Abschlussprüfung Sommer 2021

Fachinformatikerin für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Entwicklung des HANA Readers

Java Applikation zur performanten Abfrage von Daten eines SAP HANA Datenbanksystems

Abgabetermin: Oldenburg, den 12.05.2021

Prüfungsbewerber:
Alina Ahrens
[redacted]

Ausbildungsbetrieb:
EWE NETZ GmbH
Cloppenburger Straße 302
26133 Oldenburg

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AES-Algorithmus	Advanced Encryption Standard-Algorithmus
BI	Business Intelligence
DSGVO	Datenschutz-Grundverordnung
EPK	Ereignisgesteuerte Prozesskette
ERM	Entity-Relationship-Modell
EWE TEL	EWE TEL GmbH
EWE NETZ	EWE NETZ GmbH
G-PR	Geschäftsfeld Produkte Energie und Telekommunikation
GUI	Graphical User-Interface
IDE	Integrated Development Environment
MVC	Model-View-Controller Pattern
SHA-256	Secure Hash Algorithm

1 Einleitung

In dieser Projektdokumentation werden die Aufgaben und Ergebnisse des IHK-Abschlussprojektes der Autorin erläutert und dokumentiert. Das Projekt wurde vom 22.04.2021 bis zum 03.05.2021 im Rahmen der Ausbildung zur Fachinformatikerin für Anwendungsentwicklung durchgeführt.

1.1 Projektumfeld

Das Projekt fand im Umfeld der EWE TEL GmbH (EWE TEL) statt. Wie der Ausbildungsbetrieb EWE NETZ GmbH (EWE NETZ) ist EWE TEL ein hundertprozentiges Tochterunternehmen der EWE AG und gehört damit dem EWE-Konzern an. Die EWE AG steuert und verwaltet als Holding-Gesellschaft des Konzerns die Aktivitäten der Tochterunternehmen und Geschäftsfelder. Der Vorstandsvorsitzende des Konzerns ist Stefan Dohler (vgl. EWE AG [2021c]). Der EWE-Konzern verknüpft die Bereiche Energie, Telekommunikation und Informationstechnologie-Dienstleistungen miteinander (vgl. EWE AG [2021b]). 2019 umfasste der Konzern ca. 8.800 Mitarbeiter und erwirtschaftete einen Umsatz von rund 5,7 Mrd. Euro und ein Periodenergebnis von rund 130 Mio. Euro (vgl. EWE AG [2021a]). EWE TEL ist ein Telekommunikationsunternehmen mit Sitz in Oldenburg. Der EWE-Konzern plante 1996 in den Telekommunikationsmarkt einzusteigen, was durch die Gründung von EWE TEL erfolgte. Das Unternehmen ist überwiegend im Nordwesten Deutschlands für Privatkunden und deutschlandweit für Geschäftskunden tätig und bietet verschiedene Produkte im Bereich Festnetz und Internet, Mobilfunk und individuelle Geschäftskundenlösungen an. Seit 2020 repräsentiert der EWE-Konzern nach außen nicht seine verschiedenen Tochtergesellschaften, sondern Geschäftsfelder, in denen diese zu verorten sind. Die Produkte der Tochtergesellschaften werden nicht mehr unter der eigenen Marke angeboten (z. B. EWE TEL), sondern einheitlich unter der Marke *EWE*. Die Geschäftsfelder können in die vier Segmente Markt, Personal und Recht, Technik sowie Finanzen einsortiert werden.

Das Abschlussprojekt wurde im Umfeld des Teams Business Intelligence (BI) des Geschäftsfeld Produkte Energie und Telekommunikation (G-PR) durchgeführt. Das Team ist Auftraggeber und Kunde des Projektes. Wie in der Abbildung im Anhang A.1: Organigramm auf Seite i abgebildet, ist BI hierarchisch direkt unter dem G-PR angelegt, welches zum Segment Markt gehört. Das Team BI ist für Datenanalysen und verschiedene Datenanalysewerkzeuge zuständig, um unterschiedliche Bereiche des Konzerns zu unterstützen. Dazu werden Dashboards genutzt, die verschiedene Daten visualisieren. Langfristig sollen so Verhaltensmuster von Kunden erkannt und die angebotenen Produkte verbessert werden. Die Daten für die Datenanalyse sind dabei in einem SAP HANA Datenbanksystem gespeichert. In dem Datenbanksystem gibt es drei Mandanten: einen für Telekommunikationsdaten, einen für Energiedaten und einen für Telekommunikations- und Energiedaten bei der gemeinschaftlichen Datenverarbeitung unter Berücksichtigung der Datenschutz-Grundverordnung (DSGVO).

1.2 Projektziel

Für die Bereitstellung verschiedener Dashboards erstellen die Mitarbeiter von BI verschiedene Views und Abfragen, deren Korrektheit im Anschluss durch eine Qualitätskontrolle sichergestellt werden muss. Dazu werden Stichproben aus unterschiedlichen Datenbanktabellen, die teilweise in verschiedenen Datenbankschemata zu finden sind, genommen. Diese werden mit den Ergebnissen der erstellten Views oder Abfragen abgeglichen. Für diesen Abgleich soll eine Java-Anwendung bereitgestellt werden, die das performante Abfragen von Daten in mehreren Datenbanktabellen gleichzeitig möglich macht. Für die Ergebnisanzeige sollen die Nutzer der Anwendung vordefinierte Ergebnistabellen erstellen können, um das Suchergebnis auf entsprechend benötigte Attribute einzuschränken. Es gibt für jeden der drei Mandanten bei EWE jeweils drei Datenbanksysteme (Entwicklungs-, Konsolidierungs-, und Produktivsystem). Die Anwendung muss alle Systeme über Hostname, Port, Username und Passwort anbinden können.

1.3 Projektbegründung

Vor diesem Projekt wurde für die Suche nach bestimmten Ausprägungen in den Datenbanktabellen ein Plug-In von SAP für die Integrated Development Environment (IDE) Eclipse genutzt. Mit dem Plug-In dauert die Suche in einer Tabelle bzw. View etwa 90 Sekunden. Ingesamt werden ca. 320 Tabellen/Views im Monat geöffnet, sodass die Suche etwa acht Stunden im Monat einnimmt. Ein weiteres Problem des Plug-Ins neben der schlechten Performance ist, dass es nicht verlässlich arbeitet. Sind z. B. mehrere Tabellen gleichzeitig geöffnet, stürzt die Anwendung in etwa fünf Prozent der Fälle ab. Das erhöht die benötigte Zeit im Monat auf 8,4 Stunden. Außerdem stürzt die Anwendung ab, wenn die Netzwerkverbindung verloren geht.

Da EWE TEL stark mit eigenentwickelten Individuallösungen arbeitet, wird auch in diesem Fall eine Individuallösung präferiert. Ein Marktvergleich, Auswahlverfahren sowie die Einführung und Integration einer Drittherstellersoftware wären gegenüber der Entwicklung einer eigenen Anwendung wirtschaftlich unverhältnismäßig. Eine Individuallösung kann genau die Bedürfnisse des Auftraggebers abdecken und ist im Nachhinein einfacher um weitere Funktionen erweiterbar. Am Ende des Projektes soll deshalb eine prototypische Anwendung programmiert worden sein, mit der in verschiedenen Tabellen und Views eines SAP HANA Datenbanksystems gesucht werden kann. Die Anwendung soll performanter sein als das bisher genutzte Plug-In von SAP und damit den Mitarbeitern von BI das Abfragen von Daten und damit gleichzeitig die Qualitätssicherung von Views erleichtern. Für den Anwendungsfall der Suche auf einem SAP HANA Datenbanksystem kann das bisher genutzte Plug-In durch die neue Anwendung abgelöst werden.

1.4 Projektschnittstellen und Projektabgrenzung

Die Anwendung interargiert mit den drei Mandanten des SAP HANA Datenbanksystems von EWE und dem Dateisystem auf dem Rechner des Nutzers. BI genehmigt das Projekt und stellt die benötigten

Mittel wie z. B. ein Benutzerkonto für die Datenbanksysteme zur Verfügung. Da die Mitarbeiter von BI gleichzeitig Auftraggeber und Anwender des Projektes sind, wird das Ergebnis dem Team vorgestellt. Bei dem Projektergebnis soll es sich um einen Prototypen und nicht um ein Produktivsystem handeln, sodass das Deployment nicht Teil des Projektes ist.

2 Projektplanung

In diesem Kapitel wird die Planung des Projektes vorgestellt. Es wird sowohl die zeitliche als auch die inhaltliche Einteilung der Projektphasen dargestellt.

2.1 Projektphasen

Das Projekt wurde vom 22.04.2021 bis zum 03.05.2021 mit einer Gesamtdauer von 70 Stunden durchgeführt und in sieben Phasen eingeteilt: die Analysephase, die Planungsphase, die Implementierungsphase, die Testphase, die Ergebnisanalyse, die Dokumentation und die Übergabe des Projektes. Jede Projektphase kann dabei in kleinere Teile unterteilt werden. Eine detaillierte Übersicht über die Projektphasen ist im Anhang A.2: Detaillierte Zeitplanung auf Seite ii abgebildet.

2.2 Abweichungen vom Projektantrag

Nach Durchsicht des Projektantrags wurde vom Prüfungsausschuss zurückgemeldet, dass die Wirtschaftlichkeitsbetrachtung in der zeitlichen Planung zu spät eingeplant wurde. Daher werden die Nutzen- und die Wirtschaftlichkeitsanalyse nicht in der Phase *Ergebnisanalyse*, sondern in der Phase *Anforderungsanalyse* durchgeführt.

2.3 Ressourcenplanung

Für die Umsetzung des Projektes waren verschiedene Ressourcen notwendig. Diese lassen sich in Hardware-, Software- und Personalressourcen unterteilen. Die notwendige Hardware waren ein Standalone-Laptop und ein Büroarbeitsplatz. Der Standalone-Laptop wurde von dem Unternehmen bereitgestellt. Als Büroarbeitsplatz wurde aufgrund der Maßnahmen gegen das Coronavirus der private Arbeitsplatz im Homeoffice genutzt. Dieser ist mit dem Standalone-Laptop, einem Monitor, Maus und Tastatur sowie einem Headset zur Kommunikation mit den Kollegen ausgestattet.

Bei der genutzten Software wurde darauf geachtet, dass keine zusätzlichen Kosten durch Lizenzen entstehen. Deswegen wurde hauptsächlich freie Software oder Software mit bereits vorhandenen Lizenzen verwendet. Zur Implementierung wurde die Entwicklungsumgebung Eclipse IDE in der Version 2020-06 verwendet. Die Versionsverwaltung wurde mit Git umgesetzt. Für den Zugriff auf das SAP

HANA-Datenbanksystem waren Zugriffsberechtigungen sowie ein VPN-Zugang notwendig. Für die Dokumentationen wurde verschiedene Software genutzt. Die Benutzerdokumentation wurde mit Microsoft Office Word verfasst und die Entwicklerdokumentation wurde durch Javadoc generiert. Die vorliegende Projektdokumentation wurde mithilfe von LaTeX verfasst. Hierzu wurde der Online-LaTeX-Editor overleaf verwendet. Zur Kommunikation mit den Betreuern wurde ein Kommunikationstool benötigt. Hier wurde Microsoft Teams verwendet, da dies das Standard-Kommunikationstool im EWE-Konzern ist.

An Personalressourcen wurden sieben Stunden für einen fachlichen Betreuer einkalkuliert, der bei fachlichen Themen unterstützen konnte und Teile der Analyse- sowie Planungsphase begleitete. Außerdem wurden sechs Stunden für einen technischen Betreuer eingeplant, der bei technischen Themen unterstützen und Feedback zu der Entwicklung geben konnte.

2.4 Vorgehensmodell

Bei der Bearbeitung des Projektes wurde nach dem Wasserfallmodell vorgegangen. Das Wasserfallmodell ist ein Vorgehensmodell der klassischen Softwareentwicklung, bei dem die Projektphasen sequentiell durchlaufen und bearbeitet werden (vgl. Fittkau und Ruf [2008], S. 31). Es wurden jedoch auch Aspekte aus einem iterativen Entwicklungsprozess eingebunden. Um sicherzustellen, dass die Umsetzung des Projektes den gewünschten Anforderungen und Vorstellungen des Betreuers entsprach, wurden ihm Zwischenergebnisse präsentiert und notwendige Änderungen besprochen. Dabei wurde allerdings auf feste Zeitfenster bzw. Sprints oder Sprintziele mit Inkrementen (wie z. B. bei Scrum) verzichtet (vgl. Fittkau und Ruf [2008], S. 36-37).

3 Analysephase

In diesem Kapitel werden die Ergebnisse der Analysephase beschrieben. Dazu gehören eine Ist-Analyse, eine Wirtschaftlichkeitsanalyse und mögliche Anwendungsfälle. Außerdem wurden Qualitätsanforderungen und funktionale Anforderungen erhoben und in einem Lastenheft dokumentiert.

3.1 Ist-Analyse

Die Mitarbeiter des Teams BI erstellen verschiedene Views und Abfragen, deren Korrektheit später durch eine Qualitätskontrolle gesichert werden muss. Dazu werden Stichproben aus unterschiedlichen Datenbanktabellen, die teilweise in verschiedenen Datenbankschemata zu finden sind, genommen. Diese werden mit den Ergebnissen der erstellten Views oder Abfragen abgeglichen. Derzeit nutzen die Mitarbeiter ein Plug-In von SAP innerhalb der IDE Eclipse, um auf das Datenbanksystem zuzugreifen und Tabellen sowie Views zu prüfen. Die Daten der entsprechenden Tabelle können mit dem Plug-In von SAP über eine *Data Preview* (Datenvorschau) angezeigt werden. Danach muss die Spalte mit dem entsprechenden Attribut nach der gesuchten Ausprägung gefiltert werden, um das gewünschte Ergebnis

zu erhalten. Dies nimmt pro Durchlauf etwa 90 Sekunden in Anspruch, wobei jede Ausprägung in mehreren Tabellen bzw. Views gesucht wird. Für jede Tabelle muss eine separate Abfrage gestartet werden. Ingesamt werden ca. 320 Tabellen/Views im Monat geöffnet, sodass die Suche etwa acht Stunden im Monat einnimmt. Je mehr Tabellen gleichzeitig geöffnet sind, desto inperformanter ist das Plug-In. Außerdem ist das Plug-In nicht verlässlich: Wenn mehrere Tabellen gleichzeitig geöffnet sind, stürzt die Anwendung in ca. fünf Prozent der Fälle ab. Daher benötigen die Mitarbeiter der Abteilung eine Anwendung, die das performante Abfragen von Daten in mehreren Tabellen gleichzeitig möglich macht. Für die Suche sollen dabei die Datenbankabellen, in denen gesucht werden soll, das Attribut, das die Ausprägung haben soll und die entsprechende Ausprägung angegeben werden. Die Anwendung muss die Dantenbanksysteme der drei Mandanten von EWE über Hostname, Port, Username und Passwort anbinden können. Außerdem soll die Anwendung verlässlicher sein und nicht abstürzen, wenn die Netzwerkverbindung ausfällt oder mehrere Tabellen gleichzeitig geöffnet werden.

3.2 Wirtschaftlichkeitsanalyse

Wie in Abschnitt 1.3 bereits beschrieben, präferiert EWE für seine Projekte eigenentwickelte Individuallösungen. Dadurch können die Anwendungen genau an die Bedürfnisse der Nutzer angepasst werden und im Nachhinein einfacher weiterentwickelt werden. Im Folgenden wird die Wirtschaftlichkeit des Projektes genauer analysiert.

3.2.1 Projektkosten

Die Kosten für die Durchführung des Projektes setzen sich aus den Personalkosten der Autorin und der Projektbetreuer zusammen. Da nur lizenzfreie Software oder Software mit bereits vorhandenen Lizenzen verwendet wurde, fallen keine weiteren Kosten an. Laut Ausbildungsvertrag beträgt das Monatsgehalt der Autorin 1.104,50 € Brutto bei zwölf Gehältern im Jahr, wobei 39 Stunden in der Woche gearbeitet wird und 30 Tage Urlaub einkalkuliert werden. Um den Stundensatz ermitteln zu können, wird zunächst die Jahresarbeitszeit ermittelt. Dabei werden abzüglich der Fehltage 220 Arbeitstage für das Jahr 2021 angesetzt.

$$\frac{39 \text{ h/Woche}}{5 \text{ Arbeitstage/Woche}} = 7,8\text{h/Arbeitstag} \tag{1}$$

$$7,8 \text{ h/Arbeitstag} \cdot 220 \text{ Arbeitstage/Jahr} = 1.716 \text{ h/Jahr} \tag{2}$$

Anschließend können die Kosten pro Jahr bestimmt werden, was in Tabelle 1 gezeigt wird. Dabei müssen je 1.000 € Brutto Weihnachts- und Urlaubsgeld, sowie die Studiengebühren für das Duale Studium in der Höhe von 7.200 € im Jahr, berücksichtigt werden. Außerdem werden der Gemeinkostenzuschlagssatz von 20% sowie der Arbeitgeberanteil zu den Sozialversicherungen von insgesamt 19,725% berücksichtigt.

	Posten	Berechnung	Kosten
	Gehalt pro Monat	1.104,50 € · 12 Monate	13.254,00 €
+	Urlaubsgeld	1.000 € · 2	2.000,00 €
=	Gehalt pro Jahr		15.254,00 €
+	Gemeinkostenzuschlagssatz	0,2 · 15.254,00 €	3.050,80 €
+	Arbeitgeberanteil	0,19725 · 15.254,00 €	3.008,85 €
+	Studiengebühren		7.200,00 €
=	**Kosten pro Jahr**		**28.513,65 €**

Tabelle 1: Berechnung der Kosten pro Jahr

Aus den Kosten und der Arbeitszeit pro Jahr lässt sich der Stundensatz der Autorin berechnen.

$$\frac{28.513{,}65\,€/\text{Jahr}}{1.716\ \text{h/Jahr}} = 16{,}62\,€/\text{Stunde} \tag{3}$$

Der Zeitaufwand des fachlichen Betreuers wurde mit sieben Stunden festgesetzt und der des technischen Betreuers wurde mit sechs Stunden festgesetzt. Für beide Betreuer fällt ein Stundensatz von [redacted] an. Daraus lässt sich der Kostenaufwand für die Betreuer und damit die Projektkosten ermitteln.

$$16{,}62\,€/\text{h} \cdot 70\ \text{h} + [\text{redacted}] = 2.593{,}40\,€\ \text{Projektkosten} \tag{4}$$

3.2.2 Amortisationsdauer

Der Anwendungsfall der Suche in mehreren Datenbanktabellen auf dem SAP HANA Datenbanksystem tritt im Monat etwa 32 mal auf, wobei jedes Mal in mehreren Tabellen oder Views gleichzeitig gesucht werden soll. Im Durchschnitt ergeben sich zehn Tabellen/Views pro Durchführung. Die Suche in einer Tabelle dauert mit dem Plug-In von SAP für die IDE Eclipse im Durchschnitt 90 Sekunden. Außerdem stürzt das Plug-In in ca. fünf Prozent der Fälle ab. Daraus lässt sich der Zeitaufwand und die daraus resultierenden Kosten für die Suche im Monat berechnen. Für den ausführenden Mitarbeiter wird mit einem Stundensatz von [redacted] gerechnet.

$$32\ \text{Durchführungen} \cdot 10\ \text{Tabellen} \cdot 90\ \text{s} = 8\ \text{h/Monat} \tag{5}$$

$$8\ \text{h/Monat} \cdot 105\ \% = 8{,}4\ \text{h/Monat} \tag{6}$$

$$8{,}4\ \text{h/Monat} \cdot [\text{redacted}] = 924\,€\ /\text{Monat} \tag{7}$$

Mit dem HANA Reader wird die Dauer einer Suche geschätzt auf maximal 10 Sekunden gekürzt, wobei hier auch in mehreren Tabellen gleichzeitig gesucht werden kann. Für die Auswahl der Tabellen und

Attribute werden 60 Sekunden Dauer angenommen. Daraus kann der Zeitaufwand und die Kosten für die Suche im Monat mit dem HANA Reader berechnet werden.

$$32 \text{ Durchführungen} \cdot 70 \text{ s} = 37 \text{ min } 20 \text{ s /Monat} = 0,6222 \text{ h/Monat} \quad (8)$$

$$0,6222 \text{ h/Monat} \cdot [redacted] = 68,44 \text{ € /Monat} \quad (9)$$

Aus den resultierenden Kosten für das zuvor genutzte Plug-In und die jetzt genutzte Anwendung ergibt sich eine Einsparung von 924 € - 68,44 € = 855,56 € /Monat = 10.266,72 € /Jahr. Daraus und aus den Projektkosten kann die Amortisationsdauer für das Projekt ermittelt werden. Die Amortisationszeit beträgt $\frac{2.593,40\,€}{10.266,72\,€/\text{Jahr}} \approx 0,253$ Jahre ≈ 3 Monate. Eine graphische Darstellung befindet sich im Anhang A.3: Amortisationsdauer auf Seite iii. Aus der kurzen Amortisationsdauer ergibt sich, dass die Umsetzung des HANA Readers wirtschaftlich sinnvoll ist.

3.3 Nicht-monetärer Nutzen

Neben dem monetären Nutzen können einige nicht-monetäre Vorteile des HANA Readers festgestellt werden. Das Plug-In von SAP für die IDE Eclipse zeigt zu den erstellten Verbindungen die Datenbankschemata, Views und Tabellen in einer komplexen Baumstruktur. Es kann also länger dauern, bis die gewünschte View oder Tabelle gefunden wurde. In der Anwendung, die im Rahmen des Projektes programmiert werden soll, ist die Komplexität deutlich geringer, da in der Baumstruktur direkt unter den angelegten Verbindungen nur die selbst erstellten, benötigten Ergebnistabellen aufgeführt sind. Hier können Verbindungen und vordefinierte Ergebnistabellen ohne Auswirkungen auf die Datenbanksysteme gelöscht werden. Außerdem bietet das Erstellen vordefinierter Ergebnistabellen die Möglichkeit, nur benötigte Attribute anzugeben, um den Fokus auf wesentliche Inhalte zu setzen. Ein weiterer Vorteil ist die Erweiterbarkeit und Änderbarkeit der Anwendung. Es können jederzeit weitere Funktionen hinzugefügt werden, die das bisher genutzte Plug-In nicht hat. Außerdem können Funktionen oder Benutzeroberflächen im Nachhinein geändert werden.

3.4 Anwendungsfälle

Das Projekt soll verschiedene Anwendungsfälle abdecken, die in dem Use-Case Diagramm im Anhang A.4: Use Case-Diagramm auf Seite iv dargestellt sind. Die Anwendungsfälle *Verbindung testen* und *Sucheregebnisse anzeigen* greifen auf das SAP HANA-Datenbanksystem zu. *Verbindungen* und *Ergebnistabellen* werden im Dateisystem des Nutzers gespeichert. Die typische Vorgehensweise eines Mitarbeiters, der die Anwendung das erste Mal verwendet, zeigt die vereinfachte Ereignisgesteuerte Prozesskette (EPK) im Anhang A.5: Ereignisgesteuerte Prozesskette auf Seite v.

3.5 Qualitätsanforderungen

Neben den funktionalen Anforderungen soll die Anwendung einige Anforderungen hinsichtlich der Qualität erfüllen. Das Suchen in mehreren Datenbanktabellen gleichzeitig soll mit der Anwendung mindestens 25% performanter sein als mit dem bisher genutzten Plug-In für Eclipse IDE. Die Anwendung soll verlässlicher sein als das bisher genutzte System und nicht abstürzen, wenn die Netzwerkverbindung verloren geht. Außerdem gibt es noch eine Anforderung hinsichtlich der Sicherheit der Anwendung. Die Passwörter für die Verbindungen sollen verschlüsselt auf dem lokalen Rechner abgelegt werden.

3.6 Soll-Analyse und Lastenheft

Im Rahmen der Soll-Analyse wurde gemeinsam mit den Auftraggebern ein Lastenheft erstellt, welches die fachlichen Anforderungen und die Qualitätsanforderungen beinhaltet. Ein Auszug des Lastenheftes mit Fokus auf die funktionalen Anforderungen befindet sich im Anhang A.6: Lastenheft (Auszug) auf Seite vi.

4 Entwurfsphase

In der Entwurfsphase wurden die Zielplattform bestimmt, die Benutzeroberfläche, das Datenmodell und die Geschäftslogik entworfen sowie die Maßnahmen zur Qualitätssicherung entwickelt.

4.1 Zielplattform

Die Anwendung soll kompatibel zur aktuellen IT-Infrastruktur von EWE TEL sein, in der grundsätzlich mit der Programmiersprache Java programmiert wird. Dies war ein wichtiges Kriterium bei der Auswahl der Zielplattform, da die Verwendung der Standardsprache die Wartung und Weiterentwicklung der Anwendung durch andere Mitarbeiter vereinfacht. Ein weiterer Grund für die Verwendung der Programmiersprache Java ist, dass SAP für Java eine Bibliothek bereitstellt, um eine Verbindung zu einem SAP HANA Datenbanksystem aufzubauen.

4.2 Architekturdesign

Für den Entwurf und die Implementierung der Anwendung wurde das Model-View-Controller Pattern (MVC) verwendet. Bei MVC wird ein interaktives System in die Komponenten Model, View und Controller geteilt. Das Model hält und verarbeitet die konkreten Daten. Die View stellt die Daten dar und präsentiert die Benutzerschnittstelle, der Controller interpretiert und verarbeitet die Eingaben des Benutzers. Das dient dazu, dass die Daten unabhängig von der Darstellung sind und die Benutzerschnittstelle flexibel ausgetauscht werden kann, beispielsweise wenn anstelle einer Graphical

User-Interface (GUI) in Zukunft eine webbasierte Benutzeroberfläche verwendet werden soll. (Goll [2014], S. 377-379) Damit Änderungen der Daten direkt auf der GUI angezeigt werden können, wurde außerdem das Observer Pattern verwendet. Bei dem Observer Pattern informiert ein Objekt abhängige Objekte über eine Änderung des Zustands und eine Aktualisierung der abhängigen Elemente wird automatisch eingeleitet (Goll [2014], S. 163). Das Observer Pattern wird in der Anwendung u. a. verwendet, um neu hinzugefügte Verbindungen direkt auf der GUI anzuzeigen.

4.3 Entwurf der Benutzeroberfläche

Da es sich bei der Anwendung um eine Desktop-Anwendung handeln soll, wurde sich für eine graphische Benutzeroberfläche (GUI) entschieden. Diese wurde durch verschiedene View-Klassen als JavaFX-Applikation mit Java 8 umgesetzt. Beim Start der Anwendung wird zunächst ein Dialogfenster gezeigt, in dem der Nutzer das Passwort für die Anwendung eingeben muss. Dann öffnet sich das Startfenster der Anwendung. Dieses ist vom Aufbau dem bisher genutzten Plug-In von SAP für die IDE Eclipse angelehnt: Der Nutzer kann auf der linken Seite in einer Baumstruktur gespeicherte Verbindungen und angelegte Ergebnistabellen einsehen. In der Mitte des Fensters können die Ergebnistabellen und Attribute ausgewählt werden und die Suche kann gestartet werden. Dann öffnet sich das Ergebnisfenster mit einem Tab pro Tabelle, in dem der Nutzer die Suchergebnisse einsehen kann. Außerdem gibt es für das Anlegen und Bearbeiten neuer Verbindungen und vordefinierter Ergebnistabellen jeweils ein Fenster für die Eingabe der benötigten Werte. Diese können über die Menüleiste, über ein Kontextmenü in der Baumstruktur oder über Buttons oberhalb der Baumstruktur aus dem Startfenster heraus geöffnet werden. Um die Attribute für die Ergebnistabellen auszuwählen, gibt es ein weiteres Fenster mit einer Auswahlliste. Die Entwürfe der Benutzeroberfläche sind im Anhang A.7: Oberflächenentwürfe auf Seite vii dargestellt.

4.4 Datenmodell

Die in der Anwendung verwendeten Models halten die für die Anwendung benötigten Daten. Dazu gehören die Models *Connection* (Verbindungen) und *Table* (vordefinierte Ergebnistabelle). Diese beiden Entitäten werden in einem ERM im Anhang A.9: ERM der Models Connection und Table auf Seite x dargestellt. Die einzelnen Objekte der Entitäten sollen nach dem Key-Value-Prinzip als Property-Dateien im Dateisystem auf dem lokalen Rechner des Nutzers gespeichert werden. Dabei sollen die Passwörter für die Verbindungen verschlüsselt in den Dateien abgelegt werden. Für die Speicherung der Dateien wird von der Anwendung eine Ordnerstruktur angelegt, damit der Nutzer die Dateien einsehen und vordefinierte Ergebnistabellen mit Kollegen teilen kann.

4.5 Geschäftslogik

Das Projekt soll sich in die Schichten *Model*, *View* und *Controller* aufteilen, wie es im Package-Diagramm im Anhang A.8: Package-Diagramm auf Seite ix gezeigt wird. Die Geschäftslogik wird

in Service- und Util-Klassen implementiert, deren Methoden von den Controller-Klassen aufgerufen werden können. Dabei soll für jeden Anwendungsfall, z. B. die Suche in der Datenbank, eine eigene Controller-Klasse und eine zugehörige Service-Klasse implementiert werden. Util-Klassen stellen Methoden bereit, die für mehrere Anwendungsfälle genutzt werden können. Ein Beispiel für eine Util-Klasse ist eine Klasse zum Speichern und Laden verschiedener Dateien aus dem Dateisystem.

4.6 Maßnahmen zur Qualitätssicherung

Eine wichtige Qualitätsanforderung des HANA Readers ist, dass das Suchen in mehreren Datenbanktabellen gleichzeitig mit der Anwendung mindestens 25% schneller sein soll als mit dem bisher genutzten Plug-In für Eclipse. Dazu wird für mehrere Suchdurchläufe mit unterschiedlich vielen Tabellen auf dem HANA Reader eine Zeitmessung gestartet, um den Durchschnitt zu errechnen. Dies kann mit dem für das Eclipse IDE Plug-In festgestellten Wert von 90 Sekunden pro Tabelle verglichen werden. Eine weitere Qualitätsanforderung hinsichtlich der Sicherheit der Anwendung ist, dass die Passwörter für die Verbindungen verschlüsselt im Dateisystem abgelegt werden sollen. Dazu soll ein Anmeldepasswort eingeführt werden, welches gleichzeitig als Grundlage für einen symmetrischen Schlüssel dient, um die Passwörter mit dem Advanced Encryption Standard-Algorithmus (AES-Algorithmus) zu ver- und entschlüsseln. Das Anmeldepasswort wird bei Einrichtung der Applikation gehasht. Das Eingabepasswort des Nutzers kann mit diesem Hash validiert werden. Nur dann kann der Nutzer die Anwendung verwenden. Um die generelle Funktionalität der Anwendung sicherzustellen, sollen außerdem verschiedene Tests durchgeführt werden. Einerseits sollen dabei manuelle Systemtests durch die Autorin und die Auftraggeber durchgeführt werden, andererseits sollen auch automatische Unit-Tests geschrieben werden.

5 Implementierungsphase

In der Implementierungsphase wurden die Ergebnisse der Entwurfsphase implementiert. Dazu gehört die Datenstruktur, die Benutzeroberfläche und die Geschäftslogik.

5.1 Implementierung der Datenstrukturen

Die entworfene Datenstruktur der Models `Connection` und `Table` wurde als ERM im Anhang A.9: ERM der Models Connection und Table auf Seite x dargestellt. Diese beiden Models wurden als eigene Java-Klassen angelegt, die die Attribute der Entitäten enthalten. Die Klasse des Table-Models enthält zusätzlich ein Connection-Objekt als Attribut. Die Getter- und Setter-Methoden wurden dabei automatisch generiert. Um die Namen der Models in der View an den entsprechenden Stellen (z. B. in der Baumstruktur) anzeigen zu können, wurden die `toString`-Methoden der beiden Models entsprechend überschrieben. Bei der Speicherung der Models wurde sich für die Speicherung der Objekte in Property-Dateien im Dateisystem auf dem lokalen Rechner des Nutzers entschieden. Beispiele mit

Dummy-Daten für diese Dateien finden sich im Anhang A.10: Property-Dateien auf Seite x. Die Objekte des Connection-Models können eindeutig mit dem Namen der Verbindung als Primärschlüssel identifiziert werden. Das Passwort der Verbindung wird verschlüsselt gespeichert. Die Objekte des Table-Models speichern als Fremdschlüssel zur Identifizierung der Connection den Namen der Verbindung. Die Attribute werden als ein String mit Kommata getrennt gespeichert. Table-Objekte können eindeutig mit dem Namen der Tabelle als Primärschlüssel identifiziert werden.

5.2 Implementierung der Benutzeroberfläche

Die Benutzeroberfläche der Applikation wurde mit JavaFX programmiert. Dabei wurde sich gegen die Verwendung eines Scene-Builders und einer FXML-Datei entschieden. An sich kann die Verwendung eines Scene-Builders die Erstellung der Oberfläche erleichtern, allerdings muss die erzeugte FXML-Datei in einigen Fällen manuell angepasst werden. Bei EWE TEL haben nur wenige Mitarbeiter Kenntnis über die Nutzung von FXML. Durch die programmatische Programmierung der Oberfläche wird daher der Zugang, die Wartbarkeit und die Erweiterbarkeit der Applikation sichergestellt. Als Grundlage für die Erstellung der Oberfläche dienten die in Abschnitt 4.3 auf Seite 9 beschriebenen Entwürfe. Screenshots der Oberfläche mit Dummy-Daten befinden sich im Anhang A.11: Screenshots der Anwendung auf Seite xi. Ein Ausschnitt des Quellcodes für die Erstellung des Startfensters ist ebenfalls im Anhang A.12: Ausschnitt aus der Klasse ApplicationGUI zur Erstellung des Startfensters auf Seite xiv zu sehen. In dem Ausschnitt werden Methoden gezeigt, mit denen die Baumstruktur mit den Verbindungen und Ergebnistabellen des Startfensters erstellt wird. Diese Methoden werden auch durch die `update`-Methode im Rahmen des Observer Patterns aufgerufen, um den Teil der Oberfläche mit den geänderten Daten im Model, z. B. beim Anlegen neuer Verbindungen, neu zu zeichnen.

5.3 Implementierung der Geschäftslogik

Die Implementierung der Oberfläche und der Datenstrukturen ermöglicht die Implementierung der Geschäftslogik. Diese ist dabei hauptsächlich in verschiedenen Service- oder Util-Klassen implementiert worden, die von Controller-Klassen aufgerufen werden. Für jede View-Klasse, insbesondere für die Verwaltung der Verbindungen, der vordefinierten Ergebnistabellen und die Suche auf dem Datenbanksystem, wurden je eine Controller- und eine Service-Klasse erstellt. Außerdem gibt es Util-Klassen für die Ver- und Entschlüsselung der Passwörter sowie eine Repository-Klasse für die Interaktionen mit der Datenbank.

5.3.1 Implementierung der Datenbankverwaltung

Die Datenbankverwaltung umfasst das Anlegen, Bearbeiten und Löschen verschiedener Verbindungen und Ergebnistabellen. Die Eingabefenster für das Anlegen und Bearbeiten der Models können aus dem Startfenster heraus geöffnet werden. Dazu werden die entsprechenden Methoden aus der Klasse

`ApplicationController` aufgerufen, die im Anhang A.13: Klasse ApplicationController auf Seite xv gezeigt werden.

Wenn eine Verbindung angelegt und gespeichert werden soll, wird zunächst die Verbindung zur Datenbank geprüft. Die eingegebenen Werte werden ausgelesen und es wird eine Instanz der Klasse `DatabaseConnectionService` erstellt, welche sich ausschnittweise im Anhang A.14: Klasse DatabaseConnectionService (Ausschnitt) auf Seite xvi befindet. Um testweise eine Verbindung aufzubauen, wird die Methode `connectToDatabase` aufgerufen. Dabei kann eine `SQLInvalidAuthorizationSpecExceptionSapDB` oder eine `SQLException` geworfen werden. Diese werden durch das Anzeigen entsprechender Alerts bei Aufruf der Methode zum Speichern einer Verbindung behandelt. Falls alle Eingaben korrekt sind und erfolgreich eine Verbindung zur Datenbank aufgebaut werden konnte, wird die Verbindung als Connection-Objekt in einer ArrayList im Connection-Service hinterlegt und anschließend als Property-Datei im Dateisystem des Nutzers gespeichert.

Für die Speicherung der Verbindung als Property-Datei im Dateisystem wurde die Klasse `SaveAndLoadFromFileSystemService` erstellt, die ausschnittweise im Anhang A.15: Klasse SaveAndLoadFromFileSystemService (Ausschnitt) auf Seite xvii gezeigt wird. Die Methode `saveConnectionAsFile` speichert eine Verbindung im Dateisystem. Bei der Speicherung im Dateisystem wird das Passwort der Verbindung verschlüsselt. Dazu wurde eine Klasse `PasswordEncrypterAndDecrypter` geschrieben, welche sich im Anhang A.16: Klasse PasswordEncrypterAndDecrypter auf Seite xviii befindet. In der Methode `generateKey` wird zu dem Masterpasswort mit dem Secure Hash Algorithm (SHA-256) ein symmetrischer Schlüssel generiert, der in der Methode `encrypt` verwendet wird, um das Passwort mit dem AES-Algorithmus zu verschlüsseln.

Beim Öffnen des HANA Readers werden die Verbindungen geladen, wozu die Methode `loadConnectionsFromFileSystem` der Klasse `SaveAndLoadFromFileSystemService` genutzt wird. Dabei wird das Passwort mit der Methode `decrypt` der Klasse `PasswordEncrypterAndDecrypter` wieder entschlüsselt. Der Ablauf zum Erstellen und Speichern vordefinierter Ergebnistabellen ist ähnlich wie der beschriebene Ablauf zum Anlegen und Speichern neuer Verbindungen. Auch Ergebnistabellen werden im Dateisystem gespeichert und beim Öffnen der Anwendung aus dem Dateisystem geladen.

5.3.2 Implementierung der Suche

Die Hauptfunktion des HANA Readers ist die Suche auf dem SAP HANA Datenbanksystem von EWE. Alle wichtigen Angaben für die Suche wie Datenbanktabelle, Datenbankschema und Verbindung mit Hostname und Port der Datenbank sind in den vordefinierten Ergebnistabellen gespeichert, die der Nutzer in der Anwendung erstellen kann. Hier können auch die Attribute angegeben werden, die in der Ergebnisanzeige zu sehen sein sollen. Um die Suche zu starten, muss der Nutzer mindestens eine Ergebnistabelle und ein dazugehöriges Attribut aus den Combo-Boxen im Startfenster der Anwendung auswählen. Außerdem muss eine Ausprägung angegeben werden, die das ausgewählte Attribut haben soll. Der Nutzer kann zusätzlich den Operator für die Suche ändern und die maximale Anzahl der Ergebniszeilen festlegen. Bevor die Suche gestartet wird, wird zunächst die Verbindung zur Datenbank

getestet, indem aus der Klasse `DatabaseConnectionService` die Methode `testConnection` aufgerufen wird. Falls die Verbindung erfolgreich war, wird die Suche gestartet. Die entsprechenden Werte für die Suche werden ausgelesen und es wird die Methode `searchInDatabase` aufgerufen. Anschließend wird ein SQL-Statement generiert, das z. B. so aussehen könnte:

```
SELECT TOP 100 "Kundennr", "Vertragsnr", "Produktnr"
FROM "EWETEST"."Vertrag" WHERE "Kundennr" = '02229799';
```

Von der Datenbank wird ein `ResultSet` zurückgegeben. Dieses kann ausgelesen werden, um daraus eine `TableView` für die Ergebnisanzeige zu erstellen. Falls in mehreren Tabellen gleichzeitig gesucht werden soll, werden die Methoden in einer Schleife für jede Tabelle aufgerufen, um für jede Tabelle eine `TableView` zu erstellen. Daraus wird ein `TabbedPane` für die Ergebnisanzeige erstellt.

6 Testphase

Während der Implementierungsphase wurden regelmäßig Tests der Anwendung durchgeführt, um die generelle Funktionalität sicherzustellen. Dabei wurden auch fehlerhafte Daten eingegeben und überprüft, ob diese durch eine entspechende Fehlerbehandlung abgefangen werden. Beispielsweise wurde beim Anlegen einer Verbindung zur Datenbank die `SQLInvalidAuthorizationSpecExceptionSapDB`, die geworfen wird, wenn der Nutzer nicht authentifiziert werden konnte, in einem anderen Catch-Block abgefangen als die Superklasse `SQLException`. Dadurch kann dem Nutzer eine entsprechende Fehlermeldung gezeigt werden. Dies wurde durch fehlerhafte Eingaben des Benutzernamens und des Passwortes getestet. Nach wesentlichen Implementationsschritten wurde die Anwendung auch von den Auftraggebern getestet, um mögliche Fehler in verschiedenen Szenarien aufzudecken und Feedback einzuholen. Außerdem wurden Unit-Tests mit Hilfe des JUnit 5-Testframeworks für Java geschrieben. Die Testklasse `PasswordEncrypterAndDecrypterTest` für die bereits in Abschnitt 5.3 beschriebene Klasse `PasswordEncrypterAndDecrypter` befindet sich im Anhang A.17: Testklasse PasswordEncrypterAndDecrypterTest auf Seite xix. Dort ist auch der Aufruf des Tests in Eclipse zu sehen.

7 Ergebnisanalyse

Im Rahmen eines Abnahmetests durch die Auftraggeber konnte sichergestellt werden, dass der HANA Reader alle funktionalen Anforderungen erfüllt. Auch die Qualitätsanforderung zur Verschlüsselung der Passwörter wurde erfüllt. Um zu prüfen, ob die Anwendung min. 25% performanter ist als das bisher genutzte Plug-In, wurden mit dem HANA Reader mehrere Testdurchläufe mit jeweils einer und zehn Tabellen gemacht und die Zeit gestoppt. Ein Ausschnitt aus der Tabelle mit den gemessenen Zeiten befindet sich im Anhang A.18: Zeitmessung des HANA Readers auf Seite xxi. Dabei wurde die Auswahlzeit der Ergebnistabellen und Attribute nicht mitgemessen, diese wird in der Berechnung auf 60 Sekunden festgesetzt. Durchschnittlich ergeben sich 1,475 Sekunden Wartezeit für eine Tabelle und

3,097 Sekunden für zehn Tabellen. Verglichen mit den 90 Sekunden des Plug-Ins für eine Tabelle und 15 Minuten für zehn Tabellen ist der HANA Reader deutlich performanter als das Plug-In. Die Qualitätsanforderung hinsichtlich der Performance ist damit ebenfalls erfüllt. Da der HANA Reader während aller Testdurchläufe nicht einmal abgestürzt ist, kann auch die dritte Qualitätsanforderung hinsichtlich der Verlässlichkeit des HANA Readers als erfüllt betrachtet werden. Aus den gemessenen Zeiten geht auch hervor, dass das Projekt minimal wirtschaftlicher ist, als ursprünglich angenommen. Es ergeben sich bei 32 Durchführungen im Monat Kosten von $32 \cdot 63,10\ \text{s} \cdot$ [redacted] $= 61,70\ €$ Kosten/Monat. Bei einer Einsparung von $(924\ € - 61,70\ €) \cdot 12 = 10.347,60\ €$/Jahr ergibt sich eine Amortisationszeit von $\frac{2.593,40\ €}{10.347,60\ €/\text{Jahr}} \approx 0,251$ Jahre ≈ 3 Monate.

8 Dokumentation

Im Rahmen des Projektes wurden eine Benutzerdokumentation und eine Enwicklerdokumentation erstellt. Die Benutzerdokumentation wurde mit Microsoft Office Word verfasst. Ein Ausschnitt aus der erstellten Benutzerdokumentation befindet sich im Anhang A.21: Benutzerdokumentation auf Seite xxiv. Die Entwicklerdokumentation wurde mittels JavaDoc automatisch generiert. Ein beispielhafter Auszug aus der Dokumentation der Klasse `PasswordEncrypterAndDecrypter` findet sich im Anhang A.20: Entwicklerdokumentation auf Seite xxii.

9 Übergabe des Projektes

Im Rahmen der Projektübergabe wurde gemeinsam mit dem Auftraggeber ein Abnahmetest durchgeführt, um zu prüfen, ob alle im Lastenheft dokumentierten Anforderungen an das Projekt umgesetzt wurden. Dabei konnte festgestellt werden, dass das Projektergebnis abgenommen werden kann, da es keine Mängel aufweist. Dies wurde in dem Abnahmeprotokoll im Anhang A.22: Abnahmeprotokoll auf Seite xxv dokumentiert.

10 Fazit

Es konnten alle ermittelten Anforderungen aus der Analysephase umgesetzt werden, daher wurde das Projektziel erreicht. Die zeitliche Planung des Projektes im Anhang A.2: Detaillierte Zeitplanung auf Seite ii konnte bis auf wenige Abweichungen eingehalten werden. Auch die für die Betreuer eingeplanten Zeiten konnten eingehalten werden, sodass sich die Projektkosten nicht geändert haben. Die Tabelle im Anhang A.19: Soll-/Ist-Vergleich der Zeitplanung auf Seite xxi zeigt die Differenzen der geplanten und tatsächlichen Zeiten. Dadurch, dass während der Implementierungsphase bereits alle Klassen und Methoden mit JavaDoc-Kommentaren versehen wurden, mussten in der Dokumentationsphase nur ein paar kleine Änderungen vorgenommen werden, bevor die Entwicklerdokumentation automatisch generiert wurde. Auch in der Testphase wurde eine Stunde Pufferzeit gewonnen, da während der

Implementierungsphase bereits die meisten Fehler aufgefallen sind und behoben werden konnten. Die Implementierungsphase fiel hauptsächlich aufgrund der Verschlüsselung der Passwörter länger aus als ursprünglich geplant. Außerdem wurde mehr Zeit für die Ergebnisanalyse und Überprüfung der Qualitätsanforderungen benötigt.

10.1 Gewonnene Erkenntnisse

Während der Umsetzung des Projektes konnte ich meine Fähigkeiten im Umgang mit der Programmiersprache Java verbessern. Ich habe gelernt, dass der Einsatz verschiedener Entwurfsmuster wie MVC sinnvoll ist und die Arbeit erleichtert. Durch die Code-Reviews und das Feedback meines Betreuers habe ich gelernt, dass es bei der Entwicklung größerer Programme vieles zu beachten gibt und eine Planung der Handlungsschritte im Vorraus sinnvoll ist. Für mein Projekt eignete sich das Wasserfallmodell sehr gut, dennoch war es sinnvoll, dass ich in Zusammenarbeit mit den Betreuern Aspekte eines inkrementellen Vorgehens eingebracht habe, damit einige wechselnde Anforderungen und zusätzliche Funktionen beachtet werden konnten. Gerade im Rahmen größerer Projekte erkenne ich, dass es sinnvoller ist, inkrementelle, agile Vorgehensweisen einzusetzen, um dem Kunden regelmäßig eine erweiterte, verbesserte Software zur Verfügung zu stellen. Dadurch kann flexibler auf Anforderungen eingegangen werden.

10.2 Ausblick

Der HANA Reader soll nun über einen längeren Zeitraum produktiv eingesetzt werden, um die Funktionen zu testen und die Vorteile zu erkennen. Anschließend kann darüber entschieden werden, ob er langfristig im Unternehmen eingesetzt werden soll und welche anderen Abteilungen von der Anwendung profitieren könnten. Ebenso ist es denkbar, dass weitere Funktionen zum HANA Reader hinzugefügt werden sollen. Bisher wurde bereits die Implementierung einer Filterfunktion oder die Weitersuche nach Werten in der Ergebnisanzeige angesprochen.

Literaturverzeichnis

EWE AG 2021a

EWE AG: *Daten und Fakten*. https://www.ewe.com/de/investor-relations/daten-und-fakten. Version: 2021. – letzter Aufruf am: 23.04.2021

EWE AG 2021b

EWE AG: *EWE-Strategie*. https://www.ewe.com/de/konzern/ueber-uns/strategie. Version: 2021. – letzter Aufruf am: 15.04.2021

EWE AG 2021c

EWE AG: *Leitung des EWE-Konzerns*. https://www.ewe.com/de/konzern/ueber-uns/unternehmensleitung. Version: 2021. – letzter Aufruf am: 15.04.2021

Fittkau und Ruf 2008

Fittkau, Thomas ; Ruf, Walter: *Ganzheitliches IT-Projektmanagement*. München : Oldenbourg Wissenschaftsverlag, 2008. – ISBN 978–3–486–58567–4

Goll 2014

Goll, Joachim: *Architekur- und Entwurfsmuster der Softwaretechnik*. Bd. 2. Wiesbaden : Springer Vieweg, 2014. – ISBN 978–3–658–05532–5

A Anhang

A.1 Organigramm

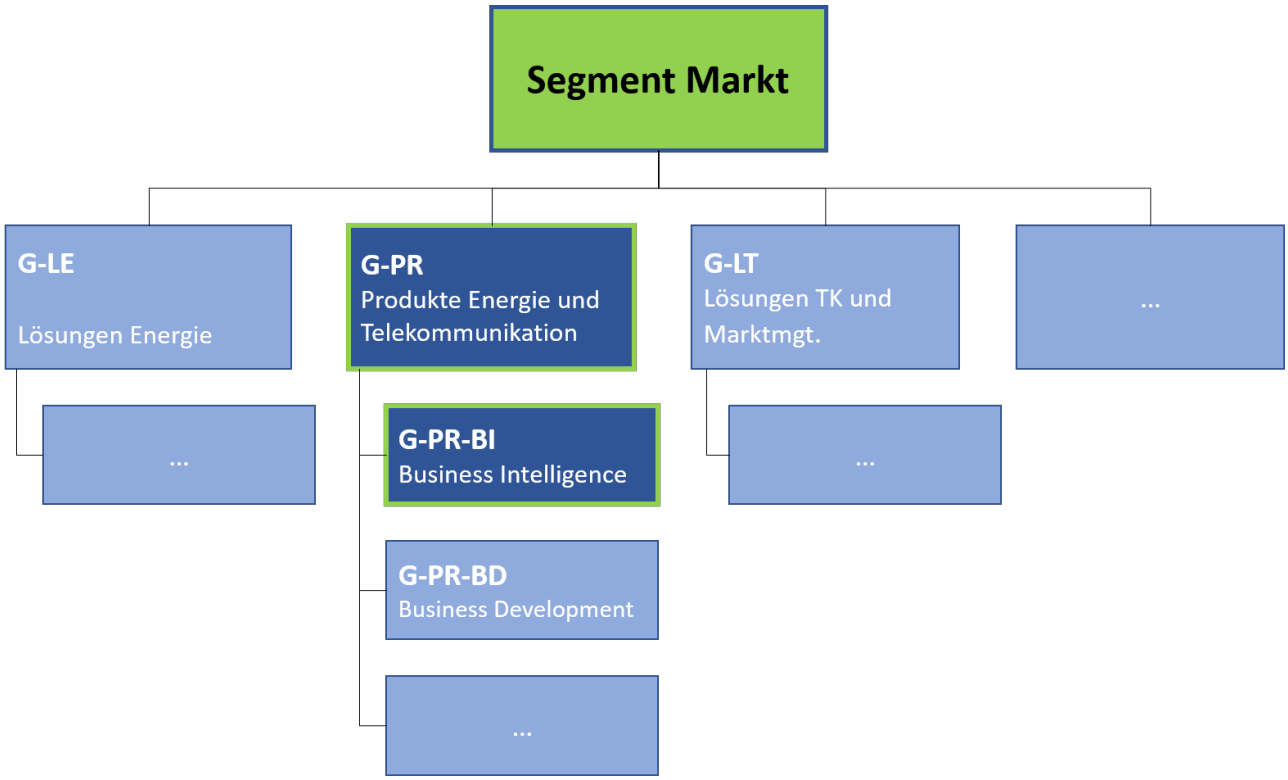


Abbildung 1: Ausschnitt des Organigramms

A.2 Detaillierte Zeitplanung

Analysephase	**6 h**
1. Ist-Analyse durchführen	2 h
2. Soll-Analyse durchführen	2 h
3. Anforderungen erheben	1 h
4. Anforderungen dokumentieren	1 h
Entwurf	**10 h**
1. Entwurf der Geschäftslogik	4 h
2. Entwurf der Oberfläche	4 h
3. Entwurf der Test-/Anwendungsfälle	2 h
Implementierung	**34 h**
1. Implementierung der Oberfläche	10 h
2. Implementierung der Datenbankverwaltung	8 h
3. Implementierung der Suche	6 h
4. Implementierung der Ergebnisanzeige	10 h
Testphase	**6 h**
1. Durchführung von Tests	2 h
2. Fehlerbehebung	4 h
Ergebnisanalyse	**6 h**
1. Nutzenanalyse	2 h
2. Wirtschaftlichkeitsanalyse	2 h
3. Überprüfung der Anforderungserfüllung	1 h
4. Überprüfung der Wirtschaftlichkeitsanalyse	1 h
Dokumentation	**7 h**
1. Erstellen der Benutzerdokumentation	3 h
2. Erstellen der Entwicklerdokumentation	4 h
Übergabe des Projektes	**1 h**
Gesamt	**70 h**

Tabelle 2: Vollständige Zeitplanung

A.3 Amortisationsdauer

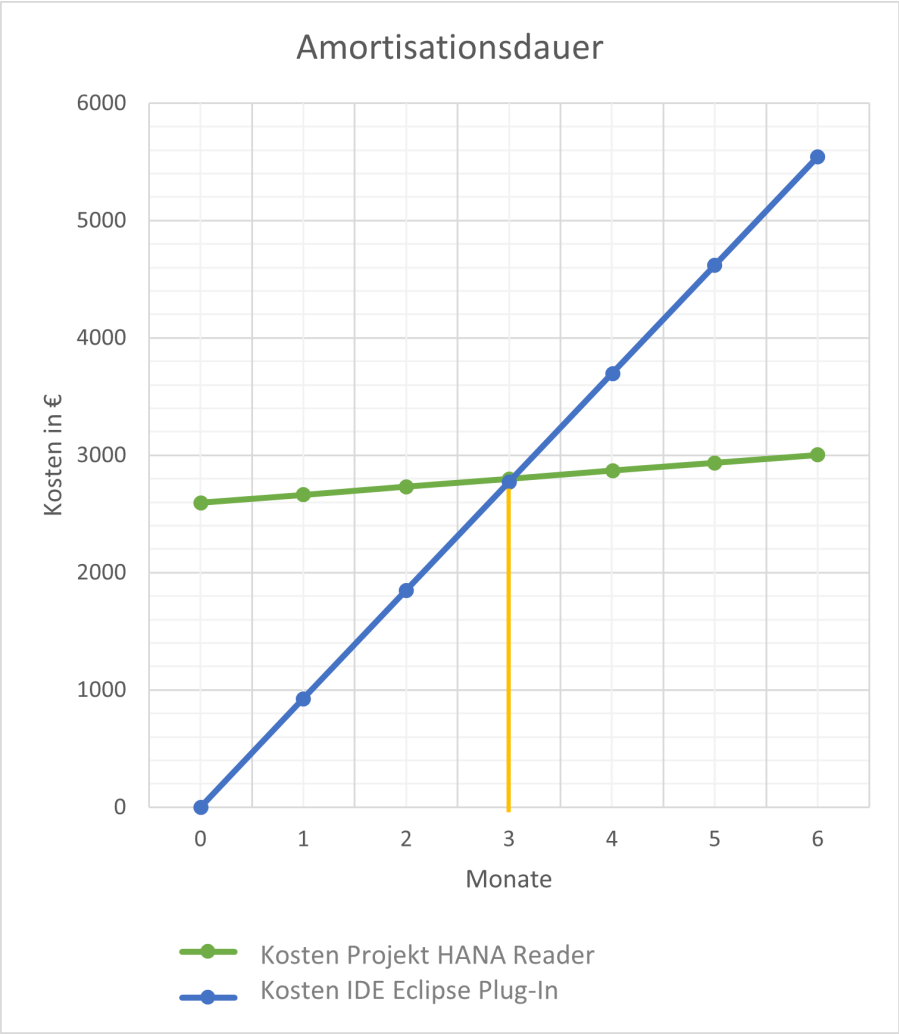


Abbildung 2: Graphische Darstellung der Amortisationsdauer

A.4 Use Case-Diagramm

Abbildung 3: Use Case-Diagramm

A.5 Ereignisgesteuerte Prozesskette

Es werden keine Organisationseinheiten oder Dokumente dargestellt.

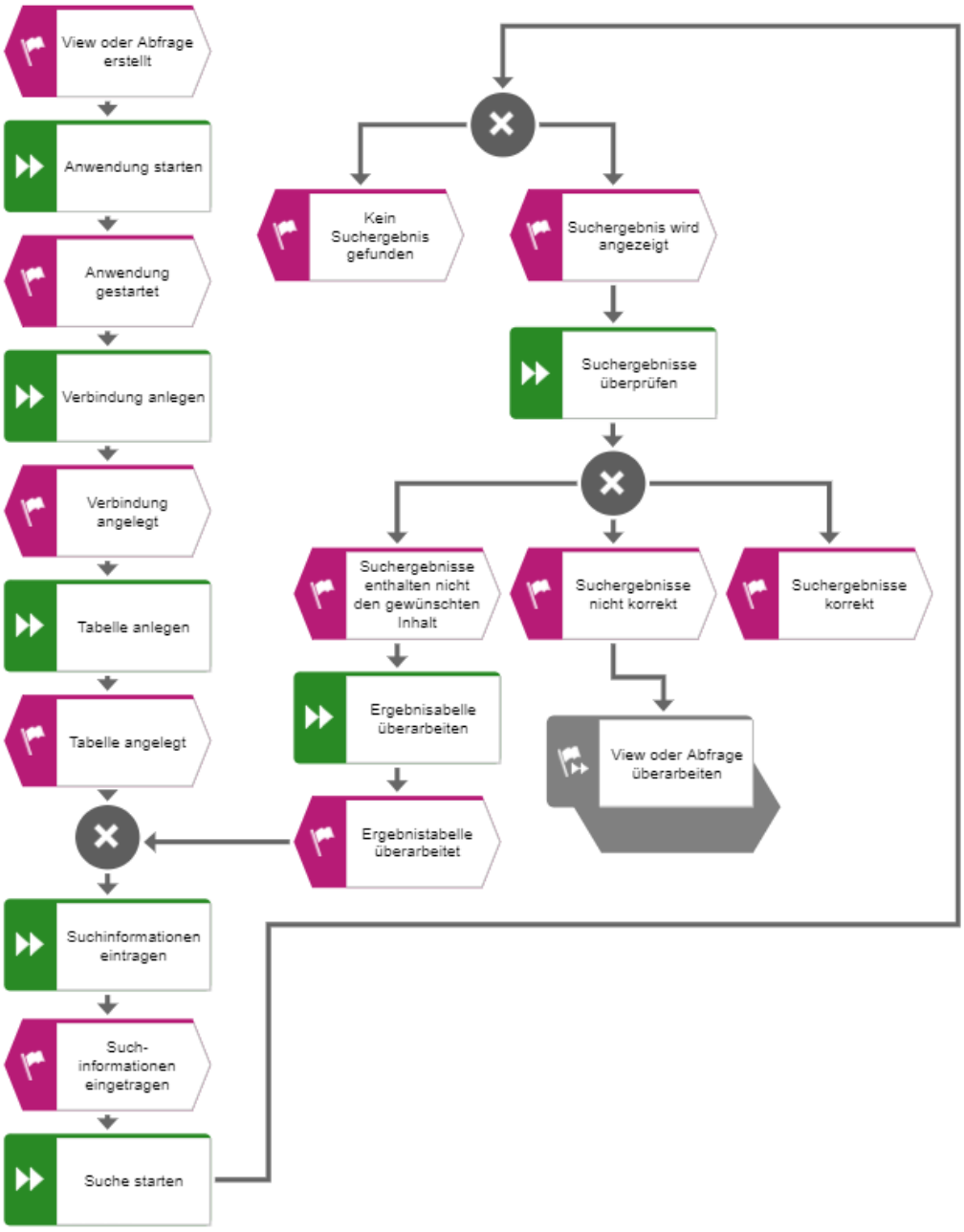


Abbildung 4: Ereignisgesteuerte Prozesskette

A.6 Lastenheft (Auszug)

Es folgt ein Auszug aus dem Lastenheft mit Fokus auf die funktionalen Anforderungen:

Die Anwendung muss folgende Anforderungen erfüllen:

1. Die Anwendung muss Verbindungen zu Datenbanken eines SAP HANA Datenbanksystems durch Eingabe von Hostname, Port, Username und Passwort verwalten können.
   1.1. Die Anwendung muss das Anlegen von Verbindungen ermöglichen.
   1.2. Die Anwendung muss das Bearbeiten von Verbindungen ermöglichen.
   1.3. Die Anwendung muss das Löschen von Verbindungen ermöglichen.
   1.4. Die Anwendung muss vor der Speicherung der Verbindung die Verbindungsdaten überprüfen und im Fehlerfall eine entsprechende Meldung anzeigen.
   1.5. Die Anwendung muss die angelegten Verbindungen nach Schließen der Anwendung lokal im Dateisystem speichern.
2. Die Anwendung muss durch das Erstellen von vordefinierten Ergebnistabellen eine Einschränkung der Suchergebnisse auf ausgewählte Attribute bereitstellen.
3. Die Anwendung muss die vordefinierten Ergebnistabellen, die zu den erstellten Verbindungen gehören, verwalten.
   3.1. Die Anwendung muss das Anlegen von Ergebnistabellen innerhalb der Anwendung ermöglichen.
   3.2. Die Anwendung muss automatisch eine Liste mit den Attributen aus der ausgewählten Datenbanktabelle anzeigen. Darüber müssen die Attribute der Ergebnistabellen über Checkboxen ausgewählt werden können. Außerdem muss die Anwendung eine komfortable Möglichkeit bereitstellen, in der Attributliste alle oder keines der Attribute auszuwählen.
   3.3. Die Anwendung muss das Bearbeiten von Ergebnistabellen innerhalb der Anwendung ermöglichen.
   3.4. Die Anwendung muss das Löschen von Ergebnistabellen innerhalb der Anwendung ermöglichen.
   3.5. Die Anwendung muss eine Möglichkeit des Teilens der Ergebnistabellen mit anderen Nutzern der Anwendung bereitstellen.
4. Die Anwendung muss eine Suchfunktion bereitstellen, die das Suchen nach bestimmten Ausprägungen in mehreren Tabellen gleichzeitig mit Angabe eines Vergleichsoperators (=, !=, >, >=, <=, <, like, not like) ermöglicht. Dabei müssen in der Anwendung eine Ergebnistabelle und ein Attribut angegeben werden.
5. Die Anwendung muss die Suchergebnisse als vordefinierte Ergebnistabelle ausgeben.
6. Die Anwendung muss die Möglichkeit der Einschränkung der Suchergebnisse auf einen Wunschwert der maximal angezeigten Ergebniszeilen bereitstellen.

A.7 Oberflächenentwürfe

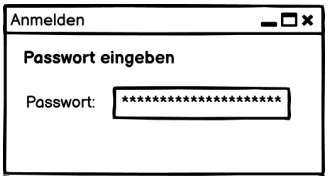


Abbildung 5: Entwurf Anmeldedialog

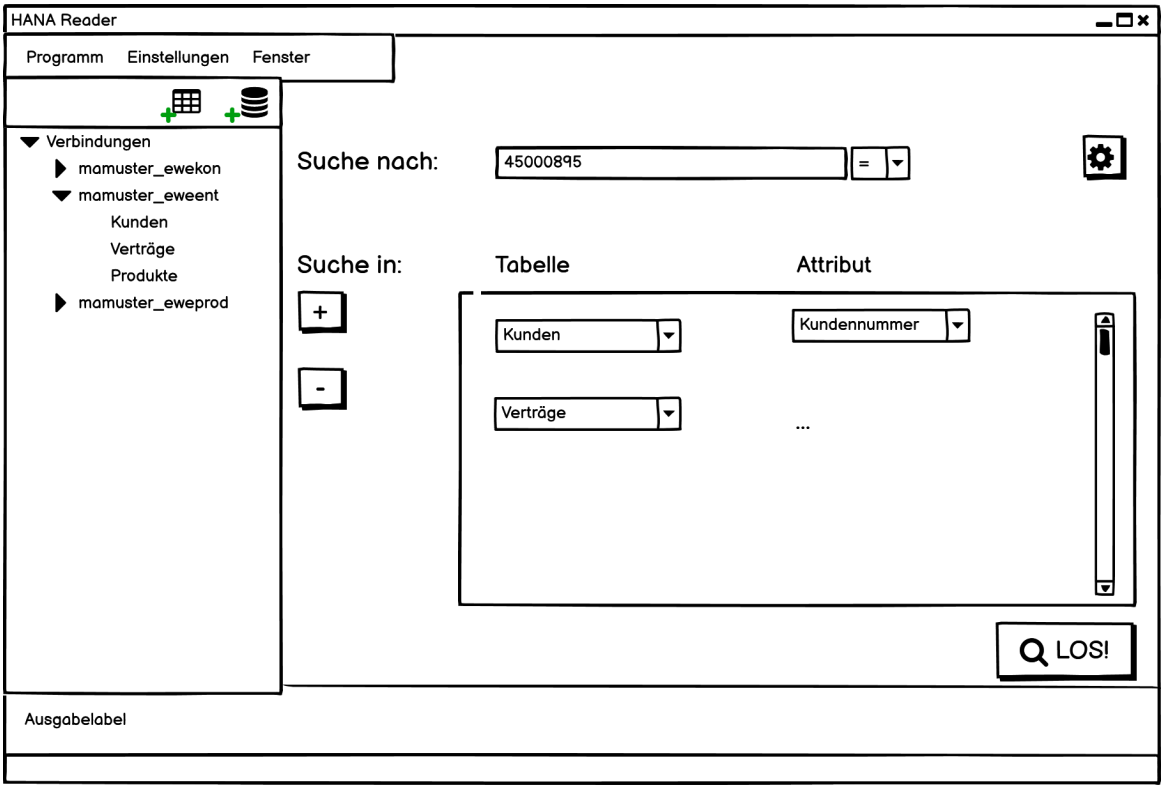


Abbildung 6: Entwurf Startfenster der Applikation

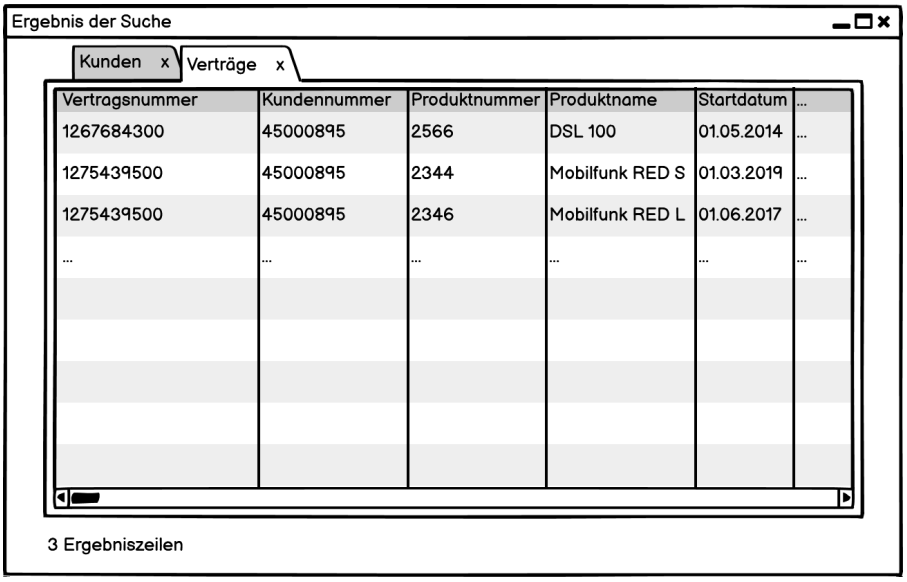


Abbildung 7: Entwurf Ergebnis der Suche

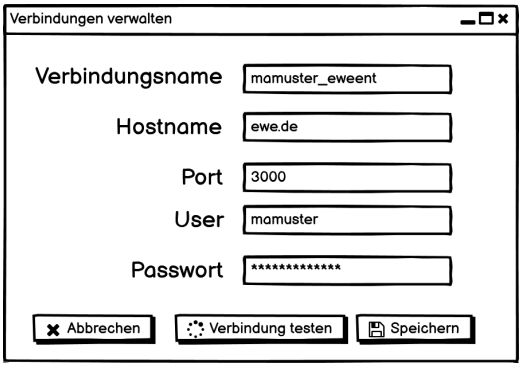


Abbildung 8: Entwurf Verbindungen verwalten

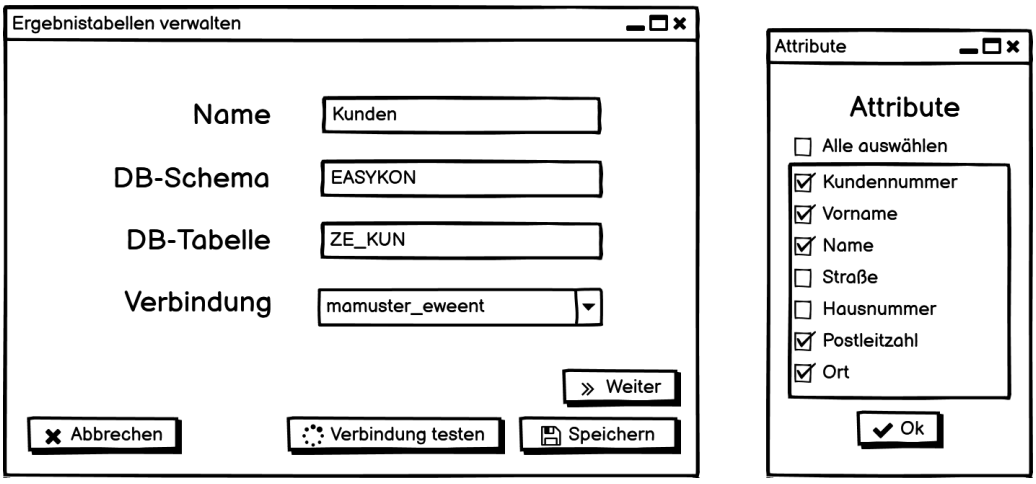


Abbildung 9: Entwurf vordefinierte Ergebnistabellen verwalten

A.8 Package-Diagramm

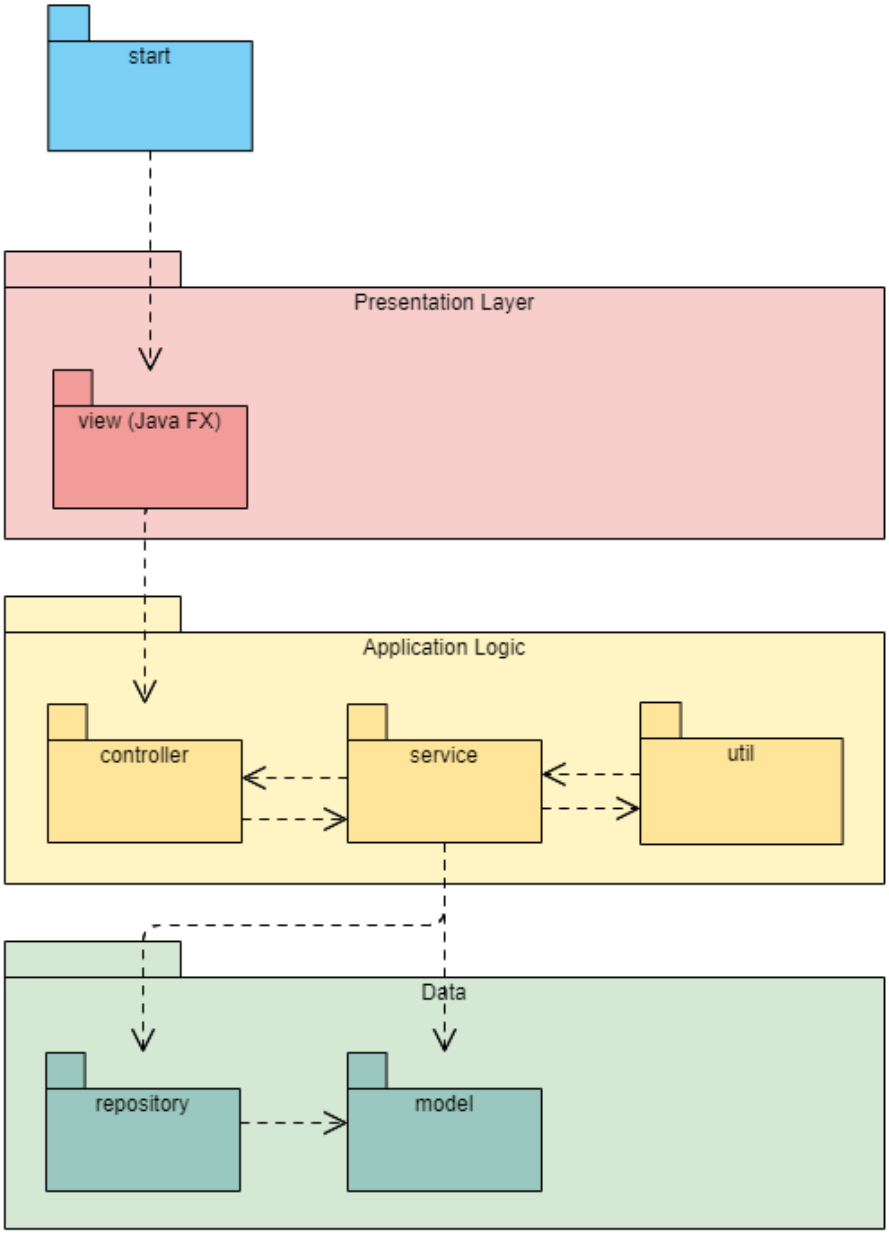


Abbildung 10: Package-Diagramm

A.9 ERM der Models Connection und Table

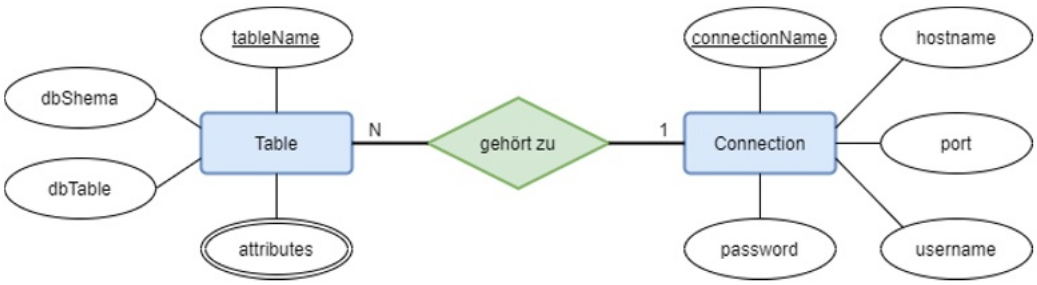


Abbildung 11: ER-Modell der Models Connection und Table

A.10 Property-Dateien

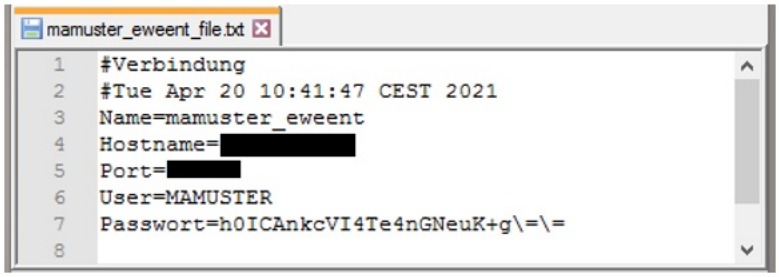


Abbildung 12: Property-Datei für ein Connection-Objekt (Dummy-Daten)

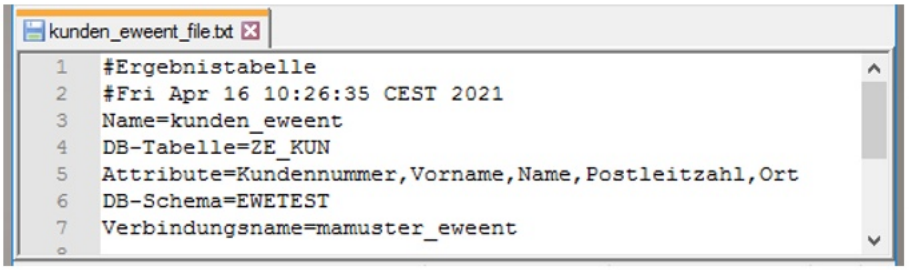


Abbildung 13: Property-Datei für ein Table-Objekt (Dummy-Daten)

A.11 Screenshots der Anwendung

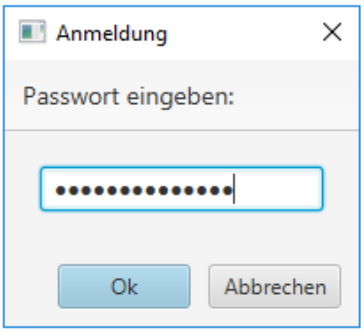


Abbildung 14: Screenshot des Anmeldefensters

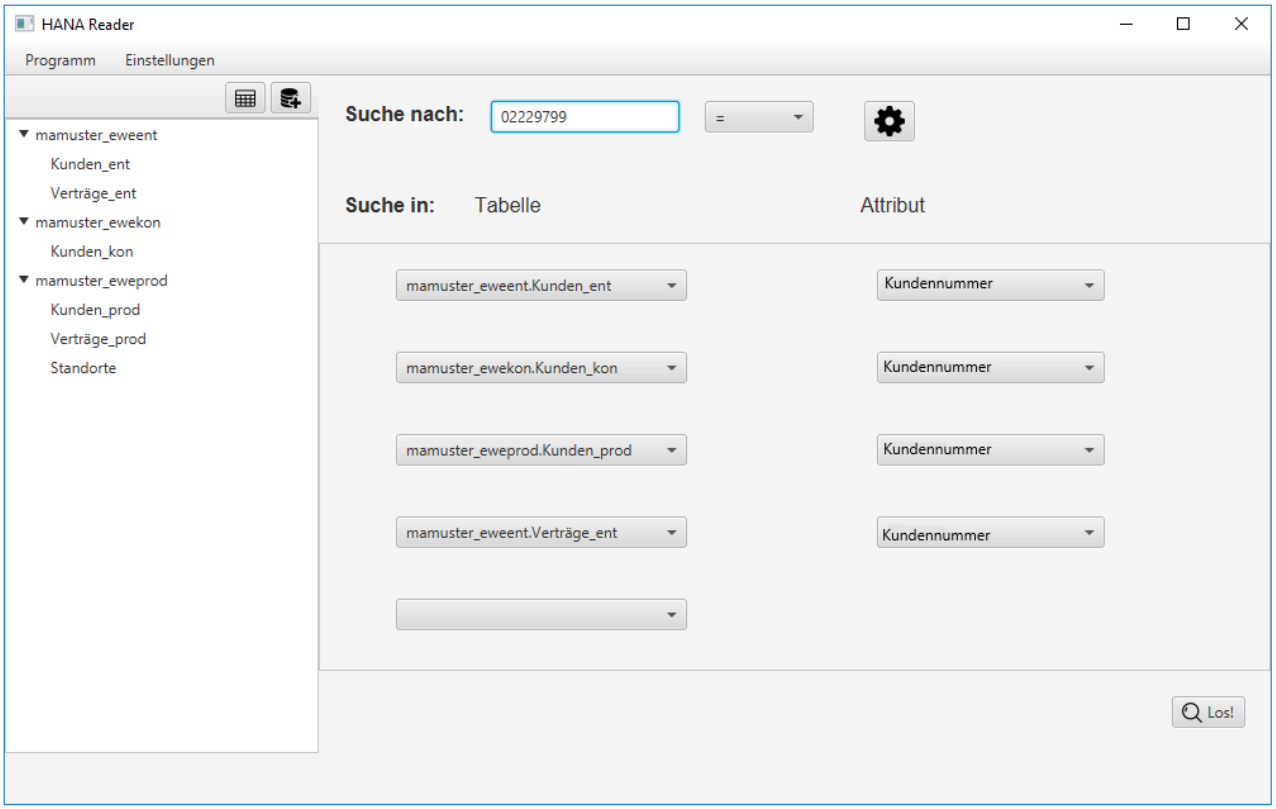


Abbildung 15: Screenshot Startfenster der Anwendung

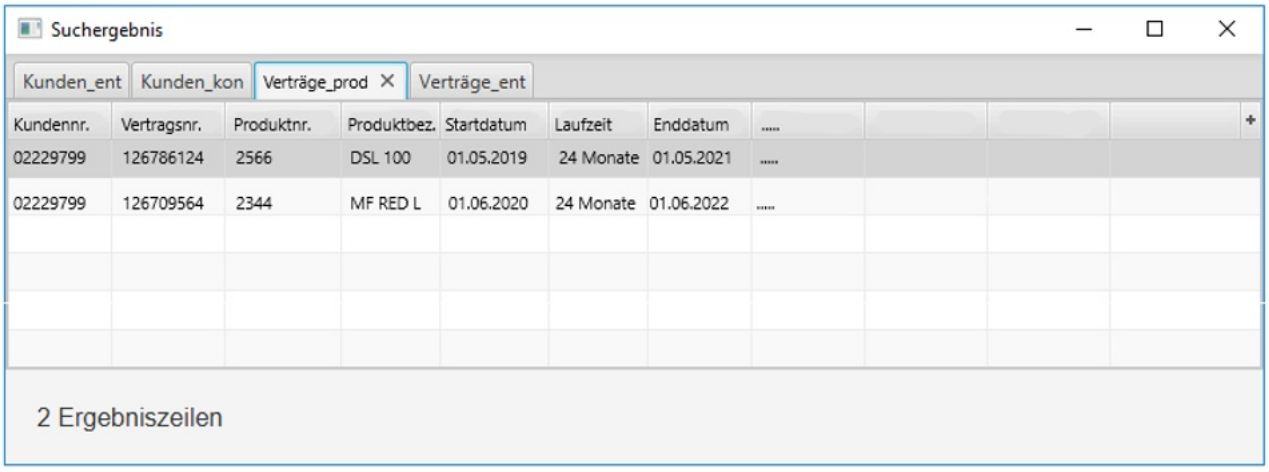


Abbildung 16: Screenshot Ergebnisanzeige der Anwendung

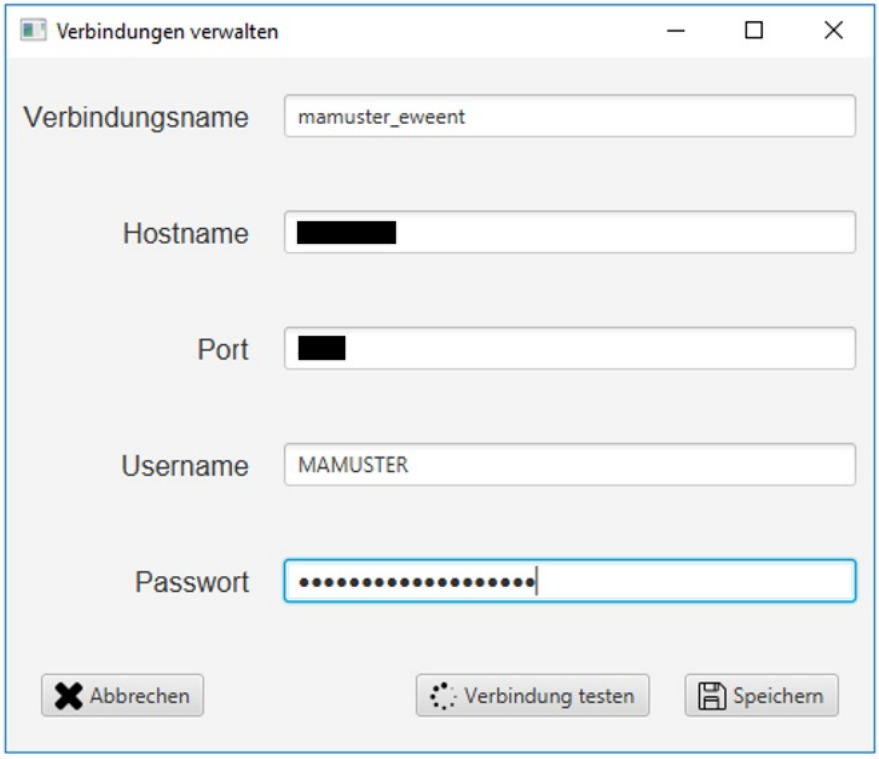


Abbildung 17: Screenshot Verbindungen verwalten

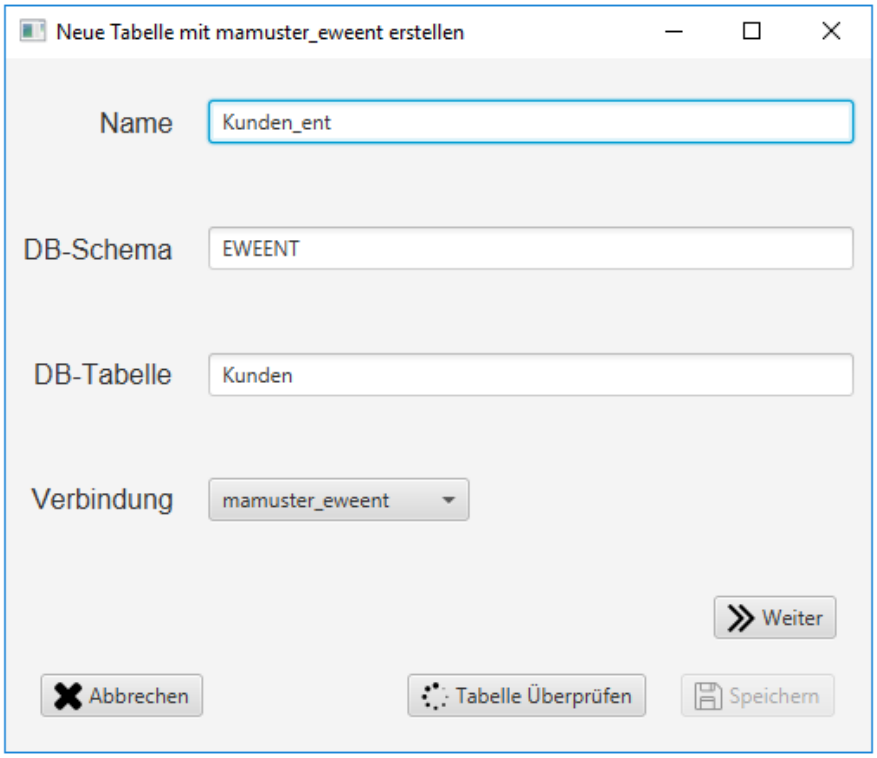


Abbildung 18: Screenshot Ergebnistabellen bearbeiten

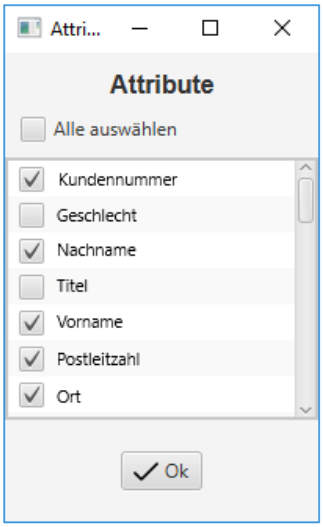


Abbildung 19: Screenshot Attribute auswählen

A.12 Ausschnitt aus der Klasse ApplicationGUI zur Erstellung des Startfensters

Kommentare werden nicht angezeigt.

```
1  public class ApplicationGUI implements Observer {
2    ...
3
4    private void createTreeView() {
5      BorderPane treeViewPane = new BorderPane();
6      ToolBar toolBar = createToolBar();
7      treeViewPane.setTop(toolBar);
8
9      TreeItem<String> rootItem = addConnectionItemsToTreeView();
10     TreeView<String> tree = new TreeView<>();
11
12     tree.setRoot(rootItem);
13     tree.setShowRoot(false);
14     tree.setEditable(true);
15
16     tree.setCellFactory((treeView) -> {
17       return new CustomTreeCell(this.connectionController, this.tableController);
18     });
19
20     treeViewPane.setCenter(tree);
21     this.rootPane.setLeft(treeViewPane);
22   }
23
24   private TreeItem<String> addConnectionItemsToTreeView() {
25     TreeItem<String> rootItem = new TreeItem<>();
26
27     if (!this.connectionList.isEmpty()) {
28       for (Connection connection : this.connectionList) {
29         TreeItem<String> connectionItem = createConnectionTreeItemWithTableItems(connection);
30         rootItem.getChildren().addAll(connectionItem);
31       }
32     }
33     return rootItem;
34   }
35
36   private TreeItem<String> createConnectionTreeItemWithTableItems(Connection connection) {
37     TreeItem<String> connectionItem = new TreeItem<>(connection.getConnectionName());
38
39     for (Table table : this.tableList) {
40       if (connection.equals(table.getConnection())) {
41         TreeItem<String> tableItem = new TreeItem<>(table.getName());
42         connectionItem.getChildren().add(tableItem);
43       }
44     }
45     return connectionItem;
46   }
47
```

```
48    @Override
49    public void update(Observable observable, Object object) {
50      Platform.runLater(() -> {
51        this.connectionList = connectionController.getConnectionService().getConnectionList();
52        this.tableList = tableController.getTableService().getTableList();
53        createTreeView();
54      });
55    }
56    ...
57  }
```

Listing 1: View-Klasse ApplicationGUI (Ausschnitt)

A.13 Klasse ApplicationController

Kommentare und Imports werden nicht angezeigt.

```
1  public class ApplicationController {
2    private ConnectionController connectionController;
3    private TableController tableController;
4
5    public ApplicationController(ConnectionController connectionController, TableController tableController) {
6      this.connectionController = connectionController;
7      this.tableController = tableController;
8    }
9
10   public void openTableGUI() {
11     TableGUI tableGUI = new TableGUI(tableController, true);
12     tableGUI.show();
13   }
14
15   public void openConnectionGUI() {
16     ConnectionGUI connectionGUI = new ConnectionGUI(connectionController, tableController, true);
17     connectionGUI.show();
18   }
19 }
```

Listing 2: Controller-Klasse ApplicationController

A.14 Klasse DatabaseConnectionService (Ausschnitt)

Kommentare, Imports und Variablendeklarationen werden nicht angezeigt.

```
1  public class DatabaseConnectionService {
2    private static final String SELECT_QUERY = "Select TOP %d %s from \"%s\".\"%s\" where \"%s\" %s '%s';";
3    private static final String DATABASE_URL_PATTERN = "jdbc:sap://%s:%s/?autocommit=false";
4     ...
5    public DatabaseConnectionService(String hostname, String port, String user, String password) {
6      this.saveAndLoadFromFileSystemService = new SaveAndLoadFromFileSystemService();
7      this.hostname = hostname;
8      this.port = port;
9      this.user = user;
10     this.password = password;
11     this.connectionSuccess = false;
12     this.authenticationSuccess = true;
13
14     this.databaseUrl = String.format(DATABASE_URL_PATTERN, this.hostname, this.port);
15   }
16
17   public void connectToDatabase() throws SQLInvalidAuthorizationSpecExceptionSapDB, SQLException{
18     this.connection = DriverManager.getConnection(this.databaseUrl, this.user, this.password);
19     this.connection.setReadOnly(true);
20   }
21
22   public ResultSet searchInDatabase(String searchString, String operator, String dbSchema, String tableName,
23       String attribute, String attributeString) throws SQLException {
24     int numberOfRows = this.saveAndLoadFromFileSystemService.loadSearchSettingsFromFile();
25     if (this.connection != null) {
26       Statement stmt = this.connection.createStatement();
27       String sql = String.format(SELECT_QUERY, numberOfRows, attributeString,
28           dbSchema, tableName, attribute, operator, searchString);
29       ResultSet resultSet = stmt.executeQuery(sql);
30       return resultSet;
31     }
32     return null;
33   }
34    ...
35 }
```

Listing 3: Repository-Klasse DatabaseConnectionService (Ausschnitt)

A.15 Klasse SaveAndLoadFromFileSystemService (Ausschnitt)

Kommentare und Imports werden nicht angezeigt.

```
1  public class SaveAndLoadFromFileSystemService {
2
3    private static final String DIRPATH_CONNECTION = "config//connections//";
4
5    ...
6
7    public void saveConnectionAsFile(Connection connection, String masterPassword)
8        throws FileNotFoundException, IOException, Exception {
9      String dirPath = DIRPATH_CONNECTION;
10     String fileName = connection.getConnectionName() + "_file.prop";
11     Path filePath = Paths.get(dirPath + fileName);
12     Files.createDirectories(filePath.getParent());
13     File file = new File(dirPath + fileName);
14
15     FileOutputStream fos = new FileOutputStream(file);
16     Properties prop = new Properties();
17
18     prop.setProperty("Name", connection.getConnectionName());
19     prop.setProperty("Hostname", connection.getHostname());
20     prop.setProperty("Port", connection.getPort());
21     prop.setProperty("User", connection.getUser());
22     prop.setProperty("Passwort", this.passwordEncrypterAndDecrypter.encrypt(connection.getPassword(),
           masterPassword));
23     prop.store(fos, "Verbindung");
24     fos.close();
25   }
26
27   public ArrayList<Connection> loadConnectionsFromFileSystem(String masterPassword)
28       throws FileNotFoundException, IOException, Exception {
29     ArrayList<Connection> connections = new ArrayList<>();
30     File dir = new File(DIRPATH_CONNECTION);
31     File[] files = dir.listFiles();
32     Properties props = new Properties();
33
34     if (files != null) {
35       for (File file : files) {
36         FileInputStream fis = new FileInputStream(file.getAbsolutePath());
37         props.load(fis);
38         fis.close();
39         Connection connection = getConnectionFromProperties(props, masterPassword);
40         connections.add(connection);
41       }
42     }
43     return connections;
44   }
45
46
```

```
47    private Connection getConnectionFromProperties(Properties props, String masterPassword) throws Exception {
48      String connectionName = props.getProperty("Name");
49      String hostname = props.getProperty("Hostname");
50      String port = props.getProperty("Port");
51      String username = props.getProperty("User");
52      String password = this.passwordEncrypterAndDecrypter.decrypt(props.getProperty("Passwort"),
53        masterPassword);
54      Connection connection = new Connection(hostname, port, username, password, connectionName);
55      return connection;
56    }
57    ...
58  }
```

Listing 4: Service-Klasse SaveAndLoadFromFileSystemService (Ausschnitt)

A.16 Klasse PasswordEncrypterAndDecrypter

Kommentare und Imports werden nicht angezeigt.

```
1  public class PasswordEncrypterAndDecrypter {
2
3    public SecretKeySpec generateKey(String masterPassword) throws Exception {
4      byte[] keyBytes = masterPassword.getBytes("UTF-8");
5      MessageDigest shaMessageDigest = MessageDigest.getInstance("SHA-256");
6      keyBytes = shaMessageDigest.digest(keyBytes);
7      keyBytes = Arrays.copyOf(keyBytes, 16);
8      SecretKeySpec secretKeySpec = new SecretKeySpec(keyBytes, "AES");
9      return secretKeySpec;
10   }
11
12   public String encrypt(String password, String masterPassword) throws Exception {
13     SecretKeySpec secretKeySpec = generateKey(masterPassword);
14     Cipher cipher = Cipher.getInstance("AES");
15     cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec);
16     byte[] encryptedByteArray = cipher.doFinal(password.getBytes("UTF-8"));
17     BASE64Encoder encoder = new BASE64Encoder();
18     String encryptedPassword = encoder.encode(encryptedByteArray);
19     return encryptedPassword;
20   }
21
22   public String decrypt(String encryptedPassword, String masterPassword) throws Exception {
23     SecretKeySpec secretKeySpec = generateKey(masterPassword);
24     BASE64Decoder decoder = new BASE64Decoder();
25     byte[] cryptedBytes = decoder.decodeBuffer(encryptedPassword);
26     Cipher cipher = Cipher.getInstance("AES/CBC/NoPadding");
27     cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, new IvParameterSpec(new byte[16]));
28     byte[] cipherData = cipher.doFinal(cryptedBytes);
29     String decryptedPassword = new String(cipherData, "UTF-8");
30     return decryptedPassword.trim();
```

```
31    }
32  }
```

Listing 5: Util-Klasse PasswordEncrypterAndDecrypter

A.17 Testklasse PasswordEncrypterAndDecrypterTest

Kommentare und Imports werden nicht angezeigt.

```
1  public class PasswordEncrypterAndDecrypterTest {
2
3    private PasswordEncrypterAndDecrypter passwordEncrypterAndDecrypter;
4
5    @BeforeEach
6    public void setUp() {
7      passwordEncrypterAndDecrypter = new PasswordEncrypterAndDecrypter();
8    }
9
10   @Test
11   @DisplayName("The encryption of a password should work")
12   public void returnEncryptedPasswordWhenPassingPlaintextPassword() {
13     String plaintextPassword = "password";
14     String exceptedCipheredPassword = "jdRoX8gQ0dpuxz4jhW1Ozw==";
15     String masterPassword = "masterPassword";
16
17     String realCipheredPassword = passwordEncrypterAndDecrypter.encrypt(plaintextPassword, masterPassword);
18
19     assertEquals(exceptedCipheredPassword, realCipheredPassword);
20   }
21
22   @Test
23   @DisplayName("The decryption of a password should work")
24   public void returnPlaintextPasswordWhenPassingEncryptedPassword() {
25     String exceptedPlaintextPassword = "password";
26     String cipheredPassword = "jdRoX8gQ0dpuxz4jhW1Ozw==";
27     String masterPassword = "masterPassword";
28
29     String realPlaintextPassword = passwordEncrypterAndDecrypter.decrypt(cipheredPassword, masterPassword);
30
31     assertEquals(exceptedPlaintextPassword, realPlaintextPassword);
32   }
33
34   @Test
35   @DisplayName("Decrypt a password with wrong masterPassword should return wrong password as plaintext")
36   public void returnWrongPlaintextPasswordPasswordWhenPassingWrongMasterPassword() {
37     String truePlaintextPassword = "password";
38     String wrongMasterPassword = "wrongMasterPassword";
39     String ciphredPassword = "jdRoX8gQ0dpuxz4jhW1Ozw==";
40
```

```
41      String wrongPlaintextPassword = passwordEncrypterAndDecrypter.decrypt(ciphredPassword,
            wrongMasterPassword);
42
43      assertNotEquals(truePlaintextPassword, wrongPlaintextPassword);
44    }
45
46    @Test
47    @DisplayName("Encrypt a password with different masterPasswords should different ciphered Password")
48    public void returnDifferentEncryptedPasswordsWhenPassingDifferentMasterPasswords() {
49      String plaintextPassword = "password";
50      String firstMasterPassword = "masterPassword1";
51      String secondMasterPassword = "masterPassword2";
52
53      String firstCiphredPassword = passwordEncrypterAndDecrypter.encrypt(plaintextPassword, firstMasterPassword)
            ;
54      String secondCiphredPassword = passwordEncrypterAndDecrypter.encrypt(plaintextPassword,
            secondMasterPassword);
55
56      assertNotEquals(firstCiphredPassword, secondCiphredPassword);
57    }
58  }
```

Listing 6: Testklasse PasswordEncrypterAndDecrypterTest

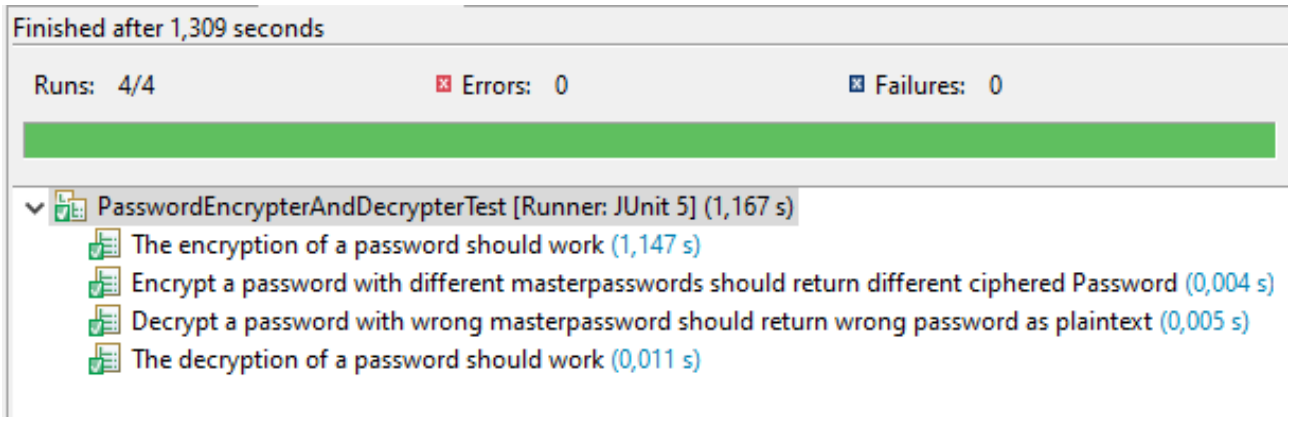


Abbildung 20: Aufruf der Testklasse in Eclipse

A.18 Zeitmessung des HANA Readers

Durchlauf	Dauer eine Tabelle	Dauer zehn Tabellen
1	1,71 s	3,03 s
2	1,24 s	2,89 s
3	1,57 s	3,09 s
4	1,24 s	3,14 s
5	1,33 s	3,4 s
6	1,68 s	2,95 s
7	1,79 s	3,24 s
8	1,22 s	2,99 s
9	1,45 s	3,06 s
10	1,52 s	3,18 s
Durchschnitt	**1,475 s**	**3,097 s**

Tabelle 3: Zeitmessung der Suche mit dem HANA Reader

A.19 Soll-/Ist-Vergleich der Zeitplanung

Phase	Geplant	Tatsächlich	Differenz
Analysephase	10 h	10 h	
Entwurfsphase	10 h	10 h	
Implementierung	34 h	37 h	+3 h
Testphase	6 h	5 h	-1 h
Ergebnisanalyse	2 h	3 h	+1 h
Dokumentation	7 h	4 h	-3 h
Übergabe des Projektes	1 h	1 h	
Gesamt	70 h	70 h	

Tabelle 4: Zeitdifferenz zur Projektplanung

A.20 Entwicklerdokumentation

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

util

Class PasswordEncrypterAndDecrypter

java.lang.Object
util.PasswordEncrypterAndDecrypter

```
public class PasswordEncrypterAndDecrypter
extends java.lang.Object
```

Class to encrypt and decrypt the passwords of a connection to save them in the file system

Author:

alahrens

Constructor Summary

Constructors

Constructor and Description
`PasswordEncrypterAndDecrypter()`

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
`java.lang.String`	`decrypt(java.lang.String encryptedPassword, java.lang.String masterpassword)` Decrypts a crypted password using the generated key from the masterpassword
`java.lang.String`	`encrypt(java.lang.String password, java.lang.String masterpassword)` Encrypts a password using the generated key from the masterpassword
`javax.crypto.spec.SecretKeySpec`	`generateKey(java.lang.String masterpassword)` Generates a key for encrypt and decrypt the passwords, uses the masterpassword

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PasswordEncrypterAndDecrypter

```
public PasswordEncrypterAndDecrypter()
```

Method Detail

generateKey

```
public javax.crypto.spec.SecretKeySpec generateKey(java.lang.String masterpassword)
```

Generates a key for encrypt and decrypt the passwords, uses the masterpassword

Parameters:

masterpassword - the masterpassword to generate the key

Returns:

the generated key

encrypt

```
public java.lang.String encrypt(java.lang.String password,
                                java.lang.String masterpassword)
```

Encrypts a password using the generated key from the masterpassword

Parameters:

password - the text to encrypt

masterpassword - the masterpassword as String

Returns:

the encrypted password

decrypt

```
public java.lang.String decrypt(java.lang.String encryptedPassword,
                                java.lang.String masterpassword)
```

Decrypts a crypted password using the generated key from the masterpassword

Parameters:

encryptedPassword - the encrypted password

masterpassword - the masterpassword as String

Returns:

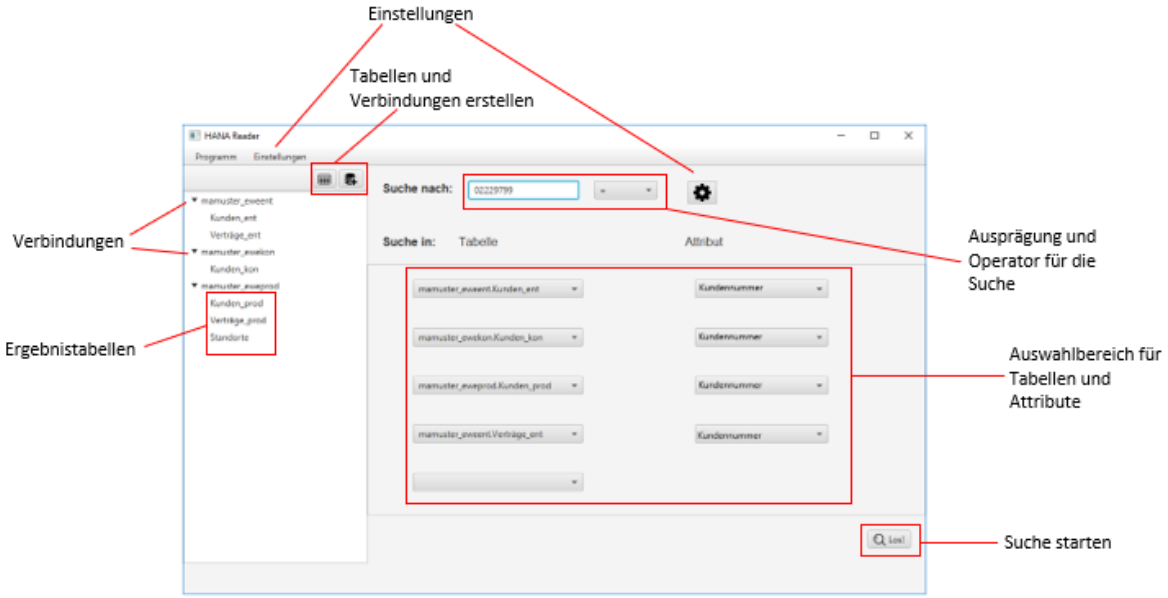
the decrypted password

A.21 Benutzerdokumentation

Ausschnitt aus der Benutzerdokumentation:

Aufbau des Startfensters

Im Startfenster können Verbindungen und Ergebnistabellen eingesehen werden. Außerdem kann die Suche gestartet werden.



Einstellungen

In den Sucheinstellungen wird festgelegt, wie viele Ergebniszeilen maximal nach der Suche angezeigt werden sollen. Das Masterpasswort kann geändert werden. Dazu wird im Verzeichnis eine Datei gespeichert. Sollte dies verloren gehen, kann zur Anmeldung wieder das Initialpasswort verwendet werden.

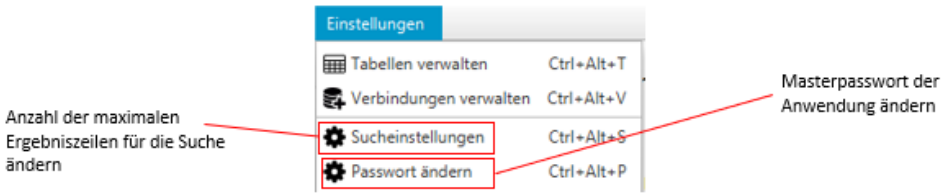


Abbildung 21: Ausschnitt aus der Benutzerdokumentation

A.22 Abnahmeprotokoll

Ausschnitt aus dem Abnahmeprotokoll:

Entwicklung des HANA Readers
Java Applikation zur performanten Abfrage von Daten
eines SAP HANA Datenbanksystems

Abnahmeprotokoll

1. **Projekt:** Entwicklung des HANA Readers – Java Applikation zur performanten Abfrage von Daten eines SAP HANA Datenbanksystems
2. **Auftraggeber:** [redacted]
3. **Auftragnehmer:** Alina Ahrens, EWE NETZ GmbH, Cloppenburger Straße 302, 26133 Oldenburg
4. **Es wurden folgende Leistungen abgenommen:**

   Durchführung eines Proof of Concept zur Suche in mehreren Tabellen eines SAP HANA Datenbanksystem. Dazu zählen:

   - Entwurf und Entwicklung der Oberfläche
   - Entwurf der Geschäftslogik
   - Implementierung der Datenbankverwaltung
   - Implementierung der Suche
   - Systemtest der Anwendung
   - Prüfen der Qualitätsanforderungen
   - Anfertigung einer Benutzer- und Entwicklerdokumentation
   - Durchführung einer Wirtschaftlichkeitsanalyse
5. **Die Ausführung der abgenommenen Leistungen wurde beendet am:**

   Montag, den 03.05.2021
6. **Die Abnahme wurde auf folgender Grundlage geprüft:**
   - Präsentation der Projektergebnisse
   - Durchführung und Besprechung der Testfälle
   - Durchführung einer Ergebnisanalyse

Prüfergebnis Abnahme: **Keine Mängel** ☒ **Mängel** ☐

Wenn Mängel, Auflistung der Mängel:

Auftraggeber (AG): [redacted]

Auftragnehmer (AN):

03.05.2021

X Alina Ahrens

Signiert von: Ahrens, Alina

Dokument Abnahmeprotokoll_HANA_Reader.docx