Universität Osnabrück

Bachelorarbeit

Service orientierte Mailintegration für eine generalisierte Projektverwaltungssoftware

Wilko Müller

Matrikelnummer: 954424

Institut für Informatik

Arbeitsgruppe Software Engineering

Erstgutachterin: Prof. Dr.-Ing. Elke Pulvermüller

Zweitgutachter: Dennis Ziegenhagen, M. Sc.

Abstract

Im Rahmen dieser Bachelorarbeit wird ein Softwarekonzept entwickelt, das eine übersichtliche Archivierung von E-Mails erleichtert.

Konzipiert für die Generalisierte Projektverwaltung der Bohnenkamp-Stiftung bindet das Konzept erstmals den dort verwendeten *Mail-Client* ein.

Ziel ist es, die uneinheitliche Abspeicherung der Mails zu beenden. Eingeführt wird ein einheitlicher Umgang mit dem E-Mailverkehr und eine projektbezogene, zentrale Speicherung.

Drei Komponenten ermöglichen die Anpassung und Erweiterung der Mailintegration: Ein *Add-In* für den *Mail-Client*, ein *projektorientierter Mailbetrachter* und ein *Web-Service*, der die Verbindung zu der Projektverwaltungssoftware herstellt.

Mit der neuen Lösung soll jeder Mitarbeiter alle einem Förderprojekt zugewiesenen Mails und deren Anhänge aufrufen können.

As part of this bachelor thesis a software concept is developed that implements a straightforward filing of e-mails.

The concept is designed to tether the generalized project management software to the mail client used in the Bohnenkamp-Stiftung.

The objective is to stop the inconsistent storing of mails. Therefore, a uniformed approach is introduced that handles the e-mail exchange in accord with a project-driven central filing.

Three components facilitate the modification and extension of the mail integration: An *Add-In* for the *mail client*, a *project orientated e-mail viewer* and a *web service* establishing a connection to the project management software.

With the new solution every employee should be able to access all the mails that are assigned to a foundation project alongside their attachments.

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit widmet sich dem Problem der fehlenden Archivierung von E-Mails und ihren Anhängen in einer projektbezogenen Verwaltungsumgebung.
Um dies zu beheben, soll der im Projekt anfallende E-Mailverkehr in eine vorhandene Software zur Projektverwaltung integriert werden.
Die Bohnenkamp-Stiftung, in der das zu entwickelnde Konzept angewandt werden soll, verwendet bisher die von Nico Kersting entwickelte Generalisierte Projektverwaltung [Kers16]. Diese dient als zentraler Datenspeicher, in dem projektbezogene Akten abgelegt werden.

Im Weiteren wird das in der Stiftung bestehende Softwaresystem analysiert, mögliche Schnittstellen für die Integration des E-Mailverkehrs aufgezeigt und eine Lösung des Problems entwickelt.

1.1 Ausgangssituation

Seit Dezember 2008 finanziert die gemeinnützige Friedel & Gisela Bohnenkamp-Stiftung Bildungsprojekte im Osnabrücker Raum, die sich zur Aufgabe setzen, Kinder und Jugendliche in ihren Potentialen zu stärken. Möglich wird die Stiftungsarbeit durch Aktienausschüttungen. Die Hälfte der Aktien der Bohnenkamp AG sind in die Stiftung eingebracht worden. Die Bohnenkamp AG ist ein führendes Handelsunternehmen für Spezialreifen, vornehmlich aus dem Landwirtschaftsbereich. Seit 2014 hat die Stiftung ihren Sitz in einer restaurierten Villa, die früher der Osnabrücker Maler Franz Hecker bewohnte.

Die Bearbeitung von eingehenden E-Mails ist für die Mitarbeiter der Bohnenkamp-Stiftung zeitintensiv. Häufig werden zur Beantwortung einer E-Mail neben den Daten aus der Projektverwaltung auch Informationen aus vorangegangenen Mails und deren Anhänge benötigt. Da Förderungen über mehrere Jahre andauern können und die Bohnenkamp-Stiftung viele Projekte unterstützt, ist es für die Mitarbeiter schwierig, den Überblick über alle laufenden Vorhaben im Kopf zu behalten. Deshalb muss der Mailverlauf eines Projekts häufig durchsucht werden. Dabei stellt sich als Problematik heraus, dass der Mail-Client nur die Möglichkeit bietet, Nachrichten nach Absender oder Datum zu sortieren, nicht aber nach dem zugehörigen Projekt. Die Beantwortung des E-Mailverkehrs kostet in einigen Fällen viel Zeit, wodurch sich die Antwortzeiten für die Ansprechpartner der Förderprojekte verlängern und die Mitarbeiter weniger Zeit für andere Arbeitstätigkeiten haben.

1.2 Anforderungen an die Mailintegration

Die Entlastung der Stiftungsmitarbeiter von der zeitintensiven Bearbeitung der eingehenden E-Mails ist zentrale Motivation für die vorliegende Arbeit. Folgende Anforderungen soll die zu entwickelnde Generalisierte Mailintegration erfüllen:

1.2.1 Integration in die Generalisierte Projektverwaltung

Um einen kompletten Mailverlauf eines Stiftungs-Projekts zu erhalten, ist bisher manuelles Suchen im E-Mailpostfach erforderlich. Da dort auch der nicht-projektbezogene Mailverkehr gespeichert ist und viele Projekte mehrjährig angelegt sind, kann das manuelle Suchen einen Vorgang mit erheblichem Zeitaufwand darstellen. Erschwerend kommt hinzu, dass sich Ansprechpartner sowie E-Mailadressen ändern können.

Generelle Informationen über Projekte beziehen die Mitarbeiter über die Generalisierte Projektverwaltung. Die Anforderungen der Reduzierung des Arbeitsaufwandes und der Übersichtlichkeit des Mailverkehrs können durch die Mailintegration in die Generalisierte Projektverwaltung erfüllt werden.

1.2.2 Unkompliziertes Einpflegen von Mails

Jede ankommende Mail mit Projektbezug muss in die Projektverwaltung eingepflegt werden. Um den Arbeitsaufwand zu minimieren, ist eine einfache Eingabe zu entwerfen.

1.2.3 Mehrbenutzerbetrieb

Die Generalisierte Projektverwaltung unterstützt eine gleichzeitige Nutzung des Systems von mehreren Mitarbeitern. Wenn auch die Mailintegration nach diesem System entwickelt wird, können die Mitarbeiter bequem von ihrem eigenen Rechner alle Vorgänge, Datenzugriffe und Speicherungen durchführen.

1.2.4 Mail-Anhänge

Die im Mail-Programm befindlichen Anhänge einzelner E-Mails enthalten unter Umständen wichtige Unterlagen mit Projektbezug. Diese sollten nicht schwer auffindbar sein oder verloren gehen. Um nicht wie bisher langwierig suchen zu müssen, ist eine Speicherung der Anhänge in der jeweiligen Akte der Generalisierten Projektverwaltung sinnvoll.

1.2.5 Erweiterbarkeit und Wiederverwendbarkeit der Software

Die Mailintegration soll möglichst einfach an andere Einsatzszenarien anpassbar sein. So muss es beispielsweise möglich sein, die Mailintegration mit einer Vielzahl von

Mail-Clients nutzen zu können. Zudem darf eine Änderung an der bestehenden Projektverwaltung nicht dazu führen, dass umfangreiche Modifikationen an der Mailintegration durchgeführt werden müssen. Stattdessen sollen nur kleinere Anpassungen notwendig sein.

1.3 Aufbau der Arbeit

In Kapitel 2 werden die grundlegenden Techniken erläutert, die von der Generalisierten Mailintegration genutzt werden. Dazu gehören Webservices, Customer-Relationship-Management-Systeme und das Prinzip einer serviceorientierten Architektur. Zudem werden verwandte Arbeiten und Systeme vorgestellt.

Kapitel 3 erläutert die Struktur der Generalisierten Mailintegration und die dahinterstehenden Ideen.

Im vierten Kapitel wird detailliert dargestellt, wie die einzelnen Komponenten der Software aufgebaut sind und erstellt wurden.

Kapitel 5 beschäftigt sich mit den Software-Testmethoden und dem Einsatz bei der Entwicklung der Generalisierten Mailintegration.

Abschließend fasst das sechste Kapitel den Inhalt der Arbeit zusammen und gibt einen Ausblick auf Erweiterungen, die in Zukunft erstellt werden können.

2 Grundlagen

Dieses Kapitel behandelt die grundlegenden Techniken und Konzepte, die für die Umsetzung der Mailintegration benötigt werden. Zudem werden ähnliche Arbeiten vorgestellt und auf ihre Anwendbarkeit im Kontext der Bohnenkamp-Stiftung untersucht.

2.1 Customer-Relationship-Management

Unter dem Begriff des Customer-Relationship-Managements (CRM) wird nach Helmke, Uebel und Dangelmaier [HeUD13; S. 7f] eine ganzheitliche Betrachtung der Beziehung eines Unternehmens zu seinen Kunden verstanden. Da der Begriff CRM aus der Betriebswirtschaft kommt und die in dieser Arbeit vorgestellte Software sich an Stiftungen richtet, müssen die Begriffe Kunde und Unternehmen etwas uminterpretiert werden. Die Stiftung lässt sich im CRM-Kontext am ehesten als Unternehmen ansehen, da sie eine Art von Dienstleistung in Form einer Förderung anbietet. Als Kunden lassen sich die Förderantragssteller betrachten, weil sie die angebotene Dienstleistung in Anspruch nehmen.

In der Literatur wird CRM in zwei Teile aufgespalten [LeHW11]. Ein Teil besteht in der Ausrichtung der Unternehmensstrategie auf ganzheitliche Betrachtung der Kundenbeziehung, der andere Teil setzt die Strategie technisch durch. In dieser Arbeit wird der zweite Aspekt, die technische Umsetzung einer Software, behandelt. Solch eine Software wird auch CRM-System genannt. Die hier vorgestellte Mailintegration ermöglicht den Mitarbeitern einer Stiftung die E-Mail-Kommunikation mit den Antragsstellern vollständig zu überblicken und zu archivieren. Somit stellt sie einen Teil eines CRM-Systems dar.

2.2 Serviceorientierte Architektur

Die Technik der Serviceorientierten Architektur wird eingesetzt, um Software-Systeme zu entwerfen. Dabei wird der Fokus auf die Entwicklung von Software-Komponenten gelegt, die über leicht zugängliche Schnittstellen kommunizieren. Zudem wird vorausgesetzt, dass die Kommunikation über eine vereinheitlichte Datenstruktur erfolgt [ChHT04]. Der große Unterschied zur klassischen Anwendungsentwicklung liegt in der Aufteilung in Services. Normalerweise wird eine Anwendung geschrieben, um genau einen Anwendungsfall zu lösen. So kann es passieren, dass für einen komplexen Anwendungsfall eine umfangreiche Software entworfen wird, die diese Problemstellung hervorragend löst, allerdings schon bei einer leichten Änderung des Einsatzszenarios ohne große Änderungen nicht mehr zu gebrauchen ist.

Unter anderem um diesen hohen Arbeitsaufwand bei Umstrukturierungen zu vermeiden, wurde die Serviceorientierte Architektur entworfen. Unter Beachtung dieser würde die eben genannte umfangreiche Software in viele weniger umfangreiche Services aufgeteilt. Dies ist vergleichbar mit der modularisierten Bauweise einer Software. Allerdings liegt der entscheidende Unterschied in der Kommunikation zwischen den Services. So ist es nicht möglich, eine in Java geschriebene Klasse in eine *.NET*-Anwendung zu integrieren, da die unterschiedlichen Programmiersprachen dies ohne eine Übersetzung in die andere Sprache unmöglich machen.

Die Serviceorientierte Architektur setzt dagegen eine vereinheitlichte Datenstruktur wie XML oder JSON voraus, mit der die Daten zwischen den Services ausgetauscht werden. So ist weder das Betriebssystem noch die Programmiersprache relevant für eine Kommunikation zwischen den Services. Zudem kann theoretisch jeder Service auf einem über das Internet erreichbaren Computer ausgeführt werden und so seine Funktionalität weltweit anbieten. Der Datenaustausch in dieser Arbeit wird genauer in Kapitel 3.4 behandelt.

Ein weiterer Nutzen, der aus dieser Architektur folgt, ist die hohe Wiederverwertbarkeit der Services in anderen Anwendungen. Sie ermöglicht durch einen hohen Grad der Modularisierung die Weiternutzung vieler Services in einem anderen Softwareprojekt [ChHT04].

Weshalb sich diese Architektur auch für die Generalisierte Mailintegration anbietet, wird im Abschnitt 3.2 erläutert.

2.3 Sprachunabhängiger Datenaustausch

Wie bereits in Kapitel 2.2 erwähnt, ist für die Serviceorientierte Architektur eine vereinheitlichte Struktur zum Datenaustausch notwendig, damit die eventuell in diversen Programmiersprachen geschriebenen Services untereinander kommunizieren können. Für den Datenaustausch bieten sich zwei formal definierte Schreibweisen an: Extensible Markup Language (XML) und JavaScript Object Notation (JSON). Beide Schreibweisen stellen Daten strukturiert und maschinenlesbar dar und befinden sich in Textform. Allerdings gibt es einige Unterschiede zwischen den beiden Sprachen, die im Folgenden vorstellt werden:

XML ist eine Markup Language, das heißt sie basiert auf Tags, die den Inhalt strukturieren. Sie beruht auf SGML (Standard Generalized Markup Language) und legt im Gegensatz zu HTML, einer Weiterentwicklung von SGML, den Fokus auf Erweiterbarkeit des Vokabulars [Frie16; S. 22-23]. Somit sind in XML beliebige Tags möglich, die Elemente jeglicher Art beschreiben können. Zu beachten sind allerdings grundlegende Regeln bei der Erstellung, da nur korrekt formatierte XML-Dateien von Parsern akzeptiert werden [Frie16; S. 35]. Im folgenden Beispiel (*Listing 1*) ist eine vereinfachte E-Mail in XML dargestellt.

```
<?xml version="1.0" encoding="UTF-8"?>
<Email>
  <CC-Empfänger>
    <Empfänger>
      <Name>Thomas Mueller</Name>
      <Emailadresse>thomas.mueller@example.com</Emailadresse>
    </Empfänger>
    <Empfänger>
      <Name>Anna Schmidt</Name>
      <Emailadresse>anna.schmidt@example.com</Emailadresse>
    </Empfänger>
  </CC-Empfänger>
  <Absender>Max.Mustermann@example.com</Absender>
  <Betreff>Liebe Grüße</Betreff>
  <Inhalt>Hallo, Viele Grüße Dein Max</Inhalt>
</Email>
```

Listing 1: XML-Beispiel

In dem obenstehenden Beispiel *(Listing 1)* wurden Tags wie <EMAIL>, <Absender> etc. kreiert. Die Struktur des Dokuments ist sowohl von Menschen lesbar als auch von einem Parser interpretierbar, falls dieser den definierten Tags eine Semantik zuweisen kann.

Die hier vorgestellte alternative Schreibweise, um Daten plattformunabhängig darzustellen, ist JSON. Im Mittelpunkt dieser Notation steht das Objekt, welches durch Listen aus Attributen erstellt wird. Ein Objekt wird immer durch ein die zugehörigen Attribute umschließendes geschweiftes Klammerpaar dargestellt. Attribute sind, wie bei XML, frei definierbar, werden allerdings nicht durch Tags markiert, sondern mittels sogenannter „Name-Value“ Paare. Damit sind in JSON Paarungen gemeint, die nach folgendem Schema aufgebaut sind:

```
“Name des Attributs“ : “Wert“
```

Folgende Darstellung (*Listing 2*) beschreibt dasselbe Objekt aus vorherigem Beispiel in JSON:

```
{
    "CC-Empfänger": [{
       "Name": "Thomas Mueller",
       "EmailAddresse": "thomas.mueller@example.com"
     },
     {
       "Name": "Anna Schmidt",
       "EmailAddresse": "anna.schmidt@example.com"
     }
    ],
    "Absender": "Max.Mustermann@example.com",
    "Betreff": "Liebe Grüße",
    "Inhalt": "Hallo, Viele Grüße Dein Max"
}
```

Listing 2: JSON-Beispiel

Auf den ersten Blick fällt in dem Beispiel auf, dass JSON deutlich weniger Zeichen benötigt als XML. Dies hängt allerdings nicht nur von den fehlenden schließenden Tags ab, sondern auch von der Möglichkeit, Arrays nativ in JSON darstellen zu können. Ein Array besteht nur aus einer Auflistung von Werten oder Objekten, ohne Namen, die von einer geschweiften Klammer umschlossen werden. Ein Nachteil von JSON gegenüber XML ist die fehlende Möglichkeit Objekten Namen zu geben, weshalb das „Email“-Tag von XML nicht übernommen werden kann.

Da die JSON-Schreibweise übersichtlicher und zudem in vielen modernen Webanwendungen wie Amazon Alexa[1] und APIs wie der Google Maps API[2] zum Einsatz kommt, wird diese auch in der Mailintegration eingesetzt.

[1] https://developer.amazon.com/docs/custom-skills/request-and-response-json-reference.html
[2] https://developers.google.com/maps/documentation/geocoding/intro

2.4 VSTO

Für die Erstellung der Mail-Client-Erweiterung der Bohnenkamp-Stiftung wurde das VSTO SDK[3] in Version 4.0 eingesetzt. Es ermöglicht die Entwicklung von Microsoft Outlook-Add-Ins für alle Outlook Versionen ab 2007. Der Quellcode kann in jeder Sprache, die von der CLR (Common Language Runtime) unterstützt wird, geschrieben werden. Als Alternative existieren für Outlook 2016 Web-basierte Add-Ins[4], die in JavaScript und HTML geschrieben werden. Im Gegensatz zu VSTO-Add-Ins, bei denen der Programmcode auf der ausführenden Maschine gelagert wird, laden Web-Add-Ins den Quelltext zur Laufzeit aus einer Webressource herunter, wenn das Add-In aufgerufen wird.

Das VSTO SDK wurde gewählt, da einerseits von der Bohnenkamp-Stiftung der Wunsch geäußert wurde, dass das Programm nicht mit dem Internet verbunden sein soll und andererseits VSTO mehr Möglichkeiten bietet, auf das ausführende System zuzugreifen, was für die Verarbeitung von E-Mail-Anhängen von Vorteil ist.

2.5 Verwandte Arbeiten

Da die Generalisierte Projektverwaltung, auf der diese Arbeit aufbaut, bereits den Projektverwaltungs-Teil des Customer-Relationship-Management Systems abdeckt, wird hier vor allem die Integration von Daten aus dem E-Mail-Verkehr in ein Customer-Relationship-Management (CRM) System behandelt.

Kommerzielle Anwendungen

Unter den kommerziellen CRM-Systemen sind die Anbieter Salesforce und Microsoft branchenführend. Die Firma Salesforce bietet im Bereich der E-Mail-Integration das Programm „Sales Cloud E-Mail-Integration“[5] an. Es arbeitet ähnlich wie die hier vorgestellte Mailintegration mit einem Add-In für den Mail-Client Outlook. Das Add-In transferiert den E-Mailverkehr zwischen Firma und Kunden in eine Cloud-basierte CRM-Anwendung. Die Software wird nach dem Grundsatz „Software as a Service“ (abgekürzt SaaS) vertrieben. Das bedeutet, dass der Nutzer die Software nicht besitzt, sondern diese nur gegen Zahlung einer Gebühr für eine bestimmte Zeit verwenden darf. Zudem benötigt der Nutzer nur eine Client-Oberfläche, da das Programm auf Servern des Softwareanbieters in der „Cloud“ läuft [ZaAy13; S. 50].

[3] http://go.microsoft.com/fwlink/?LinkId=140384
[4] https://docs.microsoft.com/de-de/outlook/add-ins/
[5] https://www.salesforce.com/de/products/sales-cloud/features/email-tracking-software/

In Gesprächen mit den Angestellten der Bohnenkamp-Stiftung wurde allerdings klar, dass eine Projektverwaltung beziehungsweise Mailintegration mit dem SaaS-Vertriebsmodell oder einer Cloud-Anbindung nicht in Frage kommt. Dies wurde mit Datenschutzbedenken hinsichtlich Cloud-basierter Anwendungen begründet.

Eine CRM-Software, die speziell auf die Bedürfnisse von Stiftungen zugeschnitten ist, bietet die CAS Software AG an. Das Produkt heißt CAS Maecenas[6] und ermöglicht die Adressverwaltung mit vollständiger Kommunikationshistorie inklusive Telefon und E-Mail. Zudem können Förderprojekte evaluiert und Förderanträge verwaltet werden. Die Anwendung wurde probeweise in der Bohnenkamp-Stiftung eingesetzt. Gegen die Nutzung des Programms sprach, dass ein spezieller Mail-Client des Softwareanbieters genutzt werden muss. Im Gegensatz zum bisher verwendeten Microsoft Outlook ist das Erstellen von Unterordnern im Mail-Postfach mit diesem nicht möglich. Die notwendige Schulung der Stiftungsangestellten zur Nutzung des neuen Mailprogramms war daher nicht lohnend. Damit die Mitarbeiter ihren gewohnten Mail-Client weiterhin benutzen können, unterstützt die Mailintegration eine Vielzahl von Mailprogrammen.

Weitere kommerzielle Anwendungen zur Stiftungsverwaltung sind FoundationPlus[7] von Zetcom und die Anwendung syprof[8] der Firma Systemgruppe. FoundationPlus basiert auf SaaS und eignet sich deshalb nicht für den Einsatz in der Bohnenkamp-Stiftung. Die Software syprof wird aufgrund des Cloud-basierten Ansatzes nicht verwendet.

Wissenschaftliche Arbeiten

In dem Buch „Pro SharePoint Solution Development“ von Ed Hild und Susie Adams [HiAd07] wird die Entwicklung einer Softwarelösung beschrieben, die mit Hilfe eines Outlook-Add-Ins E-Mails und Anhänge an einen Server sendet. Die dort vorgestellte Software ist für den Einsatz in Unternehmen ausgelegt. Das Programm arbeitet auf einem Microsoft SharePoint Repository, das als zentraler Datenspeicher dient und allen Mitarbeitern zur Verfügung steht. Nutzer der Software können mit der vom Outlook-Add-In angepassten Oberfläche die empfangenen Mails mit den Anhängen in eine Ordnerstruktur im Repository einfügen. Der SharePoint Server ist somit der zentrale Speicherort für Informationen aus E-Mail-Konversationen aller Mitarbeiter. Allerdings ist die vorgestellte Lösung sehr auf Microsoft SharePoint ausgelegt und es gibt keine Integrationsmöglichkeit für die Generalisierte Projektverwaltung.

Ein Artikel von Iqbal, Shah, James und Cichowicz [ISJC13] behandelt die Integration von mehreren Unternehmensanwendungen. Dort wird erläutert, wie durch eine die Ser-

[6] https://www.cas-communities.de/loesungen/cas-maecenas.html
[7] https://www.zetcom.com/foundationplus_de/
[8] http://www.systemgruppe.de/syprof-stiftungsverwaltung-fundraising-crm.htm

viceorientierte Architektur verwendende Anwendung sechs voneinander isoliert arbeitende Anwendungen integriert werden können. Jedoch wird dort - im Gegensatz zu dieser Arbeit - kein Mail-Client in eine Projektverwaltung integriert.

3 Konzept

In diesem Kapitel wird eine Software-Architektur vorgestellt, die von der Mailintegration umgesetzt werden soll. Die dargestellte Architektur unterstützt die Erreichung der in Abschnitt 1.2 beschriebenen Anforderungen. Zunächst wird die bestehende Architektur der Generalisierten Projektverwaltung auf mögliche Schnittstellen zur Mailintegration untersucht.

3.1 Integration in die Generalisierte Projektverwaltung

Die Integration einer neuen Software in eine bereits bestehende Software-Architektur birgt einige Herausforderungen. Schon in der Konzeptionsphase ist eine Auseinandersetzung mit der Struktur der bestehenden Software nötig, damit aus dem neuen und alten Programm eine funktionierende Einheit entsteht. Um der Lösung einen Schritt näher zu kommen, wird die Generalisierte Projektverwaltung auf mögliche Schnittstellen zum Datenaustausch zwischen Mailintegration und Projektverwaltung untersucht. Am geeignetsten wäre eine gemeinsame Basis, auf der die Projektverwaltung und Mailintegration gleichermaßen arbeiten, da so Schnittstellen eingespart werden. Dabei muss darauf geachtet werden, dass die beiden Programme nicht konkurrierend auf den Daten arbeiten, weil in diesem Fall das eine Programm gemeinsam genutzte Ressourcen dem anderen Programm unzugänglich machen würde, wie es durch einen Schreibschutz oder Ähnliches passieren könnte. Wäre dies der Fall, müsste eine Synchronisation zwischen den beiden Teilen durchgeführt werden, wodurch Mehraufwand bei der Implementierung entstehen würde. Zudem wird durch die Integration die bereits erstellte Architektur der Projektverwaltung sinnvoll erweitert.

Als Aufgabe stellt sich die Suche nach einer passenden Schnittstelle in der Projektverwaltung und die Abkapselung der Software. Die benötigte Funktionalität komplett in die Projektverwaltung zu implementieren, ist somit keine Alternative. Die Architektur der Generalisierten Projektverwaltung wird in *Abbildung 1* dargestellt.

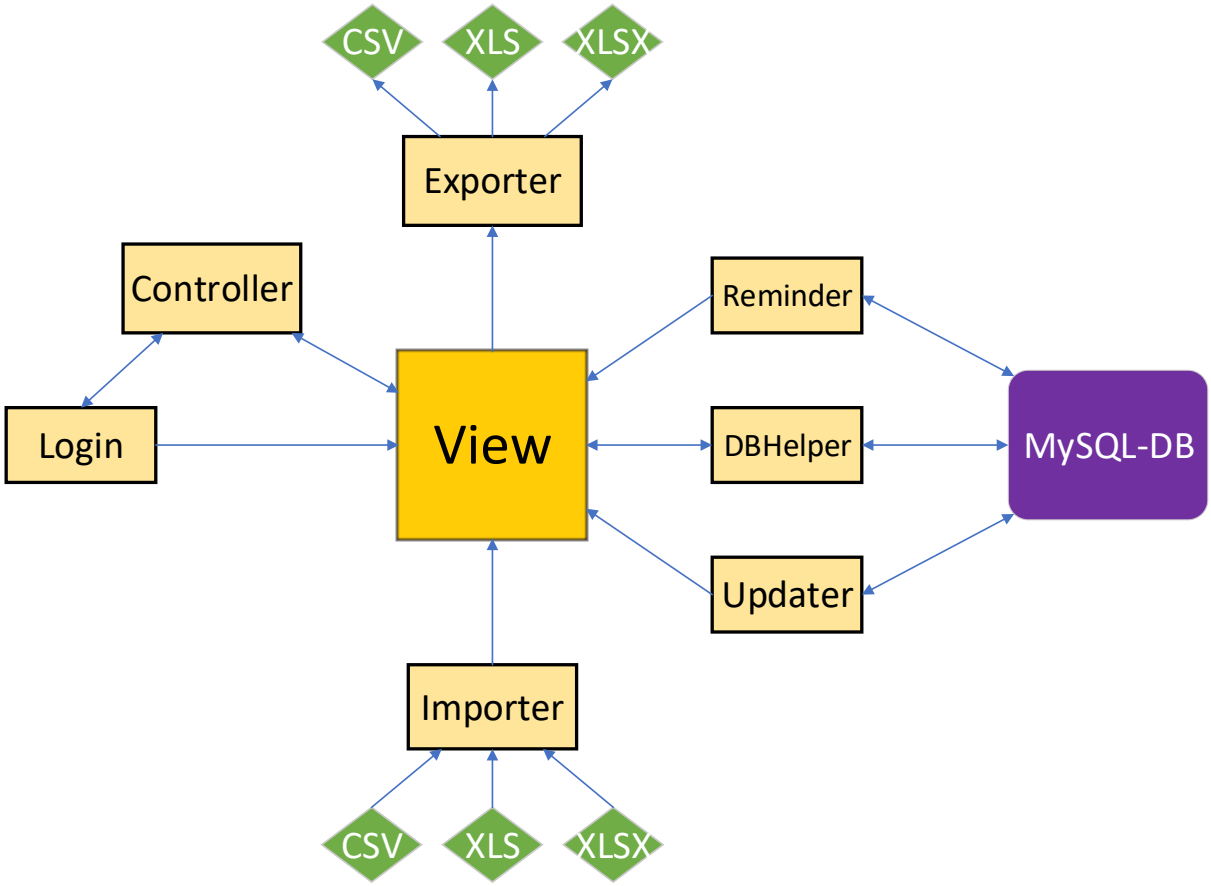

Abbildung 1: Architektur der generalisierten Projektverwaltung [Kers16; S.18]

Die Projektverwaltung ist entlang des Model-View-Controller (MVC) Entwurfsmusters gestaltet. In der obenstehenden Grafik werden Klassen durch Rechtecke, Dateiformate durch Rauten und Datenbanken durch abgerundete Rechtecke dargestellt. An den folgenden Schnittstellen ist die Mailintegration möglich *(siehe Abbildung 2)*:

- Die ersichtlichste Schnittstelle ist die Einbindung der Mailintegration direkt in den Code. Dazu müssten große Änderungen an allen Modulen der Generalisierten Projektverwaltung durchgeführt werden. Ein Vorteil dieses Ansatzes ist, dass der Quellcode sich in nur einer Anwendung befindet und im Vergleich zu einem Framework leichter überblicken lässt. Allerdings müssen für jede funktionelle Änderung an der Mailintegration im Zweifelsfall die kompletten Klassen *View* und *Controller* aufgrund des MVC-Entwurfsmusters angepasst werden [Free16; S. 54ff].

- Ein alternativer Einstiegspunkt für die Einbindung wären die Import- und Exportklassen. Diese dienen der Sicherung, beziehungsweise dem Import großer Mengen von Projektdaten, die in CSV, XLS oder XLSX formatiert sind. Die

Klassen müssten so geändert werden, dass auch E-Mail-Daten durch sie verarbeitet und weitergeleitet werden könnten. Zudem ist die Datenbank so zu verändern, dass sie auch die E-Mail-Daten der Projekte aufnehmen kann.

- Da die Datenbank in jedem Fall angepasst werden muss, ist eine Anbindung der Mailintegration direkt an die Datenbank sinnvoll. Die Datenbank wurde mit dem MySQL-Datenbanksystem erstellt. Daraus ergibt sich die Möglichkeit, mit einer Vielzahl von Programmiersprachen auf diese zuzugreifen, da der zugehörige Client in alle gängigen Sprachen portiert wurde. Aus diesen Gründen bietet sich an, die Generalisierte Mailintegration über die Datenbank an die Projektverwaltung anzubinden.

3.2 Modularisierung

Einen hohen Stellenwert bei der Entwicklung der Software nehmen die in Abschnitt 1.2.5 aufgeführten Anforderungen Wiederverwendbarkeit und Erweiterbarkeit ein.

Um diese beiden Aspekte umzusetzen, wird die Mailintegration in drei große Module *(Abbildung 2)* aufgeteilt, die jeweils einen Bearbeitungsschritt im Prozess der Mailverarbeitung darstellen.

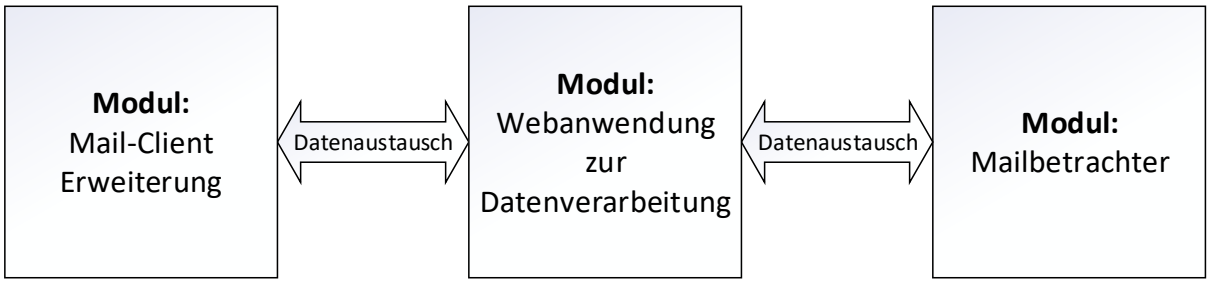

Abbildung 2: Modulübersicht der Mailintegration

Im Folgenden werden die drei Module vorgestellt.

3.3 Mail-Client-Erweiterung

Da die Nutzung der Generalisierten Mailintegration sich in die bestehenden Arbeitsabläufe der Bohnenkamp-Stiftung einfügen soll, ist eine Verarbeitung der Mails direkt am Entstehungsort sinnvoll. Deshalb ist eine Erweiterung des Mail-Clients zweckmäßig, die es den Nutzern erlaubt, empfangene und gesendete E-Mails in die Projektverwaltung zu integrieren. So wird das Öffnen eines zusätzlichen Programms, welches

diese Funktionalität ansonsten umsetzen würde, eingespart. Die Einsparung eines Arbeitsschritts ist bei der Integration der Mails in die Projektverwaltung von besonderer Bedeutung. Dieser Arbeitsschritt müsste sonst proportional zum Mailaufkommen der Institution durchgeführt werden und verursachte zusätzliche Arbeit.

Die Entwicklung der Erweiterung des Mail-Clients als abgekapseltes Modul bietet sich aus konzeptioneller und technischer Sichtweise an.

- Vom technischen Standpunkt gesehen werden Erweiterungen für gängige Mail-Clients in speziellen Frameworks geschrieben. Mozilla Thunderbird Erweiterungen [Mozi18] werden beispielsweise in XUL (einer auf XML basierten Sprache zur Beschreibung von Oberflächen) und JavaScript entwickelt[9]. Alternativ werden sie in einer eigenen Laufzeitumgebung ausgeführt, wie Microsoft Outlook-Add-Ins in VSTO [Whit05; Kapitel 10], die zur Erweiterung von Microsoft Office-Anwendungen eingesetzt werden.
- Aus konzeptioneller Sicht muss trotz der technischen Abgrenzung auf eine klar definierte Schnittstelle zwischen Erweiterung und der restlichen Anwendung geachtet werden. Diese Abgrenzung erlaubt es, die Mail-Client-Erweiterung auszutauschen. Eine neu entwickelte Erweiterung eines Mail-Clients braucht nur die definierte Schnittstelle zu implementieren, um mit der restlichen Mailintegration zusammenarbeiten zu können.

3.4 Webanwendung zur Mailverarbeitung

Grundlage für die Modularisierung der Mail-Client Erweiterung ist die im vorigen Abschnitt genannte Schnittstelle, über die der Kontakt zur restlichen Anwendung hergestellt wird. Diese Schnittstelle wird in der Mailintegration von einer Webanwendung bereitgestellt. Der Webserver, auf dem die Anwendung läuft, verarbeitet nicht nur Anfragen über die Schnittstellen, sondern konvertiert zudem noch die einkommenden Daten. Beispielsweise werden aus den von der Mail-Client-Erweiterung empfangenen Daten SQL-Abfragen kreiert. In der folgenden Abbildung *(Abbildung 3)* wird der Webserver zur Mailverarbeitung einschließlich Schnittstellen schematisch dargestellt. Die drei Module der Mailintegration sind genauso wie die Projektverwaltung als große hellgrüne Vierecke dargestellt. Die dunkelgrünen Kästen stellen interne Prozesse oder Speicher dar. Schnittstellen der Module und Anwendungen werden als kleinere angedockte Kästen abgebildet, zwischen denen Datenaustausch (hier als Pfeil präsentiert) geschieht.

[9] https://wiki.mozilla.org/Thunderbird/Add-ons_Guide_57

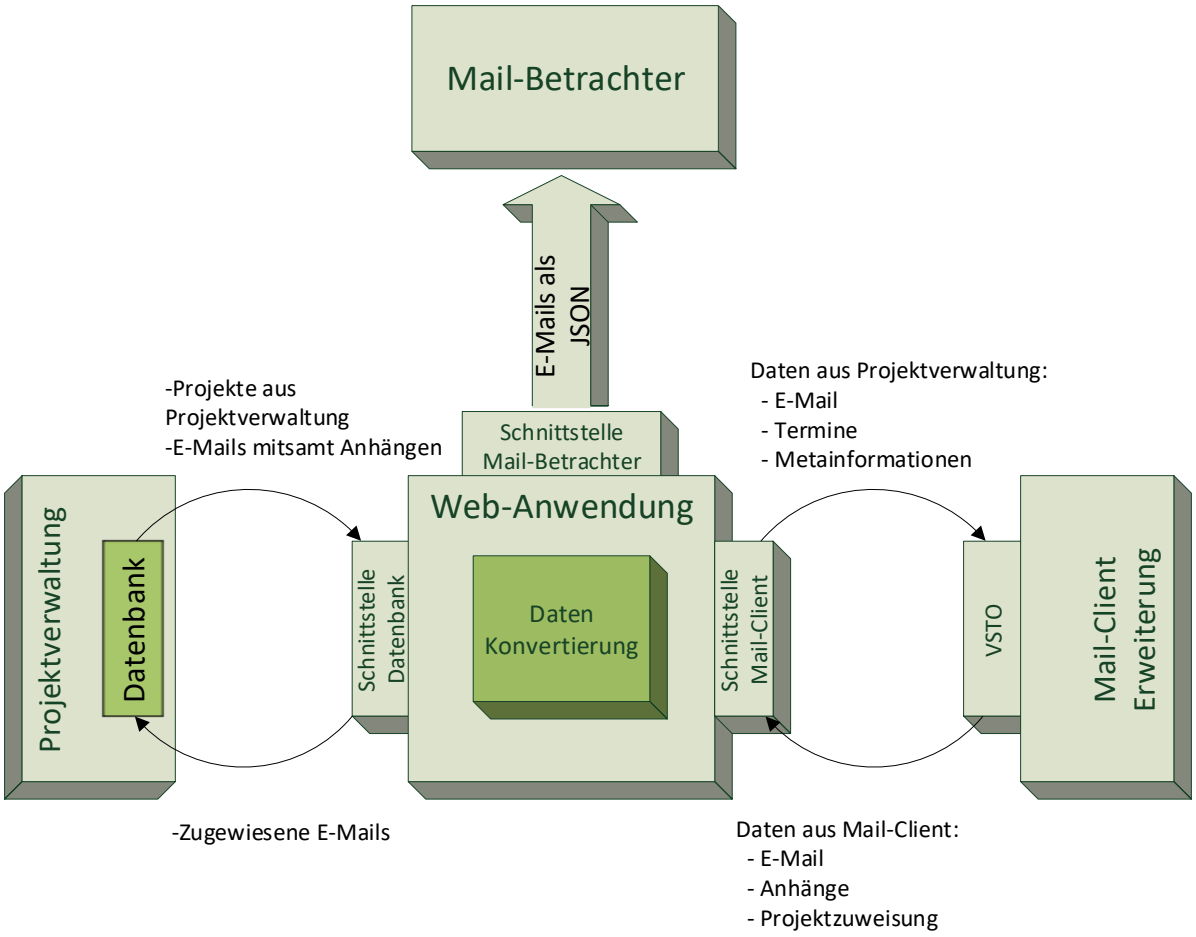

Abbildung 3: Übersicht der Web-Anwendung einschließlich Schnittstellen

Die Schnittstelle zwischen der Projektverwaltungsdatenbank und dem Webserver stellt die Create-, Read-, Update- und Delete-Funktionalität (CRUD) über SQL-Abfragen her. Über die Schnittstelle schickt der Webserver komplette E-Mails, aufgeteilt in mehrere SQL-Insert-Befehle (Create), an die passenden Tabellen. Zudem können die restlichen CRUD-Befehle ausgeführt werden. Das Modell der Datenbank wird genauer in den Abschnitten 3.6 und 4.2 erläutert. In die entgegengesetzte Richtung werden einerseits durch SQL-Select-Befehle (Read) Projekte aus der Projektverwaltung ausgelesen. Diese Liste von Projekten ist wichtig für die Zuordnung von E-Mails in der Mail-Client-Erweiterung. Andererseits können die von der Mailintegration gespeicherten Mails mit ihren Anhängen und anderen Metadaten aus der Datenbank wieder ausgelesen werden. Diese Funktionalität ist erforderlich für den im nächsten Abschnitt vorgestellten Mailbetrachter.

Die Schnittstelle zwischen Webserver und Mail-Client läuft über den Austausch von strukturierten Daten (JSON) mittels HTTP.

3.5 Projektorientierter Mailbetrachter

Der projektorientierte Mailbetrachter übernimmt im Rahmen der Generalisierten Mailintegration die Funktion der graphischen Benutzeroberfläche. Der Mailbetrachter soll die in der Mailintegration erfassten Daten wie E-Mails und Anhänge strukturiert darstellen. Der Hauptzweck der Anwendung ist die übersichtliche Ansicht der projektrelevanten E-Mails eines vom Nutzer ausgewählten Projektes. Dieser Überblick ist wichtig, da viele Förderprojekte einer Stiftung über mehrere Jahre laufen.

Es stellt sich die Frage, inwiefern der Mailbetrachter in die bereits bestehende Oberfläche der Projektverwaltung integriert werden soll und kann. Es stehen sich zwei Ansätze gegenüber:

- Der erste Ansatz würde die Betrachtungsmöglichkeit für E-Mails direkt in die Projektverwaltungsoberfläche integrieren. Der Mailbetrachter wäre somit ein weiteres Modul der in Python geschriebenen Projektverwaltung. Dieser Ansatz wird aus den folgenden Gründen nicht verfolgt: Zum einen wird das Prinzip der Wiederverwendbarkeit von Modulen missachtet, welches einen hohen Stellenwert in dieser Arbeit hat. Das hängt damit zusammen, dass ein in Python geschriebenes Modul ohne eine Webservice-ähnliche Schnittstelle nur schwierig in eine veränderte Softwareumgebung eingepasst werden kann. Zudem wurde die grafische Nutzeroberfläche der Projektverwaltung mit dem GTK 3 Toolkit entwickelt, welches einige Schwächen aufweist, auf die bereits in der Arbeit zur Generalisierten Projektverwaltung hingewiesen wurde [Kers16; S. 19]. Durch die fehlende Dokumentation zu dem verwendeten Toolkit gestaltet sich die Weiterentwicklung der Oberfläche schwierig.

- Der zweite Ansatz besteht darin, eine eigenständige Software zu implementieren, die sich als Ergänzung in die Oberfläche einfügt. Das bedeutet, der Mailbetrachter wird mit einem aktuelleren und besser dokumentierten Toolkit entwickelt und dockt an eine Schnittstelle des im vorigen Abschnitt behandelten Webservers an *(siehe Abbildung 3)*. Zudem wird die bestehende Projektverwaltungsoberfläche nur an einigen Stellen um Buttons erweitert, die beispielsweise das passende Projekt in der Mailbetrachtung aufrufen. Eine detailliertere Darstellung der Umsetzung ist im Anschnitt 4.4 zu finden.

3.6 Generalisierte Maildarstellung

Um das Konzept der Modularisierung erfolgreich durchzuführen, muss nicht nur die Notation, sondern auch die Struktur der zwischen den Services übertragenen Daten definiert werden. Der am häufigsten genutzte Datentyp ist die E-Mail. Für diese muss eine generalisierte Darstellung gefunden werden, damit die Services untereinander auf

einer gemeinsamen Basis agieren und kommunizieren können. Die generalisierte Darstellung soll nur die für die Mailintegration notwendigen Daten beinhalten, da diese von dem jeweiligen Mail-Client erfasst werden. Wenn nur das nötige Minimum an Mail-Daten vom Client erfasst werden muss, wird sichergestellt, dass kein Mail-Client durch optionale Features von der Interoperabilität mit der Mailintegration ausgeschlossen wird.

Folgende Struktur besitzt die generalisierte Maildarstellung (*Listing 3*) in JSON:

```
{
    "Attachments": [],
    "ConversationID": "",
    "MailBodyHTML": " ",
    "ProjectString": "",
    "Recipient": "",
    "RecipientMailAddress": "",
    "SHA256Hash": [],
    "SafeFileName": "",
    "SendDate": "",
    "SenderMailAddress": "",
    "SenderName": "",
    "Subject": ""
}
```

Listing 3: Maildarstellung in JSON

Die nicht selbsterklärenden Daten werden hier kurz vorgestellt: Mit der „ConversationID“ wird eine E-Mail-Konversation gekennzeichnet. Zu einer Konversation[10] gehören alle E-Mails mit gleichem Betreff, ohne Präfixe wie „RE:“ oder „FWD“. Der SHA265 Hashwert dient der Identifizierung einer E-Mail in der Mailintegration. Dieser wird vom Mail-Client aus dem Betreff, Absender, Inhalt und weiterem erstellt und ist ein Array von Integer Werten. Eine detaillierte Erklärung über die Konstruk-

[10] siehe https://docs.microsoft.com/en-us/dotnet/api/microsoft.office.interop.outlook._mailitem.conversationid?redirectedfrom=MSDN&view=outlook-pia#Microsoft_Office_Interop_Outlook__MailItem_ConversationID

tion steht in Abschnitt 4.1. Zudem wird ein Array von Anhängen unter dem Attributnamen „Attachments“ abgespeichert. Die Anhänge werden im folgenden Abschnitt behandelt.

3.7 Generalisierte Darstellung von Mail-Anhängen

Die Anhänge von E-Mails werden wie die E-Mails in ein generalisiertes Format gebracht. Die Umwandlung geschieht in der Mail-Client Erweiterung. Ein Anhang sieht wie folgt aus:

```
"Attachments": [{
     "AttachmentData": [],
     "DisplayName": "",
     "FileName": "",
     "SHA256Hash": []
     }]
```

Listing 4: Darstellung eines Mail-Anhangs in JSON

„AttachmentData“ ist ein Array, welches die Rohdaten des Anhangs als Folge von acht Bit Integer (0-255) beinhaltet. Zusätzlich dazu wird noch ein Name(DisplayName) des Anhangs abgespeichert und ein Windows-kompatibler Dateiname(FileName). Der Hashwert dient genau wie bei der E-Mail als Identifikator. Die Konstruktion wird genauer in Kapitel 4.1 erläutert.

4 Umsetzung

Dieses Kapitel beschreibt die Umsetzung der in Kapitel drei vorgestellten Architektur. Dabei wird auf die wesentlichen Aspekte zur Implementation der Generalisierten Mailintegration eingegangen.

4.1 Outlook-Add-In

Anpassung der Oberfläche

Um eine, wie in Abschnitt 1.2.2 beschriebene, einfache Bedienung der Mailintegration zu ermöglichen, wird die Oberfläche von Outlook angepasst. Die folgenden zwei Funktionen werden für diesen Zweck der Benutzeroberfläche hinzugefügt:

- Die erste Funktion weist eine E-Mail einem Projekt aus der Projektverwaltung zu.
- Die zweite Funktion ermöglicht die Rückgängigmachung einer zuvor getätigten Zuweisung.
- Zudem soll es möglich sein, E-Mails wieder aus der Mailverwaltung der Projektverwaltung zu entfernen.

Die beiden Operationen werden jeweils durch das Klicken eines Steuerelements ausgelöst. Die Buttons befinden sich im Kontextmenü (*Abbildung 4*), welches sich mittels Rechtsklick auf eine Mail oder Mailauswahl öffnet.

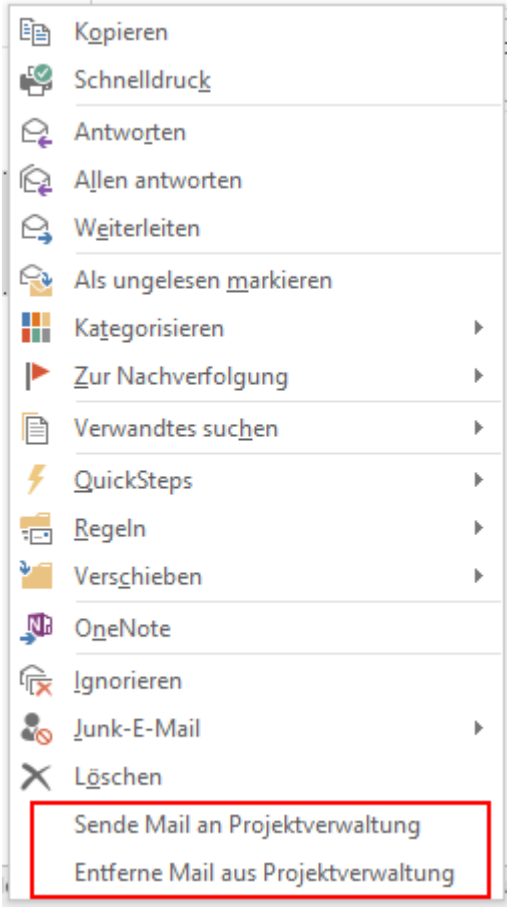

Abbildung 4: angepasstes Kontextmenü in Outlook

Damit sich das Kontextmenü erweitern lässt, muss ein „Ribbon“ (Erweiterung für das obere Menüband) erstellt werden. In dem Menü sollen das Löschen und Hinzufügen einer E-Mail jeweils als neuer Menüpunkt definiert werden. *Listing 5* zeigt den Ausschnitt aus der Menübanderweiterung, der die neue Funktionalität hinzufügt.

```
<contextMenus>
    <contextMenu idMso="ContextMenuMailItem">
      <button id="ContextMenuMailItemSend"
              label="Sende Mail an Projektverwaltung"
              onAction="ContextMenuMailItemSend_Click"/>
      <button id="ContextMenuMailItemDelete"
              label="Entferne Mail aus Projektverwaltung"
              onAction="ContextMenuMailItemDelete_Click"/>
    </contextMenu>
    <contextMenu idMso="ContextMenuMultipleItems">
      <button id="ContextMenuMultipleItems"
              label="Sende Mailauswahl an Projektverwaltung"
              onAction="ContextMenuMultipleItems_Click"/>
      <button id="ContextMenuMultipleItemsDelete"
              label="Entferne Mails aus Projektverwaltung"
              onAction="ContextMenuMailItemDelete_Click"/>
    <dynamicMenu id="MyDynamicMenu"
                 label= "Sende Auswahl an Projekt"
                 getContent="GetProjMenuContent" />
    </contextMenu>
  </contextMenus>
```

Listing 5: Kontextmenüerweiterung

Dort wird zwischen dem Kontextmenü unterschieden, welches beim Rechtsklick auf nur eine E-Mail geöffnet wird (mit dem Namen „*ContextMenuMailItem*“) und dem sich öffnenden Kontextmenü bei einer Auswahl von Mails „*ContextMenuMultipleItems*“. Beiden Menüs werden Buttons hinzugefügt, denen jeweils der passende *EventHandler* zugewiesen wird, der im zum *Ribbon* gehörigen C#-Quellcode definiert ist. Die weitere Verarbeitung der an die Projektverwaltung zu schickenden Mailauswahl wird im nächsten Abschnitt erläutert.

Ein Nachteil der Kontextmenüerweiterung ist, dass im oberen Menüband von Outlook ein neuer Reiter entsteht, auch wenn er nicht benötigt wird. In diesen ließe sich beispielsweise eine Anleitung zur Verwendung der Mailintegration einfügen.

Eine Alternative zur Erweiterung des Kontextmenüs ist die Anpassung des oberen Menübands. In diesem können Schaltflächen einfügt werden, die genau wie die Buttons im Kontextmenü anpassbare Funktionen ausführen. Zu diesem Zwecke werden zwei Knöpfe ins Menüband integriert, die Mails an die Projektverwaltung senden bzw. Mails daraus entfernen. Allerdings muss dann immer noch die entsprechende Mail im Posteingang ausgewählt werden. Die Kontextmenü-Aktionskette ist deutlich schneller durchzuführen als das Klicken auf die Schaltfläche zum Senden im Menübandeintrag. Aus diesem Grund ist die Kontextmenüerweiterung zu bevorzugen.

Projektauswahlfenster

Ist die Auswahl der an die Projektverwaltung zu sendenden Mails erfolgt und der „Sende an Projektverwaltung“-Button wurde gedrückt, findet eine mehrstufige Verarbeitung statt. Der EventHandler des „Entfernen/Hinzufügen“-Buttons einer E-Mail ruft im Controller des Add-Ins eine Methode auf, die mit der Verarbeitung beginnt. Im Folgenden wird der Ablauf für das Hinzufügen einer E-Mail zur Projektverwaltung geschildert.

Durch den Klick im Kontextmenü wird im Controller des Add-Ins eine Methode aufgerufen, die ein neues Fenster öffnet. Das Fenster ist mit dem *Windows Forms* Toolkit zur Erstellung grafischer Benutzeroberflächen entwickelt worden und dient dazu, die ausgewählte Mail einem Projekt aus der Projektverwaltung zuzuweisen. Diese Oberfläche wird Projektzuweisung genannt und im Quellcode durch die Klasse *ProjSelectForm* beschrieben. Das Projektzuweisungsfenster (*Abbildung 5*) sieht folgendermaßen aus:

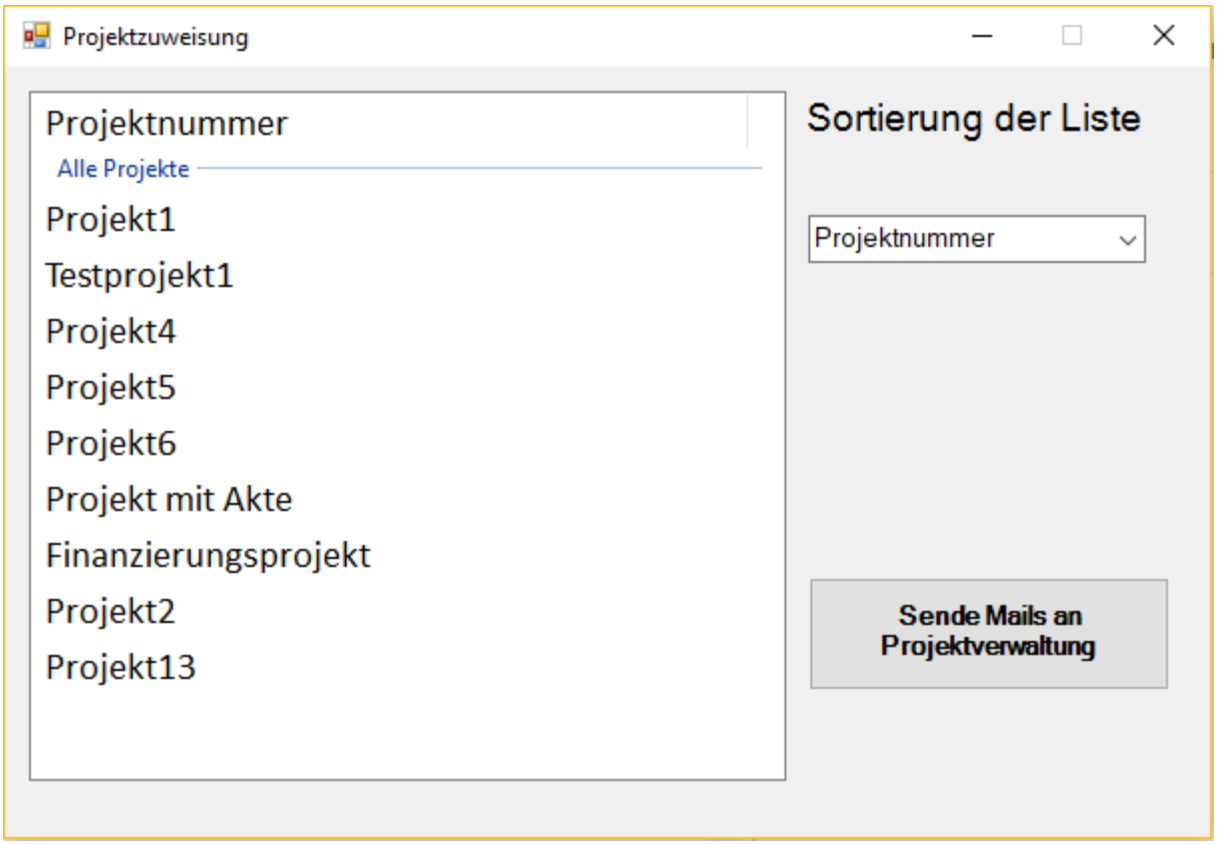

Abbildung 5: Projektzuweisungsfenster

In der Projektzuweisung werden alle Projekte an der linken Seite angezeigt, die in der Projektverwaltung existieren. Dazu müssen die Namen der Projekte aus der Projektverwaltungsdatenbank abgerufen werden. Beim Öffnen des Projektauswahlfensters im Controller des Outlook-Add-Ins wird die Methode *List<string> GetListOfProjects(string typeOfList)* aufgerufen. Mit Angabe der zur gewünschten Sortierung passenden Webserver-Adresse gibt sie eine Liste von Strings zurück. Die Strings in der Liste sind die nach dem angegebenen Kriterium sortierten Namen der Projekte.

Die Methode *GetListOfProjects* sieht wie folgt aus *(Listing 6)*:

```
List<string> IAddInController.GetListOfProjects(Uri typeOfListAddress)
    {
        string receivedJSON =
            (_view as IAddInView).GetProjects(typeOfListAddress);
        MemoryStream stream1 =
            new MemoryStream(Encoding.UTF8.GetBytes(receivedJSON))
        {
            Position = 0
        };
        DataContractJsonSerializer ser =
            new DataContractJsonSerializer(typeof(ListOfProjects));
        ListOfProjects listOfProjects =
            ser.ReadObject(stream1) as ListOfProjects;
        List<string> listOfProjectNames = new List<string>();
        foreach (var item in listOfProjects.ProjectData)
        {
            listOfProjectNames.Add(item.ProjectName);
        }
        return listOfProjectNames;
    }
```

Listing 6: Methode zum Erstellen der Projektnamenliste

Zuerst wird von der angegebenen *URI* ein String mithilfe der *GetProjects*-Methode heruntergeladen, der die Liste der Projektnamen zusammen mit der jeweils zugehörigen Projektnummer in JSON enthält. Der String wird dann in ein *MemoryStream* eingelesen, wobei die Zeichenkodierung beachtet werden muss, die der Webserver verwendet. Die nun im Binärformat vorliegende Liste wird mit Hilfe des *DataContractJsonSerializer* in den vorher definierten Typ *ListOfProjects* serialisiert. Da für die Projektzuweisung nur die Namen der Projekte relevant sind, wird eine Liste erstellt, die nur die Projektnamen enthält. Diese Liste wird an die aufrufende Funktion zurückgegeben.

Ist ein Projekt ausgewählt und die Zuweisung vom Nutzer bestätigt, wird im Controller des Add-Ins die Methode *string SendMailSelectionToWebApp(MailItem, string projectName, Explorer activeExplorer)* aufgerufen. Dort wird für jede E-Mail, die zur Projektverwaltung gesendet wird, die Methode *MailItemToJson(MailItem mailItem, String projectName, Explorer activeExplorer*) aufgerufen, die das Outlook-

Mailitem-Objekt in das JSON-Mailformat der Mailintegration umwandelt. Ein Outlook-Mailitem repräsentiert eine E-Mail mitsamt ihren Attributen wie Anhänge, Absender, Betreff etc., die sich im aktiven Outlook Explorer befindet. Der aktive Explorer ist in diesem Fall immer die aktuell geöffnete Instanz des Programms, in der das Add-In aktiv ist. Im Folgenden werden die wichtigsten Teile der Methode vorgestellt.

Als Erstes werden, falls vorhanden, die Anhänge der E-Mail verarbeitet. Dazu wird:

- für die Konvertierung in JSON der *DataContractJsonSerializer* eingesetzt.
- ein *SimpleAttachment*-Objekt zur Serialisierung des Anhangs erstellt.

Die Klasse *SimpleAttachment* enthält die nötigen Attribute eines Mailanhangs *(siehe Listing 4)* und stellt diese für die Mailintegration bereit. Da der direkte Zugriff auf die Daten des Mailanhangs aus VSTO nicht möglich ist, werden die Daten temporär auf dem Computer zwischengespeichert und anschließend wieder in ein Bytearray eingelesen. Die Einschränkung des Zugriffs durch Outlook liegt an einer Sicherheitsrichtlinie und lässt sich nur auf diese Weise lösen.

Die Attribute *AttachmentData, DisplayName* und *FileName* können direkt aus dem Outlook *Attachment*-Objekt ausgelesen werden. Zudem wird ein Hashwert berechnet, mit dem sich der Anhang in der Projektverwaltung eindeutig identifizieren lässt. Dazu wird der SHA256-Algorithmus auf das Binärarray angewendet, welches den Anhang enthält.

Alle Anhänge der an die Projektverwaltung zu sendenden Mail werden zunächst als *SimpleAttachment*-Objekte in einer Liste zusammengeführt, bevor sie zusammen mit der restlichen Mail in JSON konvertiert werden.

Sind die Anhänge verarbeitet worden, beginnt die Erstellung der Attribute, um die in Abschnitt 3.6 vorgestellte JSON-Struktur zu füllen. Für die Serialisierung mit dem *DataContractJsonSerializer* ist, genauso wie für die Anhänge, die Klasse *SimpleMail* notwendig, die die gewünschte Struktur des JSON widerspiegelt. Dem Attribut *Attachments* wird die zuvor erstellte Liste der Anhänge zugewiesen. Die *ConversationID* wird aus dem *MailItem* ausgelesen.

Problem Zeichenkodierung

Eine Schwierigkeit stellt die korrekte Repräsentation der E-Mail in HTML dar. Es existiert zwar das Attribut *HTMLBody* im *MailItem*, allerdings fehlt hierbei die Angabe der in der E-Mail verwendeten Zeichenkodierung. Deshalb gehen viele Programme zur HTML-Darstellung (Webbrowser etc.) davon aus, dass eine ANSI-Kodierung (beispielsweise Windows 1252) verwendet wurde. Somit kann nicht mehr sichergestellt werden, dass der Inhalt der E-Mail korrekt dargestellt wird, da die vorliegenden Bytes falsch dekodiert werden. Heutzutage hat sich der Unicode in Form von UTF-8 oder UTF-16 [Korp06; S. 138ff] als Standard durchgesetzt.

Die im HTML fehlende Angabe zur Kodierung lässt sich allerdings ergänzen. Dazu muss das Attribut *Content-Type* [Palm97, S. 14f] aus der Kopfzeile der E-Mail ausgelesen werden. Dort wird der MIME-Typ [BoFr93] des Mailbodys definiert. Damit die HTML-Darstellung der Mail korrekt ist, ist das Tag „*meta charset*“ dem *HTMLBody* hinzuzufügen. Falls zum Beispiel die Mail in UTF-8 kodiert wurde, muss das folgende Tag in den HTML-Head eingefügt werden: „*<meta charset="UTF-8">*“.

Um die verwendete Kodierung in der Mail-Client-Erweiterung zu definieren, wurde die Methode *AddHtmlCharset* im Controller implementiert. Das Einfügen des fehlenden Tags *(Abbildung 6)* in den *HTMLBody* des *MailItem* läuft wie folgt ab:

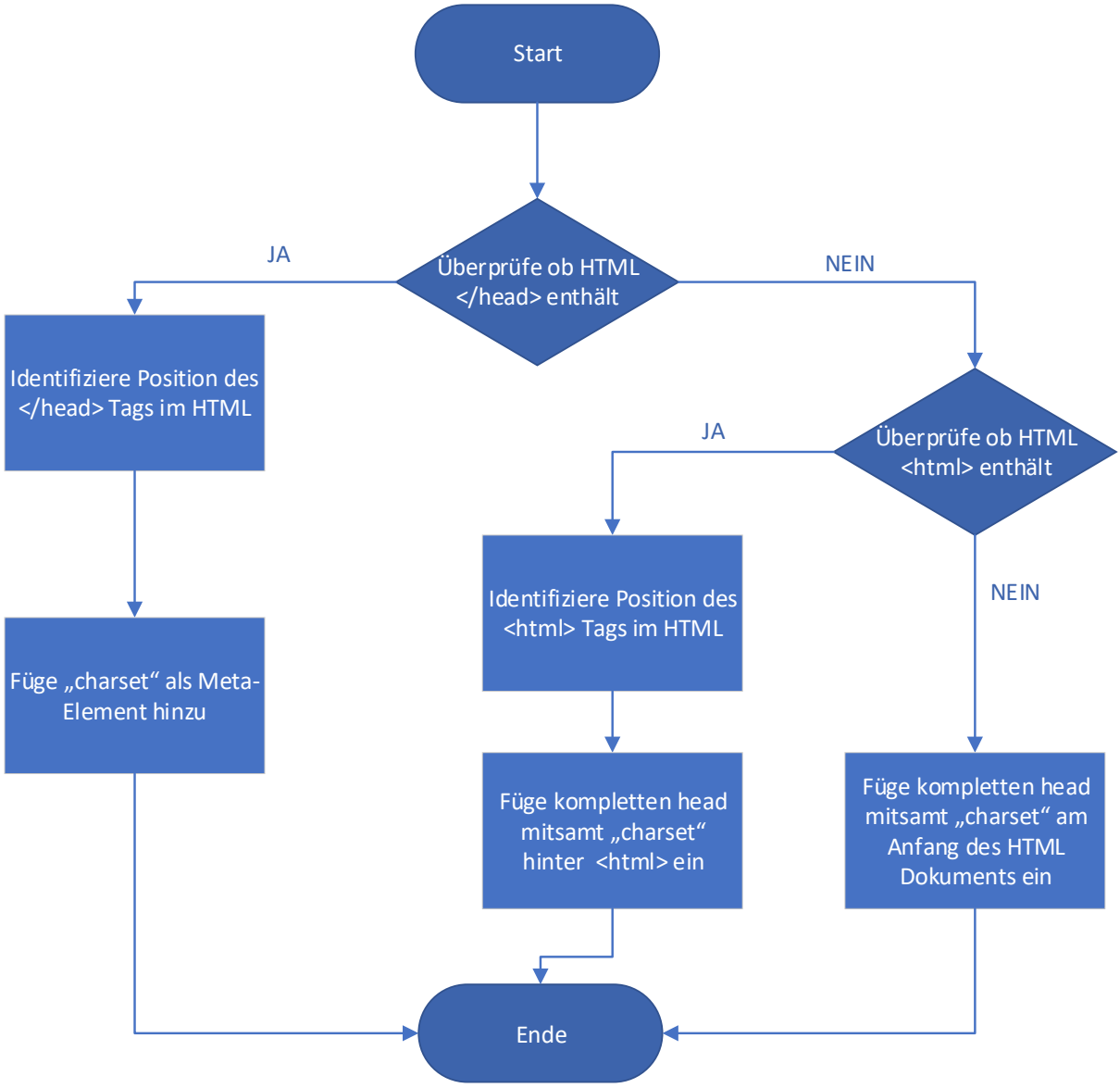

Abbildung 6: Einfügen der Zeichenkodierung

Nach Anwenden der Methode wird der *HTMLBody* von jedem Browser korrekt dargestellt, der den HTML 5 Standard unterstützt.[11]

Für die Konvertierung nach JSON müssen die weiteren Attribute, die von der generalisierten Maildarstellung gefordert werden, hinzugefügt werden:

- Der „ProjectString" enthält den Namen des Projekts, dem die Mail zugeordnet wird. Der Name wird im Projektauswahlfenster aus dem ausgewählten Projekt-Button extrahiert.
- Das Attribut „Recipient" speichert den anzuzeigenden Namen des Mailempfängers und „RecipientMailAddress" die zugehörige E-Mailadresse. Beide können aus dem Outlook *MailItem* ausgelesen werden.
- Der „SafeFileName" beinhaltet einen Windows-kompatiblen einzigartigen Dateinamen[12], unter dem die E-Mail im Dateisystem gespeichert werden kann.
- Das Datum und die Uhrzeit, zu der die Mail empfangen wurde, wird im Attribut „SendDate" vermerkt. Um eine Konvertierung im Webserver zu vermeiden, wird das Datum und die Zeit direkt im Mailclient in einem kompatiblen Format als String gespeichert: *„YYYY-MM-DD HH:MM:SSZ"*.
- Die E-Maildresse und der Name des Absenders werden genauso wie der Betreff der Mail aus dem *MailItem* abgerufen und jeweils in „*SenderMailAddress*", „*SenderName*" und „Subject" eingetragen.

Identifizierung durch Hashwerte

Damit die in die Projektverwaltung eingepflegten E-Mails in der Datenbank eindeutig identifiziert werden können, wird - vergleichbar zum Hashwert eines generalisierten Anhangs - ein Attribut benötigt, dass für jede Mail einzigartig ist. Für diesen Zweck besitzt jede E-Mail in der Projektverwaltung einen Hashwert, der folgendermaßen berechnet wird:

Das erstellte „*SimpleMail*"-Objekt wird mit der Methode *ObjectToByteArray(Object object)* in eine Bytedarstellung gebracht. Allerdings ist das Attribut „*SHA256Hash*" logischerweise noch nicht zugewiesen worden, also ist es „*null*". Zudem ist der „*ProjectString*" noch nicht auf das zugewiesene Projekt gesetzt worden. So besitzt dieselbe Mail, die zwei unterschiedlichen Projekten zugewiesen wurde, denselben Hashwert. Dadurch kann verhindert werden, dass in der Projektverwaltungsdatenbank E-Mails unnötig mehrmals abgespeichert werden. Stattdessen wird der in der Datenbank vorhandenen E-Mail ein anderes Projekt zugewiesen. Aus dem entstandenen Bytearray wird dann ein SHA256-Hash berechnet und zusammen mit dem zuvor

[11] Dies folgt daraus, dass das Meta-Attribut charset HTML 5 spezifisch ist.
[12] siehe https://docs.microsoft.com/en-us/windows/desktop/fileio/naming-a-file#naming-conventions

extrahierten Projektnamen als Bytearray im „*SHA256Hash*“-Attribut des *SimpleMail*-Objekts abgespeichert.

Abschließend wird das *SimpleMail*-Objekt mit dem *DataContractJsonSerializer* nach JSON konvertiert. Der JSON-String muss mit UTF-8 kodiert werden, da der Webserver diese Zeichenkodierung erwartet.

Mögliche Umsetzung in Thunderbird

Eine Open-Source Alternative zu Microsoft Outlook ist Mozilla Thunderbird. Da die Mailintegration modular gestaltet ist, werden im Folgenden die grundlegenden Bestandteile einer Umsetzung der Mail-Client-Erweiterung für Mozilla Thunderbird vorgestellt.

Die Anpassung der Oberfläche lässt sich sehr ähnlich zu der Outlook-Erweiterung durchführen. Dazu werden in die Datei *thunderbird-overlay.xul* die beiden Einträge „Sende/Entferne Mail“ aus der Projektverwaltung dem Kontextmenü hinzugefügt, welches sich durch einen Rechtsklick auf die jeweilige Mail in der Mailübersicht öffnet.

Mit Hilfe der XUL[13] und JavaScript wird eine Oberfläche erstellt, die dem Projektzuweisungsfenster gleicht, in dem auch die Zuweisung der ausgewählten Mails zu einem Projekt geschieht. Dazu muss die JavaScript-Funktion aus der Web-Anwendung der Mailintegration die Projekte abrufen.

Der Klick auf das gewünschte Projekt löst dann eine JavaScript-Funktion aus, die Zugriff auf die Maildatenbank von Thunderbird besitzt. Diese kann über die *nsIMsgDBHdr*-Schnittstelle die Daten abrufen, die für die generalisierte Maildarstellung notwendig sind. Aus den Daten wird ein JSON-Objekt erstellt, das über http an den Webserver geschickt wird.

Somit ist die Struktur von dem Outlook-Add-In relativ leicht übertragbar auf ein Thunderbird Plug-In, welches die gleiche Funktionalität bietet.

4.2 Erweiterung der Datenbank

Um die erfassten E-Mails mit ihren Anhängen in die Projektverwaltung, wie in Abschnitt 1.2.1 erläutert, integrieren zu können, ist die Anpassung und Erweiterung ihrer Datenstruktur nötig. Da der einzige persistente Speicherort der Generalisierten Projektverwaltung die MySQL-Datenbank ist, muss diese insoweit erweitert werden, dass die von der Mailintegration erfassten Daten aufgenommen werden können.

[13] https://developer.mozilla.org/de/docs/Mozilla/Tech/XUL

Ausgehend von der bisherigen Datenbank der Projektverwaltung [Kers16; S. 14] müssen für die Mailintegration die folgenden Tabellen und Beziehungen hinzugefügt werden (*Abbildung 7*):

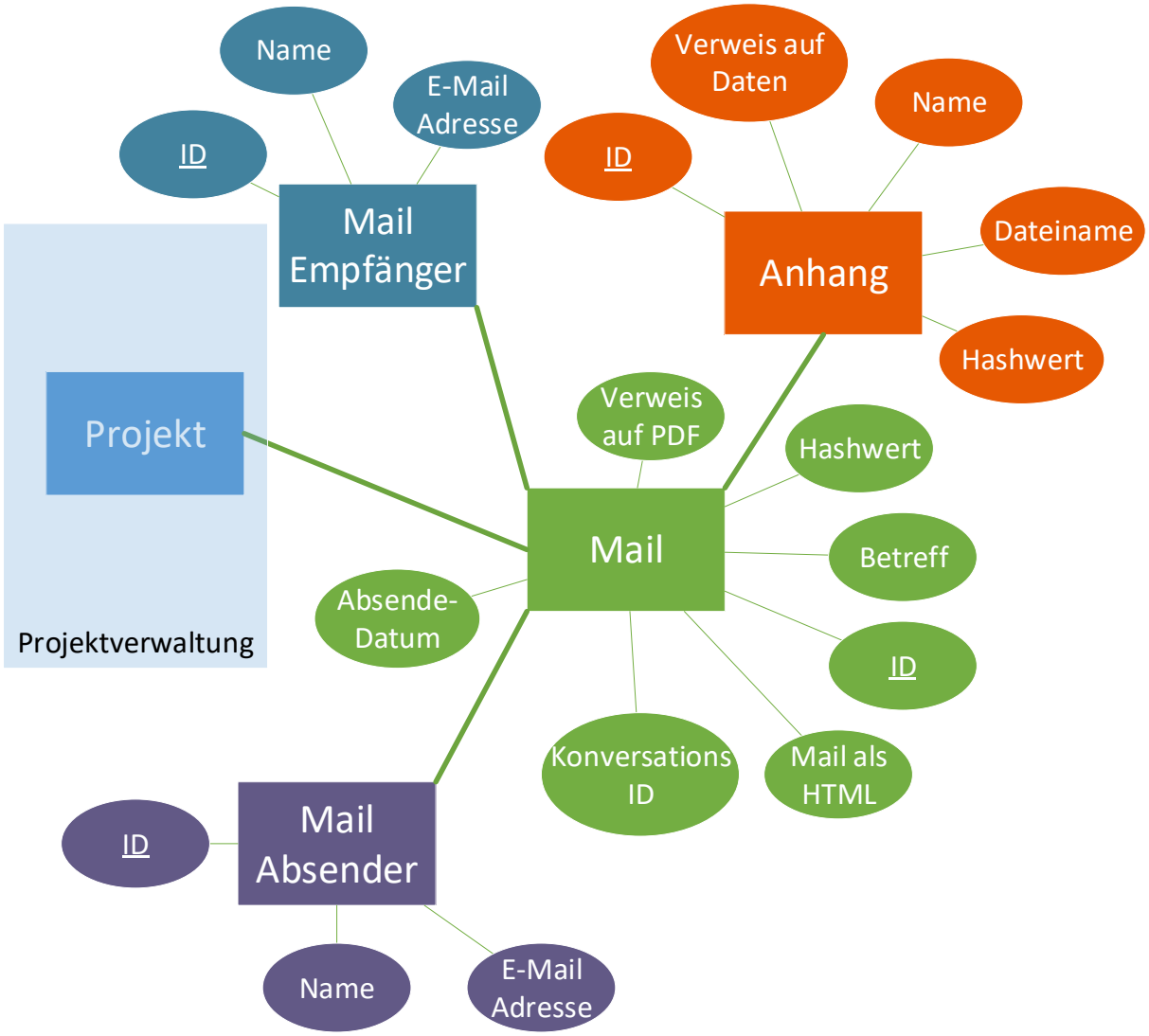

Abbildung 7: ER-Diagramm der Datenbankerweiterung in Chen-Notation

Bei der Konzeption der Datenbankerweiterung sollen Redundanzen vermieden werden. Deshalb wurde die Datenbank auf funktionale Abhängigkeiten untersucht und in die dritte Normalform überführt [Kleu13; S. 88]. So ergeben sich die folgenden Tabellen, die in die Datenbank eingefügt werden:

Mail

Die Tabelle *Mail* enthält die in Abschnitt 3.6 aufgeführten Daten der generalisierten Maildarstellung. Um Redundanzen zu vermeiden, wird auf das zugewiesene Projekt, den Absender und den Empfänger durch eine Relation verwiesen. So können mittels

SQL-JOIN auch komplexere Abfragen wie *„Wann hat das Projekt gestartet, das einer gegebenen Mail zugewiesen wurde“*, realisiert werden.

Anhang

Die Generalisierte Darstellung von Anhängen spiegelt sich in der Tabelle *Anhang* wider. Die Daten des Anhangs werden nicht in der Datenbank gespeichert, da sich dies negativ auf deren Performanz auswirkt. Stattdessen werden Verweise auf die zugehörigen Daten abgespeichert. Eine Besonderheit ist, dass in der Tabelle Mail nicht, wie in der JSON-Repräsentation, auf die zugehörigen Anhänge verwiesen wird. Stattdessen referenziert jede Entität der Tabelle *Anhang* auf eine Mail-Entität. Das folgt aus der Normalisierung der Tabelle *Mail*. Wenn auf die Anhänge in der *Mail*-Tabelle verwiesen würde, wäre die erste Normalform [Kleu13; S. 83] verletzt. Dies hängt damit zusammen, dass bei einer E-Mail mit mehreren Anhängen in einer Spalte mehrere Werte stehen würden.

Empfänger und Sender

Die Tabellen *Empfänger* und *Sender* beinhalten die E-Mailaddresse und den Namen des Empfängers, beziehungsweise Absenders. Diese Daten müssen aus der Mail entfernt werden, da sie nicht funktional abhängig von genau einer E-Mail sind.

4.3 Webserver als Datenbank-Schnittstelle

Das Bindeglied zwischen Mail-Client-Erweiterung und Projektverwaltung stellt eine Web-Anwendung dar. Diese kann entweder für eine Person lokal auf ihrem Rechner installiert werden oder beispielsweise auf einem Server, der sich in einem Intranet befindet. Bei Installation auf einem Server besteht die Möglichkeit, dass gleichzeitig mehrere Mitarbeiter die Mailintegration nutzen können. Somit kann die in Abschnitt 1.2.3 genannte Anforderung umgesetzt werden.

Die Web-Anwendung läuft auf dem *Flask*-Framework[14], das auf Python basiert. Dieses bietet die Möglichkeit HTTP-Requests nach dem RESTful-Paradigma [Patn17; S. 7f] zu kreieren. Als Folge kann der Python-Code durch Adressen mit „sprechenden“ Namen aufgerufen werden. Da *Flask* zudem ein Microframework mit geringem Overhead ist und die benötigte Funktionalität eines Webservers bereitstellt, bietet sich der Einsatz an. Alternativ könnte man die HTTP-Requests mit der *http*-Klasse aus der Python Standardbibliothek umsetzen. Dann müsste aus den Methoden dieser Klasse ein Webserver erstellt werden. Dies ist nicht nur aufwändiger zu programmieren, sondern erzeugt auch weniger übersichtlichen Quellcode als *Flask*.

[14] siehe http://flask.pocoo.org/

Die Web-Anwendung hat den Zweck, eine Rest-Schnittstelle zur Datenbank der Projektverwaltung bereitzustellen. Da diese von möglichst vielen Mail-Clients genutzt werden soll, erfolgt der Datenaustausch über JSON-Objekte, die per http an die Web-App und zurück zum Mail-Client geschickt werden. Zudem werden aus den http-Anfragen SQL-Befehle konstruiert, um auf die Datenbank zuzugreifen. Im Folgenden werden die wichtigsten Bestandteile der Web-Applikation in Funktion und Umsetzung vorgestellt.

Datenstrukturen

Damit die Kommunikation über JSON-Objekte zwischen Mail-Client-Erweiterung und der Web-App gelingt, muss die definierte JSON-Struktur zur Datenübertragung der Web-App bekannt sein. Die Web-App deserialisiert die über die jeweilige Schnittstelle empfangenen Daten in Python-Objekte. Am häufigsten wird die Umwandlung von E-Mails im JSON-Format durchgeführt. Für diesen Zweck wurden die Klassen *mail* und *attachment* angelegt, die über dieselben Attribute wie ihr Pendant im Mail-Client verfügen. Die als String empfangenen JSON-Objekte werden mithilfe der *loads*-Methode aus der Python-Bibliothek *json* in ein Python-*dictionary* konvertiert. Die folgende Methode (*Listing 7*) führt die beschriebene Konvertierung der Mail und die der Anhänge durch:

```
"""unloads the json sent by outlook and creates object"""
def jsonMailToObject(mailAsJSON):
    loaded = json.loads(mailAsJSON)
    # save the json to transfer it to object
    attachmentJson = loaded['Attachments']
    # fill the list with the attachments
    attachmentObjects = list()
    for attach in attachmentJson:
        tempAttachObject = attachment(attach['AttachmentData'], None,
                                      attach['AttachmentType'],
                                      attach['DisplayName'],
                                      attach['FileName'],
                                      attach['SHA256Hash'])
        attachmentObjects.append(tempAttachObject)

    mailObject = mail(loaded['SenderName'], loaded['SenderMailAddress'],
                      loaded['SendDate'], loaded['Recipient'],
                      loaded['RecipientMailAddress'], loaded['Subject'],
                      urllib.request.unquote(loaded['MailBodyHTML'],
                      encoding='utf-8', errors='replace'),
                      loaded['ProjectString'],
                      loaded['SafeFileName'], loaded['SHA256Hash'],
                      loaded['ConversationID'],  attachmentObjects)
    return mailObject
```

Listing 7: Konvertierung der JSON-Maildarstellung in ein Python-Objekt

Zuletzt wird das durch die Konvertierung entstandene, strukturierte *dictionary* passend aufgeteilt und an die Konstruktoren für die Klassen *mail* und *attachment* übergeben.

Funktionen der Mail-Client Schnittstelle

Die Funktion *mailClientProjectTransfer()* sendet die in der Projektverwaltung verfügbaren Projekte mit ihren Projektnummern an die Mail-Client Erweiterung *(Listing 8)*.

```
# get the projectnames and projectnumbers ordered by the projectnumber
@app.route('/outlook/getprojects', methods=['GET'])
def mailClientProjectTransfer():
    # get the projects with name and number from database
    listOfProjectsQuery = 'SELECT `ProjektNr`, `Name` FROM projekt'
    listOfProjectsQueryResult = generalDBQuery(listOfProjectsQuery, None,
    'nameOfDatabase', '127.0.0.1', 'username', 'password')
    numbersAndProjects = {"ProjectData":[{"ProjectNumber": elem[0],
                          "ProjectName": elem[1]}
                          for elem in listOfProjectsQueryResult]}

    jsonList = json.dumps(numbersAndProjects)
    return jsonList
```

Listing 8: Flask-Methode zum Abrufen der Projektliste

Mit der Anweisung *@app.route()* des Flask-Frameworks wird eine Python-Funktion an eine Serveradresse gebunden. Der erste Übergabeparameter definiert den Pfad auf dem Server, unter dem die Funktion verfügbar ist. Der zweite gibt an, ob die http-Methoden GET oder POST genutzt werden soll. Dann wird die SQL-Query definiert, die aus der Tabelle „*projekt*“ die Projektnummer und den Projektnamen abruft. Nummer und Name werden mit der Funktion *generalDBQuery()*, die im Folgenden erläutert wird, an die Datenbank gesendet. Das Ergebnis der Datenbankabfrage wird in eine Python-Listenstruktur eingefügt, die mit Hilfe der Funktion *dumps()* der *json*-Bibliothek von Python in JSON konvertiert wird. Die Variable *jsonList* enthält somit einen String, der ein JSON-Objekt (*Listing 9*) nach dem folgenden Schema enthält:

```
{
     "ProjectData": [{
          "ProjectNumber": "0.1",
          "ProjectName": "Projekt1"
     }, {
          "ProjectNumber": "0.2",
          "ProjectName": "Projekt2"
     }]
}
```

Listing 9: Projektliste in JSON-Darstellung

Dieser String wird in der Funktion zurückgegeben, wodurch er an den Client zurückgeschickt wird, von dem die Anfrage gekommen ist.

Die Kommunikation mit der Datenbank wird über die Methode *generalDBQuery()* durchgeführt. Sie besitzt die Signatur *generalDBQuery(query, insert_data_tuple, databaseName, address, username, password)*. Die Funktion sorgt dafür, dass in den Funktionen der Web-Applikation von der konkreten Implementation der Datenbank abstrahiert wird. So müsste bei einer Umstellung auf eine andere SQL-Datenbank-Implementation nur *generalDBQuery()* angepasst werden. Der Übergabeparameter *query* ist die durchzuführende SQL-Abfrage als String. Das zweite Argument *insert_data_tuple* wird nur benötigt, falls *query* ein SQL-INSERT enthält. Dann werden die in die Datenbank einzufügenden Daten als Python-Tupel übergeben. Der Name der Datenbank sowie die Zugangsdaten werden unter *address*, *username* und *password* übergeben. Mit den Zugangsdaten wird daraufhin durch den *connector* der Python MySQL-Bibliothek eine Verbindung zur Datenbank aufgebaut. Ist die Verbindung aufgebaut, wird die *query* gegebenenfalls zusammen mit dem *insert_data_tuple* durch die *cursor.execute(query, insert_data_tuple)*-Funktion ausgeführt. Das Ergebnis der Query wird an die aufrufende Funktion zurückgegeben.

Die Hauptfunktionalität der Mail-Client Schnittstelle stellt die *sendMailAndAttachmentsToDB()*-Funktion dar. Sie fügt eine im Voraus in ein Python-Objekt verwandelte E-Mail einschließlich vorhandener Anhänge in die Datenbank der Projektverwaltung ein. Die allgemeine Vorgehensweise wird im folgenden Programmablaufplan (*Abbildung 8*) vorgestellt.

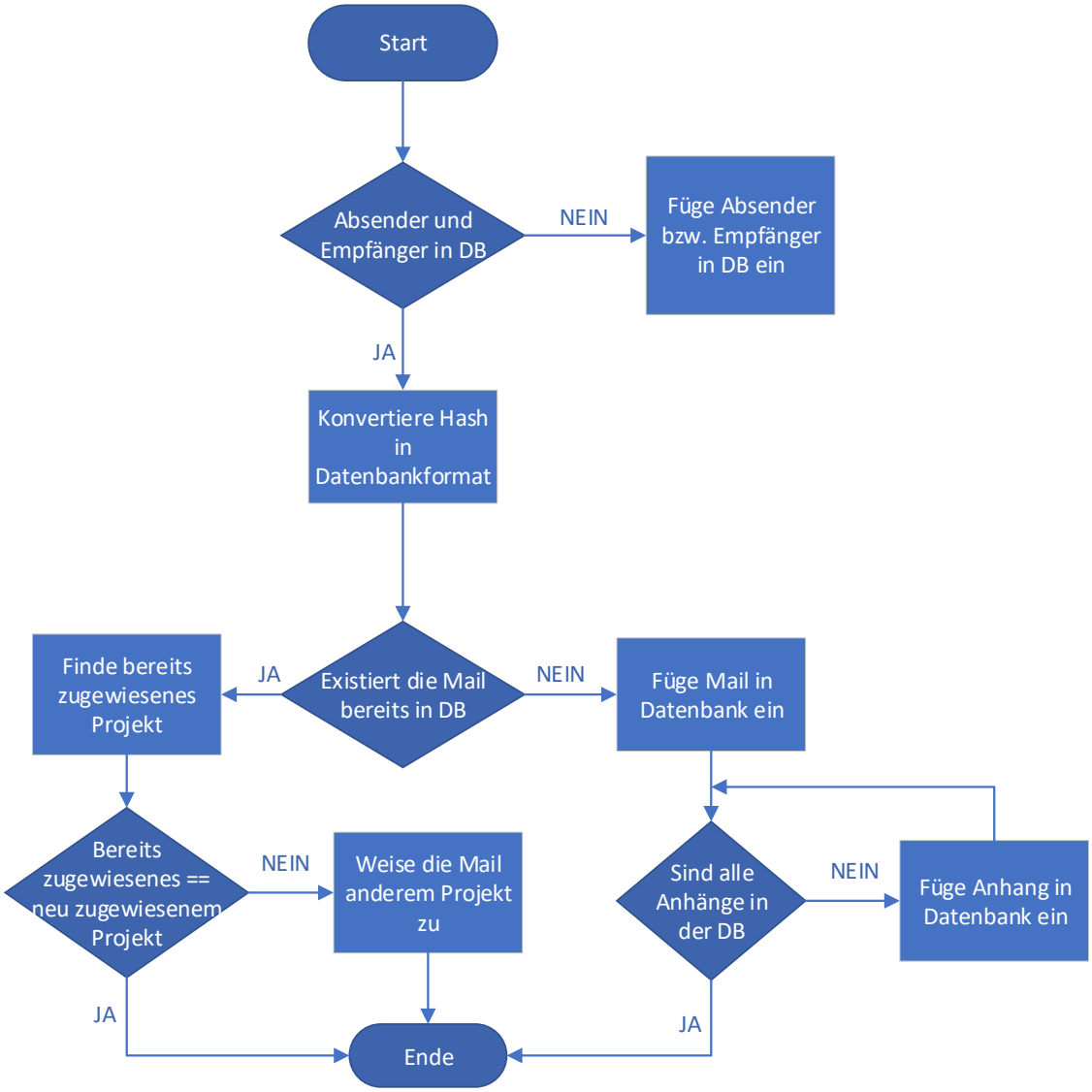

Abbildung 8: Senden von Mail mit Anhängen an die Datenbank

Zu Beginn der Funktion wird durch SQL-Abfragen überprüft, ob der Absender und der Empfänger des übergebenen Mail-Objekts bereits in der jeweiligen Tabelle der Datenbank existiert. Ist dies für einen der beiden nicht der Fall, wird die Entität eingefügt. Da die Datenbank den Hashwert der E-Mail im SQL *varbinary(256)*-Format erwartet, muss die vom Mail-Client empfangene Liste von Bits in einen String gewandelt werden. Dieser wird durch die Funktion *sha256ListToString(hashList)* erstellt und enthält den Hashwert hexadezimal kodiert beginnend mit dem Präfix „0x“. Danach wird mithilfe eines regulären Ausdrucks überprüft, ob der String, der Datum und Zeit enthält, dem erwarteten Format entspricht.

Damit dieselbe E-Mail nicht mehrmals in der Datenbank abgespeichert wird, überprüft das Programm mit Hilfe der Methode (*Listing 10*) *checkMailExistsDB(mailObject)*, ob sich die Mail bereits in der Datenbank befindet. Die Methode sieht wie folgt aus:

```
def checkMailExistsDB(mailObject):
    shaString = sha256ListToString(mailObject.sha256Hash)
    shaWithPrefix = "0x" + shaString
    checkMailQuery = "SELECT * FROM mail WHERE `hashvalue` = '%s'
                      LIMIT 1" % (shaWithPrefix)
    checkMailResult = generalDBQuery(checkMailQuery, None,
        'databaseName', '127.0.0.1', 'username', 'password')
    if checkMailResult:
        # the mail already exists in the database
        return True
    else:
        # the mail does not exist in the database
        return False
```

Listing 10: Methode zur Überprüfung der Existenz einer Mail in der Datenbank

Falls die Methode *true* zurückgibt, also die Mail bereits in der Datenbank vorhanden ist, wird herausgefunden, welchem Projekt die Mail in der Datenbank bereits zugewiesen ist. Ist das neu zugewiesene Projekt gleich dem Projekt in der Datenbank, wird nichts unternommen und die Funktion ist beendet. Andernfalls wird das zugewiesene Projekt für die Mail in der Datenbank geändert.

Existiert die Mail allerdings nicht in der Datenbank, wird sie zunächst in die Tabelle Mail eingefügt. Danach wird überprüft, ob die Anhänge der Mail bereits in der Datenbank existieren. Dazu wird über die Liste der Anhänge des Mail-Objekts iteriert und für jedes Anhang-Objekt eine Datenbankabfrage durchgeführt, die nach dem jeweiligen Hashwert des Anhangs sucht. Wird dieser nicht gefunden, wird der Anhang in die Datenbank eingefügt. Dazu wird die ID der zum Anhang gehörigen Mail aus der Datenbank gelesen und zusammen mit den restlichen Daten des Anhangs eingefügt. Zu beachten ist hier, dass die Datei des Anhangs nur als Verweis in der Datenbank gespeichert wird (siehe Abschnitt 3.7).

Funktionen der Schnittstelle des Mail-Betrachters

Über die Schnittstelle zwischen Web-Anwendung und Mail-Betrachter werden Daten wie Mails, Projekte, Termine und Anhänge aus der Datenbank an das Programm zur Mailbetrachtung weitergeleitet. Da es sich um einen reinen Betrachter handelt, können die Mails und Anhänge nicht verändert werden, weshalb auch keine Daten vom Betrachter in Richtung Web-Anwendung gesendet werden. *(siehe Abbildung 3)*

Exemplarisch wird hier die Methode (*Listing 11*) *getMailsFromProject(project-number)* vorgestellt. Sie wird aufgerufen, indem der Mail-Betrachter per http-Befehl „*POST*“ die Projektnummer des gewünschten Projekts an den Webserver über die Route „*/outlook/GetMailsFromProject*“ sendet.

```
""" requests all mails from a specific project from the database """
def getMailsFromProject(projectnumber):
    getMailsFromProjectQuery = "SELECT * FROM `mail`
        WHERE `projekt_id` = '%s'" % (projectnumber)
    mailsFromDB = generalDBQuery(getMailsFromProjectQuery, None,
        'databaseName', '127.0.0.1', 'username', 'password')
    # convert the received data to a json serializable format
    listOfMails = list(mailsFromDB)
    jsonCompatible = {"Mails":[{"ID": elem[0],
        "Project_ID": elem[1],
        "Send_Date": elem[2].strftime("%Y-%m-%d %H:%M:%S"),
        "Sender_ID": elem[3], "Recipient_ID": elem[4],
        "Subject": elem[5], "Mail_Body_Html": elem[6],
        "Conversation_ID": elem[7], "PDF_Path": elem[8],
        "Hashvalue": elem[9].decode('utf-8').replace("'", '"')}
        for elem in listOfMails]}
    jsonListMails = json.dumps(jsonCompatible)
    return jsonListMails
```

Listing 11: Abrufen aller Mails eines Projekts

Zu Beginn werden alle Mails aus der Datenbank abgerufen, die dem durch *project-number* definierten Projekt zugewiesen sind. Die empfangenen Mails werden in ein JSON-Objekt unter dem Namen „*Mails*“ als Array eingefügt. Der Datum-/Zeit-String muss konvertiert werden, damit das Einlesen in der C# *.NET*-Umgebung des Mail-Betrachters einfacher realisiert werden kann. Zurückgegeben wird die erstellte Liste von Mails eines Projekts an die Mailbetrachter-Software über die http-response.

Konvertierung der Mails zu PDF

Eine von der Bohnenkamp-Stiftung gewünschte Funktionalität ist die Speicherung der E-Mails als PDF in der Dateistruktur, die auch als Akte bezeichnet wird. Die Akte ist Bestandteil der Projektverwaltung und befindet sich im Dateisystem des Servers, auf dem die Software läuft. Damit die hier vorgestellte Web-Anwendung auf die Akte zugreifen kann, muss sie entweder auf demselben Server laufen oder per Netzwerk Zugriff darauf erhalten.

Die Konvertierung der E-Mail von HTML nach PDF könnte in der Mailintegration entweder direkt im Mail-Client oder aber in der Web-Anwendung durchgeführt werden. Die erste Methode hat den entscheidenden Nachteil, dass der verwendete Mail-Client die Funktion zur Konvertierung besitzen muss. Somit würde die Flexibilität bei der Auswahl des Mail-Clients eingeschränkt.

Aus diesem Grund wird die Umwandlung der E-Mails in das PDF-Format stattdessen erst in der Web-Applikation vorgenommen. Dazu wird die vom Mail-Client empfangene Mail in der Web-Anwendung in ihrer HTML-Repräsentation an einen HTML-zu-PDF-

Konverter gesendet. Ausgewählt wurde der Konverter „wkhtmltopdf“[15], da dieser ohne eine grafische Nutzeroberfläche arbeitet und mit Hilfe des Python-Wrappers „PDFKit“[16] sich einfach durch die Web-Anwendung bedienen lässt.

Die in *Listing 12* vorgestellte Funktion *saveMailAsPdf(mailObject)* wird für jede von der Web-Anwendung empfangenen E-Mail ausgeführt.

```
def saveMailAsPdf(mailObject):
    path_wkhtmltopdf = r'path\wkhtmltopdf.exe'
    config = pdfkit.configuration(wkhtmltopdf=path_wkhtmltopdf)
    # save the mail in a folder named after the conversation  id,
    # so all mails from one conversation are saved in one folder
    pathToAkte = getPathToAkte(mailObject.projectName)
    safeFilePath = pathToAkte + os.path.sep +
        str(mailObject.conversationID) + os.path.sep
    # create all needed folders if they are not existing
    directoryOfConversation = os.path.dirname(safeFilePath)
    if not os.path.exists(directoryOfConversation):
        os.makedirs(directoryOfConversation)
    directoryOfConversation += os.path.sep +
        mailObject.safeFileName + r'.pdf'

    pdfkit.from_string(mailObject.mailBodyHTML,
        directoryOfConversation, configuration=config)
    return directoryOfConversation
```

Listing 12: Konvertierung der Mail in Pdf

Damit die Konvertierung gelingt, muss das Programm „wkhtmltopdf“ auf dem ausführenden Rechner installiert sein und der Installations-Pfad an „PDFKit“ im Quellcode übergeben werden. Der Aufruf der Methode pdfkit.from_string() in der drittletzten Zeile von *Listing 12* konvertiert den String, der den HTML-Text enthält, in das PDF-Format. Sie speichert die entstandene Datei in einem Unterordner für die zugehörige E-Mail-Konversation ab. Durch die so entstandene Dateistruktur können die Nutzer schnell über die Oberfläche der Projektverwaltung auf die gewünschte PDF-Datei zugreifen.

Ein Nachteil der Konvertierung mit „wkhtmltopdf“ ist, dass bei E-Mails im HTML-Format, falls vorhanden, immer die Darstellung für Mobilgeräte ausgewählt wird. So sind die generierten PDF-Dateien nicht für die Darstellung auf breiteren Monitoren optimiert.

Zusätzlich zu den generierten PDF-Dateien werden in der Akte auch die Anhänge der jeweiligen Mail abgespeichert. Dies wird im folgenden Abschnitt erläutert. Somit wird die Akte der zentrale Zugriffsort für jegliche Art von projektbezogenen Dateien.

[15] https://wkhtmltopdf.org/
[16] https://pypi.org/project/pdfkit/

Speicherung der Mail-Anhänge

Die Speicherung der Anhänge wird genauso wie die PDF-Konvertierung bei Ankunft der Mail in der Web-Applikation durchgeführt. Da in der Datenbank nur Meta-Daten und ein Verweis auf den Anhang gespeichert werden, muss der Anhang selbst extern gespeichert werden. Während der Entwicklung wurde auch die Speicherung des kompletten Anhangs innerhalb der Datenbank getestet. Dies führt allerdings bei größeren an Mails angehängten Dateien(>1MB) zu einem Abbruch der Transaktion, da die maximale Größe einer Transaktion überschritten wird. Eine Erhöhung der Transaktionsgröße in der Einstellungsdatei der *MariaDB*-Datenbank[17] führt zum Verlust von bereits bestehenden Daten der Projektverwaltung. Deshalb werden die Anhänge auf dem Server in der Akte der Projektverwaltung abgespeichert. In der Datenbank wird ein Verweis auf die Datei erstellt.

4.4 Mailbetrachter

Die Betrachtung der zu einem Projekt zugewiesenen E-Mails erfolgt in der Mailbetrachter-Komponente der Mailintegration. Mit ihr lässt sich der komplette Verlauf der E-Mail-Konversation eines Projekts nachvollziehen.

Der Mailbetrachter ist eine eigenständige Anwendung, die in C# geschrieben ist und im *.NET*-Framework[18] läuft. Für die grafische Darstellung wird das Windows-Forms-Toolkit verwendet. Die benötigten Daten bezieht die Anwendung über eine Schnittstelle zur Web-Anwendung der Mailintegration.

Anzeige von E-Mails

Die Hauptfunktionalität des Programms, die Betrachtung der E-Mails, wird über die in der Datenbank gespeicherten HTML-Versionen der Mails realisiert. Dazu wird in der Oberfläche des Mailbetrachters ein Webbrowser-Fenster integriert. Dieses ist eine *WebBrowser*-Instanz aus dem Windows-Forms-Toolkit und zeigt die vom Nutzer ausgewählte E-Mail an.

Damit die E-Mail angezeigt werden kann, müssen vorher allerdings einige Schritte durchgeführt werden. Zu beachten ist, dass der Mailbetrachter auf zwei Weisen aufgerufen werden kann. Entweder startet das Programm direkt aus der Projektübersicht der Projektverwaltung oder es wird aus der geöffneten Projektansicht eines Projekts gestartet. Im ersten Fall muss der Nutzer nach dem Start des Programms auswählen, für welches Projekt die Mail-Konversation angezeigt werden soll.

[17] https://mariadb.org/
[18] https://www.microsoft.com/net

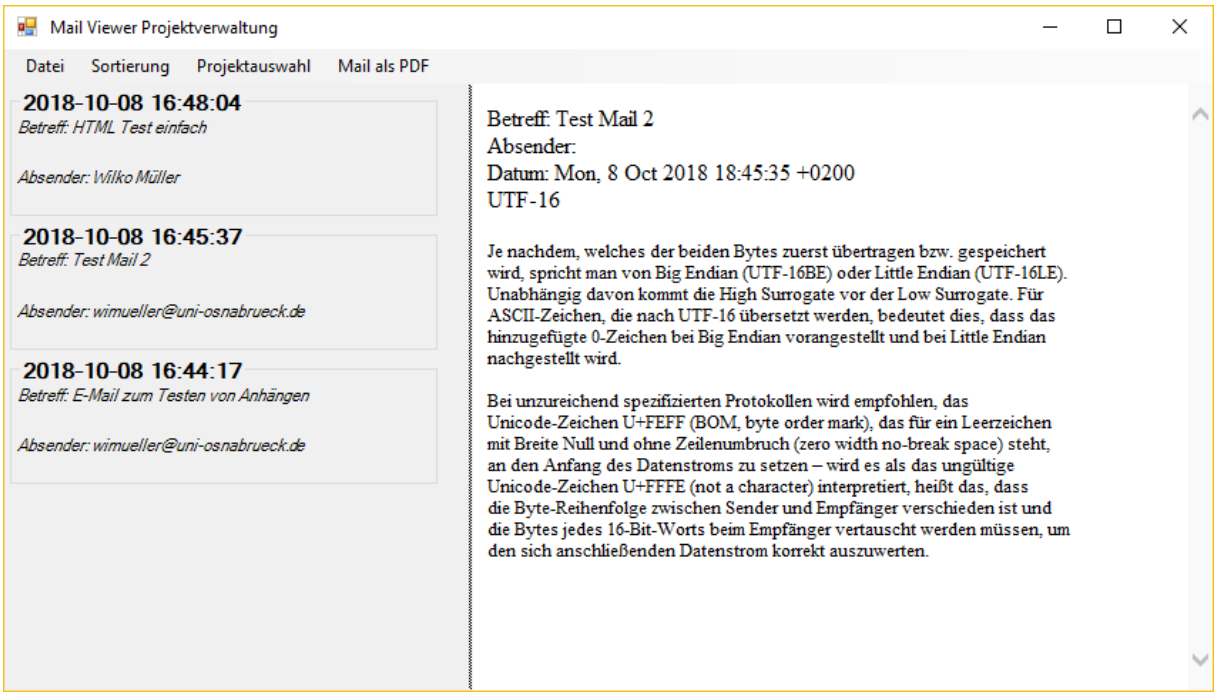

Abbildung 9: Oberfläche des Mail-Betrachters

Dazu klickt der Nutzer auf den Menüpunkt „Projektauswahl“. *(siehe Abbildung 9)* Daraufhin lädt das Programm die Liste der Projekte aus der Projektverwaltung im JSON-Format über die Schnittstelle zur Web-Anwendung. Durch Klicken des Menüpunkts öffnet sich ein Dropdown-Menü, in dem die geladenen Projekte zur Auswahl stehen. Für den Fall, dass der Mailbetrachter direkt aus der Detailansicht eines Projekts geöffnet wird, ist dieses Projekt direkt ausgewählt. Daraufhin werden alle Mails abgerufen, die dem ausgewählten Projekt zugeordnet sind.

5 Testverfahren

Damit das Einpflegen von Erweiterungen in die Mailintegration erleichtert wird, sind eine Reihe von Tests geschrieben worden. Diese können mit Hilfe des Tools „Test-Explorer“ von Visual Studio 2017 automatisiert ausgeführt werden. Durch die entworfenen Tests werden zwei Schnittstellen auf Funktionalität überprüft, zum einen die zwischen Outlook-Add-In und Web-Anwendung und zum anderen die Schnittstelle zwischen Mailbetrachter und Web-Anwendung. Da die Verbindung zwischen Mail-Client-Add-In und Web-Anwendung aufgrund des Hinzufügens von E-Mails für die Funktionalität der Mailintegration von besonderer Bedeutung ist, muss dieses Interface besonders gut getestet werden. Zudem ist zu erwarten, dass eine zukünftige Erweiterung der Software-Änderungen an einem der beiden beteiligten Module Änderungen auslösen wird. Im ersten Fall würde Funktionalität hinzugefügt, im zweiten Fall würde die Schnittstelle zur Web-Anwendung verändert.

Die durch solche Umgestaltungen verursachten Änderungen werden durch die entworfenen Komponententests auf Kompatibilität mit der bestehenden Software überprüft. Dazu müssen im Vorhinein für die zu testenden Funktionen Soll-Zustände definiert werden. Anhand dieser kann nach Ausführung der jeweiligen Funktion entschieden werden, ob sie ein erwartetes Ergebnis zurückgibt. Wird zum Beispiel das Format der Projektliste, die die Mail-Client-Erweiterung zur Projektzuweisung benötigt, geändert, kann dies durch eine Testmethode schnell ermittelt werden. Sie überprüft durch einen vorher definierten regulären Ausdruck, der den Soll-Zustand beschreibt, wie die Ausgabe der Methode definiert sein muss, damit eine korrekte Funktionalität der Mailintegration gewährleistet ist.

Durch die Automatisierbarkeit der Tests kann nach jeder größeren Modifikation an der Mailintegration einfach überprüft werden, ob die durchgeführten Änderungen die Funktionalität der restlichen Anwendung nicht beeinflussen. Scheitern einige der durchgeführten Tests, müssen solange Änderungen an dem hinzugefügten Quellcode durchgeführt und diese Tests wiederholt werden, bis bei allen Tests der Soll- dem Ist-Zustand entspricht. Dieses Verfahren wird auch als Regressionstesten bezeichnet [Blac02; S.102f].

5.1 Testen der VSTO-Schnittstelle

Das Testen der Schnittstelle zwischen dem Outlook-Add-In und der Outlook-Hauptanwendung gestaltet sich im Vergleich zu den anderen Interfaces schwieriger. Das hängt damit zusammen, dass das Add-In aufgrund der vielen Abhängigkeiten zu Outlook nicht getrennt vom Mail-Client laufen kann. Trotzdem soll programmatisch getestet werden, ob die Erweiterung korrekt mit Outlook interagiert.

In *Abbildung 10* wird die Ausführung einer Microsoft Office-Erweiterung, zu der auch Outlook Add-Ins zählen, schematisch dargestellt. Auf der linken Seite der Darstellung ist zu sehen, wie die Anwendung eine Erweiterung lädt. Die zu testende Schnittstelle ist auf der rechten Seite abgebildet. Zu überprüfen ist, ob die Aufrufe des Outlook-Objektmodells die erwarteten Ergebnisse zurückgeben. Ein Beispiel für einen solchen Aufruf ist das Abrufen der aktuell in Outlook ausgewählten E-Mail. Dazu fordert das Add-In eine Objektreferenz auf das aktive Outlook Explorer Fenster an. Über die erhaltene Referenz ist die Erweiterung in der Lage, unter anderem auf die momentan ausgewählte E-Mail als *Mailobject* des Outlook-Objektmodells zuzugreifen.

Ein zu testender Anwendungsfall ist, ob das Outlook-Add-In die E-Mails korrekt verarbeitet und an die Web-Anwendung sendet. Dafür muss zunächst, wie im vorigen Absatz beschrieben, ein *Mailobject* abgerufen und eine Testklasse erstellt werden. Die Testklasse kann zwar genauso wie das Add-In auf die *Primäre Interopassembly* (PIA) zugreifen, allerdings nicht auf Variablen und Methoden der Erweiterung. Grund dafür

ist, dass das Outlook-Add-In als COM-Add-In[19] implementiert ist, wodurch es im Prozess der Hauptanwendung (hier Outlook) läuft.

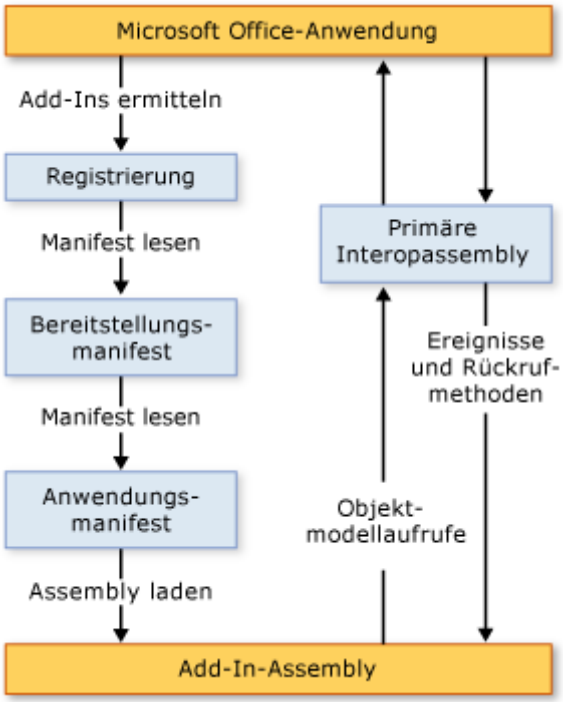

Abbildung 10: Funktionsweise eines VSTO-Add-Ins [Micr17]

Damit die Testklasse trotzdem Zugriff auf das Add-In erhält, braucht die Erweiterung eine Hilfsklasse, die den aktiven Explorer und weitere Objekte aus Outlook für andere Klassen zur Verfügung stellt. Die zu testende Klasse muss über die COM-Schnittstelle sichtbar sein und die Helferklassenschnittstelle explizit implementieren.

Durch dieses Konstrukt ist es möglich, die sonst nicht erreichbaren Outlook-Funktionalitäten im Zusammenspiel mit dem Add-In zu testen.

6 Zusammenfasung

Das Ziel dieser Arbeit war die Entwicklung einer Anwendung, die Verläufe der E-Mail-Kommunikation projektorientiert archiviert. Nachdem sich die Datenbank als passende Schnittstelle herausgestellt hatte, löste eine in drei Module abgekapselte Anwendung das Problem der Mailintegration.

[19] https://support.microsoft.com/de-de/help/291392/excel-com-add-ins-and-automation-add-ins

Diese drei Module bilden die Mailintegration:

Das *Mail-Client-AddIn* dient dazu, eine projektbezogene E-Mail dem zugehörigen Projekt aus der Projektverwaltung zuzuweisen. Da diese Funktionalität in einer Erweiterung des Mailprogramms umgesetzt ist, braucht der Anwender sich nicht in eine neue Anwendung einzuarbeiten.

Die *Web-Anwendung* leistet die Konvertierungs- und Datenverarbeitungsdienste, die für das Zusammenspiel der Komponenten notwendig sind. Sie arbeitet im Hintergrund und ist die zentrale Schnittstelle der Mailintegration, die die Verbindung zur Projektverwaltung herstellt und Möglichkeiten für zukünftige Erweiterungen bietet.

Der *Mail-Betrachter* dient zur strukturierten Betrachtung der archivierten Daten. Diese Software-Komponente ermöglicht es, sich die Mails nach Projekten sortiert anzeigen zu lassen. Zudem kann auf die Anhänge der E-Mails zugegriffen werden.

Aufgrund der gewählten dreigliedrigen Struktur kann nun jedes der Module in eine andere Software-Umgebung eingefügt werden.

Durch den Einsatz der sprachunabhängigen Datenrepräsentation JSON wurde ermöglicht, dass der in C# geschriebene *Mail-Betrachter* Objekte direkt an die in *Python* geschriebene Web-Anwendung senden kann. So können Erweiterungen für die Mailintegration in jeder beliebigen Programmiersprache geschrieben werden, die diese Datenrepräsentation unterstützt.

Eine Möglichkeit, den Arbeitsaufwand des Benutzers weiter zu verringern, wäre die automatische Erkennung und Zuweisung projektbezogener E-Mails. Erreichbar wäre dies durch eine künstliche Intelligenz, wie sie beispielsweise Xinlong Bao in dem lernenden System CALO (Cognitive Agent That Learns and Organizes) entwickelt und im Rahmen einer Doktorarbeit vorgestellt hat [Bao09].

Zusätzlich könnten Termine aus den E-Mails extrahiert werden und direkt von der Mailintegration in die Generalisierte Projektverwaltung übertragen werden.

Falls die Mailintegration zukünftig auf das Internet erweitert werden soll, sind Sicherheitsmaßnahmen notwendig. Dazu zählen insbesondere Passwörter zur Zugangsbeschränkung und die Verschlüsselung der übertragenen Daten, damit Dritte keinen Zugriff auf die vertraulichen Daten der Projektverwaltung und E-Mails bekommen. Im bisher vorgesehenen lokalen Einsatzrahmen kann ohne diese Maßnahmen gearbeitet werden.

Literaturverzeichnis

[Bao09] Bao, Xinlong: *Applying Machine Learning, Recommendation, and Integration*, Doktorarbeit an der Oregon State University, Corvallis 2009.

[Blac02] Black, Rex: *Managing the Testing Process*, Wiley Publishing, New York 2002.

[BoFr93] Borenstein, N. und Freed, N.: *MIME Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, 1993, https://tools.ietf.org/html/rfc1521, Abruf am 5.11.2018.

[ChHT04] Channabasavaiah, Kishore, Kerrie Holley und Edward M. Tuggle, Jr.: *Migrating to a service-oriented architecture*, Somers, IBM Corporation Software Group 2004.

[Free16] Freeman, Adam: *Pro ASP.NET Core MVC*, Apress, Berkley 2016.

[Frie16] Friesen, Jeff: *Java XML und JSON*, Springer Science+Buisness Media, New York 2016.

[HeUD13] Helmke, Stefan, Matthias Uebel und Wilhelm Dangelmaier: *Inhalte des CRM-Ansatzes*, in: Helmke, Stefan, Matthias Uebel und Wilhelm Dangelmaier (Hrsg.): *Effektives Customer Relationship Management*, Springer Gabler, Wiesbaden 2013, S. 3-22.

[HiAd07] Hild, Ed und Susie Adams: *Pro SharePoint Solution Development: Combining .NET, SharePoint, and Office 2007*, Apress, New York 2007.

[ISJC13] Iqbal, Rahat, Nazaraf Shah, Anne James und Tomasz Cichowicz: *Integration, optimization and usability of enterprise applications*, in: Journal of Network and Computer Application, Band 36, Ausgabe 6, 2013, S. 1480-1488.

[Kers16] Kersting, Nico: *Generalisierte Projektverwaltung*, Bachelorarbeit im Fach Informatik an der Universität Osnabrück, 2016.

[Kleu13] Kleuker, Stephan: *Grundkurs Datenbankentwicklung*, 3. Aufl., Springer Vieweg, Wiesbaden 2013.

[Korp06] Korpela, Jukka K.: *Unicode Explained*, Sebastopol, O'Reilly Media, 2006.

[LeHW11] Leußer, Wolfgang, Hajo Hipper und Klaus D. Wilde: *CRM – Grundlagen, Konzepte und Prozesse*, in: Hippner, Hajo, Beate Hubrich und Klaus D. Wilde (Hrsg.): *Grundlagen des CRM*, 3. Aufl., Gabler Verlag, Wiesbaden 2011, S. 15-56.

[Micr17] Microsoft: *Architektur von VSTO-Add-Ins*, 2017, https://docs.microsoft.com/de-de/visualstudio/vsto/architecture-of-vsto-add-ins?view=vs-2017, Abruf am 5.11.2018.

[Mozi18] Mozilla: *Thunderbird/Add-ons Guide 75*, 2018, https://wiki.mozilla.org/Thunderbird/Add-ons_Guide_57, Abruf am 5.11.2018.

[Palm97] Palme, J.: Common Internet Message Headers, 1997, S. 14f, https://tools.ietf.org/html/rfc2076#section-3.13, Abruf am 5.11.2018.

[Patn17] Patni, Sanjay: *Pro RESTful APIs*, Apress, New York 2017.

[Whit05] Whitechapel, Andrew: *Microsoft .NET Development for Microsoft Office*, Microsoft Press, Redmond 2005.

[ZaAy13] Zafar, Fareeha und Omer Muhammad Ayoub (2013): *Social, Dynamic and Custom-Based Clouds: Architecture, Services and Frameworks*, in: Mahmood, Zaigham (Hrsg.): *Cloud Computing*, Springer, London 2013, S. 47-66.

Erklärung zur selbständigen Abfassung der Bachelorarbeit

Ich versichere, dass ich die eingereichte Bachelorarbeit[1] selbstständig und ohne unerlaubte Hilfe verfasst habe. Anderer als der von mir angegebenen Hilfsmittel und Schriften habe ich mich nicht bedient. Alle wörtlich oder sinngemäß den Schriften anderer Autoren entnommenen Stellen habe ich kenntlich gemacht.

……
Ort, Datum, Unterschrift

[1] Bei einer Gruppenarbeit gilt o. für den entsprechend gekennzeichneten Anteil der Arbeit.