 umgesetzt, wodurch eine klassische Desktopanwendung entsteht, deren Interaktion vollständig über die REST-Schnittstelle erfolgt. PySide6 basiert auf Qt6, welches eines der modernsten GUI-Frameworks ist. Im Vergleich zu anderen GUI-Frameworks wie Tkinter bietet es eine Vielzahl an Widgets, unter anderem auch geeignete Widgets zur Darstellung von CSV-Daten in Tabellen.[19]

4.6 Qualitäts- und Testkonzept

Die Validierung des MVP erfolgt vollständig manuell, da im Rahmen der Seminararbeit keine automatisierten Tests implementiert werden. Dennoch wird der Testprozess so strukturiert, dass die wichtigsten Funktionen reproduzierbar geprüft werden können.

Für die Tests werden repräsentative CSV-Dateien aus der Buchhaltung sowie künstlich erzeugte Fehlerbeispiele verwendet. Zu den geprüften Szenarien gehören insbesondere uneinheitliche oder mehrzeilige Header, fehlende Werte sowie unterschiedliche Datums- und Zahlenformate. Für jedes Szenario wird geprüft, ob die Anwendung die Struktur der Datei korrekt erkennt, sinnvolle Korrekturvorschläge macht und eine konsistente Weiterverarbeitung erlaubt.

Die Validierung erfolgt über das Webinterface und die bereitgestellten REST-Endpunkte. Dabei werden folgende Schritte durchgeführt:

- Upload einer CSV-Datei über das Webinterface und Prüfung der erkannten Header.
- Kontrolle der automatisch zugewiesenen Datentypen und Korrekturvorschläge.
- Prüfung der bereinigten und normalisierten Daten in der tabellarischen Darstellung.
- Überprüfung der Aggregationen und Berechnungen in der Finanzübersicht.

Ein Test gilt als erfolgreich, wenn alle Daten ohne Fehler eingelesen, korrekt normalisiert und in der Finanzübersicht ohne Abweichungen dargestellt werden. Werden Probleme erkannt, zeigt das System verständliche Fehlermeldungen an und verhindert fehlerhafte Weiterverarbeitung.

Zwar werden im MVP keine automatisierten Tests implementiert, jedoch ermöglicht die klare Trennung zwischen Benutzeroberfläche, REST-Service und Geschäftslogik eine spätere Erweiterung um Unit- und Integrationstests ohne strukturelle Änderungen.

4.7 Zusammenfassung des Konzepts

Das Konzept des MVP definiert eine modulare und erweiterbare Architektur, die die Verarbeitung unterschiedlichster CSV-Exporte ermöglicht und eine klare Trennung zwischen Benutzeroberfläche, Geschäftslogik und Datenhaltung vorsieht. Durch die Nutzung einer internen REST-Schnittstelle bleibt das System langfristig flexibel und kann sowohl um weitere Frontends als auch um alternative Datenbanken ergänzt werden. Die Datenmodellierung kombiniert eine stabile fachliche Struktur mit dynamischen Tabellen für die Importdaten, während das UI auf eine verständliche und reduzierte Bedienführung ausgelegt ist. Insgesamt bildet das Konzept eine kompakte, auf die Kernanforderungen fokussierte Grundlage für die Umsetzung eines funktionsfähigen MVP.

5 Implementierung des MVP

Das folgende Kapitel beschreibt die Implementierung des MVP auf Grundlage der in Kapitel 4 beschriebenen Softwarearchitektur. Das Ziel dieses Kapitels ist nicht die vollständige Darstellung jedes Programmteils, sondern die Herausarbeitung der wesentlichen Prinzipien, die das Verhalten und die Qualität des MVP bestimmen.

5.1 Verwendete Software

Die nachfolgende Tabelle listet die verwendeten Softwaremodule und gibt einen kurzen Überblick über deren Funktion.

Komponente	**Technologie**	**Einsatzbereich**
PySide6	Python / Qt	Desktop-Benutzeroberfläche, Dialoge, Widgets
Qt Widgets	Qt	Tabellen, Dialoge, Layouts und Eingabekomponenten
QSS	Qt Style Sheets	Gestaltung der GUI
FastAPI	Python	REST-API, Routing, Request-Validierung
Uvicorn	Python	ASGI-Server zur Ausführung der FastAPI-Anwendung
Pydantic	Python	Datenvalidierung, Serialisierung, API-Modelle
HTTPX	Python	REST-Client für die Kommunikation zwischen GUI und Backend
SQLite3	Relationale DB	Lokale Speicherung von Projekten, Importtabellen, Finanzübersichten

Tabelle 5.1: Übersicht der im MVP verwendeten Technologien

5.2 Implementierung der Architektur

Die Umsetzung folgt konsequent der zuvor beschriebenen Client-Server-Trennung. Frontend und Backend kommunizieren über eine interne REST-API miteinander. Die Desktopanwendung agiert als reiner Client und implementiert selbst keine Logik zur Datenverarbeitung. Sämtliche Anfragen wie etwa der Import von CSV-Dateien, die Generierung von Finanzübersichten oder das Erstellen von Projekten werden über HTTP an das lokal laufende FastAPI-Backend gesendet.

Die Schichten kommunizieren ausschließlich über definierte Schnittstellen, wodurch die Geschäftslogik vollständig von der Datenhaltung isoliert bleibt. Durch diese Trennung ist die Software für eine spätere Umstellung von SQLite zu einer anderen Datenbank wie MariaDB oder PostgreSQL bestens vorbereitet.

5.2.1 Implementierung der Repository-Schicht

Über die Repository-Schicht werden sämtliche Datenbankzugriffe gekapselt. Sie stellt einheitliche Interfaces für die CRUD-Operationen bereit. Dadurch wird die Geschäftslogik von den SQL-Anweisungen und Datenbankspezifika getrennt. Da durch die stark variierenden Tabellenstrukturen, welche dynamisch zur Laufzeit generiert werden keine festen Schemata verwendet werden können, wurde auf den Einsatz eines ORMs verzichtet. ORMs wie SQLAlchemy gehen davon aus, dass die Schemata weitgehend stabil und im Vorhinein bekannt sind. Das steht in direktem Konflikt mit den dynamisch erzeugten Datenbanktabellen der CSV-Importe.[20] Die Repository-Schicht ersetzt die ORM-Funktionalität und sorgt so für eine höhere Wartbarkeit und bessere Testbarkeit.[14]

Listing 5.1 zeigt einen beispielhaften Ausschnitt aus dem Interface.

```python
# interfaces.py
class ProjectRepository(ABC):
@abstractmethod
def get_by_name(self, name: str) -> Optional[Project]: ...

@abstractmethod
def create(self, project: ProjectCreate) -> Project: ...
```

Listing 5.1: Interface Beispiel

Die Implementierung des Interfaces erfolgt analog in den entsprechenden Repositories. Im Falle des MVP ist dass das SQLite Repository.

```python
  class SQLiteProjectRepository(ProjectRepository):
    def get_by_name(self, name: str) -> Optional[Project]:
      with self._db.connection() as conn:
        cursor = conn.execute(
          "SELECT * FROM projects WHERE name = ?", (name,)
        )
      row = cursor.fetchone()
    return Project(**row) if row else None
```

Listing 5.2: Repository Beispiel

5.2.2 Implementierung der Service-Schicht

Die Serviceschicht enthält die gesamte Geschäftslogik und steuert die Repository-Operationen. Sie ist vollständig frei von SQL-Befehlen und besitzt keine Abhängigkeit von der konkreten Datenbank. Als zentrales Beispiel dient hier der ImportService. Er nimmt einen Projektnamen, Dateipfad, Configuration und import_type als Parameter und fügt anhand dieser Informationen die CSV-Datei in die Datenbank ein.

```python
class ImportService:
  def __init__(self, project_repo, import_repo):
    self._project_repo = project_repo
    self._import_repo = import_repo

  def import_csv(self, project_name: str, ...):
    project = self._project_repo.get_by_name(project_name)
    if not project:
      project = self._project_repo.create(...)

    # weitere Schritte: Tabellenerstellung, Datentypinferenz, Speicherung
```

Listing 5.3: Service Beispiel

Die Serviceschicht ist bewusst so implementiert, dass alle komplexen Operationen zentralisiert sind. Zusätzlich zum `ImportService` existieren der `FinancialService`, welcher die Berechnung der Budgets und Finanzübersichten orchestriert. Der `ProjectService` dient der Erstellung und Verwaltung von Projekten. Als letzten Service stellt die Software den `PlannedEntryService` bereit. Dieser ist für sämtliche Operationen die mit dem Erstellen und Verwalten von Plandaten in Form von zukünftigen Rechnungen und der Personalkosten in Zusammenhang stehen.

5.2.3 Implementierung der API

Die REST-API bildet die Schnittstelle zwischen GUI und Geschäftslogik. Hier werden die Funktionen aus den Services über klar abgegrenzte REST-Endpunkte dem Frontend zur Verfügung gestellt. Als Framework zur effizienten Erstellung der API kommt FastAPI zum Einsatz. FastAPI ist ein modernes, schnelles und hochperformantes Webframework zur Erstellung von APIs mit Python auf Basis von Standard-Python-Typhinweisen.[21] Eine beispielhafte Routendefinition zur Erzeugung eines Projekts ist in Listing 5.4 zu sehen.

```python
# routes/projects.py
@router.post("/projects")
def create_project(data: ProjectCreate, service: ProjectService = Depends(...)):
  return service.create_project(data)
```

Listing 5.4: REST-Endpunkt Beispiel

Durch die `router.post()` Annotation wird spezifiziert, dass es sich um eine POST-Anfrage handelt. Alle Post-Anfragen an die Route .../projects werden also von dieser Funktion bedient. Alle Endpunkte sind den Services entsprechend getrennt in:

- financial
- imports
- planned_entries
- projects

Zu allen Endpunkten existiert eine von FastAPI automatisch generierte Beschreibung nach der OpenAPI-Spezifikation. Die API ist als interaktive Dokumentation über eine Weboberfläche die mittels SwaggerUI aus der OpenAPI generiert wird erreichbar. Das bietet große Vorteile bei der Wartbarkeit und Erweiterbarkeit der Software. Außerdem lassen sich alle Endpunkte direkt über das Webinterface testen. Die SwaggerUI Dokumentation aller REST-Endpunkte ist den Abbildungen Abschnitt A.2 im Anhang zu entnehmen.

5.2.4 Implementierung des Client-Layers und der GUI

Der Client-Layer kapselt sämtliche HTTP-Kommunikation mit dem REST-Backend. Zentrale Komponente ist die APIClient Klasse. Diese stellt mithilfe von httpx einen HTTP-Client bereit.

Die GUI greift nicht direkt auf HTTP-Endpunkte zu, sondern verwendet ausschließlich Methoden des APIClient. Für alle fachlichen Funktionen wie Projektverwaltung, CSV-Import, Abruf der Finanzübersicht existieren jeweils klar benannte Methoden, die die REST-Routen kapseln. Listing 5.5 zeigt die ApiClient Kapselung der Route `/api/v1/projects`. Über diese wird eine Auflistung aller Projekt zurückgeliefert.

```
1  def list_projects(self) -> list[Project]:
2    """Get all projects."""
3    response = self._client.get("/api/v1/projects")
4    response.raise_for_status()
5    return [Project(**p) for p in response.json()]
```

Listing 5.5: ApiClient Beispiel

Die Antworten des Backends werden ebenfalls im Client-Layer konsumiert und in Pydantic-Modelle überführt. Mit diesen Modellen arbeitet das GUI.

Durch die strikte Entkopplung von der Präsentationsschicht und dem Transportprotokoll ist die Testbarkeit deutlich erhöht. In Unit-Tests können Aufrufe an das Backend einfach durch Mockups von ApiClient-Instanzen ersetzt werden, ohne die GUI selbst anpassen zu müssen.

Die Implementierung der GUI basiert vollständig auf PySide6 und folgt einem modularen Aufbau. Zentrale UI-Komponenten sind in unterschiedliche Klassen aufgeteilt. Der Einstiegspunkt ist das main_window, welches als Container für die unterschiedlichen Funktionsbereiche dient.

Ein zentraler Aspekt der Implementierung ist der Umgang mit dynamischen Datenstrukturen. Da die Benutzenden beliebige CSV-Exporte mit variierender Spaltenstruktur importieren können. Aus diesem Grund muss die GUI in der Lage sein, Tabellenansichten komplett dynamisch zu erzeugen. Dazu werden die Headerinformationen aus dem Backend gelesen und die Tabellenansichten entsprechend aufgebaut.

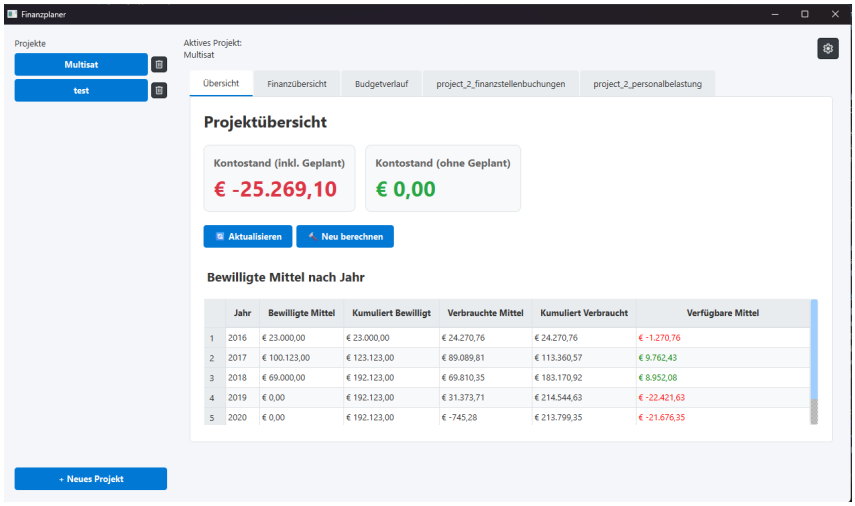

Abb. 5.1: GUI Projektübersicht

Das entwickelte Frontend orientiert sich eng an den zuvor erstellten Mockups. Abbildung 5.1 zeigt die Projektübersicht, auf der die berechneten Budgets sowie der aktuelle Kontostand dargestellt werden. Auf der linken Seite befindet sich die Liste aller Projekte. Der Kontostand ist in zwei Bereiche unterteilt: Links wird der Wert unter Einbezug der geplanten Ausgaben angezeigt, rechts hingegen wird der Kontostand ausschließlich auf Basis der tatsächlich gebuchten Werte angezeigt.

Im unteren Bereich sind die bewilligten Mittel nach Jahren gruppiert in einer Tabelle dargestellt. Die Spalte **Bewilligte Mittel** enthält die jeweils für ein Jahr zugesprochenen Beträge. **Kumuliert bewilligt** zeigt die Summe aller bis zu diesem Jahr bewilligten Mittel. **Verbrauchte Mittel** geben die tatsächlich im jeweiligen Jahr angefallenen Ausgaben an, während **Kumuliert verbraucht** die insgesamt bis zu diesem Jahr aufgelaufenen Ausgaben ausweist. Die Spalte **Verfügbare Mittel** entspricht schließlich der Differenz zwischen den kumulierten bewilligten und den kumulierten verbrauchten Mitteln.

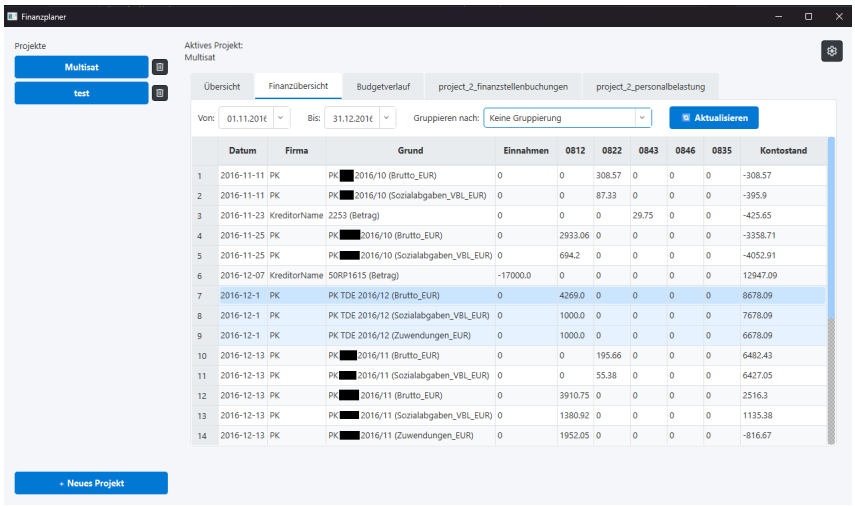

Abb. 5.2: GUI Finanzübersicht

Die Finanzübersicht zeigt die individuellen Rechnungen und lässt sich beliebig zeitlich eingrenzen. In Abbildung 5.2 werden beispielsweise die Buchungen aus Dezember 2016 gezeigt. Diese sind entsprechend der Filterkonfigurationen in Abbildung 5.3 den individuellen Spalten zugeordnet. Zu beachten ist, dass alle blau hinterlegten Zeilen Plandaten sind, welche noch nicht final gebucht wurden.

Oben rechts gelangt man über das Zahnrad zur Projektkonfiguration. Dort lassen sich sämtliche Daten für das Projekt konfigurieren.

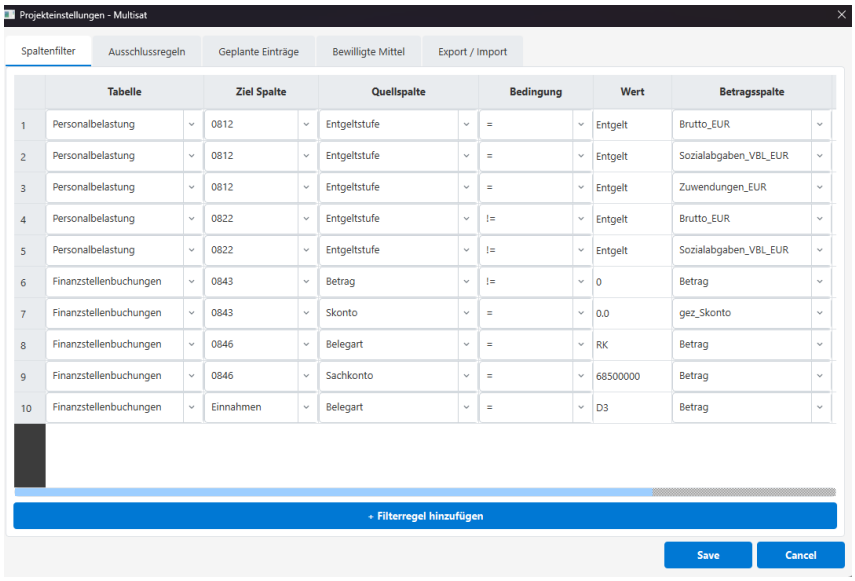

Abb. 5.3: GUI Filter

Die in Abbildung 5.3 gezeigten Filter lassen sich über den Tab Export / Import entsprechend exportieren und importieren, was eine einfache Wiederverwendbarkeit ermöglicht. Die Filter funktionieren exakt so wie in Kapitel 4 beschrieben.

6 Diskussion und Ausblick

In diesem Kapitel werden die im Rahmen der Entwicklung gewonnenen Ergebnisse kritisch eingeordnet, zentrale Architekturentscheidungen bewertet, sowie Grenzen und Weiterentwicklungsmöglichkeiten des MVP aufgezeigt.

Die wesentlichen funktionalen Anforderungen konnten erfüllt werden. Dazu zählen insbesondere die Verwaltung mehrerer Projekte, der Import und die Bereinigung von CSV-Daten, das Hinterlegen bewilligter Mittel sowie die Definition und Anwendung von Filterregeln. Auch die persistente Speicherung projektbezogener Konfigurationen wurde erfolgreich umgesetzt. Nicht realisiert wurde hingegen der Export der aggregierten Finanzübersichten, da das endgültige Ausgabeformat noch nicht abschließend spezifiziert war. Insgesamt zeigt der reduzierte Funktionsumfang im Vergleich zu SAP-Fiori-Anwendungen eine deutliche Vereinfachung der Arbeitsabläufe.

Die gewählte Architektur überzeugt durch Modularität und Erweiterbarkeit, führt jedoch zu einem höheren initialen Entwicklungsaufwand. Das Repository-Pattern erleichtert die Implementierung spezifischer SQL-Abfragen, während ein ORM größere Datenbankunabhängigkeit geboten hätte. Die Nutzung eines REST-Backends erfordert zwar zusätzliche Routen, eröffnet jedoch langfristig die Möglichkeit, unterschiedliche Frontends anzubinden.

Im Hinblick auf die Zielsetzung, eine Entscheidungsgrundlage für eine mögliche Weiterentwicklung der Software zu schaffen, zeigt das MVP klar, dass die automatisierte Verarbeitung der SAP-CSV-Daten technisch beherrschbar ist und die bestehenden manuellen Arbeitsschritte der Buchhaltung erheblich vereinfacht werden können. Die prototypische Umsetzung ermöglicht damit eine erste qualitative Einschätzung des Nutzens einer vollständigen Anwendung, insbesondere in Bezug auf die Reduktion von Fehlerquellen, die Standardisierung der Datenaufbereitung und die Wiederverwendbarkeit der Konfigurationen.

Gleichzeitig treten die Grenzen des MVP deutlich hervor und zeigen, an welchen Stellen weitere Investition notwendig wären, um einen produktiven Einsatz zu ermöglichen. Die grafische Benutzeroberfläche ist funktional, jedoch nicht produktionsreif. Eingabevalidierungen sind bislang nur rudimentär umgesetzt, und zahlreiche Randfälle werden bewusst nicht berücksichtigt. Der aktuelle Filtermechanismus erlaubt lediglich einfache Bedingungen. Für einen robusten Einsatz ist die Unterstützung logischer Operatoren wie **AND** und **OR** notwendig. Ebenfalls nicht umgesetzt wurden eine Mehrbenutzerfähigkeit mit rollenbasierten Rechten, ein umfassendes Logging sowie die direkte Anbindung an SAP-OData-Dienste.

Aus den gewonnenen Erkenntnissen ergeben sich mehrere Entwicklungsrichtungen. Vorrangig ist die Erweiterung des Filtersystems, gefolgt von Verbesserungen der Benutzeroberfläche und der Implementierung eines Exports der Finanzübersichten um diese als Verwendungsnachweise nutzen zu können. Langfristig bietet sich zudem die Integration mehrerer Benutzerrollen und eine direkte SAP-Anbindung an, um den manuellen CSV-Prozess zu ersetzen oder zu komplementieren. Zudem stellt die automatische Ersetzung von Plandaten durch tatsächliche SAP-Buchungen, sobald diese vorliegen, eine wichtige Erweiterung dar, da dadurch die Aktualität und Verlässlichkeit der Finanzübersicht deutlich verbessert würden.

7 Fazit

Das entwickelte MVP zeigt, dass eine lokal ausführbare Software zur strukturierten Verarbeitung von Finanzdaten technisch umsetzbar und fachlich hilfreich ist. Die zentralen Funktionen wurden erfolgreich implementiert und bilden eine solide Grundlage für weitere Schritte. Die Analyse verdeutlicht jedoch, dass für einen produktiven Einsatz zusätzliche Erweiterungen erforderlich sind, insbesondere im Bereich der Filterlogik, der Benutzerführung und der Mehrbenutzerfähigkeit.

Im Sinne der in der Zielsetzung genannten Funktion als Entscheidungsvorlage lässt sich festhalten, dass die Arbeit die technischen Möglichkeiten, den erwartbaren Nutzen sowie die notwendigen Erweiterungen klar aufzeigt. Damit bietet das MVP eine belastbare Grundlage, um über eine weiterführende Investition in die Entwicklung eines vollwertigen Systems zu entscheiden.

Insgesamt bestätigt die Arbeit den Nutzen eines solchen Werkzeugs für den buchhalterischen Arbeitsalltag und schafft die Basis für eine Weiterentwicklung zu einer vollständig einsetzbaren Lösung.

Literatur

[1] RWTH AACHEN. eingeführte SAP-Module. Abteilung 5.2 – SAP – Entwicklung und Betrieb. 2025. URL: https://intranet.rwth-aachen.de/group/guest/articledetailpage/-/asset_publisher/cadSeXJzMOzq/content/id/19312687 (besucht am 23. 11. 2025).

[2] RWTH AACHEN. MaCoCo - Lehrstuhlcontrolling. RWTH Aachen. URL: https://www.controlling.rwth-aachen.de/cms/Controlling/Forschung/Forschungsprojekte/Laufende-Projekte/~mgaz/MaCoCo-Lehrstuhlcontrolling/ (besucht am 10. 12. 2025).

[3] Andrew S. TANENBAUM und Maarten van STEEN. Distributed Systems. Principles and Paradigms. 2nd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2007. ISBN: 0132392275.

[4] Roy Thomas FIELDING, Mark NOTTINGHAM und Julian RESCHKE. RFC 9110 HTTP Semantics. 2022. DOI: 10.17487/RFC9110. (Besucht am 15. 12. 2025).

[5] Roy Thomas FIELDING. Architectural Styles and the Design of Network-based Software Architectures. Dissertation. Irvine: University of California, 2000.

[6] IBM. Was ist REST-API? Hrsg. von IBM. URL: https://www.ibm.com/de-de/think/topics/rest-apis (besucht am 10. 12. 2025).

[7] Leonard RICHARDSON und Sam RUBY. RESTful Web Services. Web Services for the Real World. Web Services for the Real World. Beijing und Köln: O'Reilly, 2007. ISBN: 9780596529260. URL: https://swbplus.bsz-bw.de/bsz265585090cov.htm.

[8] OPENAPIINITIATIVE. OpenAPI Specification. 2025. URL: https://github.com/OAI/OpenAPI-Specification (besucht am 04. 12. 2025).

[9] Yakov SHAFRANOVICH. RFC 4180: Common Format and MIME Type for CSV Files. 2005. DOI: 10.17487/RFC4180. URL: https://www.rfc-editor.org/info/rfc4180 (besucht am 15. 12. 2025).

[10] Jay A. KREIBICH. Using SQLite. Previous programming experience is recommended – P. [4] of cover. Includes index. 1st ed. Sebastopol, Calif.: O'Reilly, 2010. ISBN: 9780596521189. URL: http://gbv.eblib.com/patron/FullRecord.aspx?p=580130.

[11] SAP SE. Terms of Use for SAP Websites. SAP SE. 2025. URL: https://www.sap.com/germany/about/legal/terms-of-use.html (besucht am 08. 12. 2025).

[12] SAP SE. OData. SAP SE. 2025. URL: https://help.sap.com/docs/HANA_SMART_DATA_INTEGRATION/7952ef28a6914997abc01745fef1b607/d9f0a3b09e0f4b3eb010be8bd36871e5.html (besucht am 22. 11. 2025).

[13] Robert C. MARTIN und Kevlin HENNEY. Clean Architecture. Das Praxis-Handbuch für professionelles Softwaredesign: Regeln und Paradigmen für effiziente Softwarestrukturen. ger. Deutsche Ausgabe, 1. Auflage. Frechen: mitp, 2018. ISBN: 9783958457249.

[14] AzraJabeen MOHAMED ALI. Optimizing Software Architecture: Using the Repository Pattern in Decoupling Data Access Logic. In: International Scientific Journal of Engineering and Management 1.1 (2022). DOI: 10.55041/ISJEM00104.

[15] Ben SHNEIDERMAN u. a. Designing the User Interface. eng. 6th ed. Shneiderman, Ben; Plaisant, Catherine; Cohen, Maxine; Jacobs, Steven; Elmqvist, Niklas. Harlow, United Kingdom: Pearson Education Limited, 2017. ISBN: 9781292153919. URL: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=5832492.

[16] Mukul MANTOSH. Django vs. FastAPI: Which is the Best Python Web Framework? JetBrains. 2023. URL: https://blog.jetbrains.com/pycharm/2023/12/django-vs-fastapi-which-is-the-best-python-web-framework/#introduction (besucht am 15. 12. 2025).

[17] Stanley ULILI. Flask vs FastAPI: An In-Depth Framework Comparison. Better Stack, Inc. 2025. URL: https://betterstack.com/community/guides/scaling-python/flask-vs-fastapi/#documentation-and-openapi-integration.

[18] Distinctive Features Of SQLite. SQLite. 31.05.2025. URL: https://sqlite.org/different.html.

[19] Martin FITZPATRICK. PyQt vs. Tkinter — Which Should You Choose for Your Next GUI Project? What Are the Major Differences Between these Popular Python GUI Libraries. 8.04.2025. URL: https://www.pythonguis.com/faq/pyqt-vs-tkinter/ (besucht am 11. 12. 2025).

[20] MOHAMMED TAWFIK. SQL vs. ORM: Choosing the Right Tool for the Job. 2024. URL: https://xtawfik.medium.com/sql-vs-orm-choosing-the-right-tool-for-the-job-e0bc8c6fbe62 (besucht am 08. 12. 2024).

[21] TIANGOLO. FastAPI. 2025. URL: https://fastapi.tiangolo.com/ (besucht am 04. 12. 2025).

A Anhang

A.1 Userstories

Tabelle A.1: User Stories des Systems

ID	Bereich	User Story
US-01	Projektverwaltung	Als Nutzende möchte ich neue Projekte anlegen können, um projektbezogene Finanzdaten getrennt verwalten zu können.
US-02	Projektverwaltung	Als Nutzende möchte ich projektspezifische Einstellungen und Spaltenkonfigurationen speichern können, damit wiederkehrende Importe weniger Aufwand erfordern.
US-03	Projektverwaltung	Als Nutzende möchte ich bestehende Projekte löschen können, um veraltete oder nicht mehr benötigte Datenbestände zu entfernen.
US-04	Datenimport	Als Nutzende möchte ich CSV-Dateien importieren können, damit die aus SAP exportierten Finanz- und Personaldaten im System verarbeitet werden können.
US-05	Datenimport	Als Nutzende möchte ich, dass das System fehlerhafte oder uneinheitliche Spaltenbezeichnungen automatisch erkennt und Korrekturvorschläge bietet, um den Import zu erleichtern.
US-06	Datenimport	Als Nutzende möchte ich Spaltennamen manuell anpassen können, um fehlerhafte Header zu korrigieren oder die Daten verständlicher darzustellen.
US-07	Datenimport	Als Nutzende möchte ich für jede Spalte den passenden Datentyp festlegen können, damit Werte korrekt verarbeitet und aggregiert werden.
US-08	Datenimport	Als Nutzende möchte ich Spalten- und Typzuordnungen als Masken speichern und wiederverwenden können, um konsistente Importe sicherzustellen.
US-09	Datenimport	Als Nutzende möchte ich zusätzliche Datensätze manuell ergänzen können, insbesondere zukünftige Ausgaben, die in SAP noch nicht erfasst sind.
US-10	Datenanalyse	Als Nutzende möchte ich Daten nach bestimmten Kriterien filtern können, um nur die relevanten Einträge betrachten zu müssen.

ID	Bereich	User Story
US-11	Datenanalyse	Als Nutzende möchte ich Datensätze nach beliebigen Attributen gruppieren können, um aggregierte Finanzübersichten zu erstellen.
US-12	Datenanalyse	Als Nutzende möchte ich eine monatliche projektbezogene Finanzübersicht erhalten, um Einnahmen, Ausgaben und Personalkosten nachvollziehen zu können.
US-13	Datenanalyse	Als Nutzende möchte ich, dass Plandaten in die Auswertung einbezogen werden, um zukünftige Entwicklungen des Projektbudgets beurteilen zu können.
US-14	Datenanalyse	Als Nutzende möchte ich Daten mittels regelbasierter Zuordnungen Kategorien wie Einnahmen, Ausgaben oder Personalkosten zuordnen können, ohne SQL-Kenntnisse zu benötigen.
US-15	Export	Als Nutzende möchte ich erzeugte Übersichten als CSV-Datei exportieren können, um sie weiterzuverarbeiten oder weiterzugeben.
US-16	Export	Als Nutzende möchte ich Übersichten auch als PDF oder XLS exportieren können, um formale Verwendungsnachweise oder Berichte zu erstellen.
US-17	Benutzeroberfläche	Als Nutzende möchte ich eine klar strukturierte und intuitive Benutzeroberfläche vorfinden, damit ich effizient ohne technische Kenntnisse arbeiten kann.
US-18	Benutzeroberfläche	Als Nutzende möchte ich Regeln, Filter und Zuordnungen visuell konfigurieren können, um komplexe Verarbeitungsschritte ohne Programmierkenntnisse durchführen zu können.
US-19	Benutzeroberfläche	Als Nutzende möchte ich bei kritischen Aktionen wie dem Löschen eines Projekts eine Sicherheitsabfrage erhalten, um unbeabsichtigte Datenverluste zu vermeiden.
US-20	Systemanforderungen	Als Nutzende möchte ich, dass das System große CSV-Dateien performant verarbeiten kann, damit der Arbeitsablauf nicht unterbrochen wird.
US-21	Systemanforderungen	Als Nutzende möchte ich, dass alle Daten ausschließlich lokal gespeichert werden, um die Vertraulichkeit sensibler Finanz- und Personaldaten zu gewährleisten.
US-22	Systemanforderungen	Als Auftraggebende möchte ich ein modular aufgebautes System, das zukünftige Erweiterungen wie neue Schnittstellen oder Exporttypen ohne grundlegende Änderungen erlaubt.
US-23	Systemanforderungen	Als Nutzende möchte ich klare, verständliche und hilfreiche Fehlermeldungen erhalten, um Probleme selbstständig beheben zu können.

A.2 Rest-Api Swagger UI

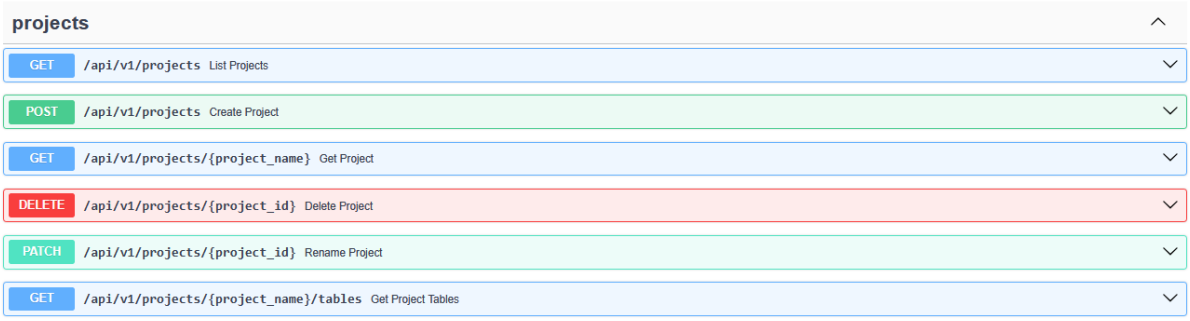

Abb. A.1

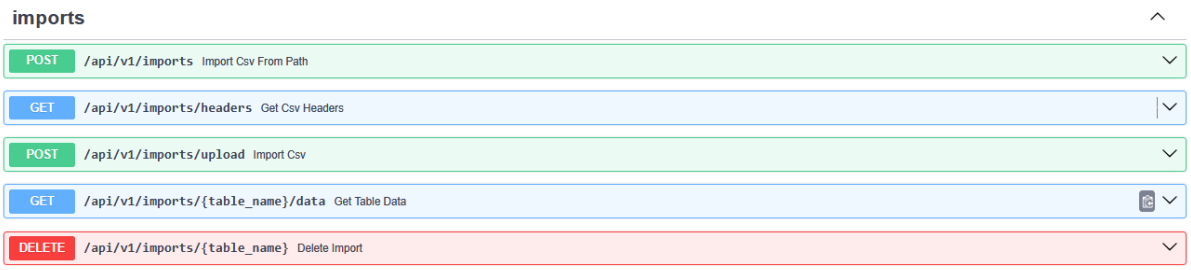

Abb. A.2

Abb. A.3

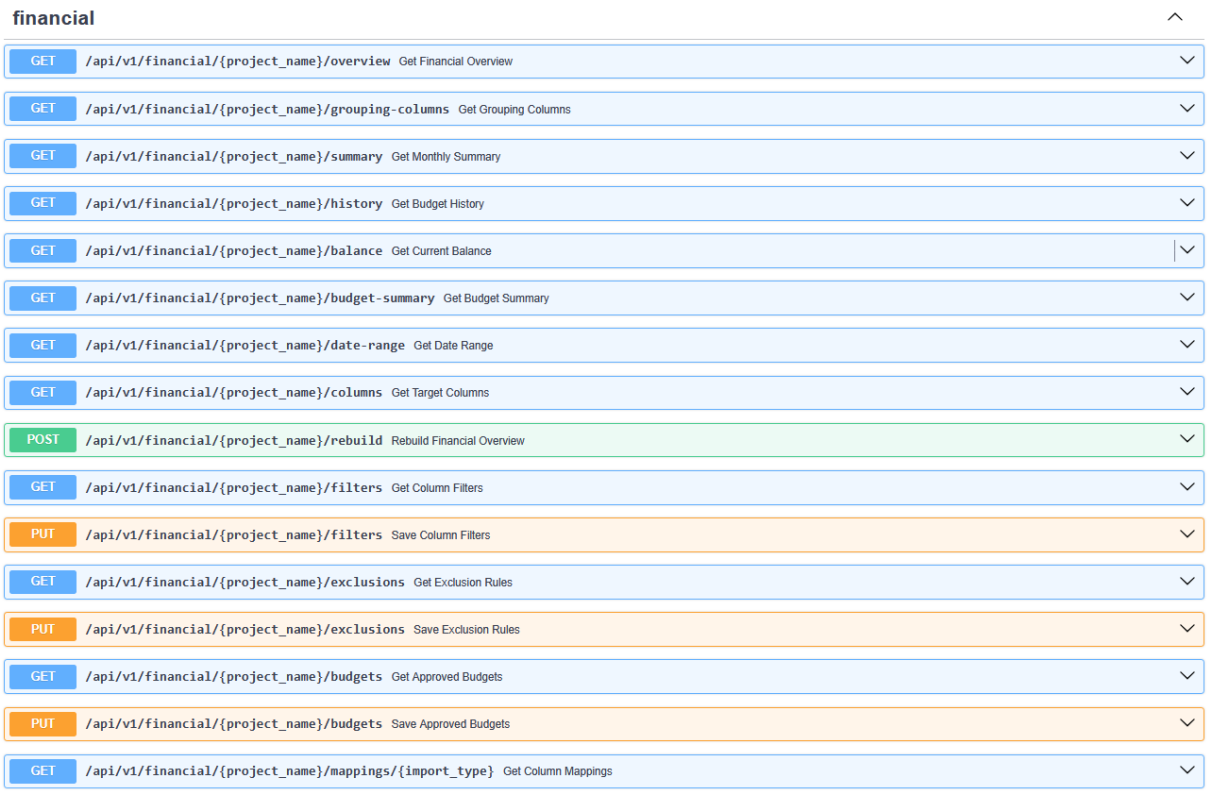
financial
GET /api/v1/financial/{project_name}/overview Get Financial Overview
GET /api/v1/financial/{project_name}/grouping-columns Get Grouping Columns
GET /api/v1/financial/{project_name}/summary Get Monthly Summary
GET /api/v1/financial/{project_name}/history Get Budget History
GET /api/v1/financial/{project_name}/balance Get Current Balance
GET /api/v1/financial/{project_name}/budget-summary Get Budget Summary
GET /api/v1/financial/{project_name}/date-range Get Date Range
GET /api/v1/financial/{project_name}/columns Get Target Columns
POST /api/v1/financial/{project_name}/rebuild Rebuild Financial Overview
GET /api/v1/financial/{project_name}/filters Get Column Filters
PUT /api/v1/financial/{project_name}/filters Save Column Filters
GET /api/v1/financial/{project_name}/exclusions Get Exclusion Rules
PUT /api/v1/financial/{project_name}/exclusions Save Exclusion Rules
GET /api/v1/financial/{project_name}/budgets Get Approved Budgets
PUT /api/v1/financial/{project_name}/budgets Save Approved Budgets
GET /api/v1/financial/{project_name}/mappings/{import_type} Get Column Mappings

Abb. A.4

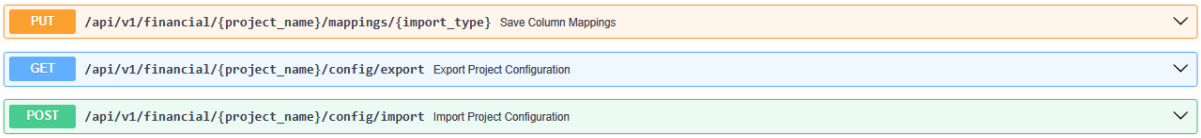
PUT /api/v1/financial/{project_name}/mappings/{import_type} Save Column Mappings
GET /api/v1/financial/{project_name}/config/export Export Project Configuration
POST /api/v1/financial/{project_name}/config/import Import Project Configuration

Abb. A.5

A.3 Datenbankschema

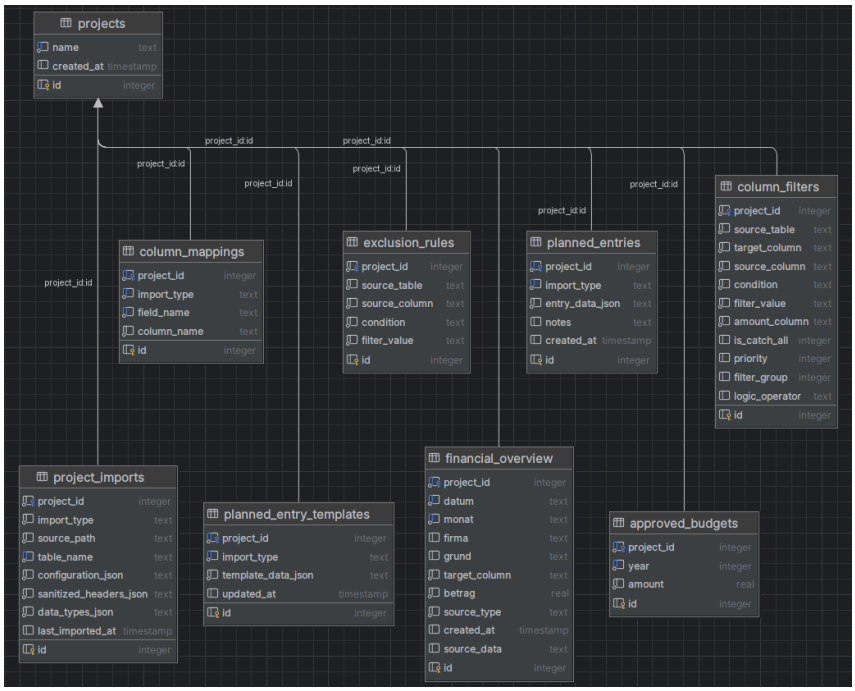

Abb. A.6

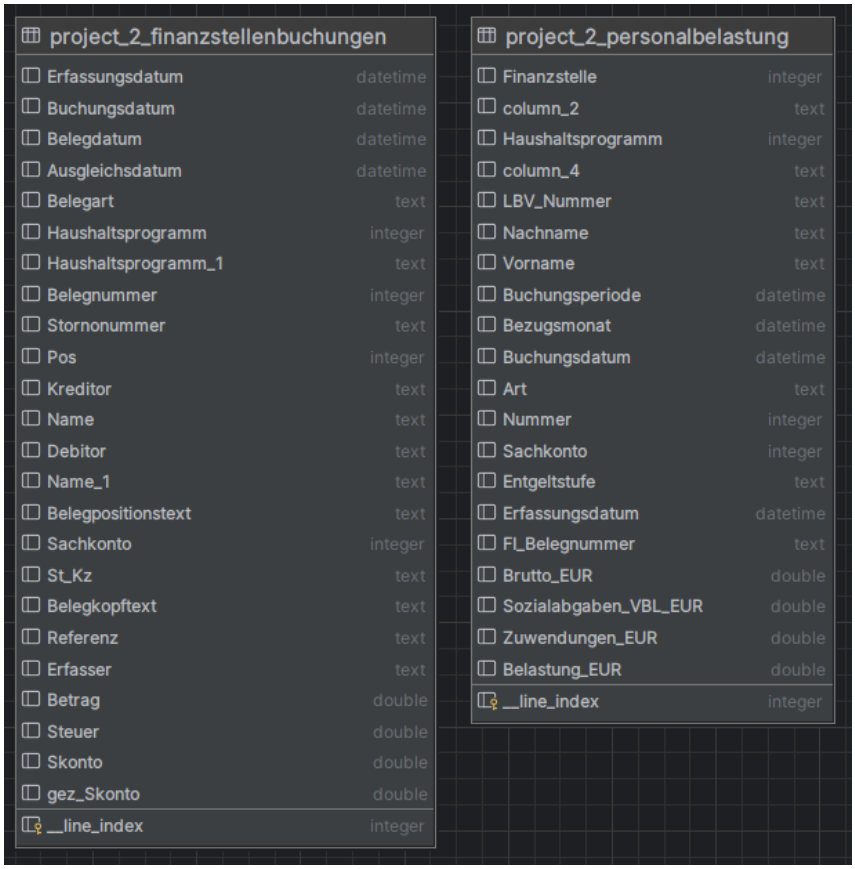
project_2_finanzstellenbuchungen
Erfassungsdatum datetime
Buchungsdatum datetime
Belegdatum datetime
Ausgleichsdatum datetime
Belegart text
Haushaltsprogramm integer
Haushaltsprogramm_1 text
Belegnummer integer
Stornonummer text
Pos integer
Kreditor text
Name text
Debitor text
Name_1 text
Belegpositionstext text
Sachkonto integer
St_Kz text
Belegkopftext text
Referenz text
Erfasser text
Betrag double
Steuer double
Skonto double
gez_Skonto double
__line_index integer
project_2_personalbelastung
Finanzstelle integer
column_2 text
Haushaltsprogramm integer
column_4 text
LBV_Nummer text
Nachname text
Vorname text
Buchungsperiode datetime
Bezugsmonat datetime
Buchungsdatum datetime
Art text
Nummer integer
Sachkonto integer
Entgeltstufe text
Erfassungsdatum datetime
FI_Belegnummer text
Brutto_EUR double
Sozialabgaben_VBL_EUR double
Zuwendungen_EUR double
Belastung_EUR double
__line_index integer

Abb. A.7